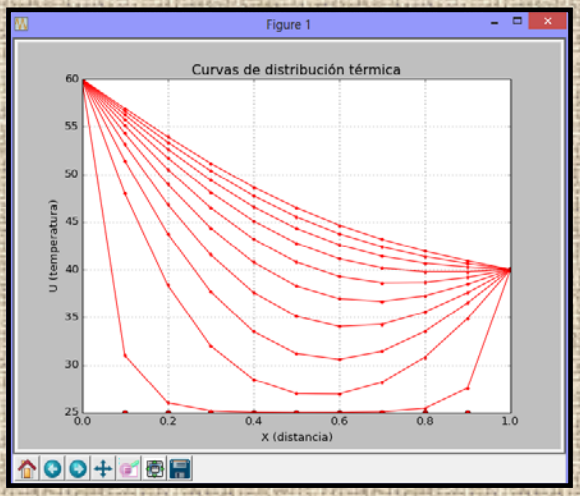
Escuela Superior Politécnica del Litoral

Facultad de Ciencias Naturales y Matemáticas
Departamento de Matemáticas

ANÁLISIS NUMÉRICO BÁSICO

Un enfoque algorítmico con el soporte de Python



Libro digital
Versión 4.4 - 2016

Luis Rodríguez Ojeda

ANÁLISIS NUMÉRICO BÁSICO

Un enfoque algorítmico con el soporte de Python

Prefacio

Esta obra es una contribución dedicada a los estudiantes que toman un curso de Análisis Numérico o Métodos Numéricos en carreras de ingeniería. El pre-requisito es haber tomado los cursos de matemáticas del ciclo básico de nivel universitario y alguna experiencia previa con un programa computacional del tipo MATLAB, Mathematica, o Python para aprovechar el poder computacional en la aplicación de los métodos numéricos de manera efectiva.

El contenido se basa en la experiencia desarrollada en varios años impartiendo los cursos de Análisis Numérico, Métodos Numéricos en las carreras de ingeniería. Esta obra pretende ser un texto complementario para estas materias. La orientación principal del material es su aplicación computacional, sin descuidar los conceptos matemáticos y algorítmicos básicos con los que se fundamentan los métodos numéricos.

Este texto contribuye también a difundir entre los estudiantes el uso del programa Python para cálculos, graficación y manejo matemático simbólico como un soporte común para todos los cursos básicos matemáticos, incluyendo Álgebra Lineal, Cálculo Diferencial e Integral, Ecuaciones Diferenciales, Análisis Numérico y otros, como lo fue anteriormente el lenguaje MATLAB.

Python dispone de funciones en librerías especiales para su aplicación en la solución de una gran cantidad de problemas matemáticos, sin embargo sería equivocado usarlas como una receta sin el conocimiento de sus fundamentos matemáticos y algorítmicos. En este curso se desarrollan algunos métodos alternativos a las que ofrecen las librerías de Python, y en algunos casos, nuevos métodos cuya programación es instructiva para orientar a los estudiantes en el desarrollo de software para matemáticas e ingeniería.

El segundo objetivo principal de esta obra es contribuir al desarrollo de textos digitales en la ESPOL de tal manera que puedan ser usados en línea e interactivamente, aunque también puedan imprimirse, pero tratando de reducir al mínimo el uso de papel para contribuir al cuidado del medio ambiente. Otra ventaja importante de los textos digitales es que pueden ser actualizados y mejorados continuamente sin costos de impresión. El texto ha sido compilado en formato que permite controlar el tamaño de la pantalla y se puede agregar un índice para búsqueda rápida de temas. Se pueden usar facilidades para resaltar y marcar texto, agregar comentarios, notas, enlaces, revisiones, etc.

Esta obra es de uso y distribución libres y estará disponible para uso público en el repositorio digital del Departamento de Matemáticas de la Facultad de Ciencias Naturales y Matemáticas en la página web de la ESPOL: **www.espol.edu.ec**

Luis Rodríguez Ojeda, MSc.
Profesor
2014
lrodrig@espol.edu.ec

CONTENIDO

ANÁLISIS NUMÉRICO
Un enfoque algorítmico con el soporte de Python

1 INTRODUCCIÓN

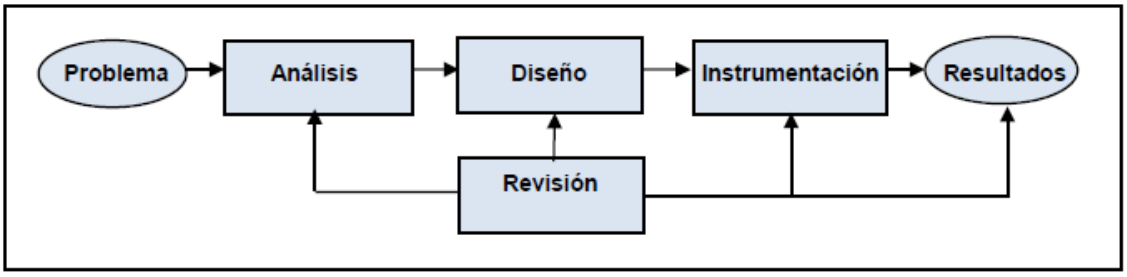
Análisis Numérico es una rama de la matemática cuyo objetivo principal es el estudio de métodos para resolver problemas numéricos complejos. El estudio de estos métodos no es reciente, pero actualmente con el apoyo de la computación se los puede usar con mucha eficiencia en la resolución de problemas que antes no era posible.

En este curso se proporciona a los estudiantes el conocimiento matemático básico para proveer soporte y formalidad a cada uno de los métodos estudiados. Se desarrolla la forma algorítmica de los métodos y finalmente se instrumenta su forma computacional usando la capacidad de cálculo, visualización y programación de Python. Este componente práctico del curso es desarrollado en un laboratorio de computación.

El estudio de cada método se complementa con ejemplos y ejercicios que pueden resolverse con la ayuda de una calculadora. Sin embargo, el objetivo principal del curso es la aplicación de los métodos para obtener respuestas con precisión controlada en la resolución de algunos problemas de ingeniería que por su complejidad requieren usar el computador.

1.1 Resolución de problemas con el computador

Suponer un problema que debemos resolver y que está en nuestro ámbito de conocimiento.



En la etapa de **Análisis** es necesario estudiar y entender el problema. Sus características, las variables y los procesos que intervienen. Asimismo, debemos conocer los datos requeridos y el objetivo esperado. En el caso de problemas de tipo numérico, el resultado de esta etapa será un **modelo matemático** que caracteriza al problema. Por ejemplo, un sistema de ecuaciones lineales.

En la etapa de **Diseño** procedemos a elegir el método numérico apropiado para resolver el modelo matemático. Debe suponerse que no se puede, o que sería muy laborioso, obtener la solución exacta mediante métodos analíticos. Los métodos numéricos permiten obtener

soluciones aproximadas con simplicidad. El resultado de esta etapa es la formulación matemática del método numérico y la elaboración de un **algoritmo** para usarlo.

En la etapa de **Instrumentación** elegimos el dispositivo de cálculo para la obtención de resultados. En problemas simples, basta una calculadora. Para problemas complejos, requerimos el computador mediante funciones predefinidas y en algunos casos, desarrollando **programas** y **funciones** en un lenguaje computacional. Esta última opción es importante para la extensión de los métodos.

Este proceso debe complementarse con la **revisión** y retroalimentación.

Es preferible invertir más tiempo en las primeras etapas, antes de llegar a la instrumentación.

1.2 Fuentes de error en la resolución de un problema numérico

En el **Análisis** pueden introducirse errores debido a suposiciones inadecuadas, simplificaciones y omisiones al construir el modelo matemático. Estos errores se denominan errores inherentes.

En el **Diseño** se pueden introducir errores en los métodos numéricos utilizados los cuales se construyen mediante fórmulas y procedimientos simplificados para obtener respuestas aproximadas. También se pueden introducir errores al usar algoritmos iterativos. Estos errores se denominan errores de truncamiento.

En la **Instrumentación** se pueden introducir errores en la representación finita de los números reales en los dispositivos de almacenamiento y en los resultados de las operaciones aritméticas. Este tipo de error se denomina error de redondeo. También se pueden introducir errores de redondeo al usar datos imprecisos.

Los errores son independientes y su efecto puede acumularse. En el caso del error de redondeo el efecto puede incrementarse si los valores que se obtienen son usados en forma consecutiva en una secuencia de cálculos.

Debido a que los métodos numéricos en general permiten obtener únicamente aproximaciones para la respuesta de un problema, es necesario definir alguna medida para cuantificar el error en el resultado obtenido. Normalmente no es posible determinar exactamente este valor por lo que al menos debe establecerse algún criterio para estimarlo o acotarlo. Esta información es útil para conocer la precisión de los resultados calculados.

1.2.1 Consistencia numérica

En la aplicación de los métodos numéricos que utilizan datos obtenidos mediante observación o medición, si los datos tienen una precisión limitada, la precisión que se obtiene al aplicar el método numérico también estará limitada a esa precisión.

Por otra parte, si un método numérico se desarrolla con varios componentes, la precisión de todos los ellos debería ser similar para que el método tenga consistencia numérica.

1.3 El modelo matemático

Al resolver un problema con el computador, la parte más laboriosa normalmente es el análisis del problema y la obtención del modelo matemático que finalmente se usará para llegar a la solución.

El modelo matemático es la descripción matemática del problema que se intenta resolver. Esta formulación requiere conocer el ámbito del problema y los instrumentos matemáticos para su definición.

1.4 Algoritmos numéricos

Un algoritmo es una descripción ordenada de los pasos necesarios para resolver un problema. El diseño de un algoritmo para resolver un problema numérico requiere conocer en detalle la formulación matemática, las restricciones de su aplicación, los datos y algún criterio para validar y aceptar los resultados obtenidos.

Esta descripción facilita la instrumentación computacional del método numérico. En problemas simples puede omitirse la elaboración del algoritmo e ir directamente a la codificación computacional.

1.5 Instrumentación computacional

En este curso se usará el lenguaje computacional **Python** para instrumentar los algoritmos correspondientes a los métodos numéricos estudiados. Se escribirán funciones que puedan llamarse desde la ventana interactiva o desde un programa que contenga los datos del problema que se desea resolver. Esta actividad es necesaria para entender los métodos numéricos, evitando así depender de instrumentaciones particulares disponibles en los paquetes computacionales. Desarrollar software para matemáticas e ingeniería es una actividad formativa que facilita posteriormente enfrentar la solución computacional de problemas nuevos o más complejos.

Con los métodos numéricos desarrollados se puede construir una librería para mejorar, agrandar o sustituir las funciones existentes en los lenguajes, de tal manera que los usuarios interesados puedan resolver computacionalmente muchos problemas numéricos con mayor facilidad y comprensión.

El traductor **Python** y muchas librerías de utilidad pueden descargarse de la red internet

Algunas librerías importantes de apoyo a los métodos numéricos son las siguientes.

La librería **Pylab** es útil para graficar funciones. Esta librería está incluida en la librería **Matplotlib**.

La librería **SymPy** tiene facilidades para manejo matemático simbólico.

La librería **NumPy** incluye muchas funciones numéricas relacionadas con Álgebra Lineal.

La librería **SciPy** contiene funciones para aplicaciones matemáticas avanzadas.

Python y las librerías son de uso público y pertenecen a la categoría de software libre

Los usuarios interesados en complementar el conocimiento del lenguaje Python y el manejo de algunas de estas librerías pueden descargar el texto digital **Python Programación** de la página del Departamento de Matemáticas de la Facultad de Ciencias Naturales y Matemáticas en la página web de la ESPOL, o revisar otras fuentes de información en la red internet:

www.espol.edu.ec

1.6 Un ejemplo inicial

A continuación se proporciona un ejemplo para seguir el procedimiento descrito.

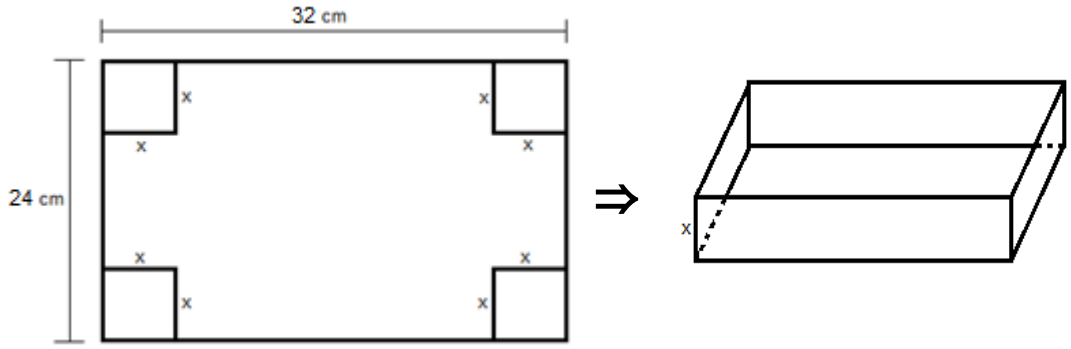
Problema. Se necesita un recipiente rectangular, sin tapa, de un litro de capacidad. Para construirlo se debe usar una lámina rectangular de **32** cm de largo y **24** cm de ancho. El procedimiento será recortar un cuadrado idéntico en cada una de las cuatro esquinas y doblar los bordes de la lámina para formar el recipiente.

Determine la medida del lado del cuadrado que se debe recortar en cada esquina para que el recipiente tenga la capacidad requerida.

Análisis

Para formular el modelo matemático el problema debe entenderse detalladamente. En este ejemplo un dibujo facilita la elaboración del modelo.

Sea: **x**: medida del lado de los cuadrados que se deben recortar para formar la caja



Lados de la caja: **x, (32 – 2x), (24 – 2x)** en cm

Volumen: **1 litro = 1000 cm^3**

Ecuación: **$1000 = (32 - 2x)(24 - 2x)x$**

$250 - 192x + 28x^2 - x^3 = 0$

El modelo matemático es una ecuación polinómica de tercer grado que no se puede resolver directamente con la conocida fórmula de la ecuación cuadrática, por lo tanto se utilizará un método numérico.

Algoritmo e instrumentación

Los cálculos serán realizados directamente en la ventana interactiva de Python usando algunas funciones disponibles en sus librerías.

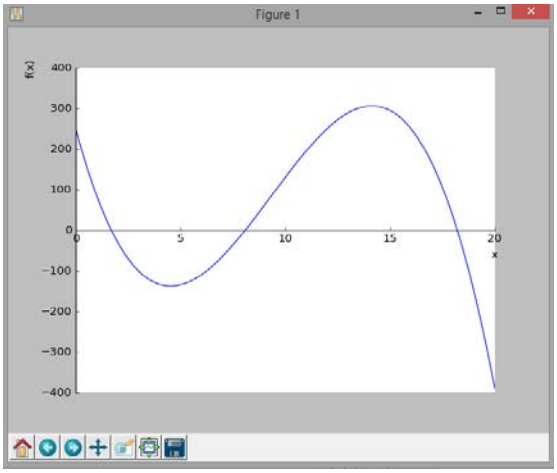
Posteriormente se revisarán métodos para resolver este y otros tipos de modelos matemáticos. El estudio de estos métodos, la construcción de algoritmos y su

instrumentación computacional constituye la base para entender estos objetos matemáticos-computacionales y extender el conocimiento al desarrollo de métodos para resolver problemas nuevos y más complejos.

Gráfico y solución de la ecuación usando la librería **SymPy**.

A la derecha se escriben comentarios acerca de las instrucciones utilizadas.

```
>>> from sympy import*                  Cargar la librería simbólica
>>> x=Symbol('x')                       Definir símbolo matemático
>>> f=250-192*x+28*x**2-x**3            Definición de la fórmula
>>> plot(f,(x,0,20))                    Construir el gráfico
```



Se observan tres respuestas reales positivas para la ecuación.

```
>>> r=solve(f)                          Método simbólico para resolver la ecuación
>>> r[0].evalf(5)                       Las respuestas están en r[0], r[1], r[2]
1.6963                                  Mostrar cada respuesta con 5 dígitos
>>> r[1].evalf(5)
8.0932
>>> r[2].evalf(5)
18.211
```

Las tres son respuestas para la ecuación.

La tercera respuesta no es aceptable para el problema, pues se obtendrían valores negativos para las otras dimensiones de la caja. La primera respuesta pudiera ser descartada porque la caja tendría una apariencia muy plana. Elegimos la segunda como la respuesta adecuada para el problema.

Respuesta:

Lados de la caja en centímetros: **8.0932, 15.8136, 7.8136** aproximadamente.

1.7 Preguntas

1. ¿Cual etapa del proceso de resolución de un problema numérico requiere más atención?

2. ¿Qué conocimientos son necesarios para formular un modelo matemático?

3. En el ejemplo de la caja ¿Cual sería la desventaja de intentar obtener experimentalmente la solución mediante prueba y error en lugar de analizar el modelo matemático?

4. ¿Que es más crítico: el error de truncamiento o el error de redondeo?

5. ¿Cuál es la ventaja de instrumentar computacionalmente un método numérico?

6. Cual es la desventaja de usar funciones definidas en paquetes de software para resolver los problemas

7. ¿Por que es importante validar los resultados obtenidos?

2 TIPOS DE MÉTODOS NUMÉRICOS

Existen dos estrategias para diseñar métodos numéricos y es importante conocer sus características para elegirlos adecuadamente, así como su instrumentación computacional

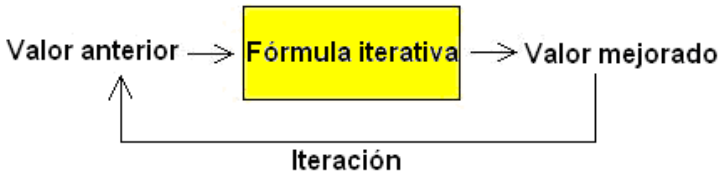
2.1 Métodos iterativos

Los métodos iterativos son procedimientos para acercarse a la respuesta mediante aproximaciones sucesivas. Estos métodos incluyen fórmulas que tienen la propiedad de producir un resultado más cercano a la respuesta a partir de un valor inicial estimado. El resultado obtenido se puede usar nuevamente como valor anterior para continuar mejorando la respuesta.

Se deben considerar algunos aspectos tales como la elección del valor inicial, la propiedad de convergencia de la fórmula y el criterio para terminar las iteraciones.

Estos métodos son auto-correctivos. La precisión de la respuesta está dada por la distancia entre el último valor calculado y la respuesta esperada. Esto constituye el error de truncamiento.

El gráfico describe la estructura de un método iterativo:



Cada ciclo se denomina iteración. Si la fórmula converge, en cada iteración la respuesta estará más cerca del resultado buscado. Aunque en general no es posible llegar a la respuesta exacta, se puede acercar a ella tanto como lo permita la aritmética computacional del dispositivo de cálculo.

El siguiente ejemplo elemental es una introducción al diseño de los métodos numéricos iterativos para resolver un problema formulado matemáticamente. Se omite alguna formalidad en la deducción de la formulación del método numérico, tema que será ampliado en el siguiente capítulo.

Ejemplo. Para introducir el uso de los métodos iterativos se usará una conocida fórmula iterativa para calcular la raíz cuadrada de un número real positivo.

Determinar $\mathbf{r} = \sqrt{\mathbf{n}}$, siendo $n \in \mathbf{R}^+$

Fórmula iterativa, en donde **x** es un valor estimado para **r**

$$t = \frac{1}{2}(x + \frac{n}{x})$$

Esta fórmula tiene la propiedad que si **x** es un valor estimado para **r**, entonces el valor calculado **t** estará más cerca a **r**. La fórmula se puede usar nuevamente y cada vez, el resultado estará mas cerca a la respuesta. Este comportamiento se denomina convergencia.

Algoritmo

```
Algoritmo: Raíz cuadrada
Entra:n:    Dato (valor real)
      x:    Valor inicial
Sale: t:    Solución aproximada
Repita
      t ← 0.5(x + n/x)
      x ← t
Hasta que t² sea muy cercano a n
```

La "←" significa "asignar"

Con la fórmula iterativa anterior calcular $\mathbf{r} = \sqrt{\mathbf{n}}$, para **n = 7**

Valor inicial: **x = 3**

```
t = 0.5(3 + 7/3) = 2.6666666666
x = t
t = 0.5(2.6666666666 + 7/2.6666666666) = 2.6458333333
x = t
t = 0.5(2.6458333333 + 7/2.6458333333) = 2.6457513123
x = t
t = 0.5(2.6457513123 + 7/2.6457513123) = 2.6457513110
x = t
t = 0.5(2.6457513110 + 7/2.6457513110) = 2.6457513110
```

Los dos últimos resultados tienen diez dígitos decimales que no cambian, por lo tanto podemos suponer que la respuesta tiene esa precisión. Se observa la rápida convergencia de la sucesión de números generada. Sin embargo es necesario verificar que la solución es aceptable pues si los números convergen a un valor, este valor no necesariamente es la respuesta correcta. Esta validación debe realizarse en el modelo matemático y también en el problema propuesto:

```
t² = 2.6457513110² = 6.9999999996
```

Se comprueba que el último valor calculado es aproximadamente la raíz cuadrada de 7.

En la siguiente prueba, se utilizará una fórmula incorrecta para realizar los cálculos,

Calcular **r =** $\sqrt{7}$ con la siguiente fórmula iterativa:

$$y = 0.4(x + \frac{n}{x})$$

Valor inicial: **x = 3**

```
t = 0.4(3 + 7/3) = 2.1333333333333337
x = t
t = 0.4(3 + 7/3) = 2.165833333333333
x = t
t = 0.4(3 + 7/3) = 2.1591382583044765
x = t
.
.
.
t = 0.4(3 + 7/3) = 2.1602468994692865
x = t
t = 0.4(3 + 7/3) = 2.160246899469287
x = t
t = 0.4(3 + 7/3) = 2.160246899469287
```

Los dos últimos resultados tienen diez dígitos decimales que no cambian, por lo tanto podemos suponer que la respuesta tiene esa precisión.

Validación del resultado:

$t^2 = 2.160246899469287^2 = 4.666666666666668$

La fórmula converge pero el resultado final no es la raíz cuadrada de 7.

Esto plantea la importancia de verificar si la formulación del método numérico utilizado es correcta y la necesidad de constatar si la respuesta es aceptable para el problema propuesto representado en el modelo matemático.

En general, como se explicará en los siguientes capítulos, los métodos numéricos se enfocan y diseñan para resolver una clase o tipo de problemas y luego se aplican a la resolución de un problema específico.

El ejemplo anterior es un caso particular del problema general: la solución de ecuaciones no lineales: **f(x) = 0**.

2.1.1 Convergencia de los métodos iterativos

Es la propiedad que tienen las fórmulas iterativas de un método numérico para producir resultados cada vez más cercanos a la respuesta esperada.

Definición: Convergencia de una fórmula iterativa

Sean **r** : Respuesta del problema (valor desconocido)
x_i : Valor calculado en la iteración **i** (valor aproximado)

Si la fórmula iterativa converge, entonces

$$\lim_{i \to \infty} x_i \to r$$

.

.

2.1.2 Error de truncamiento

La distancia entre la respuesta esperada y el valor calculado con una fórmula iterativa se denomina error de truncamiento.

Definición: Error de truncamiento

Sean **r** : Respuesta del problema (valor desconocido)
x_i : Valor calculado en la iteración **i** (valor aproximado)
x_{i+1}: Valor calculado en la iteración **i + 1** (valor aproximado)
Entonces
E_i = **$r - x_i$** : Error de truncamiento en la iteración **i**
E_{i+1} = **$r - x_{i+1}$**: Error de truncamiento en la iteración **i + 1**

2.1.3 Finalización de un proceso iterativo

Si la fórmula iterativa converge, la distancia entre valores consecutivos se debe reducir y se puede usar como una medida para el error de truncamiento. Con la definición de convergencia se puede establecer un criterio para finalizar el proceso iterativo.

Consideremos los resultados de dos iteraciones consecutivas: x_i, x_{i+1}

Si el método converge, $\lim_{i \to \infty} x_i \to r$ y también $\lim_{i \to \infty} x_{i+1} \to r$

Al restar estas dos expresiones: $\lim_{i \to \infty} x_{i+1} - x_i \to 0$, se puede establecer un criterio de convergencia

Definición: Criterio para finalizar un proceso iterativo (error absoluto)

Sea **E** algún valor positivo arbitrariamente pequeño.

Si el método converge, se cumplirá que a partir de alguna iteración **i:**

$$|x_{i+1} - x_i| < E \quad .$$

Este valor **E** es el **error absoluto** y se usa como una medida para el error de la respuesta calculada.

La precisión utilizada en los cálculos aritméticos, debe ser coherente con la precisión del método numérico y con los exactitud del modelo matemático y de los datos utilizados.

Adicionalmente, es necesario verificar que la respuesta final sea **válida para el modelo matemático** y **aceptable para el problema que se está resolviendo**.

Ejemplo. Se desea que la respuesta calculada para un problema con un método iterativo tenga un error absoluto menor que **0.0001**. Entonces el algoritmo deberá terminar cuando se cumpla que $|x_{i+1} - x_i| < 0.0001$. Los cálculos deben realizarse al menos con la misma precisión.

Para que el criterio del error sea independiente de la magnitud del resultado, conviene usar la definición del error relativo:

Definición: Criterio para finalizar un proceso iterativo (error relativo)

Sea **e** algún valor positivo arbitrariamente pequeño.
Si el método converge, se cumplirá que a partir de alguna iteración **i:**

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < e \quad .$$

Este valor **e** es el **error relativo** y puede usarse como una medida para el error de la respuesta calculada, independiente de la magnitud. Para calcular el error relativo se toma el último valor como el más cercano a la respuesta.

Ejemplo. Se desea que la respuesta calculada para un problema con un método iterativo tenga un error relativo menor que **0.1%.** Entonces el algoritmo deberá terminar cuando se cumpla que

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < 0.001$$

También debería distinguirse entre **error** y **precisión**.

Ejemplo. Si el error es **1%**, la precisión es **99%**.

2.1.4 Error de truncamiento y estimación del error

Debe distinguirse entre estas dos medidas del error

$r - x_i$: Error de truncamiento en la iteración i. Su valor es desconocido pues **r** es la respuesta que se desea calcular

$x_{i+1} - x_i$: Estimación del error de truncamiento en la iteración **i**. La diferencia entre dos aproximaciones consecutivas se usa como una estimación para el error de truncamiento si el método converge.

2.1.5 Eficiencia de un método iterativo

Sean E_i , E_{i+1} los errores de truncamiento en las iteraciones **i**, **i+1** respectivamente. Se supondrá que estos valores son pequeños y menores a 1.

Si a partir de alguna iteración **i** esta relación puede especificarse como $|E_{i+1}| = k\,|E_i|$, siendo **k** alguna constante positiva menor que uno, entonces se dice que la convergencia es **lineal** o de **primer orden** y **k** es el factor de convergencia. Se puede usar la notación **O( )** y escribir $E_{i+1} = O(E_i)$ para expresar de una manera simple el orden de esta relación, y se lee "orden de".

Si en un método esta relación es más fuerte tal como $E_{i+1} = O(E_i^2)$ entonces el error se reducirá más rápidamente y se dice que el método tiene convergencia **cuadrática** o de **segundo orden**.

Definición: Orden de convergencia de un método iterativo

Sean E_i , E_{i+1} los errores en las iteraciones consecutivas **i**, **i + 1** respectivamente
Si se pueden relacionar estos errores en la forma:

$$E_{i+1} = O(E_i^n)$$

Entonces se dice que el método iterativo tiene convergencia de orden **n**.

Si un método iterativo tiene convergencia mayor que lineal, entonces si el método converge, lo hará más rápidamente. .

2.1.6 Intervalo de existencia e intervalo de convergencia

El intervalo de existencia de la respuesta de un problema es la región en la cual se puede determinar que se encuentra la solución.

El intervalo de convergencia es la región en la cual se puede asegurar que la secuencia de los valores calculados en las iteraciones converge a la respuesta del problema.

2.1.7 Elección del valor inicial

Los métodos iterativos normalmente requieren que el valor inicial sea elegido apropiadamente. Si es elegido al azar, puede ocurrir que no se produzca la convergencia.

Para algunos problemas, mediante un análisis previo puede definirse un intervalo de **convergencia** tal que si el valor inicial y los valores calculados en cada iteración permanecen en esta región, el método converge.

2.1.8 Preguntas

Conteste las siguientes preguntas

1. ¿Por que el error de redondeo no debe ser mayor que el error de truncamiento?

2. Una ventaja de los métodos iterativos es que son auto-correctivos, es decir que si se introduce algún error aritmético en una iteración, en las siguientes puede ser corregido. ¿Cuando no ocurriría esta auto-corrección?

3. El ejemplo del método numérico para calcular $\sqrt{7}$ produce una secuencia numérica. ¿Le parece que la convergencia es lineal o cuadrática?

4. Cual intervalo incluye al otro intervalo: intervalo de existencia e intervalo de convergencia.

5. Sea **E** error absoluto, suponer menor que 1. **¿**Cual es verdadero?:

a) $\mathbf{O(E) + O(E^2) = O(E)}$
b) $\mathbf{O(E) + O(E^2) = O(E^2)}$

2.2 Métodos directos

Son procedimientos para obtener resultados realizando una secuencia finita de operaciones aritméticas. La cantidad de cálculos aritméticos depende del tamaño del problema. El resultado obtenido será exacto siempre que se puedan conservar en forma exacta los valores calculados en las operaciones aritméticas, caso contrario se introducirán los **errores de redondeo**.

Ejemplo. Instrumentar un método directo para resolver un sistema triangular inferior de ecuaciones lineales.

Modelo matemático

$$
\begin{aligned}
a_{1,1}x_1 &= b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 &= b_2 \\
a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 &= b_3 \\
&\ldots \\
a_{n,1}x_1 + a_{n,2}x_2 + a_{x,3}x_3 + \ldots + a_{n,n}x_n &= b_n
\end{aligned}
$$

En donde: $a_{i,j}$: coeficientes (datos)
b_i : constantes (datos)
x_i : variables (resultados)

La formulación matemática del método se obtiene despejando sucesivamente x_i de la ecuación **i**

$$x_1 = \frac{1}{a_{1,1}}(b_1)$$

$$x_2 = \frac{1}{a_{2,2}}(b_2 - a_{2,1}x_1)$$

$$x_3 = \frac{1}{a_{3,3}}(b_3 - a_{3,1}x_1 - a_{3,2}x_2)$$

$$\ldots$$

En general:

$$x_i = \frac{1}{a_{i,i}}(b_i - a_{i,1}x_1 - a_{i,2}x_2 - \ldots - a_{i,i-1}x_{i-1}),\ i = 2, 3, \ldots, n,\ a_{i,i} \neq 0$$

Este procedimiento de obtención de fórmulas para los métodos directos se usará en este curso en varios capítulos. Pudiéramos llamarlo “inducción incompleta”. Consiste en desarrollar fórmulas para varios casos incrementando el tamaño de problema y usar esta información para extender la formulación al problema general de tamaño n. La demostración formal requeriría el uso de la técnica de inducción matemática.

Es conveniente describir el uso de la formulación mediante un algoritmo, el cual facilita su uso manual y su instrumentación computacional como se verá posteriormente.

Algoritmo

```
Algoritmo: Triangular
Entra:n:    Número de ecuaciones
      a_i,j   Coeficientes
      b_i     Constantes
Sale: x     Solución calculada
x_1=b_1/a_1,1
Para  i = 2, 3, . . ., n
      s ← 0
      Para j = 1, 2, . . ., i-1
           s ← s + a_i,j x_j
      Fin
      x_i ← (b_i – s)/a_i,i
 Fin
```

Este algoritmo es un caso particular del problema general de los sistemas de ecuaciones lineales.

Los métodos numéricos normalmente se crean para resolver una clase o tipo general de problemas. Los algoritmos pueden usarse manualmente para resolver algún ejemplo con la ayuda de una calculadora. Este desarrollo demuestra que se entiende el método y permite su instrumentación posterior mediante un programa o función en un lenguaje computacional.

Ejemplo. Usar la formulación del algoritmo Triangular para resolver el sistema triangular:

$$
\begin{aligned}
3x_1 &= 2\\
7x_1 + 5x_2 &= 3\\
8x_1 + 2x_2 + 9x_3 &= 6
\end{aligned}
$$

$x_1 = b_1/a_{1,1} = 2/3 \cong 0.6667$ Valores aproximados

$x_2 = (b_2 - a_{2,1}*x_1)/a_{2,2} \cong (3 - 7*0.6667)/5 \cong -0.3333$

$x_3 = (b_3 - a_{3,1}*x_1 - a_{3,2}*x_2)/a_{3,3} \cong (6 - 8*0.6667 - 2*(-0.3333))/9 \cong 0.1481$

La solución se debe verificar sustituyéndola en el sistema dado. Se puede comprobar que las ecuaciones se satisfacen pero en forma aproximada, no excata, debido a los errores de redondeo.

Estos resultados no se pueden mejorar recalculándolos nuevamente con las fórmulas, como en los métodos iterativos. En los métodos directos las fórmulas son usadas una sola vez. Si se desea reducir el error, los cálculos deben ser realizados con mayor precisión.

2.2.1 Error de redondeo

Los métodos numéricos operan con datos que pueden ser inexactos y con dispositivos para representar a los números reales. El error de redondeo se atribuye a la imposibilidad de almacenar todas las cifras de estos números y a la imprecisión de los instrumentos de medición con los cuales se obtienen los datos.

Definición: Error de redondeo absoluto

Sean **X:** Valor exacto (normalmente desconocido)
$\overline{X}$: Valor aproximado (observado o calculado)
$E = X - \overline{X}$ Error de redondeo .

Definición: Error de redondeo relativo

Sean **X:** Valor exacto (normalmente desconocido)
$\overline{X}$: Valor aproximado (observado o calculado)
E: Error de redondeo
$e = \frac{E}{X} \cong \frac{E}{\overline{X}}$: Error de redondeo relativo. (**X,** $\overline{X}$ diferentes de cero)

Debido a que normalmente no es posible calcular exactamente el valor de **E**, se debe intentar al menos acotar su valor.

A diferencia del error de truncamiento, se dispone solamente de un resultado para estimar el error de redondeo. Igualmente, es necesario evaluar el modelo matemático con el valor calculado.

2.2.2 Error en la representación de los números reales

Las operaciones aritméticas pueden producir resultados que no se pueden representar exactamente en los dispositivos de almacenamiento. Si estos errores se producen en forma recurrente entonces el error propagado pudiera crecer en forma significativa dependiendo de la cantidad de operaciones requeridas. Esta cantidad de operaciones está determinada por la eficiencia del algoritmo.

Ejemplo. Suponga que un dispositivo puede almacenar únicamente los cuatro primeros dígitos decimales de un número real y trunca los restantes (esto es redondeo inferior).

Se requiere almacenar el número:

$$X = 536.78$$

Primero expresemos el número en forma normalizada, es decir sin enteros y ajustando su magnitud con potencias de 10:

$$X = 0.53678x10^{3}$$

Ahora descomponemos el número en dos partes

$$X = 0.5367x10^{3} + 0.00008x10^{3}$$

El valor almacenado es un valor aproximado

$$\overline{X} = 0.5367x10^{3}$$

El error de redondeo por la limitación del dispositivo de almacenamiento es

$$E = 0.00008x10^{3} = 0.8x10^{3-4} = 0.8x10^{-1}$$

En general, si **n** es la cantidad de enteros del número normalizado con potencias de 10, y **m** es la cantidad de cifras decimales que se pueden almacenar en el dispositivo, entonces si se truncan los decimales sin ajustar la cifra anterior, el error de redondeo absoluto está acotado por:

$$|E| < 1 * 10^{n-m}$$

Mientras que el error relativo:

$$|e| < \frac{\max(|E|)}{\min(|X|)} = \frac{1*10^{n-m}}{0.1*10^{n}} = 10*10^{-m}$$ (Solo depende del almacenamiento)

2.2.3 Error de redondeo en las operaciones aritméticas

En los métodos directos debe considerarse el error que se propaga en las operaciones aritméticas, el cual puede ser significativo cuando la cantidad de cálculos requeridos es grande. A continuación se analizan la suma y el producto. A estos expresiones se deberá agregar el error de redondeo que se produce al almacenar el resultado en algún dispositivo.

a) Error de redondeo en la suma

Sean **X, Y :** Valores exactos

$\overline{X}$, $\overline{Y}$ **:** Valores aproximados

Con la definición de error de redondeo

$$E_X = X - \overline{X}, \quad E_Y = Y - \overline{Y}$$

$$S = X + Y$$

$$S = (\overline{X} + E_X) + (\overline{Y} + E_Y) = (\overline{X} + \overline{Y}) + (E_X + E_Y)$$

$$\overline{S} = \overline{X} + \overline{Y}$$ Valor que se almacena

}

Error de redondeo absoluto en la suma

$$E_{X+Y} = E_X + E_Y$$

$$|E_{X+Y}| \leq |E_X| + |E_Y|$$

Error de redondeo relativo en la suma

$$e_{X+Y} = \frac{E_{X+Y}}{\overline{X}+\overline{Y}} = \frac{E_X + E_Y}{\overline{X}+\overline{Y}} = \frac{E_X}{\overline{X}+\overline{Y}} + \frac{E_Y}{\overline{X}+\overline{Y}}$$

$$e_{X+Y} = \frac{\overline{X}}{\overline{X}+\overline{Y}}\frac{E_X}{\overline{X}} + \frac{\overline{Y}}{\overline{X}+\overline{Y}}\frac{E_Y}{\overline{Y}}$$

$$e_{X+Y} = \frac{\overline{X}}{\overline{X}+\overline{Y}} e_x + \frac{\overline{Y}}{\overline{X}+\overline{Y}} e_Y$$

$$|e_{X+Y}| \leq \left|\frac{E_X}{\overline{X}+\overline{Y}}\right| + \left|\frac{E_Y}{\overline{X}+\overline{Y}}\right|$$

$$|e_{X+Y}| \leq \left|\frac{\overline{X}}{\overline{X}+\overline{Y}} e_x\right| + \left|\frac{\overline{Y}}{\overline{X}+\overline{Y}} e_Y\right|$$

Se puede extender a la resta

$$E_{X-Y} = E_X - E_Y$$

$$|E_{X-Y}| \leq |E_X| + |-E_Y|$$

$$e_{X-Y} = \frac{E_{X-Y}}{\overline{X}-\overline{Y}} = \frac{E_X - E_Y}{\overline{X}-\overline{Y}} = \frac{E_X}{\overline{X}-\overline{Y}} - \frac{E_Y}{\overline{X}-\overline{Y}}$$

$$|e_{X-Y}| \leq \left|\frac{E_X}{\overline{X}-\overline{Y}}\right| + \left|-\frac{E_Y}{\overline{X}-\overline{Y}}\right|$$

$$|e_{X-Y}| \leq \left|\frac{\overline{X}}{\overline{X}-\overline{Y}} e_x\right| + \left|-\frac{\overline{Y}}{\overline{X}-\overline{Y}} e_Y\right|$$

b) Error de redondeo en la multiplicación

$$P = X\,Y$$

$$P = (\bar{X} + E_X)\,(\bar{Y} + E_Y) = \bar{X}\,\bar{Y} + \bar{X}\,E_Y + \bar{Y}\,E_X + E_X E_Y$$

$$P = \bar{X}\,\bar{Y} + \bar{X}\,E_Y + \bar{Y}\,E_X$$ El último término se descarta por ser muy pequeño

$$\bar{P} = \bar{X}\,\bar{Y}$$ Valor que se almacena

Error de redondeo absoluto en la multiplicación

$$E_{XY} = \bar{X}\,E_Y + \bar{Y}\,E_X$$

$$|E_{XY}| \leq |\bar{X}\,E_Y| + |\bar{Y}\,E_X|$$

La magnitud del error de redondeo en la multiplicación puede ser tan grande como la suma de los errores de redondeo de los operandos ponderada por cada uno de sus respectivos valores.

Error de redondeo relativo en la multiplicación

$$e_{XY} = \frac{E_{XY}}{\bar{X}\bar{Y}} = \frac{\bar{X}E_Y + \bar{Y}E_X}{\bar{X}\bar{Y}} = \frac{\bar{X}E_Y}{\bar{X}\bar{Y}} + \frac{\bar{Y}E_X}{\bar{X}\bar{Y}} = \frac{E_Y}{\bar{Y}} + \frac{E_X}{\bar{X}}$$

$$e_{XY} = e_x + e_Y$$

$$|e_{XY}| \leq |e_x| + |e_Y|$$

En general, si los valores de los operandos tienen ambos el mismo signo y son valores mayores que 1, se puede concluir que la operación aritmética de multiplicación puede propagar más error de redondeo que la suma.

Adicionalmente, si el resultado de cada operación aritmética debe escribirse o almacenarse, hay que agregar el error de redondeo debido a la limitación de la representación en forma exacta de los números reales en los dispositivos de cálculo.

Ejemplo del crecimiento del error de redondeo en la representación de los números reales en la memoria de una computadora

```
>>> 1/3
0.3333333333333333                    El error es insignificante
>>> x=0
>>> for i in range(1000000):          Sumar 1/3 un millón de veces
      x=x+1/3                         El error se propaga al siguiente
                                      resultado
>>> print(x)
333333.3333322413                     El error acumulado es significativo
```

Por los errores de redondeo, sumar puede producir un resultado diferente que multiplicar:

```
>>> x=1000000*1/3
>>> print(x)
333333.3333333333
```

2.2.4 Casos de propagación del error de redondeo en operaciones aritméticas

1) Error de redondeo en mediciones

La velocidad de una partícula es constante e igual a **4 m/s**, medida con un error de **0.1 m/s** durante un tiempo de recorrido de **5 seg**. medido con error de **0.1 seg**. Determine el error absoluto y el error relativo en el valor de la distancia recorrida.

v = 4, $E_v = 0.1$ (velocidad)
t = 5, $E_t = 0.1$ (tiempo)
d = vt (distancia recorrida)

$$E_d = \bar{v}\,E_t + \bar{t}\,E_v = 4(0.1) + 5(0.1) = 0.9$$ (error absoluto)

d = vt = 20 con un rango de variación: **$19.1 \le d \le 20.9$**

$$e_d = \frac{E_v}{\bar{v}} + \frac{E_t}{\bar{t}} = \frac{0.1}{4} + \frac{0.1}{5} = 0.045 = 4.5\%$$ (error relativo)

2) Resta de números con valores muy cercanos

Suponer que se deben restar dos números muy cercanos **X=578.1**, **Y=577.8** con error de redondeo en el segundo decimal: **$E_X = E_Y = 0.05$** aproximadamente. Ambos errores pueden ser del mismo signo o de diferente signo, depende de la forma como se obtuvieron los datos

$$e_X = \frac{E_X}{\bar{X}} \cong 0.000086 = 0.0086\%\ ,\quad e_Y = \frac{E_Y}{\bar{Y}} \cong 0.000086 = 0.0086\%$$ (Errores relativos)

Cota para el error de redondeo relativo:

$$e_{X-Y} = \frac{E_{X-Y}}{\bar{X}-\bar{Y}} = \frac{E_X - E_Y}{\bar{X}-\bar{Y}} = \frac{E_X}{\bar{X}-\bar{Y}} - \frac{E_Y}{\bar{X}-\bar{Y}}$$

$$|e_{X-Y}| \le \left|\frac{E_X}{\bar{X}-\bar{Y}}\right| + \left|-\frac{E_Y}{\bar{X}-\bar{Y}}\right|$$

$$|e_{X-Y}| \le \left|\frac{0.05}{578.1-577.8}\right| + \left|-\frac{0.05}{578.1-577.8}\right| \cong 0.3333 = 33.33\%$$

El aumento en la cota del error en el resultado es muy significativo con respecto a los operandos. Se concluye que se debe evitar restar números cuya magnitud sea muy cercana pues el cociente al ser muy pequeño, hará que el error relativo sea significativo.

Adicionalmente habría que agregar el efecto del error de redondeo al almacenar el resultado.

3) Suma de números de diferente magnitud

Es suficiente considerar tres números: **X, Y, Z**. Suponer por simplicidad que los datos son valores positivos exactos, por lo tanto $\mathbf{e_X = 0, e_Y = 0, e_z = 0}$

$$\mathbf{S = X + Y + Z,} \quad \text{con} \quad \mathbf{X > Y > Z}$$

Suponer que la suma se realiza en el orden usual: **S = (X + Y) + Z**

$$\mathbf{e_{X+Y} = \frac{\overline{X}}{\overline{X}+\overline{Y}} e_X + \frac{\overline{Y}}{\overline{X}+\overline{Y}} e_Y + r_1 = r_1}$$

r_1: error de redondeo al almacenar la suma

$$\mathbf{e_{(X+Y)+Z} = \frac{\overline{X}+\overline{Y}}{\overline{X}+\overline{Y}+\overline{Z}} e_{X+Y} + \frac{\overline{Z}}{\overline{X}+\overline{Y}+\overline{Z}} e_Z + r_2 = \frac{\overline{X}+\overline{Y}}{\overline{X}+\overline{Y}+\overline{Z}} r_1 + r_2 = \frac{(\overline{X}+\overline{Y})r_1 + (\overline{X}+\overline{Y}+\overline{Z})r_2}{\overline{X}+\overline{Y}+\overline{Z}}}$$

r_2: error de redondeo al almacenar la suma

Si cada resultado se almacena en un dispositivo que tiene **m** cifras, su cota de error de redondeo:

$$\mathbf{|\, r_1, r_2 \,| < 10 * 10^{-m}}$$

$$\mathbf{|\, e_{(X+Y)+Z} \,| < \frac{(2\overline{X}+2\overline{Y}+\overline{Z})10 * 10^{-m}}{\overline{X}+\overline{Y}+\overline{Z}}}$$

Si la suma se hiciera en orden opuesto, se obtendría

$$\mathbf{|\, e_{(Z+Y)+X} \,| < \frac{(2\overline{Z}+2\overline{Y}+\overline{X})10 * 10^{-m}}{\overline{Z}+\overline{Y}+\overline{X}}}$$

Si $\mathbf{\overline{X} > \overline{Z}}$, se puede concluir que la suma de los números debe realizarse comenzando con los números de menor magnitud, pues la cota del error será menor.

2.2.5 Eficiencia de los métodos directos

La eficiencia de un algoritmo y su programación está relacionada con el tiempo necesario para obtener la solución. Este tiempo depende de la cantidad de operaciones que se deben realizar. Así, si se tienen dos algoritmos para resolver un mismo problema, es más eficiente el que requiere menos operaciones para producir el mismo resultado.

Sea **n** el tamaño del problema, y **T(n)** una función que mide la eficiencia del algoritmo (cantidad de operaciones requeridas). Para obtener **T(n)** se pueden realizar pruebas en el computador con diferentes valores de **n** registrando el tiempo real de ejecución. Este tiempo es proporcional a la cantidad de operaciones que se realizaron, por lo tanto se puede usar para estimar la función **T(n)**.

Esta forma experimental para determinar **T(n)** tiene el inconveniente de usar la instrumentación computacional del algoritmo para realizar las pruebas. Es preferible conocer la eficiencia del algoritmo antes de invertir el esfuerzo de la programación computacional para preveer que sea un algoritmo aceptable.

Para determinar la eficiencia de un algoritmo antes de su instrumentación se puede analizar la estructura del algoritmo o realizar un recorrido del mismo en forma abstracta.

Ejemplo. El siguiente algoritmo en Python calcula la suma de los primeros n números naturales. Encontrar T(n)

Sea **T** la cantidad de sumas que se realizan

```
. . .
s = 0
for i in range(n):
   s = s + i
```

La suma está dentro de una repetición que se realiza **n** veces, por lo tanto,

T(n) = n

Ejemplo. El siguiente algoritmo en Python suma los elementos de una matriz cuadrada **a** de orden **n**. Encontrar **T(n)**

Sea **T** la cantidad de sumas

```
. . .
s = 0
for i in range(n):
   for j in range(n):
      s = s + a[i,j]
```

La suma está incluida en una repetición doble. La variable **i,** cambia **n** veces y para cada uno de sus valores, la variable **j** cambia **n** veces. Por lo tanto.

$T(n) = n^2$

Ejemplo. El siguiente algoritmo en Pythpn es una modificación del anterior. Suponga que se desea sumar únicamente los elementos de la sub-matriz triangular superior. Obtener **T(n)**

```
. . .
s = 0
for i in range(n):
   for j in range(i,n):
      s = s + a[i,j]
```

Si no es evidente la forma de **T(n)**, se puede recorrer el algoritmo y anotar la cantidad de sumas que se realizan

Valor Cantidad de ciclos

i	**j**
0	n
1	n-1
2	n-2
...	**...**
n-1	1

Entonces, **T(n) = 1 + 2 + ... + n =** $\frac{n}{2}(n+1) = \frac{n^2}{2} + \frac{n}{2}$ (suma de una serie aritmética)

Otra manera de obtener la función **T(n)** consiste en programar el conteo de los ciclos de un algoritmo para obtener puntos de su eficiencia.

Ejemplo. Programa en Python para conteo de ciclos para el ejemplo anterior

```
n=int(input('Ingrese n: '))
c=0
for i in range(n):
    for j in range(i,n):
        c=c+1
print(c)
```

Debido a que son dos ciclos, **T(n)** debe ser un polinomio algebraico de segundo grado. Para obtener este polinomio son suficientes tres puntos **(n, c):**

Se realizaron tres pruebas del programa de conteo y se obtuvieron los resultados :

```
>>>
Ingrese n: 4
10
>>>
Ingrese n: 5
15
>>>
Ingrese n: 6
21
```

Con estos resultados se puede construir el polinomio de interpolación que representa a **T(n)**. Este polinomio se lo puede obtener manualmente o con un método computacional.

El resultado que se obtiene es **:**

$T(n) = t^2/2 + t/2$

Resultado que coincide con el obtenido anteriormente con un conteo directo manual

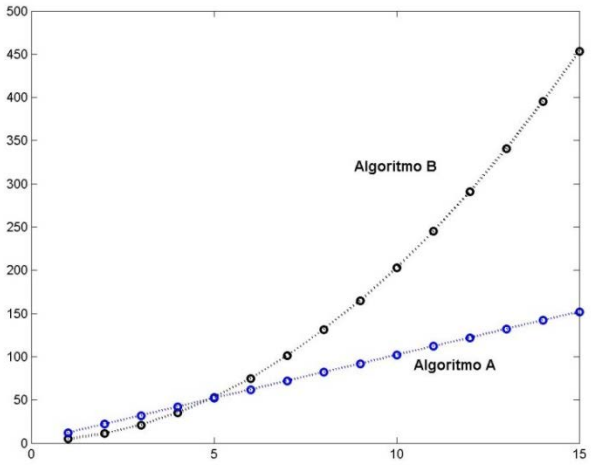
Para registrar el tiempo real de ejecución de un proceso (programa o función) se puede usar la función **`clock()`** de la librería **`time`** de Python :

```
>>> from time import*
>>> t1=clock(); ... proceso ... ;t2=clock();print(t2-t1)
```

2.2.6 La notación O( )

Supongamos que para resolver un problema se han diseñado dos algoritmos: **A** y **B**, con eficiencias $T_A(n) = 10n+2$, $T_B(n) = 2n^2 + 3$, respectivamente. ¿Cual algoritmo es más eficiente?.

Para valores pequeños de **n**, $T_B(n) < T_A(n)$, pero para valores grandes de **n**, $T_B(n) > T_A(n)$. Es de interés práctico determinar la eficiencia de los algoritmos para valores grandes de **n**, por lo tanto el algoritmo **A** es más eficiente que el algoritmo **B** como se puede observar en el siguiente gráfico:



Si **T(n)** incluye términos de **n** que tienen diferente orden, es suficiente considerar el término de mayor orden pues es el que determina la eficiencia del algoritmo cuando **n** es grande. Para compararlos, no es necesario incluir los coeficientes y las constantes.

Ejemplo. Suponga $\mathbf{T(n) = n^2 + n + 10.}$ Evaluar **T** para algunos valores de **n**

T(2) = 4 + 2 + 10
T(5) = 25 + 5 + 10
T(20) = 400 + 20 + 10
T(100) = 10000 + 100 + 10

Se observa que a medida que **n** crece, **T** depende principalmente del término dominante $\mathbf{n^2}$. Este hecho se puede expresar usando la notación **O( )** la cual indica el "orden" de la eficiencia del algoritmo, y se puede escribir: **T(n)** = $\mathbf{O(n^2)}$ lo cual significa que la eficiencia es proporcional a $\mathbf{n^2}$, y se dice que el algoritmo tiene eficiencia de segundo orden.

En general, dado un problema de tamaño **n**, la medida de la eficiencia **T(n)** de un algoritmo se puede expresar con la notación **O(g(n))** siendo **g(n)** alguna expresión tal como: **n,** $\mathbf{n^2}$**,** $\mathbf{n^3}$**, ..., log(n), n log(n), ...,** $\mathbf{2^n}$**, n!, ...** etc, la cual expresa el orden de la cantidad de operaciones que requiere el algoritmo.

Es de interés medir la eficiencia de los algoritmos antes de su instrumentación computacional. En el siguiente cuadro se ha tabulado **T(n)** con algunos valores de **n** y para algunas funciones típicas **g(n)**.

Tabulación de T(n) con algunos valores de n para algunas funciones típicas g(n)

n	[log(n)]	n	[n log(n)]	n^2	n^3	2^n	n!
1	0	1	0	1	1	2	1
3	1	3	3	9	27	8	6
5	1	5	8	25	125	32	120
7	1	7	13	49	343	128	5040
9	2	9	19	81	729	512	$3.62x10^5$
11	2	11	26	121	1331	2048	$3.99x10^7$
13	2	13	33	169	2197	8192	$6.22x10^9$
15	2	15	40	225	3375	32768	$1.30x10^{12}$
17	2	17	48	289	4913	$1.31x10^5$	$3.55x10^{14}$
19	2	19	55	361	6859	$5.24x10^5$	$1.21x10^{17}$
21	3	21	63	441	9261	$2.09x10^6$	$5.10x10^{19}$
23	3	23	72	529	12167	$8.38x10^6$	$2.58x10^{22}$
25	3	25	80	625	15625	$3.35x10^7$	$1.55x10^{25}$
50	3	50	195	2500	125000	$1.12x10^{15}$	$3.04x10^{64}$
100	4	100	460	10000	1000000	$1.26x10^{30}$	$9.33x10^{157}$

Los algoritmos en las dos últimas columnas son de tipo **exponencial** y **factorial** respectivamente. Se puede observar que aún con valores relativamente pequeños de **n** el valor **T(n)** es extremadamente alto. Los algoritmos con este tipo de eficiencia se denominan **no factibles** pues ningún computador actual pudiera calcular la solución en un tiempo aceptable para valores de **n** grandes. La mayoría de los algoritmos que corresponden a los métodos numéricos son de tipo polinomial con **g(n) = n,** $\mathbf{n^2}$**,** $\mathbf{n^3}$

2.2.7 Ejercicios

1. Encuentre la cota del error relativo en la siguiente operación aritmética:

$T = X(Y + Z)$

El error de redondeo relativo de los operandos es respectivamente $\mathbf{e_X}$, $\mathbf{e_Y}$, $\mathbf{e_z}$, y el error de redondeo relativo en el dispositivo de almacenamiento es $\mathbf{r_m}$

a) Primero se realiza la multiplicación y luego la suma
b) Primero se realiza la suma y luego la multiplicación

2. Suponga que tiene tres algoritmos: A, B, C con eficiencia respectivamente:

$T_A(n) = 5n + 50$
$T_B(n) = 10n\ ln(n) + 5$
$T_C(n) = 3n^2 + 1$

a) Determine **n** a partir del cual A es más eficiente que B
b) Determine **n** a partir del cual B es más eficiente que C
c) Coloque los algoritmos ordenados según el criterio de eficiencia establecido

3. Exprese la eficiencia de los algoritmos A, B, C con la notación **O( )**

4. Los computadores actuales pueden realizar 100 millones de operaciones aritméticas en un segundo. Calcule cuanto tiempo tardaría este computador para resolver un problema de tamaño **n=50** si el algoritmo es de tipo:

a) Polinomial de tercer grado
b) Exponencial
c) Factorial

5. Determine la función de eficiencia **T(n)** del algoritmo **triangular** incluido en el capítulo 1 y exprésela con la notación **O( )**. Suponga que es de interés conocer la cantidad total de ciclos que se realizan.

6. Dado el siguiente algoritmo

```
Leer n
Mientras n>0 repita
    d ← mod(n,2)        Produce el residuo entero de la división n/2
    n ← fix(n/2)        Asigna el cociente entero de la división n/2
    Mostrar d
fin
```

a) Recorra el algoritmo con **n = 73**

b) Suponga que **T(n)** representa la cantidad de operaciones aritméticas de división que se realizan para resolver el problema de tamaño **n**. Encuentre **T(n)** y exprésela con la notación **O( )** Para obtener **T(n)** observe el hecho de que en cada ciclo el valor de **n** se reduce aproximadamente a la mitad.

7. Sea **n** tamaño del problema, n>1. ¿Cual es verdadero?:

a) $\mathbf{O(n) + O(n^2) = O(n)}$
b) $\mathbf{O(n) + O(n^2) = O(n^2)}$

3 RAÍCES REALES DE ECUACIONES NO-LINEALES

Sea **f: R→R**. Dada la ecuación **f(x)** = **0**, se debe encontrar un valor real **r** tal que **f(r)** = **0**. Entonces **r** es una raíz real de la ecuación

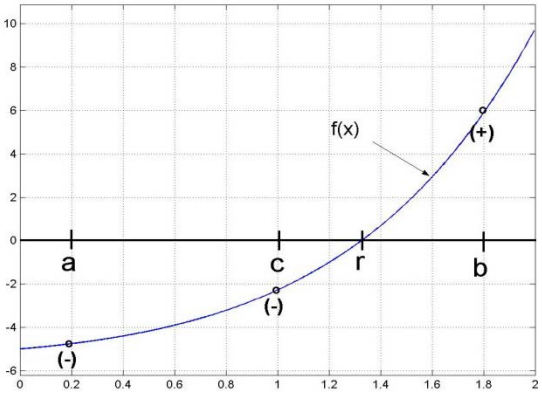
Si no es posible obtener la raíz directamente, entonces se debe recurrir a los métodos numéricos iterativos para calcular **r** en forma aproximada con alguna precisión controlada. Se han creado muchos métodos numéricos para resolver este problema clásico, pero con el uso de computadoras para el cálculo, conviene revisar solamente algunos de estos métodos que tengan características significativamente diferentes.

3.1 Método de la bisección

Sea **f: R→R**. Suponer que **f** es continua en **[a, b],** y que además **f(a)** y **f(b)** tienen signos diferentes.

El método de la bisección es un método simple y convergente para calcular **r**. Consiste en calcular el punto medio **c=(a+b)/2** del intervalo **[a, b]** y sustituirlo por el intervalo **[a, c]** ó **[c, b]** dependiendo de cual contiene a la raíz **r**. Este procedimiento se repite hasta que la distancia entre **a** y **b** sea muy pequeña, entonces el último valor calculado **c** estará muy cerca de **r**.

Interpretación gráfica del método de la bisección



En la figura se puede observar que luego de haber calculado **c,** para la siguiente iteración debe sustituirse el intervalo **[a, b]** por **[c, b]** debido a que **f(a)** y **f(c)** tienen igual signo y por lo tanto la raíz estará en el intervalo **[c, b]**

3.1.1 Existencia de la raíz real

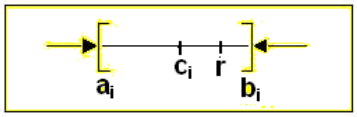
Si una función **f** es continua en un intervalo **[a, b]** y **f(a)** tiene signo diferente que **f(b)**, entonces existe por lo menos un punto **r** en **(a, b)** tal que **f(r)=0.** Si además **f'(x)** no cambia de signo en el intervalo **[a, b]**, entonces la solución es única.

La validez de esta proposición se basa en el Teorema del Valor Intermedio.

3.1.2 Convergencia del método de la bisección

Sean a_i, b_i, c_i los valores de **a, b, c** en la iteración **i=1, 2, 3, . . .** respectivamente, en donde $c_i = (a_i+b_i)/2$ es el punto medio entre a_i y b_i, es decir $a_i < c_i < b_i$,

El método de la bisección genera a partir de un intervalo inicial **(a, b)** una sucesión de intervalos $(a_1, b_1), (a_2, b_2), \ldots, (a_i, b_i)$, en donde se ha sustituido a_i o b_i por c_i tales que $a, a_1, a_2, \ldots a_i$ constituyen una sucesión creciente y $b, b_1, b_2, \ldots, b_i$ una sucesión decreciente. Por definición del método: $c_i, r \in (a_i, b_i)$ en cada iteración **i**:



Sean $d_i = b_i - a_i$ longitud del intervalo (a_i, b_i) en la iteración **i=1, 2, 3, . . .**
$d = b - a$ longitud del intervalo inicial

Recorrido de las iteraciones

Iteración	Longitud del intervalo
1	$d_1 = d/2$
2	$d_2 = d_1/2 = d/2^2$
3	$d_3 = d_2/2 = d/2^3$
. . .	. . .
i	$d_i = d/2^i$

Entonces

$$\frac{d}{2^i} \xrightarrow[i\to\infty]{} 0 \Rightarrow d_i \xrightarrow[i\to\infty]{} 0 \Rightarrow a_i \xrightarrow[i\to\infty]{} b_i \Rightarrow c_i \xrightarrow[i\to\infty]{} r \Rightarrow \exists i>0 \mid |c_i - r| < E \text{ para cualquier } E \in R^+$$

Suponer que se desea que el último valor calculado c_i tenga error **E = 0.001,** entonces si el algoritmo termina cuando $b_i - a_i < E$, se cumplirá que $|c_i - r| < E$ y c_i será una aproximación para **r** con un error menor que 0.0001

Se puede predecir el número de iteraciones que se deben realizar con el método de la Bisección para obtener la respuesta con error permitido **E**:

En la iteración **i**: $d_i = d/2^i$
Se desea terminar cuando: $d_i < E$
Entonces se debe cumplir $d/2^i < E$
De donde se obtiene: $i > \dfrac{\log(d/E)}{\log(2)}$

Ejemplo. La ecuación **f(x) = x e^x - π = 0** tiene una raíz real en el intervalo **[0, 2].** Determine cuantas iteraciones deben realizarse con el método de la bisección para obtener un resultado con error **E=0.0001**. Por simple inspección **f(0) < 0, f(2) > 0** y **f** es contínua en **[0, 2]**

El número de iteraciones que deberán realizarse es:

i > log(2/0.0001)/log(2) ⇒ i >14.287 ⇒ 15 iteraciones

3.1.3 Algoritmo del método de la bisección

Calcular una raíz **r** real de la ecuación **f(x) = 0** con error **E**.
f es contínua en un intervalo **[a, b]** tal que **f(a)** y **f(b)** tienen signos diferentes

```
Algoritmo: Bisección
Restricción: No valida el intervalo inicial
Entra:       f, a, b, E
Sale:        c   (aproximación a la raíz r, con error E)
Mientras b-a > E
    c ← (a+b)/2
    Si f(c)=0
       Terminar
    Sino
       Si signo f(c) = signo f(a)
             a ← c                          La "←" significa "asignar"
       Sino
             b ← c
       Fin
   Fin
Fin
```

El último valor calculado **c** estará a no más de una distancia **E** de la raíz **r**.

Ejemplo. Calcule una raíz real de **f(x) = x e^x - π = 0** en el intervalo **[0, 2] con error 0.01**

Solución

La función **f** es continua y además por simple inspección: **f(0)<0**, **f(2)>0**, por lo tanto la ecuación **f(x) = 0** debe contener alguna raíz real en el intervalo **[0, 2]**

Cantidad de iteraciones

$$i > \frac{\log(d/E)}{\log(2)} = \frac{\log(2/0.01)}{\log(2)} = 7.6439 \quad \Rightarrow \quad 8 \text{ iteraciones}$$

Tabulación de los cálculos para obtener la raíz con el método de la Bisección

iteración	a	b	C	sign(f(a))	sign(f(c))
inicio	0	2	1	-	-
1	1	2	1.5	-	+
2	1	1.5	1.25	-	+
3	1	1.25	1.125	-	+
4	1	1.125	1.0625	-	-
5	1.0625	1.125	1.0938	-	+
6	1.0625	1.0938	1.0781	-	+
7	1.0625	1.0781	1.0703	-	-
8	1.0703	1.0781	1.0742		

En la octava iteración:

b – a = 1.0781-1.0703 = 0.0078 $\Rightarrow$ |r – c|<0.01

r = 1.074 con error menor que **0.01**

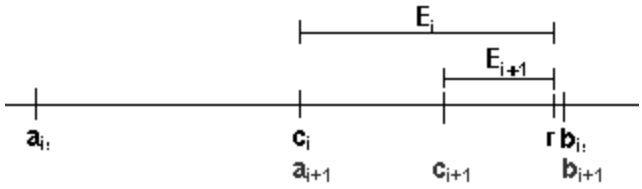
En la última iteración se observa que el intervalo que contiene a la raíz se ha reducido a **[1.0703, 1.0781]**, por lo tanto el último valor calculado de **c = 1.074** debe estar cerca de **r** con una distancia menor que **0.01**

3.1.4 Eficiencia del método de la bisección

Suponer el caso más desfavorable, en el que **r** está muy cerca de uno de los extremos del intervalo **[a, b]**:

Sean $E_i = r - c_i$: error en la iteración **i**

$E_{i+1} = r - c_{i+1}$: error en la iteración **i+1**



En cada iteración la magnitud del error se reduce en no más de la mitad respecto del error en la iteración anterior: $E_{i+1} \cong 0.5\, E_i$. Esta es una relación lineal. Con la notación **O( )** se puede escribir $E_{i+1} = O(E_i)$. Entonces, el método de la Bisección tiene **convergencia lineal** o de primer orden y su factor de convergencia es **0.5** aproximadamente.

3.1.5 Instrumentación computacional del método de la Bisección

Calcular una raíz **r** real de la ecuación **f(x) = 0**. **f** es contínua en un intervalo **[a, b]** tal que **f(a)** y **f(b)** tienen signos diferentes

Para instrumentar el algoritmo de este método se escribirá una función en Python. El nombre será **bisección**. Recibirá como parámetros **f,** definida como función de una línea, los puntos del intervalo de existencia: **a, b**, el valor **e** que representa el error absoluto requerido, y entregará **c** como aproximación a la raíz **r**.

Criterio para salir: Terminar cuando la longitud del intervalo sea menor que el valor **e**. Entonces el último valor **c** estará aproximadamente a una distancia **e** de la raíz **r**.

```
def biseccion(f, a, b, e):
    while b-a>=e:
        c=(a+b)/2
        if f(c)==0:
            return c
        else:
            if f(a)*f(c)>0:
                a=c
            else:
                b=c
    return c
```

Ejemplo. Desde la ventana interactiva de Python use la función **bisección** para calcular una raíz real de la ecuación **f(x)** = $\mathbf{xe^x - \pi = 0}$. Suponer que se desea que el error sea menor que 0.000001.

Por simple inspección se puede observar que **f** es continua y además **f(0) < 0**, **f(2) > 0**. Por lo tanto se elije como intervalo inicial: **[0, 2].** También se puede previamente graficar **f**.

En la ventana interactiva de Python:

```
>>> from biseccion import*                    Cargar el método numérico
>>> from math import*
>>> def f(x): return x*exp(x)-pi              Definición de la ecuación
>>> c=biseccion(f,0,2,0.000001)               Calcular con E=0.000001
>>> c
1.0736589431762695
>>> f(c)                                      Verificar en la ecuación
4.540916178063729e-06
```

Instrumentación alternativa usando el tipo simbólico de la librería SymPy de Python

El tipo simbólico permite definir funciones de manera más natural y facilita su operación matemática (derivar, integrar, etc) y su graficación, pero la evaluación de estas funciones requiere un formato especial con la función **subs** en la que se indica el valor que sustituye a la variable. Esta librería también incluye las funciones matemáticas.

En esta instrumentación la ecuación no se definirá como una función en línea, sino directamente como una expresión usando una variable de tipo simbólico que debe también enviarse como parámetro:

```
from sympy import*
def biseccions(f,v,a,b,e):
    while b-a>=e:
        c=(a+b)/2
        if f.subs(v,c)==0:
            return c
        else:
            if f.subs(v,a)*f.subs(v,c)>0:
                a=c
            else:
                b=c
    return c
```

El ejemplo anterior resuelto con la nueva función Bisección:

```
>>> from biseccions import*
>>> x=Symbol('x')
>>> f=x*exp(x)-pi
>>> c=biseccions(f,x,0,2,0.000001)
>>> print(c)
1.0736589431762695
>>> float(f.subs(x,c))
4.540916178063729e-06
```

NOTA. Al importar la función **biseccions**, también se importa la librería **SymPy**.

Adicionalmente se puede importar la librería con un nombre específico y usar sus funciones con esta identificación. Esto es importante cuando hay varias librerías en una aplicación y que pueden tener algunas funciones con el mismo nombre, por lo que es necesario distinguirlas. Ejemplo.

```
>>> import sympy as sp
>>> x=sp.Symbol('x')
....
```

Ejemplo. Encontrar las intersecciones en el primer cuadrante de los gráficos de las funciones: **f(x) = 4 + x cos(x+1), g(x)=e^x sen(x).**

Para graficar las funciones y visualizar las intersecciones se puede usar la librería **Pylab** de **Python**. Esta librería grafica puntos de funciones.

La librería **Pylab** se puede cargar de la manera típica:

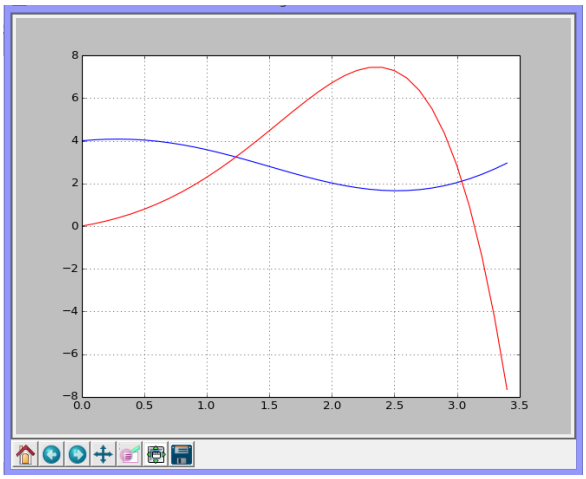
```
>>> from pylab import*
```

Pero para prevenir que haya conflicto con los nombres de las funciones que están en otras librerías que se importan para su uso, es conveniente asignar a las librerías una identificación propia como se muestra a continuación:

```
>>> import pylab as pl
```

Entonces, el uso de las funciones deberá usar la identificación **'pl'** asignada a la librería:

```
>>> x=pl.arange(0,3.5,0.1)
>>> f=4+x*pl.cos(x+1)
>>> g=pl.exp(x)*pl.sin(x)
>>> pl.plot(x,f,'-b')
>>> pl.plot(x,g,'-r')
>>> pl.grid(True)
>>> pl.show()
```



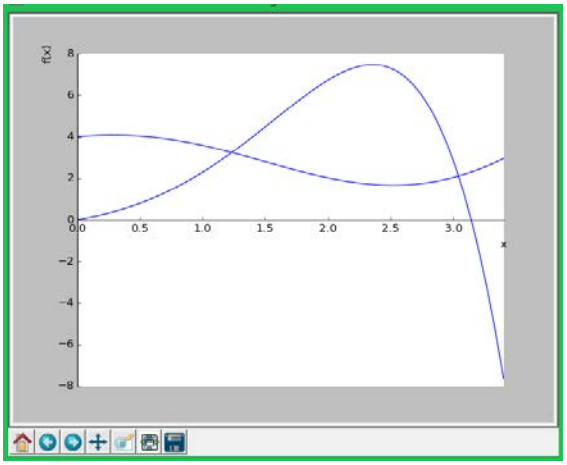
Las intersecciones son las raíces de la ecuación **h(x) = g(x)-f(x) = 0**

El cálculo de las raíces se realiza con el método de la bisección con E= 0.000001

```
>>> from biseccion import*
>>> from math import*
>>> def h(x): return exp(x)*sin(x)-4-x*cos(x+1)
>>> r=biseccion(h,1,1.5,0.000001)
>>> print(r)
1.233719825744629
>>> r=biseccion(h,2.5,3.2,0.000001)
>>> print(r)
3.0406669616699213
```

El ejemplo anterior resuelto usando el tipo simbólico de la librería **SymPy**

```
>>> from biseccions import*
>>> import sympy as sp
>>> x=sp.Symbol('x')
>>> f=4+x*sp.cos(x+1)
>>> g=sp.exp(x)*sp.sin(x)
>>> sp.plot(f,g,(x,0,3.4))
```




```
>>> h=g-f
>>> r=biseccions(h,x,1,1.5,0.000001)
>>> print(r)
1.233719825744629
>>> r=biseccions(h,x,2.5,3.2,0.000001)
>>> print(r)
3.0406669616699213
```

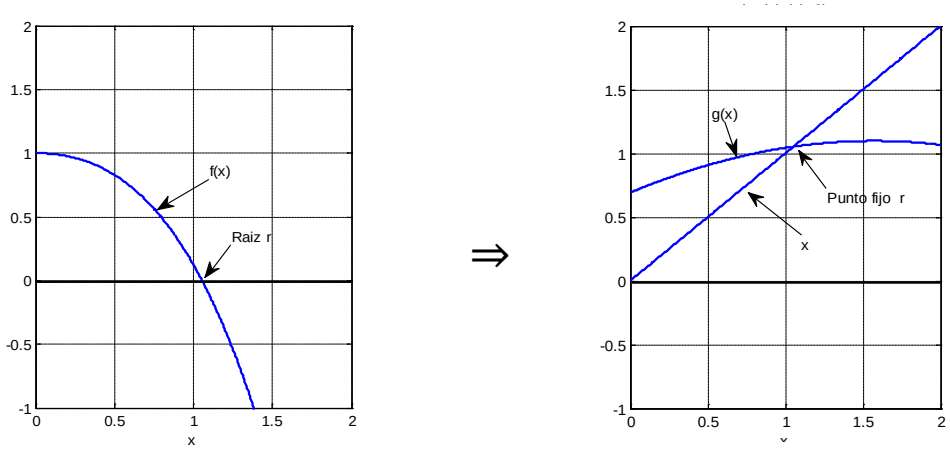
NOTAS. El usuario debe elegir la instrumentación computacional más adecuada y comprensible. Por otra parte, al graficar con Python es necesario cerrar el gráfico de la pantalla para seguir trabajando en la ventana interactiva. Para evitarlo, se sugiere abrir otra ventana interactiva de Python para realizar los cálculos. Esto permite mantener el gráfico.

3.2 Método del punto fijo

Sea **f: R→R.** Dada la ecuación **f(x) = 0**, encuentre **r** tal que **f(r) = 0**

El método del punto fijo, también conocido como método de la Iteración funcional es el fundamento matemático para construir otros métodos eficientes para el cálculo de raíces reales de ecuaciones no lineales.

Este método consiste en re-escribir la ecuación **f(x) = 0** en la forma **x = g(x).** Esta nueva ecuación debe ser equivalente a la ecuación original en el sentido que debe satisfacerse con la misma raíz, es decir la existencia de un punto fijo **r** de la ecuación **x = g(x)** es equivalente a encontrar una raíz real **r** de la ecuación **f(x) = 0: r = g(r) ⇔ f(r)=0** como muestra el gráfico:



3.2.1 Existencia del punto fijo

Sea **g** una función continua en un intervalo **[a, b]** tal que **g(a) > a** y **g(b) < b.**
Entonces la ecuación **x = g(x)** tiene al menos un punto fijo en **[a, b]**

Demostración

Sea **h(x) = g(x) - x** una función, también continua, en el intervalo **[a, b]**
Entonces si:

h(a) = g(a) - a > 0
h(b) = g(b) - b < 0

Por la continuidad de **h** existe algún valor **r** en el intervalo **[a, b]**, tal que **h(r) = 0**. Entonces **g(r) - r = 0**. Por lo tanto **r** es un punto fijo de **x=g(x)** y **r** es una raíz real de **f(x) = 0**

3.2.2 Convergencia del método del punto fijo

Sea **g** una función continua en un intervalo **[a, b]** tal que **g(a) > a** y **g(b) < b** y sea **r** un punto fijo en **[a, b].** Si $\forall x \in (a,b)(|g'(x)|<1)$, entonces la secuencia $x_{i+1} = g(x_i)$, **i = 0, 1, 2, 3, ...** converge al punto fijo **r** y **g'(x)** es el factor de convergencia:

Demostración

Sean las ecuaciones equivalentes: **f(x) = 0, x = g(x)**

Si **r** es una raíz de **f(x)=0** se cumple que $r = g(r) \Leftrightarrow f(r) = 0$

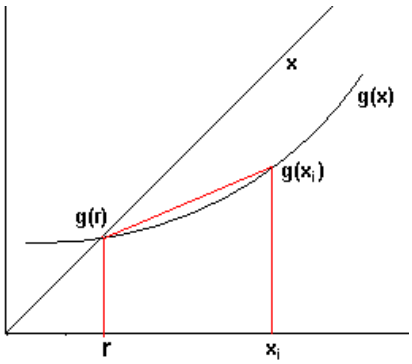
La ecuación **x = g(x)** sugiere una fórmula iterativa $x_{i+1} = g(x_i)$, **i = 0, 1, 2, 3, . . .** siendo x_0 el valor inicial elegido para comenzar la secuencia $\{x_i\}$

Definiciones:

$E_i = r - x_i$: Error de truncamiento en la iteración **i**

$E_{i+1} = r - x_{i+1}$: Error de truncamiento en la iteración **i + 1**

Analizamos el siguiente caso:



Si **g** es continua en el intervalo que incluye a **r** y a cada punto calculado x_i, por el Teorema del Valor Medio:

$$g'(z) = \frac{g(x_i) - g(r)}{x_i - r}, \text{ para algún } z \in (r, x_i)$$

Sustituyendo las definiciones anteriores:

$$g'(z) = \frac{x_{i+1} - r}{x_i - r} = \frac{E_{i+1}}{E_i} \Rightarrow E_{i+1} = g'(z)E_i, \quad i = 0, 1, 2, 3, \ldots$$

Si en cada iteración, $0 < g'(z) \leq m < 1$, entonces $E_{i+1} \leq mE_i$, **i = 0, 1, 2, 3, . . .**

$E_1 \leq mE_0$

$E_2 \leq mE_1 \leq m^2E_0$

. . .

$E_{i+1} \leq mE_i \leq m^2E_{i-1} \leq \ldots \leq m^i E_0$

$0 < m < 1 \Rightarrow \lim_{i\to\infty} m^i \to 0 \Rightarrow \lim_{i\to\infty} E_{i+1} \to 0$ y por lo tanto, $\lim_{i\to\infty} r - x_{i+1} \to 0 \Rightarrow \lim_{i\to\infty} x_{i+1} \to r$

El método del punto fijo converge a **r** y $g'(x) \in (0, 1)$ es el factor de convergencia.

Se puede extender al caso en el cual **g** tiene pendiente negativa y deducir la condición necesaria de convergencia del método del punto fijo: **|g'(x)| < 1**.

Un estimado de la cantidad de iteraciones requeridas para calcular la raíz con error absoluto ε se lo puede obtener de:

$$E_{i+1} = g'(z)E_i \leq m^i E_0 \leq \varepsilon$$

$$i \log(m) \geq \log(\frac{\varepsilon}{E_0}) \qquad \text{(Se multiplican ambos términos por un factor negativo)}$$

$$i \geq \frac{\log(\frac{\varepsilon}{E_0})}{\log(\overline{m})}$$

Debido a que el valor de **m** cambia en cada iteración, se lo reemplaza por $\overline{m}$ que representa un promedio de **g'(x)** en los extremos del intervalo de convergencia o del intervalo de existencia de la raíz, o se pudiera tomar su mayor valor. Para aproximar E_0 se pudiera usar la distancia del intervalo inicial, considerando el caso en que **r** está muy cerca de uno de los extremos, como se propuso en el método de la bisección

El valor de **i** resultante será simplemente una estimación.

3.2.3 Unicidad del punto fijo

Sea **r** un punto fijo de **x = g(x)**. Si **g'(x)** existe en el intervalo **[a, b]** y $\forall x \in [a,b](|g'(x)|<1)$, entonces el punto fijo **r** es único

Demostración

Sean **r, p** puntos fijos de **x=g(x)** con **r ≠ p**, es decir, **r = g(r)**, **p = g(p)** en **[a, b]** tal que $\forall x \in (a,b)(|g'(x)|<1)$

Por el Teorema del Valor Medio: $\exists z \in [r, p]$ tal que $g'(z) = \frac{g(p) - g(r)}{p - r}$

$\Rightarrow g'(z)(p-r) = g(p)-g(r) \Rightarrow |g'(z)|\,|(p-r)| = |p-r| \Rightarrow |g'(z)| = 1$, pues **r ≠ p**

Pero **r, p** son puntos fijos y **r, p** $\in$ **[a, b]** lo cual significa que $\forall x \in [p, r]\ |g'(x)|<1$

Esta contradicción implica que **p** debe ser igual a **r**

3.2.4 Algoritmo del punto fijo

La ecuación **x = g(x)** sugiere convertirla en una fórmula iterativa $\mathbf{x_{i+1} = g(x_i)}$, **i = 0, 1, 2, 3, . . .** siendo $\mathbf{x_0}$ el valor inicial, elegido con algún criterio. En la fórmula se usa un índice para numerar los valores calculados.

La fórmula iterativa producirá una sucesión de valores **x:** $\mathbf{x_{i+1} = g(x_i)}$, **i = 0, 1, 2, 3, ...** y se espera que tienda a un punto fijo de la ecuación **x = g(x)** lo cual implica que este resultado también satisface a la ecuación **f(x) = 0**

```
Algoritmo: Punto Fijo
Restricción: No verifica la condición de convergencia
Entra:     g, x, E
Sale:      x     (aproximación a la raíz r, con error E)
Mientras |g(x)-x|>E
    x ← g(x)
Fin
```

Ejemplo. Dada la ecuación: $\mathbf{x = g(x) = \sqrt{\frac{e^x}{3}}}$

Encuentre un **intervalo de existencia** de una raíz

Sea **f(x) = g(x) – x**

f(0) = g(0) – 0 = 0.5774 – 0 > 0
f(2) = g(2) – 2 = 1.5694 – 2 < 0

g, f son funciones contínuas $\Rightarrow$ Intervalo de existencia de la raíz: $\mathbf{I_r}$ = **[0,2]**

Encuentre un **intervalo de convergencia** de la iteración de punto fijo

Ecuación $\mathbf{x = g(x) = \sqrt{\frac{e^x}{3}}}$.

g es una función contínua, diferenciable y estrictamente creciente

$$\mathbf{g'(x) = \frac{\sqrt{3}}{6}\sqrt{e^x}}$$

$$\mathbf{g'(x) < 1 \Rightarrow \frac{\sqrt{3}}{6}\sqrt{e^x} < 1 \Rightarrow \sqrt{e^x} < 6/\sqrt{3} \Rightarrow x < 2.4849}$$

Intervalo de convergencia de la iteración del punto fijo: $\mathbf{I^*}$ = **(-∞, 2.4849]**

En este ejemplo, **el intervalo de existencia** está incluido en el **intervalo de convergencia**. También pudiera ocurrir lo opuesto, pero el valor inicial debe tomarse dentro del intervalo de convergencia.

Estime el número de iteraciones necesarias para obtener una aproximación con error $\mathbf{10^{-4}}$

$\mathbf{E_0 = r - x_0 < 2 - 0}$ (distancia del intervalo de existencia de la raíz)

$\mathbf{\overline{m} = 0.5(g'(a)+g'(b)) = 0.5(g'(0)+g'(2)) = 0.5366}$

$$\mathbf{i \geq \frac{\log(\frac{\varepsilon}{E_0})}{\log(\overline{m})} = \frac{\log(\frac{0.0001}{2})}{\log(0.5366)} = 15.90 \Rightarrow i=16}$$

Resultado aproximado pues **g'( )** cambia en cada iteración

Cálculos

$$\mathbf{x_{i+1} = g(x_i) = \sqrt{\frac{e^{x_i}}{3}}, \quad i = 0,1,2,3,...}$$

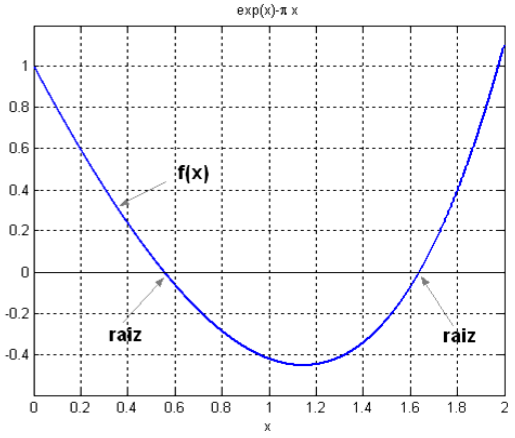
$\mathbf{x_0 = 2}$

$$\mathbf{x_1 = g(x_0) = \sqrt{\frac{e^{x_0}}{3}} = 1.569400}$$

$\mathbf{x_2 = g(x_1) = 1.265407}$
$\mathbf{x_3 = g(x_2) = 1.086973}$
. . .
$\mathbf{x_{14} = g(x_{13}) = 0.910080}$
$\mathbf{x_{15} = g(x_{14}) = 0.910040}$ (resultado con 4 decimales fijos)

Ejemplo. Calcule una raíz real de $\mathbf{f(x) = e^x - \pi x = 0}$ con el método del punto fijo.

Un gráfico de **f** nos muestra que la ecuación tiene dos raíces reales en el intervalo **[0, 2]**



Re-escribir la ecuación en la forma **x = g(x)**.

Tomando g(x) directamente de la ecuación (pueden haber varias opciones):

a) $\mathbf{x = g(x) = e^x/\pi}$
b) $\mathbf{x = g(x) = \ln(\pi x)}$
c) $\mathbf{x = g(x) = e^x - \pi x + x,}$ etc

Usaremos la primera

Escribir la fórmula iterativa:

$$x_{i+1} = g(x_i) = e^{x_i}/\pi, \quad i = 0, 1, 2, 3, \ldots$$

Elegir un valor inicial $x_0 = 0.6$ (cercano a la primera raíz, tomado del gráfico)

Calcular los siguientes valores

$$x_1 = g(x_0) = e^{x_0}/\pi = e^{0.6}/\pi = 0.5800$$
$$x_2 = g(x_1) = e^{x_1}/\pi = e^{0.5800}/\pi = 0.5685$$
$$x_3 = g(x_2) = e^{0.5685}/\pi = 0.5620$$
$$x_4 = g(x_3) = e^{0.5620}/\pi = 0.5584$$
$$x_5 = g(x_4) = e^{0.5584}/\pi = 0.5564$$
$$x_6 = g(x_5) = e^{0.5564}/\pi = 0.5552$$
$$\ldots$$

La diferencia entre cada par de valores consecutivos se reduce en cada iteración. En los últimos la diferencia está en el orden de los milésimos, por lo tanto podemos considerar que el método converge y el error está en el orden de los milésimos.

Para calcular la segunda raíz, usamos la misma fórmula iterativa:

$$x_{i+1} = g(x_i) = e^{x_i}/\pi, \quad i = 0, 1, 2, 3, \ldots$$

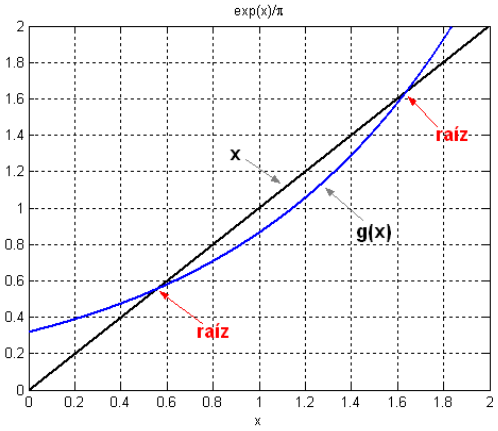
El valor inicial elegido es $x_0 = 1.7$ (cercano a la segunda raíz, tomado del gráfico)

Calcular los siguientes valores

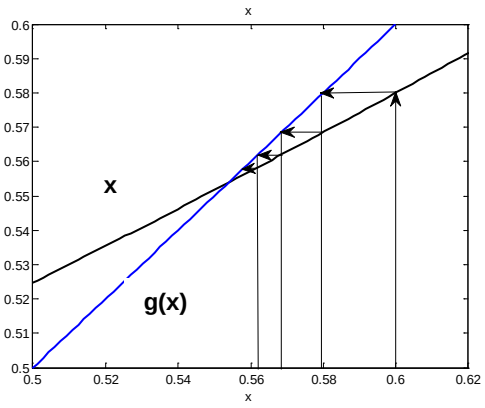
$$x_1 = g(x_0) = e^{x_0}/\pi = e^{1.7}/\pi = 1.7424$$
$$x_2 = g(x_1) = e^{x_1}/\pi = e^{1.7424}/\pi = 1.8179$$
$$x_3 = g(x_2) = e^{1.8179}/\pi = 1.9604$$
$$x_4 = g(x_3) = e^{1.9604}/\pi = 2.2608$$
$$x_5 = g(x_4) = e^{2.2608}/\pi = 3.0528$$
$$x_6 = g(x_5) = e^{3.0528}/\pi = 6.7399$$
$$x_6 = g(x_5) = e^{6.7399}/\pi = 269.1367$$
$$\ldots$$

La diferencia entre cada par de valores consecutivos aumenta en cada iteración. Se concluye que el método no converge.

En el ejemplo anterior, las raíces reales de la ecuación **f(x)=0** son las intersecciones de **f** con el eje horizontal. En el problema equivalente **x=g(x)**, las raíces reales son las intersecciones entre **g** y la recta **x**:



En el cálculo de la primera raíz, la pendiente de **g** es menor que la pendiente de **x** y se observa que la secuencia de valores generados tiende a la raíz. La interpretación gráfica del proceso de cálculo se describe en la siguiente figura.



Para la segunda raíz, la pendiente de **g** es mayor que la pendiente de **x** y se puede constatar que la secuencia de valores generados se aleja de la raíz.

Encuentre un **intervalo de convergencia** para para calcular las raíces de la ecuación anterior: $\mathbf{f(x) = e^x - \pi x = 0}$ con el método del punto fijo.

Ecuación equivalente:

$$\mathbf{x = g(x) = e^x/\pi}$$

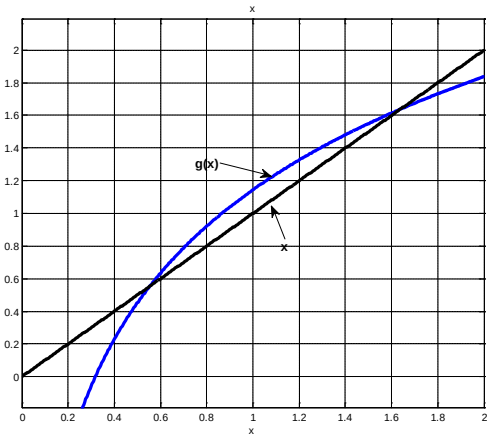
Condición de convergencia: $\mathbf{|g'(x)| < 1}$

$$\mathbf{g'(x) = e^x/\pi \Rightarrow |e^x/\pi| < 1 \Rightarrow \quad x < \ln(\pi)}$$

Intervalo de convergencia: $\mathbf{(-\infty, \ln(\pi)) \cong (-\infty, 1.1447)}$ Conjunto disjunto con el intervalo de existencia. Con lo que se concluye que la segunda raíz no se puede calcular con esta fórmula.

Ejemplo. Calcule una raíz real de la misma ecuación $\mathbf{f(x) = e^x - \pi x = 0}$ con el método del punto fijo con otra forma de la ecuación de recurrencia **x=g(x)**.

Para este ejemplo se decide usar la segunda forma: $\mathbf{x = g(x) = \ln(\pi x)}$



Según la condición de convergencia establecida, el gráfico muestra que será imposible calcular la primera raíz. La segunda raíz si podrá ser calculada, lo cual se puede verificar numéricamente.

Ejemplo. Calcular con Python la segunda raíz de la ecuación: $\mathbf{f(x)=\exp(x)-\pi x}$ mediante la fórmula de recurrencia: $\mathbf{x=g(x)=\ln(\pi x)}$

```
>>> from math import*
>>> def g(x): return log(pi*x)
>>> x=1.6                                   Valor inicial para la segunda raíz
>>> x=g(x)
>>> print(x)
1.6147335150951356
>>> x=g(x)
>>> print(x)
1.6238998227759673
>>> x=g(x)
>>> print(x)
1.62956044020348
. . .
. . .
>>> x=g(x)
>>> print(x)
1.6385043042098606
>>> x=g(x)
>>> print(x)
1.6385137019235614                          Valor luego de 15 iteraciones
```

3.2.5 Eficiencia del método del punto fijo

El método del punto fijo tiene convergencia lineal o de primer orden debido a que $\mathbf{E_i}$ y $\mathbf{E_{i+1}}$ están relacionados directamente mediante el factor de convergencia: $\mathbf{E_{i+1} = g'(z)\, E_i}$, por lo tanto este método tiene convergencia lineal: $\mathbf{E_{i+1} = O(E_i)}$ y **g'(z)** es el factor de convergencia. Si la magnitud de **g'(z)** permanece mayor a **1**, el método no converge.

3.3 Método de Newton

Sean **f: R→R**, y la ecuación **f(x) = 0.** Sea **r** tal que **f(r) = 0,** entonces **r** es una raíz real de la ecuación.

El método de Newton es una fórmula iterativa eficiente para encontrar **r.** Es un caso especial del método del punto fijo en el que la ecuación **f(x) = 0** se re-escribe en la forma **x = g(x)** eligiendo **g** de tal manera que la convergencia sea de segundo orden.

3.3.1 La fórmula de Newton

Suponer que **g** es una función diferenciable en una región que incluye a la raíz **r** y al valor $\mathbf{x_i}$ (calculado en la iteración **i**). Desarrollando con la serie de Taylor:

$$\mathbf{g(x_i) = g(r) + (x_i - r)\, g'(r) + (x_i - r)^2 g''(r)/2! + \ldots}$$

Con las definiciones del método del punto fijo:

$$\mathbf{r = g(r)}$$
$$\mathbf{x_{i+1} = g(x_i),\ i = 0, 1, 2, 3, \ldots}$$

Se obtiene:

$$\mathbf{x_{i+1} = r + (x_i - r)\, g'(r) + (x_i - r)^2 g''(r)/2! + \ldots}$$

Si se define el error de truncamiento de la siguiente forma:

$\mathbf{E_i = x_i - r}$: Error en la iteración **i**

$\mathbf{E_{i+1} = x_{i+1} - r}$: Error en la iteración **i + 1**

Finalmente se obtiene:

$$\mathbf{E_{i+1} = E_i\ g'(r) + E_i^2\, g''(r)/2! + \ldots}$$

Si se puede hacer que $\mathbf{g'(r) = 0}$, y si $\mathbf{g''(r) \neq 0}$, entonces se tendrá:

$$\mathbf{E_{i+1} = O(E_i^2)},$$

Con lo que el método tendrá convergencia cuadrática.

Procedimiento para hacer que **g'(r) = 0**

Consiste en elegir una forma apropiada para **g(x):**

Sea **g(x) = x - f(x) h(x),** en donde **h** es alguna función que deberá especificarse

Es necesario verificar que la ecuación **x = g(x)** se satisface con la raíz **r** de **f(x) = 0**

Si **f(r) = 0**, y **h(r) ≠ 0: g(r) = r - f(r) h(r) = r** $\Rightarrow$ **g(r) = r**

Derivar **g(x)** y evaluar en **r**

g'(x) = 1 - f'(x) h(x) - f(x) h'(x)
g'(r) = 1 - f'(r) h(r) - f(r) h'(r) = 1 - f'(r) h(r)

Para que la convergencia sea cuadrática se necesita que **g'(r) = 0**

g'(r) = 0 $\Rightarrow$ **0 = 1 - f'(r) h(r)** $\Rightarrow$ **h(r) = 1/f'(r)** $\Rightarrow$ **h(x) = 1/f'(x), f'(x)≠0**

Con lo que **h(x)** queda especificada para que la convergencia sea cuadrática.

Al sustituir en la fórmula propuesta se obtiene **x = g(x) = x - f(x)/f'(x)**, y se puede escribir la fórmula iterativa de Newton:

Definición: Fórmula iterativa de Newton

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad f'(x_i) \neq 0 \text{ , } i = 0, 1, 2, \ldots$$

Interpretación gráfica de la fórmula de Newton

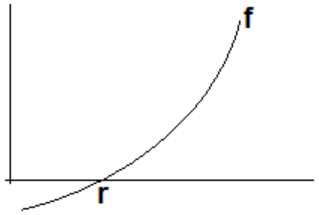
Sean

f(x)=0: Ecuación de la que se pretende calcular una raíz real **r**
Se supondrá que **f** es diferenciable y tiene forma creciente

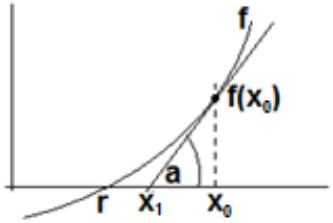
x_0 : Valor inicial elegido para calcular la raíz **r**

x_1: Valor calculado, más cercano a **r** que el valor inicial x_0

Para calcular x_1 se utilizará un procedimiento geométrico, considerando que **f** tiene la forma descrita en el siguiente gráfico:



Se elige el valor inicial x_0. Se traza una tangente a **f** en el punto $(x_0, f(x_0))$. El punto de intersección de la recta con el eje horizontal estará más cerca de **r**, y se lo designa x_1 :



Si **a** es el ángulo de intersección, el valor de x_1 se lo puede encontrar con la definición:

$$\tan(a) = f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \quad \Rightarrow \quad x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Esta fórmula iterativa se puede utilizar repetidamente, tomando como nuevo valor inicial el valor x_1 para calcular x_2. Se generará una secuencia de valores que esperamos converja a la raíz **r**.

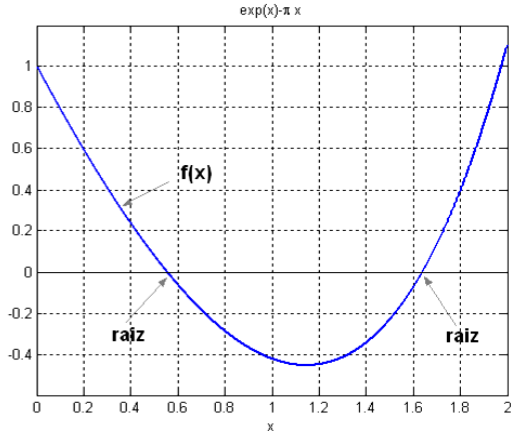
3.3.2 Algoritmo del método de Newton

Para calcular una raíz real **r** de la ecuación **f(x) = 0** con error **E** se usa la fórmula iterativa . $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$, $f'(x_i) \neq 0$. Comenzando con un valor inicial x_0 se genera una sucesión de valores x_i , **i = 0, 1, 2, 3, ...** esperando que converja a un valor que satisfaga la ecuación.

```
Algoritmo: Newton
Restricción: No detecta divergencia
Entra:      f     Ecuación a resolver
            x     Valor inicial
            E     Error que se espera en la respuesta
Sale:       r     Aproximación a la raíz con error E
r ← x – f(x)/f’(x)
Mientras |r – x| > E
     x ← r
     r ← x – f(x)/f’(x)
Fin
Mostrar(r)
```

Ejemplo. Calcule las raíces reales de $f(x) = e^x - \pi x = 0$ con el método de Newton.

Un gráfico de **f** nos muestra que la ecuación tiene dos raíces reales en el intervalo **[0, 2]**



Para la primera raíz tomamos el valor inicial: $x_0 = 0.5$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{e^{x_0} - \pi x_0}{e^{x_0} - \pi} = 0.5 - \frac{e^{0.5} - 0.5\pi}{e^{0.5} - \pi} = 0.5522$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = x_1 - \frac{e^{x_1} - \pi x_1}{e^{x_1} - \pi} = 0.5522 - \frac{e^{0.5522} - 0.5522\pi}{e^{0.5522} - \pi} = 0.5538$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = x_2 - \frac{e^{x_2} - \pi x_2}{e^{x_2} - \pi} = 0.5538 - \frac{e^{0.5538} - 0.5538\pi}{e^{0.5538} - \pi} = 0.5538$$

En los resultados se observa la rápida convergencia. En la tercera iteración el resultado tiene cuatro decimales que no cambian.

Para la segunda raíz tomamos el valor inicial: $x_0 = 1.8$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{e^{x_0} - \pi x_0}{e^{x_0} - \pi} = 1.8 - \frac{e^{1.8} - 1.8\pi}{e^{1.8} - \pi} = 1.6642$$

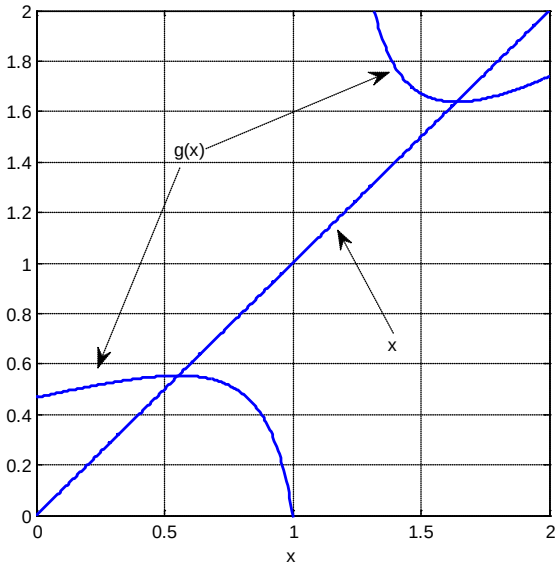
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = x_1 - \frac{e^{x_1} - \pi x_1}{e^{x_1} - \pi} = 1.6642 - \frac{e^{1.6642} - 1.6642\pi}{e^{1.6642} - \pi} = 1.6393$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = x_2 - \frac{e^{x_2} - \pi x_2}{e^{x_2} - \pi} = 1.6393 - \frac{e^{1.6393} - 1.6393\pi}{e^{1.6393} - \pi} = 1.6385$$

$$x_4 = x_3 - \frac{f(x_3)}{f'(x_3)} = x_3 - \frac{e^{x_3} - \pi x_3}{e^{x_3} - \pi} = 1.6385 - \frac{e^{1.6385} - 1.6385\pi}{e^{1.6385} - \pi} = 1.6385$$

A diferencia del método del Punto Fijo, la fórmula de Newton converge a ambas raíces. Para entender este comportamiento, una interpretación gráfica de la fórmula nos muestra la transformación geométrica que permite la convergencia para ambas raíces:

Gráfico de la fórmula de Newton interpretada como fórmula del Punto Fijo para el ejemplo anterior: $x = g(x) = x - \frac{f(x)}{f'(x)} \Rightarrow x_{i+1} = g(x_i) = x_i - \frac{e^{x_i} - \pi x_i}{e^{x_i} - \pi}$



Se observa que en la vecindad de cada raíz se tiene que **|g'(x)|<1**, adicionalmente **g'(x)** toma valores cercanos a cero en la región muy cercana a la raíz, por lo cual la convergencia es segura, es muy rápida y se puede llegar tan cerca de la raíz como se desee.

3.3.3 Existencia de un intervalo de convergencia para el método de Newton

Sea la ecuación **f(x) = 0**, y **r** una raíz real.

Si $f \in C^2[a,b]$, sabiendo que $\exists r \in [a,b]$ y que $f(r)=0$ y $f'(r) \neq 0$

Entonces $\exists I \subset R$ en el que el método de Newton converge

[a,b] es el intervalo de existencia de una raíz mientras que **I** es el intervalo de convergencia del método iterativo

Demostración

La fórmula de Newton es una forma particular del punto fijo. Para que se produzca la convergencia, la ecuación recurrente del método del punto fijo: **x=g(x)** debe cumplir la condición: $\forall x \in I |g'(x)|<1$ en algún intervalo **I** que contenga a la raíz **r**.

Para el método de Newton la ecuación recurrente es:

$$x = g(x) = x - \frac{f(x)}{f'(x)}$$

La condición de convergencia:

$$|g'(x)| < 1 \quad \Rightarrow \quad \left|\frac{f(x)f''(x)}{[f'(x)]^2}\right| < 1$$

Si **r** es una raíz de **f(x)=0,** y si $f'(r) \neq 0$, entonces:

$$g'(r) = \frac{f(r)f''(r)}{[f'(r)]^2} = 0 \quad \Rightarrow \quad \exists I \subset R (r \in I, \forall x \in I |g'(x)|<1)$$

$$\text{Si } x_i \in I \Rightarrow \lim_{i \to \infty} x_i \to r$$

Este resultado demuestra que existe algún intervalo **I** alrededor de **r** en el que la fórmula de Newton converge siempre que **g** y **g'** sean continuas **y** los valores calculados se mantengan dentro de este intervalo.

La definición de convergencia establecida: **|g'(x)| < 1** no resulta práctica para obtener un intervalo de convergencia para el método de Newton pues la fórmula $g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}$ requiere resolver una desigualdad complicada. En la siguiente sección se propone otra definición para la convergencia del método de Newton si **f(x)** cumple algunas propiedades.

3.3.4 Un teorema de convergencia local para el método de Newton

Sea la ecuación **f(x) = 0**, y **r** una raíz real

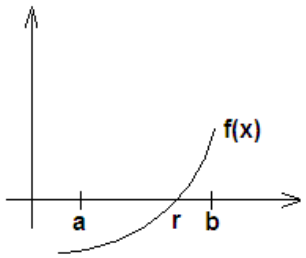
Si $f \in C^2[a,b]$ sabiendo que $\exists r \in [a,b]$ tal que $f(r)=0$ y $f'(r) \neq 0$ con las siguientes propiedades:

a) $f(x) > 0, \quad x \in (r, b)$
b) $f'(x) > 0, \quad x \in (r, b)$
c) $f''(x) > 0, \quad x \in (r, b)$

Entonces, la fórmula de Newton converge para cualquier valor inicial x_0 elegido en el intervalo **(r, b)**.

Demostración

El siguiente gráfico muestra la forma de **f** correspondiente al enunciado anterior



1) Las premisas **a)** y **b)** implican que $x_{i+1} < x_i$:

En la fórmula de Newton:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad \Rightarrow \quad x_{i+1} < x_i \qquad \textbf{(i)}$$

Siendo **f(x)**, **f'(x)** positivos, la desigualdad resulta al sumar un término positivo a la derecha.

2) La premisa **c)** implica que $r < x_{i+1}$:

Desarrollamos **f(x)** con dos términos y el error, en la serie de Taylor, alrededor de x_i :

$$f(r) = f(x_i) + (r - x_i)f'(x_i) + (r - x_i)^2 f''(z)/2! \;\Rightarrow\; f(r) > f(x_i) + (r - x_i)f'(x_i)$$

Siendo **f''(x)** positivo, la desigualdad resulta al restar un término positivo a la derecha.

$$\Rightarrow\; 0 > f(x_i) + (r - x_i)f'(x_i) \;\Rightarrow\; r < x_i - f(x_i)/f'(x_i) \;\Rightarrow\; r < x_{i+1} \qquad \textbf{(ii)}$$

Combinando los resultados **(i)** y **(ii)** se tiene: $r < x_{i+1} < x_i$, **i = 0, 1, 2,...**

Este resultado define una sucesión numérica decreciente que tiende a **r**, con lo cual se prueba la convergencia de la fórmula iterativa de Newton**:** $x_i \underset{i\to\infty}{\to} r$ si $x_0 \in (r, b)$.

Si se dispone del gráfico de **f** es fácil reconocer visualmente si se cumplen las condiciones **a)**, **b)** y **c)** y definir un intervalo de convergencia. Si se elije un valor inicial fuera de este intervalo, no se puede asegurar que el método converja.

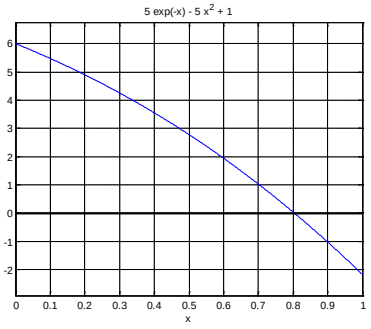
Si **f** tiene otra forma, igualmente se pueden enunciar y demostrar las condiciones para que se produzca la convergencia del método de Newton, llevando **f** a la forma analizada en la sección anterior mediante rotación o desplazamiento en los ejes horizontal o vertical.

El teorema de convergencia anterior puede usarse para obtener **analíticamente** el intervalo de convergencia si la ecuación no es complicada y se pueda analizar las propiedades de **f**, como en los ejemplos siguientes.

Ejemplo. Encuentre un intervalo de convergencia para calcular, con la fórmula de Newton, una raíz real de la siguiente ecuación: $\mathbf{f(x) = 5e^{-x} - 5x^2 + 1 = 0}$

Intervalo de existencia de una raíz real

La función **f** y sus derivadas son funciones contínuas en el conjunto **R**



$\mathbf{f(x) = 5e^{-x} - 5x^2 + 1}$
$\mathbf{f'(x) = -5e^{-x} - 10x}$
$\mathbf{f''(x) = 5e^{-x} - 10}$

$\mathbf{f(0) = 6,\ \ f(1) = -5}$

Por el Teorema del Valor Intermedio: $\exists \mathbf{r} \in [0, 1]$ tal que $\mathbf{f(r)=0}$
Además $\forall \mathbf{x} \in [0, 1]$, $\mathbf{f'(x)}$ no cambia de signo, entonces la raíz **r** es única

Intervalo de convergencia para la fórmula de Newton

Para el método de Newton, aplicándolo a la ecuación del ejemplo:

$$x = g(x) = x - \frac{f(x)}{f'(x)} \Rightarrow g'(r) = \frac{f(r)f''(r)}{[f'(r)]^2} = 0 \Rightarrow \exists I \subset R\,(r \in I, \forall x \in I |g'(x)|<1)$$

Propiedades de **f(x)**

a) $f(x)<0, \quad x \in (r, \infty)$
b) $f'(x)<0, \quad x \in (r, \infty)$
c) $f''(x)<0, \quad x \in (r, \infty)$

Si se hace una rotación alrededor del eje horizontal y se reescribe la ecuación:

$$f(x) = -(5e^{-x} - 5x^2 + 1) = 0$$

Entonces, esta **f(x)** cumple las propiedades del caso analizado en la demostración de la sección anterior: **f(x)>0, f'(x)>0, f''(x)>0,** en el intervalo: **$x \in (r, \infty)$**, y se puede concluir que el intervalo de convergencia es: **$x \in (r, \infty)$.**

Por lo tanto, un valor inicial que garantiza la convergencia es: **$x_0 = 1$** pues está dentro del intervalo de convergencia.

Cálculo de la raíz con la fórmula de Newton

$$x = g(x) = x - f(x)/f'(x) = x - (5e^{-x} - 5x^2 + 1)/(-5e^{-x} - 10x)$$

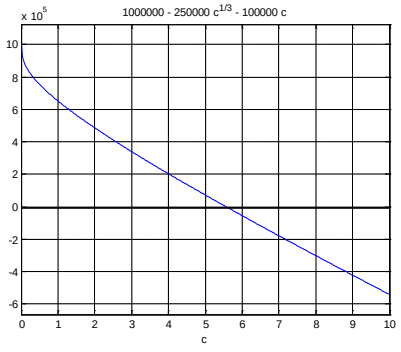
$x_0 = 1$
$x_1 = g(x_0) = 0.817507$
$x_2 = g(x_1) = 0.804607$
$x_3 = g(x_2) = 0.804544$
$x_4 = g(x_3) = 0.804544$

Ejemplo. El balance de masa de un contaminante en un lago bien mezclado se puede expresar con la ecuación: $V\frac{dc}{dt} = W - Qc - kV\sqrt[3]{c}$. Dados los valores de los parámetros: **V**=1x10^6, **Q**=1x10^5, **W**=1x10^6, **k**=0.25, en un sistema de unidades consistente, Encuentre la concentración **c** en el **estado estable**. Use la fórmula de Newton, **E**=1x10^{-4}. Previamente analice la existencia y convergencia.

Solución

Estado estable implica que $\frac{dc}{dt} = 0$

$f(c) = W - Qc - kV\sqrt[3]{c} = 10 - 2.5\sqrt[3]{c} - c = 0$ Sustituyendo y simplificando



Intervalo de existencia

f es contínua y decreciente en **[0, +∞)**

f(0) > 0

f(10) < 0 $\Rightarrow$ **r∈(0, 10)**

Por el Teorema del valor intermedio, existe una raíz real r en **(0, 10)**

Además **∀x∈(0, 10)**, **f'(x)** no cambia de signo, entonces la raíz **r** es única

Convergencia

f es estrictamente decreciente en **(0, r) ⊂ (0, 10)** y se tiene:

f(c)>0, c∈(0, r)

f'(c)<0, c∈(0, r)

f''(c)>0, c∈(0, r)

Mediante una rotacion alrededor del eje vertical: f(-x), por la correspondencia con el Teorema de convergencia local, la fórmula de Newton converge para $\mathbf{c_0 \in (0, r)}$

Elegir un intervalo de convergencia:

f(0)>0
f(5)>0
f(6)<0 $\Rightarrow$ Intervalo de convergencia: $\mathbf{c_0 \in (0, 5) \subset (0, r)}$

Iteraciones con la fórmula de Newton

$$\mathbf{f'(c) = -\frac{5}{6\sqrt[3]{c^2}} - 1}$$

$$\mathbf{c = g(c) = c - \frac{f(c)}{f'(c)} = c + \frac{6\sqrt[3]{c^2}(10 - 2.5\sqrt[3]{c} - c)}{5 + 6\sqrt[3]{c^2}} = c + \frac{6c(10 - 2.5\sqrt[3]{c} - c)}{5\sqrt[3]{c} + 6c}}$$

$\mathbf{c_0 = 5}$
$\mathbf{c_1 = g(c_0) = 5.5643,}$ $\mathbf{E \cong 0.5646}$
$\mathbf{c_2 = g(c_1) = 5.5688,}$ $\mathbf{E \cong 0.0045}$
$\mathbf{c_3 = g(c_2) = 5.5688,}$ $\mathbf{E \cong 0}$

Aplicación de la fórmula de Newton para cálculo de raíces reales

Ejemplo. Una partícula se mueve en el espacio con el vector de posición **R(t) = (2cos(t), sen(t), 0)**. Se requiere conocer el tiempo en el que el objeto se encuentra más cerca del punto **P(2, 1, 0).** Utilice el método de Newton con cuatro decimales de precisión.

Solución

Distancia entre dos puntos:

$$d(t) = \sqrt{(2\cos(t) - 2)^2 + (\text{sen}(t) - 1)^2 + (0 - 0)^2}$$

Modelo matemático: **f(t) = d'(t) = 0**

$$f(t) = d'(t) = \frac{2(\cos(t)(\text{sen}(t) - 1) - 4\text{sen}(t)(2\cos(t) - 2)}{2\sqrt{(2\cos(t) - 2)^2 + (\text{sen}(t) - 1)^2}} = 0$$

$$f(t) = \cos(t)(\text{sen}(t) - 1) - 4\text{sen}(t)(\cos(t) - 1) = 3\text{sen}(t)\cos(t) - 4\text{sen}(t) + \cos(t) = 0$$

$$f'(t) = 4\cos(t) + \text{sen}(t) - 3\cos(t)^2 + 3\text{sen}(t)^2$$

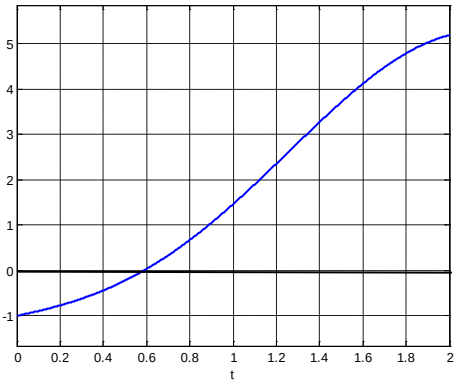


Gráfico de f(t)

Fórmula de Newton

$$t_{i+1} = t_i - \frac{f(t_i)}{f'(t_i)}, \quad i = 0,1,2,...$$

Sustituír las fórmulas anteriores en la fórmula de Newton y calcular:

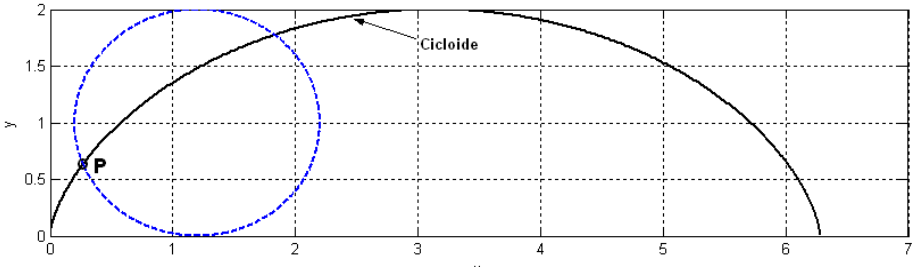
$t_0 = 0.5$ Estimación tomada del gráfico
$t_1 = 0.5938$
$t_2 = 0.5873$
$t_3 = 0.5872$
$t_4 = 0.5872$

Ejemplo. Si un círculo de radio **a** rueda en el plano a lo largo del eje horizontal, un punto **P** de la circunferencia trazará una curva denominada cicloide. Esta curva puede expresarse mediante las siguientes ecuaciones paramétricas

$$x(t) = a(t - \text{sen } t), \qquad y(t) = a(1 - \cos t)$$

Suponga que el radio es **1** metro, si **(x, y)** se miden en metros y **t** representa tiempo en segundos, determine el primer instante en el que la magnitud de la velocidad es **0.5** m/s. Use el método de Newton**, E=0.0001**

Gráfico de la cicloide



Su trayectoria:

$$u(t) = (x(t), y(t)) = (t - \text{sen } t,\ 1 - \cos t)$$

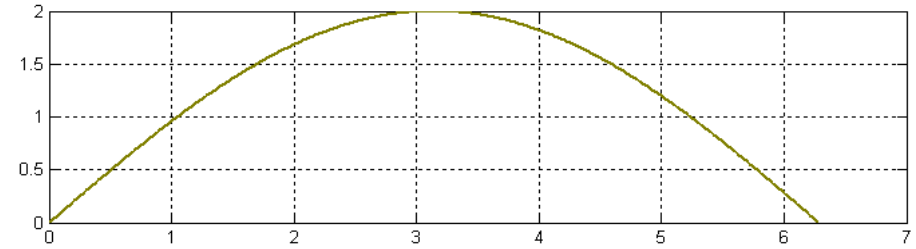
Su velocidad:

$$u'(t) = (1 - \cos t, \text{sen } t)$$

Magnitud de la velocidad:

$$\|u'(t)\| = \sqrt{(1-\cos t)^2 + (\text{sen}\, t)^2}$$

Gráfico de la magnitud de la velocidad



Dato especificado:

$$\sqrt{(1-\cos t)^2 + (\text{sen}\, t)^2} = 0.5 \quad \Rightarrow \quad f(t) = (1 - \cos t)^2 + (\text{sen}\, t)^2 - 0.25 = 0$$

Método de Newton

$$t_{i+1} = t_i - \frac{f(t_i)}{f'(t_i)} = t_i - \frac{(1-\cos t_i)^2 + (\text{sen}\, t_i)^2 - 0.25}{2(1-\cos t_i)(\text{sen}\, t_i) + 2(\text{sen}\, t_i)(\cos t_i)}$$

fórmula iterativa

$t_0 = 0.5$ del gráfico

$t_1 = \ldots = 0.505386$

$t_2 = \ldots = 0.505360$

$t_3 = \ldots = 0.505360$

3.3.5 Práctica computacional

En esta sección se describe el uso de Python para usar el método de Newton. Se lo hará directamente en la ventana interactiva.

Para calcular una raíz debe elegirse un valor inicial cercano a la respuesta esperada de acuerdo a la propiedad de convergencia estudiada para este método.

Para realizar los cálculos se usa la fórmula de Newton en notación algorítmica:

$$x_{i+1} = g(x_i) = x_i - \frac{f(x_i)}{f'(x_i)}, \quad i = 0, 1, 2, 3, \ldots$$

x_0 es el valor inicial

$x_1, x_2, x_3, \ldots$ son los valores calculados

En los lenguajes computacionales como Python no se requieren índices para indicar que el valor de una variable a la izquierda es el resultado de la evaluación de una expresión a la derecha con un valor anterior de la misma variable.

La ecuación se puede definir como una expresión matemática usando con la librería **Sympy**

La variable independiente se declara con **Symbol**. La derivada se obtiene con la función **diff** y con la función **float** se evalúan las expresiones matemáticas.

Formato sugerido para uso computacional de la fórmula de Newton en Python desde la ventana interactiva:

```
>>> from sympy import*          Cargar la librería simbólica
>>> x=Symbol('x')               Definir la variable independiente
>>> f=                          Definir la ecuación f
>>> g=x-f/diff(f)               Definir la fórmula iterativa g
>>> t=                          Elegir el valor inicial
>>> t=float(g.subs(x,t));print(t)   Evaluar la fórmula iterativa
                                    y mostrar el valor calculado
                                    float evalua en formato decimal
```

La última línea se la debe usar repetidamente para observar la convergencia o divergencia

Finalmente, si la fórmula converge, debe evaluar la ecuación con el último valor calculado:

```
>>> float(f.subs(x,t))
```

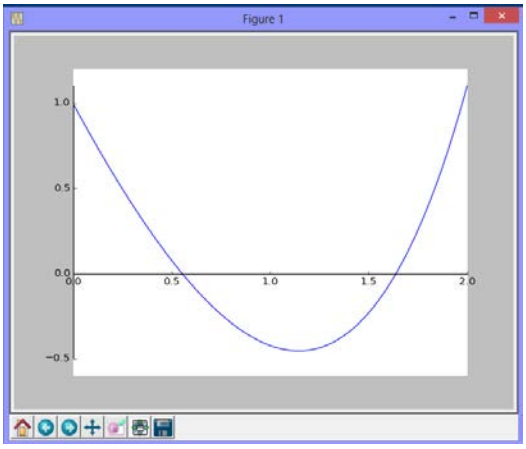
El resultado debe ser un valor cercano a cero

Cálculo computacional de raíces reales en la ventana interactiva

Ejemplo. Calcule en la ventana interactiva de Python las raíces reales de $\mathbf{f(x) = e^x - \pi x = 0}$ con la fórmula de Newton.

Es conveniente graficar la ecuación

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> plot(f,(x,0,2))
```



En el gráfico se observan dos raíces reales

NOTA. Para no cerrar el gráfico, se sugiere abrir otra ventana de Python para realizar los cálculos:

Cálculo de la primera raíz

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> g=x-f/diff(f)
>>> t=0.5                                   Valor inicial del gráfico
>>> t=float(g.subs(x,t));print(t)
0.552198029112459
>>> t=float(g.subs(x,t));print(t)
0.5538253947739784
>>> t=float(g.subs(x,t));print(t)
0.5538270366428404
>>> t=float(g.subs(x,t));print(t)
0.5538270366445136
>>> t=float(g.subs(x,t));print(t)
0.5538270366445136
```

```
>>> float(f.subs(x,t))                                    Verificar si f(t) = 0
-1.5192266539886956e-17
```

El último resultado tiene dieciseis decimales fijos. Se puede observar la rapidez con la que el método se acerca a la respuesta duplicando aproximadamente, la precisión en cada iteración. Esto concuerda con la propiedad de convergencia cuadrática. Es necesario verificar que este resultado satisface a la ecuación:

Cálculo de la segunda raíz

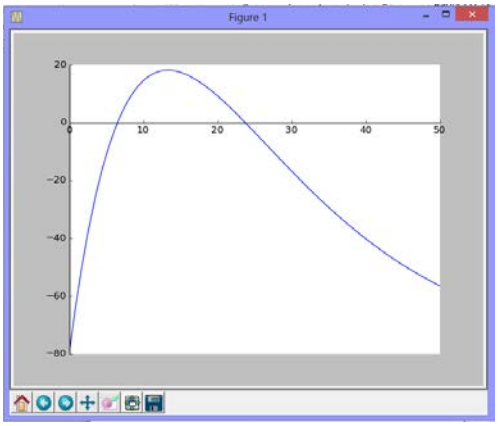
Si se comienza con **t = 1.5** se puede calcular la otra raíz real:

```
>>> t=1.5
...
...
...
>>> t=float(g.subs(x,t));print(t)
1.6385284199703636

>>> float(f.subs(x,t))                                    Verificar si f(t) = 0
4.591445985947165e-16
```

Ejemplo. Se propone el siguiente modelo para describir la demanda de un producto, en donde **x** es tiempo en meses: $\mathbf{d(x) = 20x\ e^{-0.075x}}$. Encuentre el valor de **x** para el cual la demanda alcanza por primera vez el valor de 80 unidades. Use el método de Newton para los cálculos. Elija el valor inicial del gráfico y muestre los valores intermedios.

La ecuación a resolver es: $\mathbf{f(x) = 20x\ e^{-0.075x} - 80 = 0}$

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=20*x*exp(-0.075*x)-80
>>> plot(f,(x,0,50))
>>> g=x-f/diff(f)          Fórmula iterativa
>>> t=5
>>> t=float(g.subs(x,t));print(t)
6.311945053556488
>>> t=float(g.subs(x,t));print(t)
6.520455024943886
>>> t=float(g.subs(x,t));print(t)
6.525360358429756
>>> t=float(g.subs(x,t));print(t)
6.525363029068741
>>> t=float(g.subs(x,t));print(t)
6.525363029069533
```



```
>>> t=float(g.subs(x,t));print(t)
6.525363029069533
>>> f.subs(x,t)
1.42108547152020e-14
```

Ejemplo. Una partícula se mueve en el plano **X - Y** de acuerdo con las ecuaciones paramétricas siguientes, donde **t ∈ [0, 1]** es tiempo:

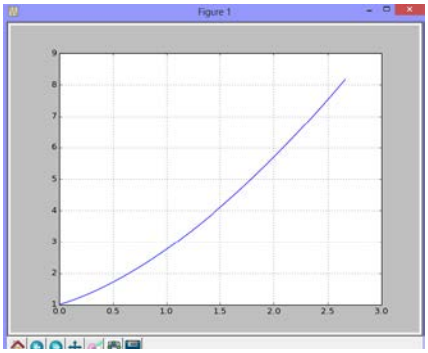
x(t)=t*exp(t); y(t)=1+t*exp(2t)

Con la fórmula de Newton calcule el tiempo en el que la partícula está más cerca de **(1,1)**

Distancia de un punto **(x, y)** al punto **(1, 1)** : $\mathbf{d = \sqrt{(x(t)-1)^2 + (y(t)-1)^2}}$

Para encontrar la menor distancia, debe resolverse la ecuación: **f(t) = d'(t) = 0**

```
>>> import pylab as pl
>>> t=pl.arange(0,1,0.01)
>>> x=t*exp(t)
>>> y=1+t*exp(2*t)
>>> pl.plot(x,y,'b')
>>> pl.grid(True)
>>> pl.show()
```




```
>>> from sympy import*
>>> t=Symbol('t')
>>> d=sqrt((t*exp(t)-1)**2+(1+t*exp(2*t)-1)**2)
>>> f=diff(d)
>>> g=t-f/diff(f)                                  Fórmula iterativa
>>> u=0.5
>>> u=float(g.subs(t,u));print(u)
0.2782466390677125
>>> u=float(g.subs(t,u));print(u)
0.25831052765669904
>>> u=float(g.subs(t,u));print(u)
0.25677759974260406
>>> u=float(g.subs(t,u));print(u)
0.2567682382596691
>>> u=float(g.subs(t,u));print(u)
0.2567682379103996
>>> u=float(g.subs(t,u));print(u)
0.2567682379103996                    tiempo para la menor distancia

>>> d.subs(t,u)
0.794004939848305                     es la menor distancia
```

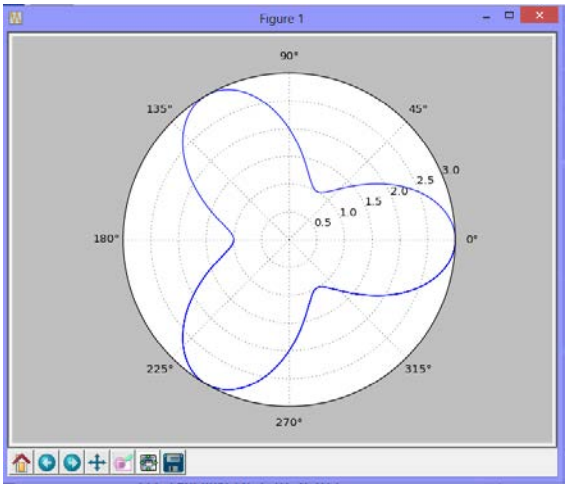
Ejemplo. Encuentre una intersección de las siguientes ecuaciones en coordenadas polares

$r = 2+\cos(3*t)$, $r = 2 - e^t$

Ecuación a resolver: $f(t) = 2+\cos(3*t) - (2 - e^t) = 0$

```
>>> import pylab as pl
>>> t=pl.arange(-pi,2*pi,0.01)
>>> r=2+cos(3*t)
>>> pl.polar(t,r)
>>> pl.grid(True)
>>> pl.show()
```

Gráfico en coordenadas polares




```
>>> from sympy import*
>>> t=Symbol('t')
>>> f=2+cos(3*t)-(2-exp(t))
>>> g=t-f/diff(f)
>>> u=-1
>>> u=float(g.subs(t,u));print(u)
-0.21374870355715347
>>> u=float(g.subs(t,u));print(u)
-0.8320496091165962
>>> u=float(g.subs(t,u));print(u)
-0.6696807111120446
>>> u=float(g.subs(t,u));print(u)
-0.696790503081824
>>> u=float(g.subs(t,u));print(u)
-0.6973288907051911
>>> u=float(g.subs(t,u));print(u)
-0.6973291231341587
>>> u=float(g.subs(t,u));print(u)
-0.6973291231342021
```

```
>>> u=float(g.subs(t,u));print(u)
-0.6973291231342021                    (ángulo: -39.95…. grados)

>>> 2+cos(3*u)
1.50208660521455                       (radio)
```

3.3.6 Instrumentación computacional del método de Newton

Para evitar iterar desde la ventana interactiva, se puede instrumentar una función que reciba la ecuación a resolver **f**, la variable independiente **v** y el valor inicial **u**. Adicionalmente se debe enviar un parámetro **e** para controlar la precisión requerida y otro parámetro **m** para el máximo de iteraciones.

La función entrega la solución calculada **r** con el error definido. Si el método no converge en el máximo de iteraciones previsto, la función entregará un valor nulo.

Variables de entrada

- **f**: Ecuación a resolver
- **v**: Variable independiente
- **u**: Valor inicial
- **e**: Error esperado en la respuesta
- **m**: Máximo de iteraciones permitido

Variables de salida

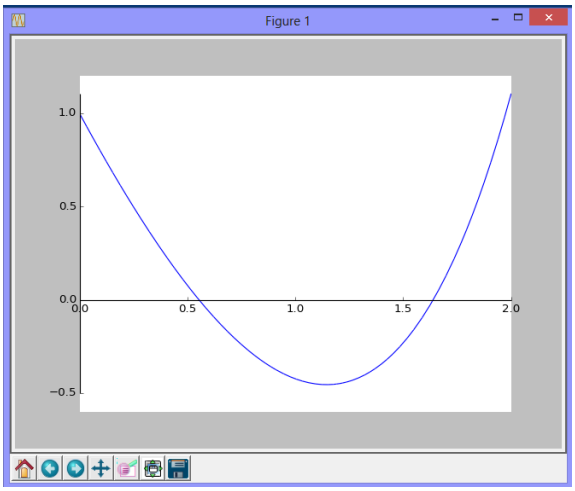
- **r**: Valor aproximado para la raíz

```
from sympy import*
def newton(f, v, u, e, m):
    g=v-f/diff(f,v)
    for i in range(m):
        r=float(g.subs(v,u))
        if abs(r-u)<e:
            return r
        u=r
    return None
```

Ejemplo. Calcular las raíces reales de **$f(x) = e^x - \pi x = 0$** con la función **newton**, $E=10^{-6}$

Un gráfico para estimar los valores iniciales

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> plot(f,(x,0,2))
```



La primera raíz real está cerca de **0.5**

NOTA. Para no cerrar el gráfico, se sugiere abrir otra ventana de Python para realizar los cálculos:

```
>>> from newton import*                    (Newton incluye a la librería Sympy)
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> r=newton(f,x,0.5,0.000001,10)
>>> print(r)
0.5538270366445136
```

Si se comienza con **1.5** se puede calcular la otra raíz real:

```
>>> r=newton(f,x,1.5,0.000001,10)
>>> print(r)
1.6385284199703631
```

Uso de la función **solve** de **Sympy** para obtener las raíces de la ecuacion del ejemplo inicial: $\mathbf{f(x) = e^x - \pi x = 0}$

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(x)-pi*x
>>> r=solve(f)
>>> len(r)
1
>>> float(r[0])
0.5538270366445136
```

La función **solve** solo permitió calcular una raíz

3.3.7 Cálculo de raíces complejas

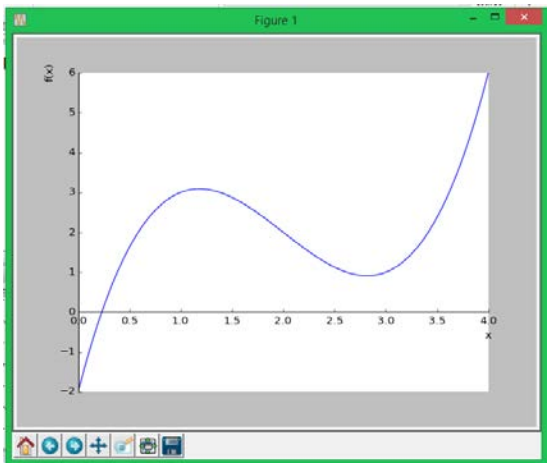
La fórmula de Newton también es aplicable al cálculo de raíces complejas. Si el valor inicial es un número complejo, la secuencia de valores generados también serán valores complejos y se puede observar la convergencia

Ejemplo. Con la fórmula de Newton calcular todas las raíces de la siguiente ecuación polinomial de tercer grado.

$$\mathbf{p(x) = x^3 - 6x^2 + 10x - 2 = 0}$$

El gráfico y el cálculo se realizarán usando los recursos de la librería simbólica de Python

```
>>> from sympy import*
>>> x=Symbol('x')
>>> p=x**3-6*x**2+10*x-2
>>> plot(p,(x,0,4))
```



Se puede observar que existe una raíz real r en el intervalo [0,0.5]. La región de convergencia está a la izquierda de r, por lo que el valor cero es un valor inicial adecuado:

NOTA. Para no cerrar el gráfico, se sugiere abrir otra ventana de Python para realizar los cálculos:

```
>>> from sympy import*
>>> x=Symbol('x')
>>> p=x**3-6*x**2+10*x-2
>>> g=x-p/diff(p)
>>> r=0
>>> r=float(g.subs(x,r));print(r)
0.2
>>> r=float(g.subs(x,r));print(r)
0.23005181347150264
>>> r=float(g.subs(x,r));print(r)
0.23070733709405902
>>> r=float(g.subs(x,r));print(r)
0.2307076457613002
>>> r=float(g.subs(x,r));print(r)
0.23070764576136857
```

Las raíces complejas están alrededor de x=3. La región de convergencia parece estar a la derecha. Para iniciar la secuencia de valores, se debe comenzar con un número complejo con un componente real en esta región:

```
>>> r=(3+1j)
>>> r=complex(g.subs(x,r));print(r)
(2.9+0.7j)
>>> r=complex(g.subs(x,r));print(r)
(2.883669486011711+0.6002602472348734j)
>>> r=complex(g.subs(x,r));print(r)
(2.884588967417228+0.5898358998650888j)
>>> r=complex(g.subs(x,r));print(r)
(2.8846461653034137+0.5897428062018362j)
>>> r=complex(g.subs(x,r));print(r)
(2.8846461771193157+0.5897428050222053j)
>>> r=complex(g.subs(x,r));print(r)
(2.8846461771193153+0.589742805022205j)
```

La otra raíz compleja será el conjugado del resultado anterior

Otra opción para calcular las raíces complejas en el ejemplo anterior es calcular la raíz real **r** y luego factorarla del polinomio:

Si r es una raíz de **p(x) = 0**, entonces **q(x)(x-r) = p(x)**. Se puede usar la fórmula de la ecuación cuadrática para obtener las raíces complejas de la ecuación con el polinomio de grado dos **q(x) = 0** que se obtiene al dividir **p(x)** para **(x-r)**.

3.4 Ejercicios y problemas de ecuaciones no-lineales

1. La suma de dos números reales positivos es 5 y el producto de sus cuadrados es 20. Encuentre estos dos números.

2. El producto de las edades en años de dos personas es 677.35 y si se suman los cubos de ambas edades se obtiene 36594.38 Encuentre cuales son estas edades.

3. Una empresa produce semanalmente una cantidad de artículos. El costo de producción semanal tiene un costo fijo de **250** y un coso de **2.50** por cada artículo producido. El ingreso semanal por venta tiene un valor de **3.50** por cada artículo vendido, más un costo de oportunidad que se ha estimado directamente proporcional a la cantidad de artículos producidos, multiplicado por el logaritmo natural. Encuentre el punto de equilibrio para este modelo.

4. En una empresa de fertilizantes el modelo **v(x) = 0.4x (30 - x)** corresponde a las ventas en miles de dólares cada mes, mientras que el costo de producción mensual, en miles de dólares es **c(x) = 5 + 10 ln(x)**, siendo **x** la cantidad producida en toneladas, **1≤x≤30**.
a) Determine la cantidad que debe producirse para que la ganancia mensual sea 40
b) Determine la cantidad mensual que se debe producir para obtener la máxima ganancia.

5. El costo semanal fijo por uso de un local para venta de cierto tipo de artículo es $50. Producir cada Kg. del artículo cuesta $2.5. El ingreso por venta es **3x + 2 ln(x)** en donde **x** representa la cantidad de kg vendidos en cada semana. Determine la cantidad de Kg. que se debe vender semanalmente a partir de la cual la empresa obtiene ganancias.

6. En una planta de abastecimiento de combustible se tiene un tanque de forma esférica. El volumen del líquido almacenado en el tanque se puede calcular mediante la siguiente fórmula: $\mathbf{v(h) = \pi h^2 (R - h/3)}$,donde h representa la altura del líquido dentro del tanque medida desde la parte inferior del tanque y R su radio ($0 \leq h \leq 2R$). Suponga que el tanque tiene un radio de 2 m. Calcule la altura que debe tener el líquido para que el tanque contenga 27 m^3. Calcule el resultado con una tolerancia $E=10^{-4}$.

7. Encuentre el punto de la curva dada por $\mathbf{y = 2x^5 - 3xe^{-x} - 10}$ ubicado en el tercer cuadrante, donde su recta tangente sea paralela al eje **X**.

8. Determine la raíz real de la ecuación: **sin(x) = ln(x)**

9. Determine la segunda raíz real positiva, con 4 decimales exactos de la ecuación:
cos(πx) = tan(πx)

10. Calcule una raíz real positiva de la ecuación **sin(πx) + 1 = x.**

11. Para que **f** sea una función de probabilidad se tiene que cumplir que su integral en el dominio de **f** debe tener un valor igual a **1**. Encuentre el valor de **b** para que la función $\mathbf{f(x)=2x^2 + x}$ sea una función de probabilidad en el dominio **[0, b]**.

12. Calcule con cuatro decimales exactos la intersección de la ecuación $\mathbf{(y-1)^2 + x^3 = 6}$, con la recta que incluye a los puntos **(-1, -1), (1, 1)** en el plano.

13. Encuentre una fórmula iterativa de convergencia cuadrática y defina un intervalo de convergencia apropiado para calcular la raíz real n-ésima de un número real. El algoritmo solamente debe incluir operaciones aritméticas elementales.

14. El siguiente es un procedimiento intuitivo para calcular una raíz real positiva de la ecuación **f(x) = 0** en un intervalo **[a, b]** con precisión **E**:

A partir de **x = a** evalúe **f(x)** incrementando **x** en un valor **d**. Inicialmente **d = (b – a)/10** Cuando **f** cambie de signo, retroceda **x** al punto anterior **x – d**, reduzca **d** al valor **d/10** y evalúe nuevamente **f** hasta que cambie de signo Repita este procedimiento hasta que **d** sea menor que **E**.

a) De manera formal escriba las condiciones necesarias para que la raíz exista, sea única y pueda ser calculada.
b) Indique el orden de convergencia y estime el factor de convergencia del método.
c) Describa el procedimiento anterior en notación algorítmica, o en MATLAB o en Python

15. Se propone resolver la ecuación $\int_0^{\mathbf{x}}(\mathbf{5}-\mathbf{e^u})\mathbf{du} = \mathbf{2}$ con el **método del punto fijo**
a) Obtenga la ecuación **f(x) = 0** resolviendo el integral
b) Mediante un gráfico aproximado, o evaluando directamente, localice la raíces reales.
c) Proponga una ecuación equivalente **x = g(x)** y determine el intervalo de convergencia para calcular una de las dos raíces.
d) Del intervalo anterior, elija un valor inicial y realice 5 iteraciones. En cada iteración verifique que se cumple la condición de convergencia del punto fijo y estime el error de truncamiento en el último resultado.

.
16. El valor neto **C** de un fondo de inversiones se lo ha modelado con $\mathbf{C(t) = Ate^{-t/3}}$ en donde **A** es el monto inicial y **t** es tiempo.

a) Encuentre el tiempo **t** en el que el fondo de inversiones **C(t)** alcanzaría el máximo
b) Determine el monto de la inversión inicial **A** necesaria para que el valor máximo de **C(t)** sea igual a **1** en el tiempo especificado anteriormente.
c) Con la inversión inicial anterior **A**, encuentre el tiempo **t** en el que el fondo de inversiones **C(t)** se reduciría a **0.25**. Use el **método de Newton**, **E =** 10^{-4}**.**

17. La siguiente ecuación relaciona el factor de fricción **f** y el número de Reynolds **Re** para flujo turbulento que circula en un tubo liso: $\mathbf{1/f = -0.4 + 1.74\ ln(Re\ f)}$
Calcule el valor de **f** para **Re = 20000**

18. Un ingeniero desea tener una cantidad de dólares acumulada en su cuenta de ahorros para su retiro luego de una cantidad de años de trabajo. Para este objetivo planea depositar un valor mensualmente. Suponga que el banco acumula el capital mensualmente mediante la siguiente fórmula:

$$A = P\left[\frac{(1+x)^n - 1}{x}\right], \text{ en donde}$$

A: Valor acumulado, **P**: Valor de cada depósito mensual
n: Cantidad de depósitos mensuales, **x:** Tasa de interés mensual
Determine la tasa de interés anual que debe pagarle el banco si desea reunir 200000 en 25 años depositando cuotas mensuales de 350

19. Una empresa compra una máquina en 20000 dólares pagando 5000 dólares cada año durante los próximos 5 años. La siguiente fórmula relaciona el costo de la máquina **P**, el pago anual **A**, el número de años **n** y el interés anual **x**: $A = P\frac{x(1+x)^n}{(1+x)^n - 1}$

Determine la tasa de interés anual **x** que se debe pagar.

20. Una empresa vende un vehículo en **P**=$34000 con una entrada de **E**=$7000 y pagos mensuales de **M**=$800 durante cinco años. Determine el interés mensual **x** que la empresa está cobrando. Use la siguiente fórmula:

$$P = E + \frac{M}{x}\left[1 - \frac{1}{(1+x)^n}\right], \quad \text{en donde } \mathbf{n} \text{ es el número total de pagos mensuales}$$

21. Un modelo de crecimiento poblacional está dado por: $\mathbf{f(t) = 5t + 2e^{0.1t}}$, en donde n es el número de habitantes, t es tiempo en años.

a) Calcule el número de habitantes que habrán en el año 25
b) Encuentre el tiempo para el cual la población es 200

22. Un modelo de crecimiento poblacional está dado por. $\mathbf{f(t) = kt + 2e^{0.1t}}$, siendo **k** una constante que debe determinarse y **t** tiempo en años. Se conoce que **f(10)=50**.

a) Determine la población en el año 25
b) Determine el año en el que la población alcanzará el valor 1000.

23. Un modelo de crecimiento poblacional está dado por $\mathbf{f(t) = k_1 t + k_2 e^{0.1t}}$

Siendo $\mathbf{k_1}$ y $\mathbf{k_2}$ constantes, y **t** tiempo en años.
Se conoce que f(10)=25, f(20)=650.
Determine el año en el que la población alcanzará el valor 5000.

24. La concentración de bacterias contaminantes **c** en un lago decrece de acuerdo con la relación: $\mathbf{c = 70e^{-1.5t} + 25e^{-0.075t}}$.
Se necesita determinar el tiempo para que la concentración de bacterias se reduzca a 15 unidades o menos.
a) Determine un intervalo de existencia de la raíz de la ecuación. Use un gráfico
b) Aproxime la raíz indicando la cota del error.

25. En un modelo de probabilidad se usa la siguiente fórmula para calcular la probabilidad **f(k)** que en el intento número k se obtenga el primer resultado favorable:

$$f(k) = p(1-p)^{k-1}, 0 \le p \le 1, k=0, 1, 2, 3, \ldots$$

a) Si en una prueba se obtuvo que **f(5) = 0.0733**, encuentre cuales son los dos posibles valores de **p** posibles en la fórmula.
b) Con el menor valor obtenido para **p** encuentre el menor valor de **k** para el que **f(k)<0.1**

26. Sean **a, b** las longitudes de los dos segmentos en los que se ha dividido una recta, siendo **a>b**. La relación $\frac{a}{b}$ que satisface a la ecuación $\frac{a+b}{a} = \frac{a}{b}$ se denomina "número áureo". Este número tiene propiedades muy interesantes que han sido estudiadas desde la antigüedad.

a) Proponga una ecuación para encontrar numéricamente el valor de la relación **a/b**
b) Localice la raíz real y encuentre un intervalo de convergencia con el Teorema de Convergencia Local si se desea usar el método de Newton.
c) Calcule la raíz con 6 decimales exactos.

27. Para simular la trayectoria de un cohete se usará el siguiente modelo:

$$y(t) = 6 + 2.13t^2 - 0.0013t^4$$

En donde **y** es la altura alcanzada, en metros y **t** es tiempo en segundos. El cohete está colocado verticalmente sobre la tierra.

a) Encuentre el tiempo de vuelo.
b) Encuentre la altura máxima del recorrido.

28. El movimiento de una partícula en el plano, se encuentra representado por las ecuaciones paramétricas:

$$x(t) = 3\,sen^3(t) - 1; \quad y(t) = 4\,sen(t)\cos(t); \quad t \ge 0$$

Donde **x, y** son las coordenadas de la posición expresadas en cm, **t** se expresa en seg.

a) Demuestre que existe un instante $t \in [0, \pi/2]$ tal que sus coordenadas **x** e **y** coinciden.
b) Encuentre con un error de 10^{-5} en qué instante las dos coordenadas serán iguales en el intervalo dado en a).

29. Los polinomios de Chebyshev $T_n(x)$ son utilizados en algunos métodos numéricos. Estos polinomios están definidos recursivamente por la siguiente formulación:

$$T_o(x) = 1, \quad T_1(x) = x$$
$$T_n(x) = 2.x.T_{n-1}(x) - T_{n-2}(x), \quad n = 0, 1, 2, 3, \ldots$$

Calcule todas las raíces reales positivas de $T_7(x)$.

30. La posición del ángulo central θ en el día t de la luna alrededor de un planeta si su período de revolución es P días y su excentricidad es e, se describe con la ecuación de Kepler:

$$2\pi t - P\theta + P e \operatorname{sen}\theta = 0$$

Encuentre la posición de la luna (ángulo central) en el día 30, sabiendo que el período de revolución es 100 días y la excentricidad 0.5.

31. Una partícula se mueve en el plano **XY** (la escala está en metros) con una trayectoria descrita por la función $\mathbf{u(t) = (x(t), y(t)) = \left(2t\, e^{t} + \sqrt[4]{t},\ 2t^{3}\right)}$, $\mathbf{t \in [0,1]}$, **t** medido en horas.

a) Grafique la trayectoria **u(t)**
b) Encuentre la posición de la partícula cuando **x=3.**
c) En que el instante la partícula se encuentra a una distancia de **4** metros del origen.

32. En una región se instalan **100** personas y su tasa de crecimiento es $\mathbf{e^{0.2x}}$, en donde **x** es tiempo en años. La cantidad inicial de recursos disponibles abastece a **120** personas. El incremento de recursos disponibles puede abastecer a una tasa de **10x** personas, en donde **x** es tiempo en años. Se desea conocer cuando los recursos no serán suficientes para abastecer a toda la población. Calcule la solución con cuatro dígitos de precisión y determine el año, mes y día en que se producirá este evento.

33. Suponga que el precio de un producto **f(x)** depende del tiempo **x** en el que se lo ofrece al mercado con la siguiente relación $\mathbf{f(x) = 25x \exp(-0.1x),\ 0 \le x \le 12,}$ en donde **x** es tiempo en meses. Se desea determinar el día en el que el precio sube a **80**.
a) Evalúe **f** con **x** en meses hasta que localice una raíz real (cambio de signo) y trace la forma aproximada de **f(x)**
b) Calcule la respuesta (mes) con $E=10^{-4}$. Exprese esta respuesta en días (1mes = 30 días)
c) Encuentre el día en el cual el precio será máximo. $E=10^{-4}$

34. Determine de ser posible, el valor del parámetro $\alpha > 0$, tal que $\int_{\alpha}^{2\alpha} xe^{x}dx = 10$.

35. Una partícula sigue una trayectoria elíptica centrada en el origen (eje mayor 4 y eje menor 3) comenzando en el punto más alto, y otra partícula sigue una trayectoria parabólica ascendente hacia la derecha comenzando en el origen (distancia focal 5). El recorrido se inicia en el mismo instante.

a) Encuentre el punto de intersección de las trayectorias.

b) Si la primera partícula tiene velocidad uniforme y pasa por el punto más alto cada minuto, determine el instante en el cual debe lanzarse la segunda partícula con aceleración 10 m/s^2 para que intercepte a la primera partícula.

36. La velocidad **V** de un paracaidista está dada por la fórmula:

$$V = \frac{gx}{c}(1 - e^{-\frac{c\,t}{x}})$$

En donde **g**=**9.81** m/s^2 (gravedad terrestre, **c**=**14** kg/s (coeficiente de arrastre)
t: tiempo en segundos, **x**: masa del paracaidista en kg.

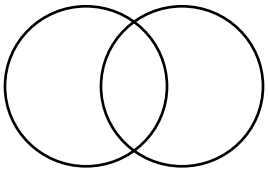
Cuando transcurrieron **7** segundos se detectó que la velocidad es **35** m/s. Determine la masa del paracaidista. E=0.001

37. Un tanquero de forma cilíndrica de 2 metros de diámetro está lleno de agua y debe entregar su contenido en partes iguales a tres lugares. El único instrumento de medición es una tira de madera que el operador sumerge desde la parte superior hasta el fondo del tanque. La humedad en la tira indica la cantidad de agua. Encuentre las alturas en las que debe marcarse la tira para que el operador del tanquero pueda conocer que ha entregado la misma cantidad de agua en los tres sitios.

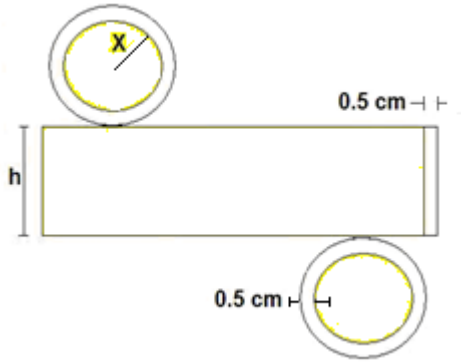
38. Una esfera de densidad 0.4 y radio 5 flota parcialmente sumergida en el agua. Encuentre la profundidad h que se encuentra sumergida la esfera.

Nota: requiere conocer la densidad del agua y el volumen de un segmento esférico.

39. En el siguiente gráfico de dos circunferencias con radio igual a 2, el área de la intersección es igual a π. Determine la distancia entre los centros de las circunferencias, E=10^{-6}



40. Un empresario desea producir recipientes cilíndricos de aluminio de un litro de capacidad. Los bordes deben tener **0.5** cm. adicionales para sellarlos. Determine las dimensiones del recipiente para que la cantidad de material utilizado en la fabricación sea mínima. Calcule una solución factible con error **10^{-4}**



3.5 Raíces reales de sistemas de ecuaciones no-lineales

En general este es un problema difícil, por lo que conviene intentar reducir el número de ecuaciones y en caso de llegar a una ecuación, poder aplicar alguno de los métodos conocidos.

Si no es posible reducir el sistema, entonces se intenta resolverlo con métodos especiales para sistemas de ecuaciones no-lineales.

Debido a que el estudio de la convergencia de estos métodos es complicado, se prefiere utilizar algún método eficiente, de tal manera que numéricamente pueda determinarse la convergencia o divergencia con los resultados obtenidos.

Una buena estrategia consiste en extender el método de Newton, cuya convergencia es de segundo orden, al caso de sistemas de ecuaciones no lineales. En esta sección se describe la fórmula para resolver un sistema de **n** ecuaciones no lineales y se la aplica a la solución de un sistema de dos ecuaciones. Al final de este capítulo se propone una demostración más formal de esta fórmula.

3.5.1 Fórmula iterativa de segundo orden para calcular raíces reales de sistemas de ecuaciones no-lineales

Sean **F: f_1, f_2, …, f_n** sistema de ecuaciones no lineales con variables **X: x_1, x_2, …, x_n**. Se requiere calcular un vector real que satisfaga al sistema **F**

En el caso de que **F** contenga una sola ecuación **f** con una variable **x**, la conocida fórmula iterativa de Newton puede escribirse de la siguiente manera:

$$x^{(k+1)} = x^{(k)} - \left(\frac{df^{(k)}}{dx}\right)^{-1} f^{(k)} \text{ , k=0, 1, 2, …} \qquad \text{(iteraciones)}$$

Si **F** contiene **n** ecuaciones, la fórmula se puede extender, siempre que las derivadas existan:

$$X^{(k+1)} = X^{(k)} - \left(\frac{\partial F^{(k)}}{\partial X}\right)^{-1} F^{(k)} = X^{(k)} - (J^{(k)})^{-1} F^{(k)}$$

En donde:

$$X^{(k+1)} = \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \dots \\ x_n^{(k+1)} \end{bmatrix}, \quad X^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \dots \\ x_n^{(k)} \end{bmatrix}, \quad F^{(k)} = \begin{bmatrix} f_1^{(k)} \\ f_2^{(k)} \\ \dots \\ f_n^{(k)} \end{bmatrix}, \quad J^{(k)} = \begin{bmatrix} \frac{\partial f_1^{(k)}}{\partial x_1} & \frac{\partial f_1^{(k)}}{\partial x_2} & \dots & \frac{\partial f_1^{(k)}}{\partial x_n} \\ \frac{\partial f_2^{(k)}}{\partial x_1} & \frac{\partial f_2^{(k)}}{\partial x_2} & \dots & \frac{\partial f_2^{(k)}}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n^{(k)}}{\partial x_1} & \frac{\partial f_n^{(k)}}{\partial x_2} & \dots & \frac{\partial f_n^{(k)}}{\partial x_n} \end{bmatrix}$$

J es la matriz jacobiana. Esta ecuación de recurrencia se puede usar iterativamente con **k = 0, 1, 2, …** partiendo de un vector inicial $X^{(0)}$ generando vectores de aproximación: $X^{(1)}$, $X^{(2)}$, $X^{(3)}$, **…**

3.5.2 Convergencia del método de Newton para sistemas de ecuaciones no lineales

En forma general la convergencia de este método para sistemas no lineales requiere que:

a) $\mathbf{f_1, f_2, \ldots f_n}$ así como sus derivadas sean continuas en la región de aplicación.
b) El determinante del Jacobiano no se anule en esta región
c) El valor inicial y los valores calculados pertenezcan a esta región, la cual incluye a la raíz que se intenta calcular

3.5.3 Algoritmo del método de Newton para sistemas de ecuaciones no lineales

Dado un sistema de ecuaciones **F = 0**, sea **J** su matriz Jacobiana. El siguiente algoritmo genera una sucesión de vectores que se espera tienda al vector solución.

```
Algoritmo: Newton para sistemas no lineales
Restricción:      No detecta divergencia
Entra:      F     (Vector con las ecuaciones)
            X     (Vector inicial)
            E     (Error o tolerancia)
Sale:       U     (Aproximación al vector solución, con error E)
J ← ∂F/∂X                       (Construir la matriz Jacobiana)
U ← X – (J)^-1 F                (Evaluar con los valores iniciales de X)
Mientras ||U-X||>E
     X ← U
     U ← X – (J)^-1 F           (Evaluar con los valores actuales de X)
Fin
```

Ejemplo. Encuentre las raíces reales del sistema:

$$f_1(x,y) = (x-2)^2 + (y-1)^2 + xy - 3 = 0$$
$$f_2(x,y) = xe^{x+y} + y - 3 = 0$$

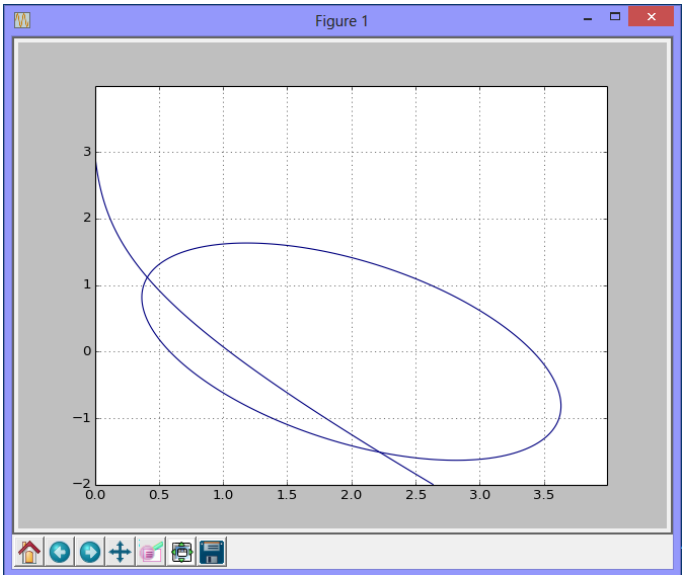
f(x,y), g(x,y) son funciones de dos variables y sus gráficos son superficies en el espacio. Es de interés encontrar sus intersecciones en el plano, es decir cuando f(x,y)=0, g(x,y)=0. Son ecuaciones con las mismas dos variables y sus gráficos pueden visualizarse en el plano. Las raíces reales son las intersecciones de sus gráficos..

Para graficar las ecuaciones, **abra una ventana** de Python y cargue la librería **Pylab**. Con las instrucciones indicadas a continuación se obtiene la figura de las dos ecuaciones en el plano X-Y Se puede observar que existen dos soluciones para el sistema de ecuaciones:

```
>>> import pylab as pl
>>> xr = pl.arange(0,4,0.01)
>>> yr = pl.arange(-2,4,0.01)
>>> [x, y] = pl.meshgrid(xr,yr)
>>> f=x*pl.exp(x+y)+y-3
>>> g=(x-2)**2+(y-1)**2+x*y-3
>>> pl.contour(x, y, f,[0])
>>> pl.contour(x, y, g,[0])
>>> pl.grid(True)
>>> pl.show()
```

Note el uso calificado de la función

El parámetro [0] indica que se grafican las superficies f, g en el plano X-Y (nivel 0)



No es posible reducir el sistema a una ecuación, por lo que se debe utilizar un método numérico para resolverlo como un sistema simultáneo con la fórmula propuesta:

Cálculo manual con el método de Newton

Ecuaciones:

$$f_1(x,y) = (x-2)^2 + (y-1)^2 + xy - 3 = 0$$
$$f_2(x,y) = xe^{x+y} + y - 3 = 0$$

Vector inicial $X^{(0)} = \begin{bmatrix} x^{(0)} \\ y^{(0)} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.0 \end{bmatrix}$ (tomado del gráfico)

Matriz jacobiana y vectores con las variables y las ecuaciones

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y - 4 & x + 2y - 2 \\ e^{x+y}(1+x) & xe^{x+y} + 1 \end{bmatrix}$$

$$X = \begin{bmatrix} x \\ y \end{bmatrix}, \quad F = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} (x-2)^2 + (y-1)^2 + xy - 3 \\ xe^{x+y} + y - 3 \end{bmatrix}$$

Ecuación de recurrencia

$$X^{(k+1)} = X^{(k)} - (J^{(k)})^{-1} F^{(k)}$$

Primera iteración: **k=0**

$$X^{(1)} = X^{(0)} - (J^{(0)})^{-1} F^{(0)}$$

$$\begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.0 \end{bmatrix} - \begin{bmatrix} 2(0.5) + 1 - 4 & 0.5 + 2(1) - 2 \\ e^{0.5+1}(1+0.5) & 0.5e^{0.5+1} + 1 \end{bmatrix}^{-1} \begin{bmatrix} (0.5-2)^2 + (1-1)^2 + 0.5(1) - 3 \\ 0.5e^{0.5+1} + 1 - 3 \end{bmatrix}$$

$$\begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.0 \end{bmatrix} - \begin{bmatrix} -2 & 0.5 \\ 6.7225 & 3.2408 \end{bmatrix}^{-1} \begin{bmatrix} -0.25 \\ 0.2408 \end{bmatrix}$$

$$\begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.0 \end{bmatrix} - \begin{bmatrix} -0.3293 & 0.0508 \\ 0.6830 & 0.2032 \end{bmatrix} \begin{bmatrix} -0.25 \\ 0.2408 \end{bmatrix} = \begin{bmatrix} 0.4055 \\ 1.1218 \end{bmatrix}$$

3.5.4 Instrumentación computacional del método de Newton para un sistema de n ecuaciones no-lineales

Sea **F:** $\mathbf{f_1, f_2, \ldots, f_n}$ ecuaciones con variables independientes **X:** $\mathbf{x_1, x_2, \ldots, x_n}$.

Ecuación de recurrencia:

$$\mathbf{X^{(k+1)} = X^{(k)} - (J^{(k)})^{-1} F^{(k)}}, \quad k=0, 1, 2, \ldots$$

En donde **J** es la matriz jacobiana del sistema

Entrada

- **F**: Vector con las ecuaciones
- **V**: Vector con las variables independientes
- **U**: Vector con valores iniciales para las variables

Salida

- **U**: Vector con los nuevos valores calculados para las variables

Nota: La convergencia será controlada desde la ventana interactiva llamando iterativamente a la función. Por las propiedades de este método, la convergencia o divergencia será muy rápida.

Alternativamente, se puede mejorar la instrumentación incorporando un ciclo con un máximo de iteraciones para que las iteraciones se realicen dentro de la función.

Las derivadas parciales se obtienen con la función **diff** y la sustitución de los valores de **U** en las variables se realiza con la función **subs**. La solución se la obtiene con la inversa de la matriz de las derivadas parciales **J**.

Estas funciones están en librerías de Python que deben cargarse. Es conveniente usar las funciones de las librerías mediante un nombre o rótulo para distinguirlas de las funciones con los mismos nombres que pudieran tener otras librerías.

```
import numpy as np
import sympy as sp
#Resolución de Sistemas no lineales
def snewton(F, V, U):
    n=len(F)
    J=np.zeros([n,n],dtype=sp.Symbol)
    T=list(np.copy(F))

    for i in range(n):                          #Construir J
        for j in range(n):
            J[i][j]=sp.diff(F[i],V[j])
    for i in range(n):                          #Evaluar J
        for j in range(n):
            for k in range(n):
                J[i][j]=J[i][j].subs(V[k],float(U[k]))
    for i in range(n):                          #Evaluar F
        for j in range(n):
            T[i]=T[i].subs(V[j],float(U[j]))
    J=np.array(J,float)
    T=np.array(T,float)
    U=U-np.dot(np.linalg.inv(J),T)              #Nuevo vector U
    return U
```

Ejemplo. Use la función **snewton** para encontrar una raíz real del sistema

$$f(x,y) = xe^{x+y} + y - 3 = 0$$
$$g(x,y) = (x-2)^2 + (y-1)^2 + xy - 3 = 0$$

El gráfico de estas ecuaciones se lo construyó con la librería Pylab de Python en una sección anterior. Para no cerrar el gráfico y poder observar los valores iniciales se sugiere abrir **otra ventana** interactiva de Python para realizar los cálculos:

```
>>> from snewton import *
>>> import sympy as sp
>>> [x,y]=sp.symbols('x,y')
>>> f=x*sp.exp(x+y)+y-3              Note el uso calificado de la función
>>> g=(x-2)**2+(y-1)**2+x*y-3
>>> F=[f,g]
>>> V=[x,y]
>>> U=[0.5,1]
>>> U=snewton(F,V,U);print(U)
 [0.405451836483295 1.12180734593318]
>>> U=snewton(F,V,U);print(U)
[0.409618877363502 1.11619120947847]
```

```
>>> U=snewton(F,V,U);print(U)
 [0.409627787030011 1.11618013799184]
>>> U=snewton(F,V,U);print(U)
 [0.409627787064807 1.11618013794281]
>>> U=snewton(F,V,U);print(U)
 [0.409627787064807 1.11618013794281]
```

Se observa la rápida convergencia.

Para verificar que son raíces reales de las ecuaciones deben evaluarse **f, g**

```
>>> f.subs(x,U[0]).subs(y,U[1])
0
>>> g.subs(x,U[0]).subs(y,U[1])
3.62557206479153e-16
```

Los valores obtenidos son muy pequeños, por lo cual se aceptan las raíces calculadas

Para calcular la otra raíz, tomamos del gráfico los valores iniciales cercanos a esta raíz.

```
>>> U=[2.4,-1.5]
>>> U=snewton(F,V,U);print(U)
 [2.26184271829705 -1.53588073184920]
>>> U=snewton(F,V,U);print(U)
 [2.22142100136910 -1.51230470581913]
>>> U=snewton(F,V,U);print(U)
 [2.22041081429453 -1.51147810488742]
>>> U=snewton(F,V,U);print(U)
 [2.22041032725647 -1.51147760884696]
>>> U=snewton(F,V,U);print(U)
 [2.22041032725637 -1.51147760884683]
>>> U=snewton(F,V,U);print(U)
 [2.22041032725637 -1.51147760884683]
```

Comprobar si es una solución del sistema

```
>>> f.subs(x,U[0]).subs(y,U[1])
1.77635683940025e-15
>>> g.subs(x,U[0]).subs(y,U[1])
-4.44089209850063e-16
```

Aparentemente, en su estado actual, la librería **SymPy** de Python no tiene instrumentado algún método para resolver sistemas no lineales como se puede encontrar en otros lenguajes como MATLAB, sin embargo el software de MATLAB solamente pudo calcular una de las dos raíces del ejemplo anterior. Con esto concluimos que el conocimiento de los métodos numéricos permite resolver problemas para los que el cálculo manual o los

programas computacionales disponibles no son suficientes para calcular todas las respuestas esperadas.

3.5.5 Obtención de la fórmula iterativa de segundo orden para calcular raíces reales de sistemas de ecuaciones no lineales

Se considera el caso de dos ecuaciones y luego se generaliza a más ecuaciones

Sean $\mathbf{f_1(x_1, x_2) = 0}$, $\mathbf{f_2(x_1, x_2) = 0}$ dos ecuaciones no-lineales con variables $\mathbf{x_1, x_2}$.

Sean $\mathbf{r_1, r_2}$ valores reales tales que $\mathbf{f_1(r_1, r_2) = 0}$, $\mathbf{f_2(r_1, r_2) = 0}$, entonces $\mathbf{(r_1, r_2)}$ constituye una raíz real del sistema y es de interés calcularla.

Suponer que $\mathbf{f_1, f_2}$ son funciones diferenciables en alguna región cercana al punto $\mathbf{(r_1, r_2)}$

Con el desarrollo de la serie de Taylor expandimos $\mathbf{f_1, f_2}$ desde el punto $\mathbf{(x_1^{(k)}, x_2^{(k)})}$ al punto $\mathbf{(x_1^{(k+1)}, x_2^{(k+1)})}$

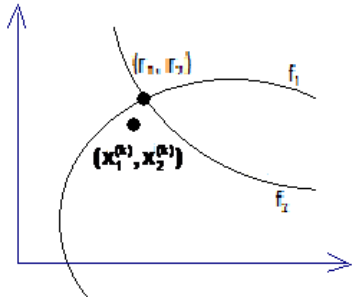
$$\mathbf{f_1^{(k+1)} = f_1^{(k)} + (x_1^{(k+1)} - x_1^{(k)})\frac{\partial f_1^{(k)}}{\partial x_1} + (x_2^{(k+1)} - x_2^{(k)})\frac{\partial f_1^{(k)}}{\partial x_2} + O(x_1^{(k+1)} - x_1^{(k)})^2 + O(x_2^{(k+1)} - x_2^{(k)})^2}$$

$$\mathbf{f_2^{(k+1)} = f_2^{(k)} + (x_1^{(k+1)} - x_1^{(k)})\frac{\partial f_2^{(k)}}{\partial x_1} + (x_2^{(k+1)} - x_2^{(k)})\frac{\partial f_2^{(k)}}{\partial x_2} + O(x_1^{(k+1)} - x_1^{(k)})^2 + O(x_2^{(k+1)} - x_2^{(k)})^2}$$

Por simplicidad se ha usado la notación: $\mathbf{f_1^{(k)} = f_1(x_1^{(k)}, x_2^{(k)})}$, $\mathbf{f_1^{(k+1)} = f_1(x_1^{(k+1)}, x_2^{(k+1)})}$, etc.

En los últimos términos de ambos desarrollos se han escrito únicamente los componentes de interés, usando la notación **O( )**.
Las siguientes suposiciones, son aceptables en una región muy cercana a $\mathbf{(r_1, r_2)}$:

$\mathbf{(x_1^{(k)}, x_2^{(k)})}$ **cercano a la raíz** (r_1, r_2)



Si el método converge cuadráticamente entonces $\mathbf{(x_1^{(k+1)}, x_2^{(k+1)})}$ estará muy cercano a $\mathbf{(r_1, r_2)}$
Por lo tanto se puede aproximar:

$$\mathbf{f_1(x_1^{(k+1)}, x_2^{(k+1)}) \approx 0}$$
$$\mathbf{f_2(x_1^{(k+1)}, x_2^{(k+1)}) \approx 0}$$

Por otra parte, si $(x_1^{(k)}, x_2^{(k)})$ es cercano a $(x_1^{(k+1)}, x_2^{(k+1)})$, las diferencias serán pequeñas y al elevarse al cuadrado se obtendrán valores más pequeños y se los omite.

Sustituyendo en el desarrollo propuesto se obtiene como aproximación el sistema lineal:

$$0 = f_1^{(k)} + (x_1^{(k+1)} - x_1^{(k)})\frac{\partial f_1^{(k)}}{\partial x_1} + (x_2^{(k+1)} - x_2^{(k)})\frac{\partial f_1^{(k)}}{\partial x_2}$$

$$0 = f_2^{(k)} + (x_1^{(k+1)} - x_1^{(k)})\frac{\partial f_2^{(k)}}{\partial x_1} + (x_2^{(k+1)} - x_2^{(k)})\frac{\partial f_2^{(k)}}{\partial x_2}$$

En notación matricial:

$$-F^{(k)} = J^{(k)}(X^{(k+1)} - X^{(k)})$$

Siendo

$$F^{(k)} = \begin{bmatrix} f_1^{(k)} \\ f_2^{(k)} \end{bmatrix}, \quad X^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix}, \quad X^{(k+1)} = \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{bmatrix}, \quad J^{(k)} = \begin{bmatrix} \frac{\partial f_1^{(k)}}{\partial x_1} & \frac{\partial f_1^{(k)}}{\partial x_2} \\ \frac{\partial f_2^{(k)}}{\partial x_1} & \frac{\partial f_2^{(k)}}{\partial x_2} \end{bmatrix}$$

$$J^{(k)}X^{(k+1)} = J^{(k)}X^{(k)} - F^{(k)}$$

$$X^{(k+1)} = X^{(k)} - (J^{(k)})^{-1}F^{(k)}, \quad |J^{(k)}| \neq 0$$

Es la ecuación de recurrencia que se puede usar iterativamente con **k=0, 1, 2, …** partiendo de un vector inicial $X^{(0)}$ generando vectores de aproximación: $X^{(1)}$, $X^{(2)}$, $X^{(3)}$, …

La notación matricial y la ecuación de recurrencia se extienden directamente a sistemas de **n** ecuaciones no lineales $f_1, f_2, \ldots, f_n$ con variables $x_1, x_2, \ldots, x_n$.

La matriz de las derivadas parciales J se denomina **Jacobiano**. La ecuación de recurrencia se reduce a la fórmula de Newton si se tiene una sola ecuación.

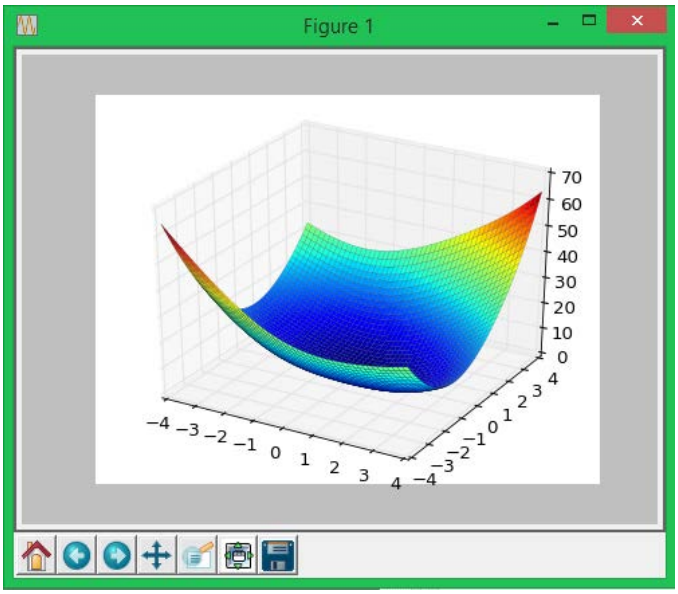
3.5.6 Aplicación del método de Newton para obtener máximos y mínimos locales de funciones no lineales multivariadas

Dada una función no lineal multivariada, el cálculo de máximos o mínimos se puede convertir en el problema equivalente de encontrar la solución común del sistema de dos ecuaciones que se obtienen igualando sus derivadas a cero. Para encontrar la solución se debe resolver el sistema de ecuaciones resultante. Siendo estas ecuaciones no lineales, se puede usar el método de Newton para sistemas de la sección anterior.

Ejemplo. Calcular el mínimo de la función **$f(x,y)=x^2+2y^2+\cos(x+y+1)+xy$**

Primer paso. Abra una **primera ventana** de Python y cargue las librerías de SymPy. Grafique y localice el mínimo o máximo. Obtenga el sistema de ecuaciones derivando la función respecto a cada variable independiente. Si la función tiene más de dos variables, solamente se podrán hacer suposiciones y pruebas para localizar el máximo.

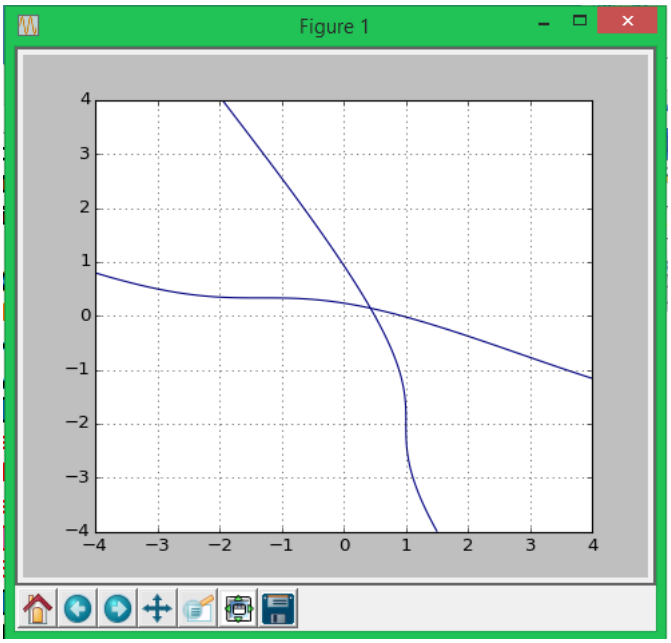
```
>>> from sympy import*
>>> from sympy.plotting import*
>>> x,y=symbols('x,y')
>>> f=x**2+2*y**2+cos(x+y+1)+x*y
>>> f1=diff(f,x)
>>> print(f1)
2*x + y - sin(x + y + 1)
>>> f2=diff(f,y)
>>> print(f2)
x + 4*y - sin(x + y + 1)
>>> plot3d(f,(x,-4,4),(y,-4,4))
```



En el gráfico se visualiza un mínimo en la vecindad del punto (1, 1)

Segundo paso. Abra una **segunda ventana** de Python y grafique las derivadas en el plano **X-Y** con la librería Pylab. Localice algun punto que satisface a las ecuaciones obtenidas con las derivadas igualadas a cero. Si hay más de dos ecuaciones, solamente se podrán hacer suposiciones y pruebas para localizar la solución..

```
>>> from pylab import*
>>> xr = arange(-4,4,0.01)
>>> yr = arange(-4,4,0.01)
>>> [x, y] = meshgrid(xr,yr)
>>> f1=2*x + y - sin(x + y + 1)
>>> f2=x + 4*y - sin(x + y + 1)
>>> contour(x, y, f1,[0])
>>> contour(x, y, f2,[0])
>>> grid(True)
>>> show()
```



Tercer paso. Abra una **tercera ventana** de Python y cargue la función **snewton.** Cargue también la libraría **SymPy**. Encuentre iterativamente la solución con la precisión requerida. Del gráfico elija un punto inicial.

```
>>> from snewton import*
>>> from sympy import*
>>> [x,y]=symbols('x,y')
>>> f=x**2+2*y**2+cos(x+y+1)+x*y
>>> f1=2*x + y - sin(x + y + 1)
>>> f2=x + 4*y - sin(x + y + 1)
>>> F=[f1,f2]
>>> V=[x,y]
```

```
>>> U=[1,1]
>>> U=snewton(F,V,U);print(U)
[ 0.58059602  0.19353201]
>>> U=snewton(F,V,U);print(U)
[ 0.43638332  0.14546111]
>>> U=snewton(F,V,U);print(U)
[ 0.42859444  0.14286481]
>>> U=snewton(F,V,U);print(U)
[ 0.42857134  0.14285711]
>>> U=snewton(F,V,U);print(U)
[ 0.42857134  0.14285711]                    Coordenadas del punto mínimo
>>> f.subs(x,U[0]).subs(y,U[1])
0.285082041122721                            Valor mínimo de la función
```

Observaciones

Se sugiere no cerrar las ventanas con los gráficos para visualizar la solución

Para este ejemplo, se define un sistema de dos ecuaciones con las dos derivadas de la función. La solución de este sistema con el método de Newton requiere construir la matriz Jacobiana que contendrá **cuatro** derivadas. Si la función original tuviese **n** variables, la matriz Jacobiana tendría **n**x**n** componentes

En la siguiente sección se revisará el método del **gradiente del máximo descenso** para obtener mínimos o máximos locales de funciones no lineales multivariadas. Este método es más eficiente que el método de Newton. Para el ejemplo, solamente requiere un vector con las **dos** derivadas de la función para calcular el mínimo. Si la función original tuviese **n** variables, el vector de las derivadas tendría **n** componentes.

La ventaja del método de Newton es su convergencia rápida. Además el algoritmo no requiere ajustar el paso de avance en la búsqueda de la solución durante el proceso iterativo.

3.5.7 Ejercicios y problemas de sistemas de ecuaciones no-lineales

1. Encuentre las soluciones del sistema de ecuaciones en la región: **-4≤x≤4, -4≤y≤4**

$$\sin(x) + e^y - xy = 5$$
$$x^2 + y^3 - 3xy = 7$$

a) Grafique las ecuaciones en el intervalo dado y observe las raíces reales.

b) Del gráfico elija valores aproximados para cada raíz y use iterativamente la función **snewton** para calcular la solución con error absoluto menor que 0.000001

c) Verifique que las soluciones calculadas satisfacen aproximadamente a las ecuaciones

2. Encuentre las soluciones del sistema de ecuaciones en la región: **-6≤x≤6, -6≤y≤6**
:

$$\cos(x+y) + xy=3$$
$$3(x - 2)^2 - 2(y - 3)^2 = 5xy$$

a) Grafique las ecuaciones en el intervalo dado y observe las raíces reales.

b) Del gráfico elija valores aproximados para cada raíz y use iterativamente la función **snewton** para calcular la solución con error absoluto menor que 0.000001

c) Verifique que las soluciones calculadas satisfacen aproximadamente a las ecuaciones

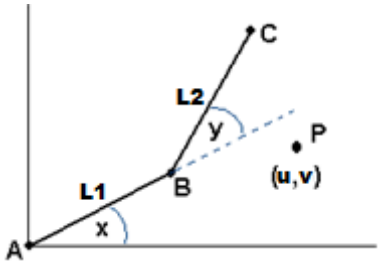
3. Encuentre las soluciones del sistema de ecuaciones en la región: **0≤x≤2, 0≤y≤2**

```
f=2*x*sin(2*x + y) + 2*(x**2 - y)*cos(2*x + y)
g=(x**2 - y)*cos(2*x + y) - sin(2*x + y)
```

Grafique las ecuaciones en el plano X,Y. Elija un punto inicial y use iterativamente la función **snewton** para calcular una solución.

4. Determine el valor de los coeficientes **a, b** para que la función $\mathbf{f(x) = (ax+b)e^{ax+b} + a}$ incluya a los puntos **(1,3), (2,4)**
a) Plantee un sistema de dos ecuaciones no lineales para obtener la respuesta
b) Elija **(0, 1)** como valores iniciales para **(a, b**). Obtenga la respuesta con E = $\mathbf{10^{-6}}$

5. El siguiente gráfico es el brazo de un robot compuesto de dos segmentos articulados en los puntos **A** y **B**. Las longitudes de los brazos son **L1** y **L2**. Determine los ángulos **X** y **Y** para que el extremo **C** coincida con el punto **P** de coordenadas **(u, v).**



a) Construya el modelo matemático mediante un sistema de dos ecuaciones no lineales

b) Plantee la formulación $\mathbf{X^{(k+1)} = X^{(k)} - (J^{(k)})^{-1}F^{(k)}}$**, k=0, 1, 2, …** para resolver el sistema

c) Elija valores para **L1, L2, u, v**. Elija valores iniciales y use la función **snewton** para obtener la solución. **E=0.0001.**

d) Analice las implicaciones de esta solución en el movimiento del brazo.

3.6 Método del gradiente de máximo descenso

El método del gradiente de máximo descenso es un algoritmo iterativo para encontrar un mínimo local de una función multivariada no lineal mediante aproximaciones sucesivas. La búsqueda de la solución sigue la dirección del gradiente descendente más pronunciado, hasta llegar a su menor valor. El proceso iterativo continua hasta que la máxima tasa de cambio del gradiente sea muy pequeña.

El método funciona con la suposición que la función tiene forma convexa alrededor del mínimo y que la distancia de avance en cada paso es elegida apropiadamente.

El método también se puede usar para encontrar máximos locales de una función multivariada f aplicándolo a la función -f. Igualmente puede usarse para el caso básico del cálculo de mínimos o máximos locales de funciones univariadas.

3.6.1 Definiciones

Vector de n variables reales

$$\mathbf{v} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

Función de n variables reales

$$f: \mathbf{R}^n \rightarrow \mathbf{R}$$

Gradiente de f. Es el vector de las derivadas parciales de la función que se desea minimizar

$$\nabla \mathbf{f} = \left[\frac{\partial \mathbf{f}}{\partial x_i} \right], \; i=1,2,\dots, n$$

Gradiente de f en un punto **v**. Es la dirección de mayor incremento de f en el punto **v**

$$\mathbf{c} = \nabla \mathbf{f}(\mathbf{v})$$

Magnitud del vector gradiente de f en un punto **v**. Es la mayor tasa de cambio del gradiente en el punto **v**

$$\| \mathbf{c} \| = \sqrt{\mathbf{c}^T \mathbf{c}}$$

Vector gradiente de f normalizado, en un punto **v**

$$\bar{\mathbf{c}} = \frac{\mathbf{c}}{\|\mathbf{c}\|}$$

Vector de la dirección de búsqueda descendente para obtener el mínimo de f en el punto **v**. Esta dirección debe ser la opuesta al gradiente de f en el punto **v**:

$$\mathbf{d} = -\bar{\mathbf{c}}$$

Longitud del paso de avance s para modificar el vector **v** en la dirección **d**:

Debe ser un valor escalar que garantice que f(**v** + s**d**) < f(**v**)

La búsqueda del mínimo de la función f consiste en modificar el vector **v** agregando el vector s**d** repetidamente. En cada iteración se debe elegir el valor de s que optimice la búsqueda.

El método converge si se llega a un punto en el cual el gradiente ya no cambia significativamente. Este punto corresponde aproximadamente al mínimo de la función f.

3.6.2 Algoritmo del gradiente de máximo descenso

Objetivo: Encontrar un mínimo local de una función multivariada f

1) Iniciar un conteo de iteraciones: k=0
   Estimar un vector inicial para la solución: $\mathbf{v}^{(k)}$
   Estimar un máximo de iteraciones m y un criterio de precisión ε

2) Calcular el vector gradiente de f evaluado en el punto $\mathbf{v}^{(k)}$

$$\mathbf{c}^{(k)} = \nabla \mathbf{f}(\mathbf{v}^{(k)})$$

3) Calcular la máxima tasa de cambio del gradiente de f en el punto $\mathbf{v}^{(k)}$:

$$\|\mathbf{c}^{(k)}\| = \sqrt{\mathbf{c}^{(k)^T}\mathbf{c}^{(k)}}$$

Si $\|\mathbf{c}^{(k)}\| < \varepsilon$ entonces
   $\mathbf{v}^{(k)}$ es un punto mínimo de **f** con precisión ε
   Finalizar (el método converge)

4) Calcular el vector normalizado de la dirección de búsqueda en el punto $\mathbf{v}^{(k)}$

$$\mathbf{d}^{(k)} = -\frac{\mathbf{c}^{(k)}}{\|\mathbf{c}^{(k)}\|}$$

5) Determinar el tamaño del paso de avance $s^{(k)}$ en la iteración k

6) Actualizar el vector de búsqueda

$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \mathbf{s}^{(k)}\mathbf{d}^{(k)}$$

7) Actualizar el conteo de iteraciones

k = k + 1

Si k < m entonces

Regresar al paso 2)

Sinó

Finalizar (el método no converge)

3.6.3 Procedimiento para determinar el tamaño óptimo del paso de avance

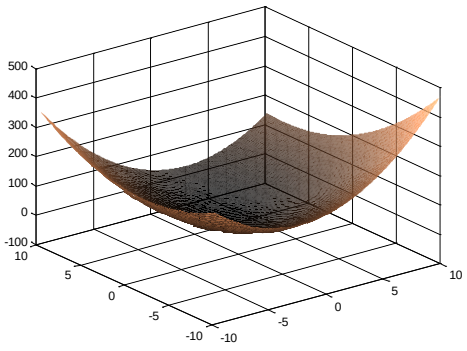
Es necesario elegir el tamaño del paso para avanzar en la búsqueda de la solución. Un tamaño muy grande puede hacer que no se pueda localizar la solución. Un tamaño muy pequeño puede hacer que la búsqueda sea ineficiente. Es preferible que el tamaño del paso se pueda modificar en cada iteración.

Un método para optimizar el tamaño del paso de búsqueda $s^{(k)}$ consiste en elegir el valor de s tal que $\mathbf{f(v^{(k+1)})}$ sea mínimo. Este valor de s puede encontrarse resolviendo en cada iteración k la ecuación:

$$\frac{\mathbf{df(v^{(k+1)})}}{\mathbf{ds}} = \frac{\mathbf{df(v^{(k)} + sd^{(k)})}}{\mathbf{ds}} = \mathbf{0}$$

Ejemplo. Encontrar un mínimo local de la función real $\mathbf{f(x,y)=2x^2-xy+y^2-7y}$ siguiendo el algoritmo de máximo descenso

Gráfico de f



Función:

$$\mathbf{f(x,y)=2x^2-xy+y^2-7y}$$

Vector de variables

$$\mathbf{v} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

Gradiente de f

$$\nabla \mathbf{f} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \end{bmatrix} = \begin{bmatrix} 4\mathbf{x} - \mathbf{y} \\ -\mathbf{x} + 2\mathbf{y} - 7 \end{bmatrix}$$

Vector inicial

$$\mathbf{v}^{(0)} = \begin{bmatrix} \mathbf{x}^{(0)} \\ \mathbf{y}^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Primera iteración: (k=0)

Vector gradiente evaluado en el punto $\mathbf{v}^{(0)}$

$$\mathbf{c}^{(0)} = \nabla \mathbf{f}(\mathbf{v}^{(0)}) = \begin{bmatrix} 4(0) - 0 \\ -0 + 2(0) - 7 \end{bmatrix} = \begin{bmatrix} 0 \\ -7 \end{bmatrix}$$

Máxima tasa de cambio del gradiente en el punto $\mathbf{v}^{(0)}$

$$\| \mathbf{c}^{(0)} \| = \sqrt{\mathbf{c}^{(0)^{\mathrm{T}}} \mathbf{c}^{(0)}} = \sqrt{\begin{bmatrix} 0 & -7 \end{bmatrix} \begin{bmatrix} 0 \\ -7 \end{bmatrix}} = 7$$

Vector normalizado de la dirección de avance

$$\mathbf{d}^{(0)} = -\frac{\mathbf{c}^{(0)}}{\| \mathbf{c}^{(0)} \|} = -\frac{1}{7} \begin{bmatrix} 0 \\ -7 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Determinar el tamaño óptimo del paso de avance

$$\mathbf{v}^{(0)} + \mathbf{s}\mathbf{d}^{(0)} = \begin{bmatrix} \mathbf{x}^{(0)} + \mathbf{s}\mathbf{d}_{\mathbf{x}}^{(0)} \\ \mathbf{y}^{(0)} + \mathbf{s}\mathbf{d}_{\mathbf{y}}^{(0)} \end{bmatrix} = \begin{bmatrix} 0 + \mathbf{s}(0) \\ 0 + \mathbf{s}(1) \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{s} \end{bmatrix}$$

$$\mathbf{f}(\mathbf{v}^{(0)} + \mathbf{s}\mathbf{d}^{(0)}) = \mathbf{f}(0, \mathbf{s}) = \mathbf{s}^2 - 7\mathbf{s}$$

$$\frac{\mathbf{df}(0, \mathbf{s})}{\mathbf{ds}} = 2\mathbf{s} - 7 = 0 \quad \Rightarrow \quad \mathbf{s}^{(0)} = \mathbf{s} = 3.5$$

Actualizar el vector solución

$$\mathbf{v}^{(1)} = \mathbf{v}^{(0)} + \mathbf{s}^{(0)} \mathbf{d}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 3.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 3.5 \end{bmatrix}$$

Evaluar la solución

$$\mathbf{f(v^{(1)})=f(0,\ 3.5)=-12.25}$$

Segunda iteración: (k=1)

Vector gradiente en el punto $\mathbf{v^{(1)}}$

$$\mathbf{c^{(1)} = \nabla f(v^{(1)}) = \begin{bmatrix} 4(0) - 3.5 \\ -0 + 2(3.5) - 7 \end{bmatrix} = \begin{bmatrix} -3.5 \\ 0 \end{bmatrix}}$$

Máxima tasa de cambio del gradiente en el punto $\mathbf{v^{(1)}}$

$$\mathbf{\| c^{(k)} \| = \sqrt{c^{(k)^T} c^{(k)}} = \sqrt{\begin{bmatrix} -3.5 & 0 \end{bmatrix} \begin{bmatrix} -3.5 \\ 0 \end{bmatrix}} = 3.5}$$

Vector normalizado de la dirección de avance

$$\mathbf{d^{(1)} = -\frac{c^{(1)}}{\| c^{(1)} \|} = -\frac{1}{3.5} \begin{bmatrix} -3.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}}$$

Determinar el tamaño óptimo del paso de avance

$$\mathbf{v^{(1)} + sd^{(1)} = \begin{bmatrix} x^{(1)} + sd_x^{(1)} \\ y^{(1)} + sd_y^{(1)} \end{bmatrix} = \begin{bmatrix} 0 + s(1) \\ 3.5 + s(0) \end{bmatrix} = \begin{bmatrix} s \\ 3.5 \end{bmatrix}}$$

$$\mathbf{f(v^{(1)} + sd^{(1)}) = f(s, 3.5) = 2s^2 - 3.5s - 12.25}$$

$$\mathbf{\frac{df(s,3.5)}{ds} = 4s - 3.5 = 0 \quad \Rightarrow \quad s^{(1)} = s = 0.875}$$

Actualizar el vector solución

$$\mathbf{v^{(2)} = v^{(1)} + s^{(1)} d^{(1)} = \begin{bmatrix} 0 \\ 3.5 \end{bmatrix} + 0.875 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.875 \\ 3.5 \end{bmatrix}}$$

Evaluar la solución

$$\mathbf{f(v^{(2)})=f(0.875,\ 3.5)=-13.78125}$$

Las iteraciones deben continuar hasta que la máxima tasa de cambio del gradiente sea muy pequeña

3.6.4 Instrumentación computacional del algoritmo del gradiente de máximo descenso

Es conveniente instrumentar computacionalmente la formulación del algoritmo de máximo descenso para experimentar y faciltar su uso práctico en la resolución de problemas.

3.6.5 Descripción y uso de las funciones de la instrumentación computacional

La instrumentación está encapsulada en un módulo denominado **gradiente** el cual contiene funciones de utilidad que se pueden llamar separadamente y también son los componentes de soporte para la función principal con el nombre **metodo_gradiente** la cual corresponde al lineamiento del algoritmo propuesto.

obtener_gradiente(f,v)
Entra
f: Función multivariada
v: Vector de variables definidas en forma simbólica
Sale
g: Vector gradiente con las derivadas parciales de f

evaluar_gradiente(g,v,u)
Entra **g:** Vector gradiente con las derivadas parciales de f
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares para las variables en **v**
Sale
c: Vector gradiente con los componentes evaluados en el punto **u**

magnitud_del_gradiente(c)
Entra
c: Vector gradiente evaluado en el punto **u**
Sale
norma: Máxima tasa de cambio del gradiente en el punto **u**

gradiente_normalizado(c)
Entra
c: Vector gradiente evaluado en el punto **u**
Sale
cn: Vector gradiente normalizado evaluado en el punto **u**

evaluar_solucion(f,v,u)
Entra
f: Función multivariada
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares para las variables en **v**
Sale
fm: Valor actual de la solución en el punto **u**

calcular_paso(f,g,v,u)

Entra

f: Función multivariada
g: Vector gradiente con las derivadas parciales de f
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares para las variables en **v**

Sale

s: Valor estimado para el tamaño del paso de avance

metodo_gradiente(f,v,u,e,m,imp=0)

Entra

f: Función multivariada
v: Vector de variables definidas en forma simbólica
u: Vector con valores escalares iniciales para las variables en **v**
e: Criterio de convergencia y precisión para el gradiente
(Es el error de truncamiento E del método iterativo)
m: Cantidad máxima permitida para las iteraciones
imp: Parámetro opcional. Si se lo omite no se mostrarán los resultados intermedios pero si se le asigna algún valor mayor a cero, se mostrarán los valores calculados en cada iteración.

Sale

Si el método converge

uk: Vector solución calculado en la última iteración
fm: Valor de la función evaluada en la solución

Si el método no converge entrega un vector nulo

Nota. Al cargar la función principal **metodo_gradiente** también se cargan las librerías que están en el encabezado de la lista de funciones.

3.6.6 Código del módulo gradiente

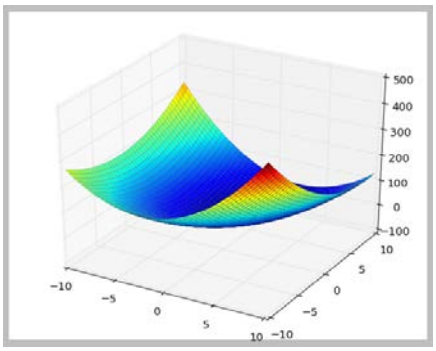
```
#Método del gradiente de máximo descenso
from sympy import*
from sympy.plotting import*
import numpy as np

def obtener_gradiente(f,v):
    n=len(v)
    g=[]
    for i in range(n):
        d=diff(f,v[i])
        g=g+[d]
    return g

def evaluar_gradiente(g,v,u):
    n=len(v)
    c=[]
    for i in range(n):
        t=g[i]
        for j in range(n):
            t=t.subs(v[j],u[j])
        c=c+[float(t)]
    return c

def magnitud_del_gradiente(c):
    norma=sqrt(np.dot(c,c))
    return norma

def gradiente_normalizado(c):
    norma=magnitud_del_gradiente(c)
    t=list(np.array(c)/norma)
    cn=[]
    for i in range(len(c)):
        cn=cn+[float(t[i])]
    return cn

def evaluar_solucion(f,v,u):
    fm=f.subs(v[0],u[0])
    for i in range(1,len(v)):
        fm=fm.subs(v[i],u[i])
    return fm
```

```
def calcular_paso(f,g,v,u):
    c=evaluar_gradiente(g,v,u)
    cn=gradiente_normalizado(c)
    t=Symbol('t')
    xt=[]
    for i in range(len(v)):
        xt=xt+[float(u[i])-t*float(cn[i])]
    fs=f.subs(v[0],xt[0])
    for i in range(1,len(v)):
        fs=fs.subs(v[i],xt[i])
    df=diff(fs,t)
    ddf=diff(df,t)
    s=1
    for i in range(5):
        s=s-float(df.subs(t,s))/float(ddf.subs(t,s))
    return s

def metodo_gradiente(f,v,u,e,m,imp=0):
    u0=u.copy()
    g=obtener_gradiente(f,v)
    for k in range(m):
        c=evaluar_gradiente(g,v,u0)
        norma=magnitud_del_gradiente(c)
        if norma<e:
            fm=evaluar_solucion(f,v,u0)
            return u0,fm
        s=calcular_paso(f,g,v,u0)
        cn=gradiente_normalizado(c)
        uk=[]
        for i in range(len(c)):
            uk=uk+[float(u0[i])-s*float(cn[i])]
        u0=uk.copy()
        if imp>0:
            print('k=',k+1,' s=',s,' vector=',u0)
    return [],None
```

3.6.7 Ejemplos de aplicación

Ejemplo 1. Calcular el mínimo de $f(x,y)=2x^2-xy+y^2-7y$. Graficar y mostrar las iteraciones y resultados. E=0.01

```
>>> from gradiente import*
>>> x,y=symbols('x,y')
>>> f=2*x**2-x*y+y**2-7*y
>>> plot3d(f,(x,-10,10),(y,-10,10))
```




```
>>> v=[x,y]
>>> u=[0,0]
>>> uk,fm=metodo_gradiente(f,v,u,0.01,20,1)
k= 1  s= 3.5  vector= [0.0, 3.5]
k= 2  s= 0.875  vector= [0.875, 3.5]
k= 3  s= 0.4375  vector= [0.875, 3.9375]
k= 4  s= 0.109375  vector= [0.984375, 3.9375]
k= 5  s= 0.0546875  vector= [0.984375, 3.9921875]
k= 6  s= 0.013671875  vector= [0.998046875, 3.9921875]
k= 7  s= 0.0068359375  vector= [0.998046875, 3.9990234375
>>> uk
[0.998046875, 3.9990234375]
>>> fm
-13.9999933242798
```

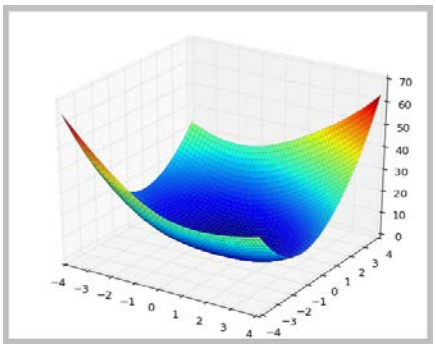
Ejemplo 2. Calcular el mínimo de $f(x,y)=2x^2-xy+y^2-7y$. Elegir otro vector inicial. Mostrar resultados. E=0.01

```
>>> from gradiente import*
>>> x,y=symbols('x,y')
>>> v=[x,y]
>>> u=[5,5]
>>> uk,fm=metodo_gradiente(f,v,u,0.01,20)
>>> uk
```

```
[1.001147654264687, 4.000286913566172]
>>> fm
-13.9999976127376
```

Ejemplo 3. Calcular el mínimo de $f(x,y)=x^2+2y^2+\cos(x+y+1)+xy$. Graficar y mostrar resultados. E=0.001

```
>>> from gradiente import*
>>> x,y=symbols('x,y')
>>> f=x**2+2*y**2+cos(x+y+1)+x*y
>>> plot3d(f,(x,-4,4),(y,-4,4))
```



```
>>> v=[x,y]
>>> u=[1,1]
>>> uk,fm=metodo_gradiente(f,v,u,0.001,20)
>>> uk
[0.42864627450129655, 0.14293529459696494]
>>> fm
0.285082064827950
```

Ejemplo 4. Calcular el mínimo de $f(x,y,z)=5(x-1)^2+3(y+2)^2+4(z+3)^2+xyz+1$. Mostrar resultados. E=0.01

```
>>> from gradiente import*
>>> x,y,z=symbols('x,y,z')
>>> f=5*(x-1)**2+3*(y+2)**2+4*(z+3)**2+x*y*z+1
>>> v=[x,y,z]
>>> u=[0,0,0]
>>> uk,fm=metodo_gradiente(f,v,u,0.01,20)
>>> uk
[0.4909226771078404, -1.7628499677299507, -2.8919840590397894]
>>> fm
5.01397838490301
```

Ejemplo 5. Calcular el mínimo de la siguiente función. E=0.01

$$f(x_1,x_2,x_3,x_4)=5(x_1-1)^2+3(x_2-2)^2+4(x_3+3)^2+(x_4-1)^4-x_1x_2x_3x_4 + 5.$$

```
>>> from gradiente import*
>>> x1,x2,x3,x4=symbols('x1,x2,x3,x4')
>>> f=5*(x1-1)**2+3*(x2-2)**2+4*(x3+3)**2+(x4-1)**4-x1*x2*x3*x4+5
>>> v=[x1,x2,x3,x4]
>>> u=[0,0,0,0]
>>> uk,fm=metodo_gradiente(f,v,u,0.01,20)
>>> uk
[1.1570991258533814, 2.141077839354268,
-3.0738289454327443, -0.23935943405879354]
>>> fm
5.74146881516544
```

4 MÉTODOS DIRECTOS PARA RESOLVER SISTEMAS DE ECUACIONES LINEALES

En este capítulo se estudia el componente algorítmico y computacional de los métodos directos para resolver sistemas de ecuaciones lineales.

Ejemplo. Un comerciante compra tres productos: **A, B, C**, pero en las facturas únicamente consta la cantidad comprada en Kg. y el valor total de la compra. Se necesita determinar el precio unitario de cada producto. Para esto dispone de tres facturas con los siguientes datos:

Factura	Cantidad de **A**	Cantidad de **B**	Cantidad de **C**	Valor pagado
1	**4**	**2**	**5**	**$60.70**
2	**2**	**5**	**8**	**$92.90**
3	**5**	**4**	**3**	**$56.30**

Solución

Sean $\mathbf{x_1, x_2, x_3}$ variables que representan al precio unitario de cada producto en dólares por Kg. Entonces, se puede escribir:

$$\mathbf{4x_1 + 2x_2 + 5x_3 = 60.70}$$
$$\mathbf{2x_1 + 5x_2 + 8x_3 = 92.90}$$
$$\mathbf{5x_1 + 4x_2 + 3x_3 = 56.30}$$

El modelo matemático resultante es un sistema lineal de tres ecuaciones con tres variables.

En general, se desea resolver un sistema de **n** ecuaciones lineales con **n** variables

$$\begin{aligned}
\mathbf{a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n} &= \mathbf{b_1} \\
\mathbf{a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n} &= \mathbf{b_2} \\
&\ldots \\
\mathbf{a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n} &= \mathbf{b_n}
\end{aligned}$$

En donde

$\mathbf{a_{i,j}} \in \Re$: Coeficientes

$\mathbf{b_i} \in \Re$: Constantes

$\mathbf{x_i} \in \Re$: Variables cuyo valor debe determinarse

En notación matricial:

$$\begin{bmatrix} \mathbf{a_{1,1}} & \mathbf{a_{1,2}} & \ldots & \mathbf{a_{1,n}} \\ \mathbf{a_{2,1}} & \mathbf{a_{2,2}} & \ldots & \mathbf{a_{2,n}} \\ \ldots & & & \ldots \\ \mathbf{a_{n,1}} & \mathbf{a_{n,1}} & \ldots & \mathbf{a_{n,n}} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \ldots \\ \mathbf{x_n} \end{bmatrix} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \ldots \\ \mathbf{b_n} \end{bmatrix}$$

Simbólicamente

$$\mathbf{AX = B}$$

Siendo

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & & & \dots \\ a_{n,1} & a_{n,1} & \dots & a_{n,n} \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}; \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

4.1 Determinantes y sistemas de ecuaciones lineales

Sea **A** la matriz de coeficientes del sistema **AX = B.** Sea $\mathbf{A^{-1}}$ su inversa y **|A|** su determinante. La relación entre **|A|** y la existencia de la solución **X** se establece con la siguiente definición:

$$\mathbf{A^{-1}} = \frac{\mathbf{[adj(A)]^T}}{\mathbf{|A|}},$$

En donde $\mathbf{[adj(A)]^T}$ es la transpuesta de la adjunta de la matriz **A**.

Si $\mathbf{|A| \neq 0}$, entonces $\mathbf{A^{-1}}$ existe, y se puede escribir:

$$\mathbf{AX = B} \quad \Rightarrow \quad \mathbf{A^{-1}AX = A^{-1}B} \quad \Rightarrow \quad \boldsymbol{I}\,\mathbf{X = A^{-1}B} \quad \Rightarrow \quad \mathbf{X = A^{-1}B}$$

En donde $\boldsymbol{I}$ es la matriz identidad. En resumen, si $\mathbf{|A| \neq 0}$ entonces **X** existe y además es único.

4.2 Método de Gauss - Jordan

La estrategia de este método consiste en transformar la matriz **A** del sistema **AX = B** y reducirla a la matriz identidad. Según el enunciado anterior, esto es posible si $\mathbf{|\,A\,| \neq 0.}$

Aplicando simultáneamente las mismas transformaciones al vector **B**, este se convertirá en el vector solución $\mathbf{A^{-1}B}$.

En caso de que esta solución exista, el procedimiento debe transformar las ecuaciones mediante operaciones lineales que no modifiquen la solución del sistema original**:**

a) Intercambiar ecuaciones
b) Multiplicar ecuaciones por alguna constante no nula
c) Sumar alguna ecuación a otra ecuación

Ejemplo. Con el Método de Gauss-Jordan resuelva el siguiente sistema de ecuaciones lineales

$$4x_1 + 2x_2 + 5x_3 = 60.70$$
$$2x_1 + 5x_2 + 8x_3 = 92.90$$
$$5x_1 + 4x_2 + 3x_3 = 56.30$$

Solución: Se define la matriz aumentada **A | B** para transformar simultáneamente **A** y **B**:

$$\mathbf{A \mid B} = \left[\begin{array}{ccc|c} 4 & 2 & 5 & 60.70 \\ 2 & 5 & 8 & 92.90 \\ 5 & 4 & 3 & 56.30 \end{array}\right]$$

Las transformaciones sucesivas de la matriz aumentada se describen en los siguientes pasos:

Dividir fila 1 para 4

1	**0.5**	**1.25**	**15.175**
2	5	8	92.9
5	4	3	56.3

Restar de cada fila, la fila 1 multiplicada por el elemento de la fila 1, columna 1

1	0.5	1.25	15.175
0	**4**	**5.5**	**62.55**
0	**1.5**	**-3.25**	**-19.575**

Dividir fila 2 para 4

1	0.5	1.25	15.175
0	**1**	**1.375**	**15.6375**
0	1.5	-3.25	-19.575

Restar de cada fila, la fila 2 multiplicada por el elemento de la fila 2, columna 2

1	0	**0.5625**	**7.35625**
0	1	1.3750	15.6375
0	0	**-5.3125**	**-43-03125**

Dividir fila 3 para -5.3125

1	0	0.5625	7.35625
0	1	1.375	15.6375
0	0	**1**	**8.1**

Restar de cada fila, la fila 3 multiplicada por el elemento de la fila 3, columna 3

1	0	0	2.8
0	1	0	4.5
0	0	1	8.1

La matriz de los coeficientes ha sido transformada a la **matriz identidad**.

Simultáneamente, las mismas transformaciones han convertido a la última columna en el vector solución:

$$\mathbf{X} = \begin{bmatrix} \mathbf{2.8} \\ \mathbf{4.5} \\ \mathbf{8.1} \end{bmatrix}$$

Como siempre, la solución debe verificarse en el sistema:

$$\mathbf{A * X} = \begin{bmatrix} \mathbf{4} & \mathbf{2} & \mathbf{5} \\ \mathbf{2} & \mathbf{5} & \mathbf{8} \\ \mathbf{2} & \mathbf{4} & \mathbf{3} \end{bmatrix} \begin{bmatrix} \mathbf{2.8} \\ \mathbf{4.5} \\ \mathbf{8.1} \end{bmatrix} = \begin{bmatrix} \mathbf{60.7} \\ \mathbf{92.9} \\ \mathbf{56.3} \end{bmatrix} = \mathbf{B}$$

Práctica computacional

En el lenguaje Python con la ayuda de los recursos disponibles en la librería **NumPy**, se pueden reproducir computacionalmente los cálculos hechos manualmente. Se describe su utilización aplicándolos en el ejemplo anterior, en la ventana interactiva.

En Python, la numeración de filas y columnas comienza en cero.

En la notación de índices, los dos puntos cuando se escriben a la derecha de un índice significa “todos los demás a la derecha”

$$\mathbf{4x_1 + 2x_2 + 5x_3 = 60.70}$$
$$\mathbf{2x_1 + 5x_2 + 8x_3 = 92.90}$$
$$\mathbf{5x_1 + 4x_2 + 3x_3 = 56.30}$$

Definición de la matriz de coeficientes

```
>>> from numpy import*
>>> a=array([[4,2,5],[2,5,8],[5,4,3]],float)
>>> print(a)
    [[4. 2. 5.]
     [2. 5. 8.]
     [5. 4. 3.]]
```

Definición del vector de constantes

```
>>> b=array([[60.7],[92.9],[56.3]],float)
>>> print(b)
    [[ 60.7]
     [ 92.9]
     [ 56.3]]
```

Matriz aumentada **a|b** (la indicación **axis=1** significa agregar una columna)

```
>>> c=concatenate([a,b],axis=1)
>>> print(c)
    [[  4.    2.    5.   60.7]
     [  2.    5.    8.   92.9]
     [  5.    4.    3.   56.3]]
```

Dividir la primera fila para el primer elemento de la diagonal

```
>>> c[0,0:]=c[0,0:]/c[0,0]
>>> print(c)
    [[  1.      0.5     1.25   15.175]
     [  2.      5.      8.     92.9  ]
     [  5.      4.      3.     56.3  ]]
```

Restar de la segunda y tercera filas la primera fila multiplicada por el elemento de la primera columna de cada fila

```
>>> c[1,0:]=c[1,0:]-c[1,0]*c[0,0:]
>>> c[2,0:]=c[2,0:]-c[2,0]*c[0,0:]
>>> print(c)
    [[  1.      0.5     1.25   15.175]
     [  0.      4.      5.5    62.55 ]
     [  0.      1.5    -3.25  -19.575]]
```

Dividir la segunda fila para el segundo elemento de la diagonal

```
>>> c[1,1:]=c[1,1:]/c[1,1]
>>> print(c)
    [[  1.       0.5      1.25    15.175 ]
     [  0.       1.       1.375   15.6375]
     [  0.       1.5     -3.25   -19.575 ]]
```

Restar de la primera y tercera filas la segunda fila multiplicada por el elemento de la segunda columna de cada fila

```
>>> c[0,1:]=c[0,1:]-c[0,1]*c[1,1:]
>>> c[2,1:]=c[2,1:]-c[2,1]*c[1,1:]
>>> print(c)
    [[  1.        0.        0.5625    7.35625]
     [  0.        1.        1.375    15.6375 ]
     [  0.        0.       -5.3125  -43.03125]]
```

Dividir la tercera fila para el tercer elemento de la diagonal

```
>>> c[2,2:]=c[2,2:]/c[2,2]
>>> print(c)
    [[  1.        0.        0.5625    7.35625]
     [  0.        1.        1.375    15.6375 ]
     [  0.        0.        1.        8.1    ]]
```

Restar de la primera y segunda filas la tercera fila multiplicada por el elemento de la tercera columna de cada fila

```
>>> c[0,2:]=c[0,2:]-c[0,2]*c[2,2:]
>>> c[1,2:]=c[1,2:]-c[1,2]*c[2,2:]
>>> print(c)
    [[ 1.   0.   0.   2.8]
     [ 0.   1.   0.   4.5]
     [ 0.   0.   1.   8.1]]
```

Mostrar la solución (todas las filas de la última columna)

```
>>> x=c[:,3]
>>> print(x)
    [ 2.8  4.5  8.1]
```

Verificar si la solución satisface al sistema de ecuaciones

```
>>> r=dot(a,x)
>>> print(r)
    [ 60.7  92.9  56.3]
```

Esta práctica ayuda a entender la instrumentación algorítmica y computacional del método de Gauss-Jordan y de otros métodos

4.2.1 Formulación del método de Gauss-Jordan

Para obtener la descripción detallada del algoritmo se define la matriz aumentada **A** con el vector **B** en forma simbólica general. Los cálculos se realizan con todos los elementos de esta matriz:

$$\mathbf{A \mid B} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} & a_{n,n+1} \end{bmatrix}$$

En donde se ha agregado la columna n+1 con el vector de las constantes:

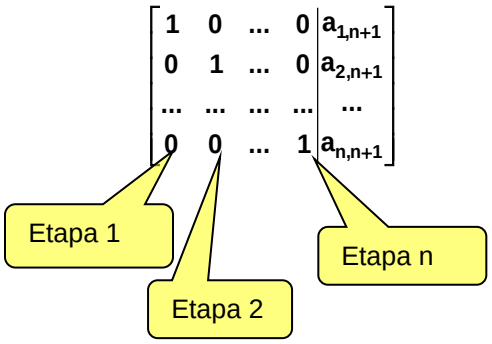
$$a_{i,n+1} = b_i, \quad i = 1, 2, 3, \ldots, n$$ (columna **n+1** de la matriz aumentada)

El objetivo es transformar esta matriz y llevarla a la forma de la matriz identidad $\mathbf{I}$:

$$\mathbf{A \mid B} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} & a_{n,n+1} \end{bmatrix} \rightarrow \ldots \rightarrow \left[\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & a_{1,n+1} \\ 0 & 1 & \cdots & 0 & a_{2,n+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & a_{n,n+1} \end{array}\right]$$

Si es posible realizar esta transformación, entonces los valores que quedan en la última columna constituirán el vector solución **X**
Las transformaciones deben ser realizadas en forma sistemática en **n** etapas, obteniendo sucesivamente en cada etapa cada columna de la matriz identidad, de izquierda a derecha.

En cada etapa, primero se hará que el elemento en la diagonal tome el valor **1**. Luego se hará que los demás elementos de la columna tomen el valor **0.**



Etapa 1

Normalizar la fila 1: (transformar el elemento $a_{1,1}$ a **1**)

$t \leftarrow a_{1,1}$, $a_{1,j} \leftarrow a_{1,j} / t$, **j=1, 2, ..., n+1;** suponer que $a_{1,1} \neq 0$

Reducir las otras filas: (transformar los otros elementos de la columna **1** a **0**)

$t \leftarrow a_{i,1}$, $a_{i,j} \leftarrow a_{i,j} - t\, a_{1,j}$, **j=1, 2, ..., n+1; i=2, 3, ..., n**

$$A \mid B = \left[\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ a_{n,1} & a_{n,1} & \cdots & a_{n,n} & a_{n,n+1} \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} 1 & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ 0 & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ 0 & a_{n,1} & \cdots & a_{n,n} & a_{n,n+1} \end{array}\right]$$

Valores transformados

Etapa 2

Normalizar la fila 2: (transformar el elemento $a_{2,2}$ a **1**)

$t \leftarrow a_{2,2}$, $a_{2,j} \leftarrow a_{2,j} / t$, **j=2, 3, ..., n+1;** suponer que $a_{2,2} \neq 0$

Reducir las otras filas: (transformar los otros elementos de la columna **2** a **0**)

$t \leftarrow a_{i,2}$, $a_{i,j} \leftarrow a_{i,j} - t\, a_{2,j}$, **j=2, 3, ..., n+1; i=1, 3, ..., n**

$$\left[\begin{array}{cccc|c} 1 & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ 0 & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ 0 & a_{n,1} & \cdots & a_{n,n} & a_{n,n+1} \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} 1 & 0 & \cdots & a_{1,n} & a_{1,n+1} \\ 0 & 1 & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ 0 & 0 & \cdots & a_{n,n} & a_{n,n+1} \end{array}\right]$$

Valores transformados

La formulación obtenida en estas dos etapas se puede generalizar

Para cada columna **e=1, 2, …, n**

Normalizar la fila **e**

$t \leftarrow a_{e,e}$, $a_{e,j} \leftarrow a_{e,j} / t$, **j=e+1, e+2, e+3, ..., n+1;** ($a_{e,e} \neq 0$)

Reducir las otras filas

$t \leftarrow a_{i,e}$, $a_{i,j} \leftarrow a_{i,j} - t\, a_{e,j}$; **i=1, 2, 3, ..., n; j=e+1, e+2, e+3, ..., n+1; i ≠ e**

La última columna de la matriz transformada contendrá la solución

4.2.2 Algoritmo de Gauss-Jordan básico

Algoritmo: Gauss-Jordan básico
Restricción: No se verifica unicidad y existencia de la solución
Entra: n Cantidad de ecuaciones
$a_{i,j}$ Coeficientes
b_i Constantes
Sale: x_i Solución calculada

```
Para i ← 1, 2, …, n                          Matriz aumentada
    a_{i,n+1} ← b_i
Fin

Para  e = 1, 2, . . ., n
    t ← a_{e,e}
    Para j=e, e+1, ..., n+1
        a_{e,j} ← a_{e,j} / t                 Normalizar la fila  e  (a_{e,e} ≠ 0)
    Fin
    Para i=1, 2, …, n;   i ≠ e
        t ← a_{i,e}
        Para j=e, e+1, ..., n+1
            a_{i,j} ← a_{i,j} – t a_{e,j}     Reducir las otras filas
        Fin
    Fin
 Fin
 Para i=1,2,...,n
     x_i ← a_{i,n+1}              La última columna contendrá la solución
 Fin
```

4.2.3 Eficiencia del método de Gauss-Jordan

El método de Gauss-Jordan es un método directo. Los métodos directos pueden ser afectados por el error de redondeo, es decir los errores en la representación de los números que se producen en las operaciones aritméticas. Para cuantificar la magnitud del error de redondeo se define la función de eficiencia del método.

Sea **n** el tamaño del problema y **T(n)** la cantidad de operaciones aritméticas que se realizan

En la normalización: $T(n) = O(n^2)$ (Dos ciclos anidados)
En la reducción: $T(n) = O(n^3)$ (Tres ciclos anidados)

Por lo tanto, este método es de tercer orden: **$T(n) = O(n^3)$**

Mediante un conteo recorriendo los ciclos del algoritmo, se puede determinar la función de eficiencia para este método directo, considerando solamente la sección crítica que incluye los tres ciclos:

e	i	j
1	n-1	n+1
2	n-1	n
.	.	.
.	.	.
.	.	.
n-1	n-1	3
n	n-1	2

$T(n) = (n-1)(2 + 3 + n + (n+1)) = (n-1)\ (3 + n\ )\ (n/2) = n^3/2 + n^2 - 3n/2$

Una función en Python para conteo de los ciclos del método de Gauss-Jordan

También se puede obtener computacionalmente T(n) mediante una función que realice el conteo de ciclos. Solamente es de interés usar una variable para registrar la cantidad de ciclos. Por la estructura del algoritmo T(n) debe ser un polinomio cúbico, por lo tanto se necesitan cuatro puntos y con estos cuatro puntos se puede construir el polinomio cúbico.

```
def conteogaussjordan(n):
    c=0
    for e in range(n):
        for i in range(n):
            if i!=e:
                for j in range(e,n+1):
                    c=c+1                    %Ciclos en la sección crítica
    return c
```

Mediante cuatro pruebas se obtuvieron los puntos:

(n, c): (10, 585), (20, 4370), (30, 14355), (40, 33540)

Con estos cuatro puntos se construye el polinomio cúbico:

$$T(n) = n^3/2 + n^2 - 3n/2 = O(n^3)$$

El polinomio es exacto. Si se usaran más puntos el resultado sería igual.

4.2.4 Instrumentación computacional del método de Gauss-Jordan básico

En esta primera versión del algoritmo se supondrá que el determinante de la matriz es diferente de cero y que no se requiere intercambiar filas.

La codificación en el lenguaje Python usa la formulación matemática y el algoritmo descritos anteriormente. La instrumentación sigue en forma detallada la formulación desarrollada a nivel de componentes de la matriz.

En una segunda versión, más adelante, se usará la notación de índices implícita de Python. Esa instrumentación es compacta. Ambas instrumentaciones son traducciones adecuadas

```
#Solución de un sistema lineal: Gauss-Jordan básico
import numpy as np
def gaussjordan1(a,b):
    n=len(b)
    c=np.concatenate([a,b],axis=1)      #matriz aumentada
    for e in range(n):
        t=c[e,e]
        for j in range(e,n+1):
            c[e,j]=c[e,j]/t             #Normalizar fila e
        for i in range(n):
            if i!=e:
                t=c[i,e]
                for j in range(e,n+1):
                    c[i,j]=c[i,j]-t*c[e,j]      #Reducir otras filas
    x=c[:,n]
    return x
```

NOTA. Las funciones en Python reciben listas o vectores y matrices por referencia, esto significa que si son modificadas dentro de la función, fuera de ella las listas también habrán sido modificadas. Para evitar que se hagan estos cambios debe crearse una copia dentro de la función.

Instrumentación del algoritmo Gauss-Jordan usando notación implícita de índices

```
#Solución de un sistema lineal: Gauss-Jordan básico
import numpy as np
def gaussjordan1(a,b):
    n=len(b)
    c=np.concatenate([a,b],axis=1)                #Matriz aumentada
    for e in range(n):
        c[e,e:]=c[e,e:]/c[e,e]                    #Normalizar fila e
        for i in range(n):
            if i!=e:
                c[i,e:]=c[i,e:]-c[i,e]*c[e,e:]    #Reducir otras filas
    x=c[:,n]
    return x
```

Ejemplo. Desde la ventana interactiva de Python, use la función Gauss-Jordan para resolver el sistema. Cualquiera de las dos instrumentaciones entrega la solución.

$$4x_1 + 2x_2 + 5x_3 = 18.00$$
$$2x_1 + 5x_2 + 8x_3 = 27.30$$
$$2x_1 + 4x_2 + 3x_3 = 16.20$$

Escriba en la ventana interactiva de Python

```
>>> from numpy import*
>>> from gaussjordan1 import*
>>> a=array([[4,2,5],[2,5,8],[2,4,3]],float)      Matriz de coeficientes
>>> b=array([[18.0],[27.3],[16.2]],float)         Vector columna de constantes
>>> print(a)
[[4. 2. 5.]
 [2. 5. 8.]
 [2. 4. 3.]]
>>> print(b)
[[ 18. ]
 [ 27.3]
 [ 16.2]]
>>> x=gaussjordan1(a,b)                           Vector solución
>>> print(x)
[ 1.2  2.1  1.8]
```

Es necesario verificar que la solución calculada satisface al modelo matemático. El producto de la matriz original **A** multiplicada por el vector solución **X** calculado debe ser igual al vector **B**:

```
>>> r=dot(a,x)
>>> print(r)
[ 18.   27.3  16.2]
```

Puede haber alguna diferencia por los errores de redondeo. El vector **R=Ax-b** se denomina "**residual**" y se puede usar para cuantificar el **error de redondeo en los cálculos**.

Medición experimental de la eficiencia mediante el registro de tiempo de ejecución

Cuando se tiene únicamente la instrumentación computacional de un algoritmo, se puede obtener experimentalmente su eficiencia registrando, para diferentes valores de **n**, el tiempo de ejecución del algoritmo. Este tiempo depende de la velocidad del procesador del dispositivo computacional, pero es proporcional a **T(n)**.

Para registrar el tiempo real de ejecución de un proceso (programa o función) se puede usar la función **clock()** de la librería **time**. La diferencia de tiempos es el tiempo de ejecución real.

```
>>> from time import*
>>> t1=clock(); ... proceso que se desa medir ... ;t2=clock();print(t2-t1)
```

Ejemplo. Registrar el tiempo real de ejecución del algoritmo básico de Gauss-Jordan :

```
>>> t1=clock();x=gaussjordan1(a,b);t2=clock();print(t2-t1)
```

Para las pruebas se pueden generar matrices y vectores grandes, con números aleatorios.

4.2.5 Obtención de la inversa de una matriz

Para encontrar la matriz inversa se puede usar el método de Gauss-Jordan.

Sea **A** una matriz cuadrada cuyo determinante es diferente de cero.
Sean $\mathbf{t_1, t_2, \ldots, t_{m-1}, t_m}$ las transformaciones lineales del método de Gauss-Jordan que transforman la matriz **A** en la matriz identidad **I** incluyendo intercambios de filas

$$\mathbf{t_m\ t_{m-1} \ldots t_2\ t_1\ A = I}$$

Entonces se puede escribir

$$\mathbf{t_m\ t_{m-1} \ldots t_2\ t_1\ A^{-1} A = A^{-1} I \quad \Rightarrow \quad t_m\ t_{m-1} \ldots t_2\ t_1\ I = A^{-1}}$$

Lo cual significa que las mismas transformaciones que convierten **A** en la matriz **I,** convertirán la matriz **I** en la matriz $\mathbf{A^{-1}}$.

Para aplicar este algoritmo, suponiendo que se desea conocer la matriz $\mathbf{A^{-1}}$, se debe aumentar la matriz anterior con la matriz **I**: **A | B | I**

Las transformaciones aplicadas simultáneamente proporcionarán finalmente el vector solución **X** y la matriz identidad $\mathbf{A^{-1}}$

Ejemplo. Con el Método de Gauss-Jordan resuelva el sistema de ecuaciones siguiente y simultáneamente obtenga la matriz inversa:

$$\begin{aligned} 4x_1 + 2x_2 + 5x_3 &= 18.00 \\ 2x_1 + 5x_2 + 8x_3 &= 27.30 \\ 2x_1 + 4x_2 + 3x_3 &= 16.20 \end{aligned}$$

Solución. La matriz aumentada es:

$$\mathbf{A \mid B} = \left[\begin{array}{ccc|c|ccc} 4 & 2 & 5 & 18.00 & 1 & 0 & 0 \\ 2 & 5 & 8 & 27.30 & 0 & 1 & 1 \\ 2 & 4 & 3 & 16.20 & 0 & 0 & 1 \end{array}\right]$$

Cálculos

Normalizar fila 1 y reducir filas 2 y 3

1	0.5	1.25	4.5	0.25	0	0
0	4	5.5	18.3	-0.5	1	0
0	3	0.5	7.2	-0.5	0	1

Normalizar fila 2 y reducir filas 1 y 3

1	0	0.5625	2.2125	0.3125	-0.125	0
0	1	1.375	4.575	-0.125	0.25	0
0	0	-3.625	-6.525	-0.125	-0.75	1

Normalizar fila 3 y reducir filas 1 y 2

1	0	0	1.2	0.2931	-0.2414	0.1552
0	1	0	2.1	-0.1724	-0.0345	0.3793
0	0	1	1.8	0.0345	0.2069	-0.2759

Solución del sistema

$$\mathbf{X} = \begin{bmatrix} \mathbf{1.2} \\ \mathbf{2.1} \\ \mathbf{1.8} \end{bmatrix}$$

Matriz inversa

$$\mathbf{A^{-1}} = \begin{bmatrix} \mathbf{0.2931} & \mathbf{-0.2414} & \mathbf{0.1552} \\ \mathbf{-0.1724} & \mathbf{-0.0345} & \mathbf{0.3793} \\ \mathbf{0.0345} & \mathbf{0.2069} & \mathbf{-0.2759} \end{bmatrix}$$

4.3 Método de Gauss

El método de Gauss es similar al método de Gauss-Jordan. Aquí se trata de transformar la matriz del sistema a una forma triangular superior**.** Si esto es posible entonces la solución se puede obtener resolviendo el sistema triangular resultante.

Ejemplo. Con el Método de Gauss resuelva el sistema de ecuaciones siguiente:

$$\begin{aligned} 4x_1 + 2x_2 + 5x_3 &= 18.00 \\ 2x_1 + 5x_2 + 8x_3 &= 27.30 \\ 2x_1 + 4x_2 + 3x_3 &= 16.20 \end{aligned}$$

Solución: Se define la matriz aumentada **A | B** para transformar simultáneamente **A** y **B**:

$$\mathbf{A \mid B} = \left[\begin{array}{ccc|c} 4 & 2 & 5 & 18.00 \\ 2 & 5 & 8 & 27.30 \\ 2 & 4 & 3 & 16.20 \end{array}\right]$$

Las transformaciones sucesivas de la matriz aumentada se describen en los siguientes cuadros

Dividir fila 1 para 4

1	0.5	1.25	4.5
2	5	8	27.3
2	4	3	16.2

Restar de cada fila, la fila 1 multiplicada por el elemento de la columna 1

1	0.5	1.25	4.5
0	4	5.5	18.3
0	3	0.5	7.2

Dividir fila 2 para 4

1	0.5	1.25	4.5
0	1	1.375	4.575
0	3	0.5	7.2

Restar de la fila, la fila 2 multiplicada por el elemento de la columna 2

1	0.5	1.25	4.5
0	1	1.375	4.575
0	0	-3.625	-6.525

Dividir fila 3 para -3.625

1	0.5	1.25	4.5
0	1	1.375	4.575
0	0	1	1.8

La matriz de los coeficientes ha sido transformada a la forma triangular superior

De este sistema se obtiene la solución mediante una sustitución directa comenzando por el final:

$$x_3 = 1.8$$
$$x_2 = 4.575 - 1.375(1.8) = 2.1$$
$$x_1 = 4.5 - 0.5(2.1) - 1.25(1.8) = 1.2$$

4.3.1 Formulación del método de Gauss

Para unificar la descripción algorítmica, es conveniente aumentar la matriz **A** con el vector **B** pues deben realizarse las mismas operaciones simultáneamente:

$$\mathbf{A \mid B} = \left[\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ a_{n,1} & a_{n,1} & \cdots & a_{n,n} & a_{n,n+1} \end{array}\right]$$

En donde la columna de los coeficientes se define:

$$a_{i,n+1} = b_i \text{ , } i=1, 2, 3,. . ., n$$

La formulación se obtiene directamente del método de Gauss-Jordan, pero la reducción de las filas únicamente se realiza en la sub-matriz triangular inferior.

Para cada columna **e = 1, 2, …, n**

Normalizar la fila **e**

$$t \leftarrow a_{e,e} \text{ , } a_{e,j} \leftarrow a_{e,j} / t \text{ , } j=e+1, e+2, e+3, ..., n+1; \qquad (a_{e,e} \neq 0)$$

Reducir las otras filas debajo de la diagonal

$$t \leftarrow a_{i,e} \text{ , } a_{i,j} \leftarrow a_{i,j} - t\, a_{e,j} \text{ , } i=e+1, e+2, ..., n; \quad j=e+1, e+2, e+3, ..., n+1$$

Las transformaciones convierten la matriz aumentada en la forma triangular superior:

$$\mathbf{A \mid B} = \left[\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ a_{n,1} & a_{n,1} & \cdots & a_{n,n} & a_{n,n+1} \end{array}\right] \rightarrow . \; . \; . \; . \rightarrow \left[\begin{array}{cccc|c} 1 & a_{1,2} & \cdots & a_{1,n} & a_{1,n+1} \\ 0 & 1 & \cdots & a_{2,n} & a_{2,n+1} \\ \cdots & & & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & a_{n,n+1} \end{array}\right]$$

De sistema triangular se puede obtener directamente la solución. Para facilitar la notación expandimos la forma triangular final obtenida:

$$\left[\begin{array}{cccccc|c} 1 & a_{1,2} & \dots & a_{1,n-2} & a_{1,n-1} & a_{1,n} & a_{1,n+1} \\ 0 & 1 & \dots & a_{2,n-2} & a_{2,n-1} & a_{2,n} & a_{2,n+1} \\ \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & a_{n-2,n-1} & a_{n-2,n} & a_{n-2,n+1} \\ 0 & 0 & \dots & 0 & 1 & a_{n-1,n} & a_{n-1,n+1} \\ 0 & 0 & \dots & 0 & 0 & 1 & a_{n,n+1} \end{array}\right]$$

La solución se obtiene comenzando desde el final:

$x_n \leftarrow a_{n,\, n+1}$

$x_{n-1} \leftarrow a_{n-1,\, n+1} - a_{n-1,\, n} x_n$

$x_{n-2} \leftarrow a_{n-2,\, n+1} - (a_{n-2,\, n-1} x_{n-1} - a_{n-2,\, n} x_n)$

. . . etc

Con la formulación anterior se define el algoritmo para el método de Gauss básico:

4.3.2 Algoritmo de Gauss básico

```
Algoritmo: Gauss básico
Restricción: No se verifica unicidad y existencia de la solución
Entra: n:    Número de ecuaciones
       a_i,j Coeficientes
       b_i   Constantes
Sale:  x_i   Solución calculada
Para i ← 1, 2, …, n                    Matriz aumentada
    a_i,n+1 ← b_i
Fin

 Para  e = 1, 2, . . ., n
     t ← a_e,e
      Para j = e, e+1, ..., n+1
       a_e,j ← a_e,j / t                Normalizar la fila  e  ( a_e,e ≠ 0 )
      Fin
      Para  i = e + 1, e + 2,… n
            t ← a_i,e
            Para j = e+1,e+2,..., n+1
               a_i,j ← a_i,j – t a_e,j  Reducir las filas debajo
            Fin
      Fin
 Fin
```

```
x_n ← a_{n, n+1}                    Resolver el sistema triangular
Para i = n-1, n-2, ..., 1
     s ← 0
     Para j = i+1, i+2, ..., n
          s ← s + a_{i,j} x_j
     Fin
     x_i ← a_{i,n+1} – s
Fin
```

4.3.3 Eficiencia del método de Gauss

Sea **n** el tamaño del problema y **T(n)** la cantidad de operaciones aritméticas que se realizan

En la normalización:	$T(n) = O(n^2)$	(dos ciclos anidados)
En la reducción:	$T(n) = O(n^3)$	(tres ciclos anidados)
En la obtención de la solución:	$T(n) = O(n^2)$	(dos ciclos anidados)

Por lo tanto, este método es de tercer orden: **$T(n) = O(n^3)$**

La obtención de **T(n)** puede ser realizada mediante el análisis matemático de la formulación o con un recorrido detallado de los ciclos del algoritmo. Se puede determinar que: **$T(n) = n^3/3 + O(n^2)$** con lo que se concluye que el método de Gauss es más eficiente que el método de Gauss-Jordan para **n** grande. Esta diferencia se la puede constatar experimentalmente resolviendo sistemas grandes y registrando el tiempo real de ejecución.

Proponemos un método computacional para obtener **T(n)**. Es suficiente escribir código computacional solamente para contar la cantidad de ciclos para diferentes valores de **n**. Los puntos **(n, T(n)** permiten obtener matemáticamente la función **T(n)**, la cual, por la estructura del algoritmo debe ser un polinomio cúbico, por lo tanto son suficientes cuatro puntos.

Una función en Python para conteo de los ciclos del método de Gauss

```
def conteogauss(n):
    c=0
    for e in range(n):
        for i in range(e+1,n):
            for j in range(e,n+1):
                c=c+1                    %Ciclos en la sección crítica
    return c
```

Se requieren solo cuatro puntos. Estos puntos se los obtuvo mediante cuatro pruebas:

(n, c): (10, 375), (20, 2850), (30, 9425), (40, 22100)

Con estos puntos se construye la función **$T(n) = n^3/3 + n^2/2 - (5n)/6 = O(n^3)$**

El polinomio es exacto. Si se usaran más puntos, el resultado sería igual.

4.3.4 Instrumentación computacional de método de Gauss básico

En esta primera versión del algoritmo se supondrá que el determinante de la matriz es diferente de cero y que no se requiere intercambiar filas. La codificación en Python sigue directamente la formulación matemática y el algoritmo descritos anteriormente.

```
#Solución de un sistema lineal: Gauss básico
import numpy as np
def gauss1(a,b):
    n=len(b)
    c=np.concatenate([a,b],axis=1)        #matriz aumentada
    for e in range(n):
        t=c[e,e]
        for j in range(e,n+1):            #Normalizar fila  e
            c[e,j]=c[e,j]/t
        for i in range(e+1,n):            #Reducir filas debajo
            t=c[i,e]
            for j in range(e,n+1):
                c[i,j]=c[i,j]-t*c[e,j]
    x=np.zeros([n,1])                     #Celdas para el vector X
    x[n-1]=c[n-1,n]
    for i in range(n-2,-1,-1):            #Sistema triangular
        s=0
        for j in range(i+1,n):
            s=s+c[i,j]*x[j]
        x[i]=c[i,n]-s
    return x
```

Instrumentación del método de Gauss básico usando notación implícita de índices

```
#Solución de un sistema lineal: Gauss básico
import numpy as np
def gauss1(a,b):
    n=len(b)
    c=np.concatenate([a,b],axis=1)            #Matriz aumentada
    for e in range(n):
        c[e,e:]=c[e,e:]/c[e,e]                #Normalizar fila  e
        for i in range(e+1,n):
            c[i,e:]=c[i,e:]-c[i,e]*c[e,e:]    #Reducir filas debajo
    x=np.zeros([n])
    x[n-1]=c[n-1,n]
    for i in range(n-2,-1,-1):                #Sistema triangular
        x[i]=c[i,n]-np.dot(x[i+1:n],c[i,i+1:n])
    return x
```

Ejemplo. Desde la ventana interactiva de Python, use la función Gauss1 para resolver

$$\begin{bmatrix} 2 & 3 & 7 \\ -2 & 5 & 6 \\ 8 & 9 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 8 \end{bmatrix}$$

Cualquiera de las dos instrumentaciones del método de Gauss desarrolladas en la página anterior entrega la solución

Escriba en la ventana interactiva de Python

```
>>> from numpy import*
>>> from gauss1 import*
>>> a=array([[2,3,7],[-2,5,6],[8,9,4]],float)      Matriz de coeficientes
>>> b=array([[3],[5],[8]],float)                   Vector columna de constantes
>>> x=gauss1(a,b)                                  Vector solución
>>> print(x)
[-0.05555556  0.91503268  0.05228758]
```

Registro experimental de la diferencia de tiempos de ejecución real de los métodos de Gauss-Jordan y Gauss

Para la prueba se generará un sistema de 1000 ecuaciones cuyas matrices y constantes son números aleatorios reales entre 0 y 1. El resultado es tiempo en segundos.

```
>>> import numpy as np
>>> from time import*
>>> from gaussjordan1 import*
>>> from gauss1 import*
>>> A=np.array([[np.random.rand() for j in range(1000)] for i in
range(1000)])
>>> B=np.array([[np.random.rand() for j in range(1)] for i in
range(1000)])
>>> t1=clock();X=gaussjordan1(A,B);t2=clock();print(t2-t1)
7.3277532340907925
>>> t1=clock();X=gauss1(A,B);t2=clock();print(t2-t1)
3.799714028466532
```

4.3.5 Estrategia de pivoteo

El método de Gauss es más eficiente que el método de Gauss-Jordan. Esto implica también menor propagación del error de redondeo. En esta sección se describe una técnica para reducir adicionalmente el error de redondeo en base al error propagado en la operación aritmética en la sección crítica del método de Gauss. Al mismo tiempo esta técnica perimte determinar si el sistema no tiene solución única.

Formulación del método de Gauss:

Etapa **e = 1, 2, . . ., n**

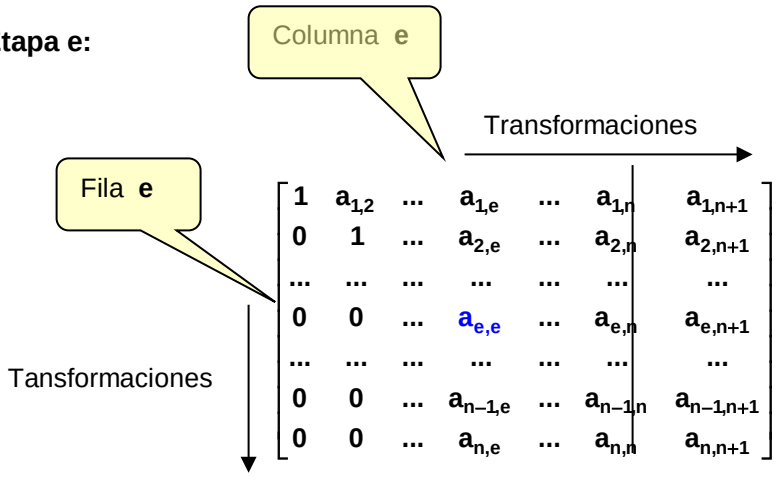
Normalizar la fila **e:**

$$t \leftarrow a_{e,e} \;,\; a_{e,j} \leftarrow a_{e,j} / t \;,\; j=e+1, e+2, e+3, ..., n+1; \quad (a_{e,e} \neq 0)$$

Reducir las otras filas debajo de la diagonal

$$t \leftarrow a_{i,e} \;,\; a_{i,j} \leftarrow a_{i,j} - t\, a_{e,j} \;,\; i=e+1, e+2, ..., n; \quad j=e+1, e+2, e+3, ..., n+1$$

Etapa e:



Recordando la definición del error de redondeo propagado en la multiplicación:

$$E_{XY} = \overline{X}\, E_Y + \overline{Y}\, E_X$$

Una estrategia para disminuir el error de redondeo consiste en reducir el valor de los operandos que intervienen en la multiplicación.

En la estrategia de **"Pivoteo Parcial",** antes de normalizar la fila **e** se busca en la columna **e** de cada fila **i = e, e+1, . . ., n** cual es el elemento con mayor magnitud. Si se usa este elemento como divisor para la fila **e**, el cociente $a_{e,j}$ tendrá el menor valor. Este factor permite disminuir el error cuando se realiza la etapa de **reducción** de las otras filas.

Por otra parte, si en esta estrategia de búsqueda, el valor elegido como el mayor divisor no es diferente de cero, se concluye que en el sistema existen ecuaciones redundantes o incompatibles, entonces el sistema no tiene solución única y el algoritmo debe terminar

4.3.6 Algoritmo de Gauss con pivoteo

Algoritmo: Gauss básico con pivoteo parcial
Entra: n: Número de ecuaciones
$a_{i,j}$ Coeficientes
b_i Constantes
Sale: x_i Solución calculada o un vector nulo si la solución no es única

```
Para i ← 1, 2, …, n                          Matriz aumentada
   a_{i,n+1} ← b_i
Fin

Para  e = 1, 2, . . ., n
    p ← e
    Para i = e+1,e+2, ..., n                 Buscar el pivote entre las filas e hasta n
        Si |a_{i,e}| > |a_{e,p}|
           p←i
        Fin
    Fin
    Intercambiar las filas e y p
    Si a_{e,e} = 0                           La solución no existe o no es única
        X ← nulo
        Terminar
    Fin
    t ← a_{e,e}
    Para j = e, e+1, ..., n+1
        a_{e,j} ← a_{e,j} / t                Normalizar la fila  e  ( a_{e,e} ≠ 0 )
    Fin
    Para  i = e + 1, e + 2,… n
        t ← a_{i,e}
        Para j = e+1,e+2,..., n+1
            a_{i,j} ← a_{i,j} – t a_{e,j}    Reducir las filas debajo de la diagonal
        Fin
    Fin
Fin
x_n ← a_{n, n+1}                             Resolver el sistema triangular superior
Para i=n-1, n-2, ..., 1
    s ← 0
    Para j=i+1, i+2, ..., n
        s ← s + a_{i,j} x_j
    Fin
    x_i ← a_{i,n+1} – s
Fin
```

4.3.7 Instrumentación computacional del método de Gauss con pivoteo

La siguiente instrumentación en Python del método de eliminación de Gauss incluye la formulación descrita y la estrategia de "pivoteo parcial" vista anteriormente. En esta instrumentación final se incluye un chequeo del divisor para prevenir el caso de que el sistema sea singular aunque.

Si un sistema es singular, el elemento pivote debería ser cero, pero debido a los errores de redondeo, no será exactamente igual a cero, por lo que se considera como criterio que el sistema es singular si el valor del pivote es menor a 10^{-20}

```
#Solución de un sistema lineal: Gauss con pivoteo
import numpy as np
def gauss(a,b):
    n=len(b)
    c=np.concatenate([a,b],axis=1)            #Matriz aumentada

    for e in range(n):
        p=e
        for i in range(e+1,n):                #Pivoteo
            if abs(c[i,e])>abs(c[p,e]):
                p=i

        for j in range(e,n+1):                #Intercambio de filas
            t=c[e,j]
            c[e,j]=c[p,j]
            c[p,j]=t

        t=c[e,e]
        if abs(t)<1e-20:                      #Sistema singular
            return []

        c[e,e:]=c[e,e:]/c[e,e]                #Normalizar fila  e
        for i in range(e+1,n):
            c[i,e:]=c[i,e:]-c[i,e]*c[e,e:]    #Reducir filas debajo

    x=zeros([n])
    x[n-1]=c[n-1,n]
    for i in range(n-2,-1,-1):                #Sistema triangular
        x[i]=c[i,n]-np.dot(x[i+1:n],c[i,i+1:n])
    return x
```

Ejemplo. Desde la ventana interactiva de Python use la función Gauss para resolver:

$$\begin{bmatrix} 2 & 3 & 7 \\ -2 & 5 & 6 \\ 8 & 9 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 8 \end{bmatrix}$$

Solución

Escriba en la ventana interactiva de Python

```
>>> from numpy import*
>>> from gauss import*
>>> a=array([[2,3,7],[-2,5,6],[8,9,4]],float)      Matriz de coeficientes
>>> b=array([[3],[5],[8]],float)                   Vector columna de constantes
>>> x=gauss(a,b)
>>> print(x)
[-0.05555556  0.91503268  0.05228758]
```

4.3.8 Funciones de Python para sistemas de ecuaciones lineales

Resolución de un sistema de ecuaciones lineales con las funciones de la librería **NumPy**

Ejemplo. Resolver el sistema

$$\begin{bmatrix} 2 & 4 & 5 \\ 3 & 1 & 4 \\ 5 & 2 & 4 \end{bmatrix} X = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}$$

```
>>> from numpy import*
>>> a=array([[2,4,5],[3,1,4],[5,2,4]],float)      Matriz de coeficientes
>>> b=array([[5],[6],[7]],float)                  Vector columna de constantes
>>> linalg.det(a)                                 Determinante de la matriz
28.999999999999989
>>> x=dot(linalg.inv(a),b)                        Mltipilcar la inversa de a por b
>>> print(x)
[[ 0.72413793],                                   Vector solución
 [-0.44827586],
 [ 1.06896552]]
```

Con la opción **set_printoptions(precision=d)** se puede controlar la cantidad de decimales **d** al imprimir los arreglos de la librería **NumPy**

```
>>> set_printoptions(precision=4)
>>> print(x)
[[ 0.7241],
 [-0.4483],
 [ 1.069 ]]
```

Verificar que la solución calculada satisface al sistema de ecuaciones:

```
>>> r=dot(a,x)
>>> print(r)
[[ 5.],
 [ 6.],
 [ 7.]]
```

En lugar de usar la inversa, se puede usar directamente el método **solve** de **NumPy:**

```
>>> x=linalg.solve(a,b)
>>> print(x)
[[ 0.72413793],
 [-0.44827586],
 [ 1.06896552]]
```

En general es preferible usar las librerías de Python asignándoles un nombre de identificación

Ejemplo.

```
>>> import numpy as np
>>> a=np.array([[2,4,5],[3,1,4],[5,2,4]],float)
>>> np.linalg.det(a)
28.999999999999989
```

4.3.9 Cálculo del determinante de una matriz

El algoritmo de Gauss transforma la matriz cuadrada de los coeficientes a la forma triangular superior. En una matriz triangular, el determinante es el producto de los coeficientes de la diagonal principal. Por lo tanto, el determinante se puede calcular multiplicando los divisores colocados en la diagonal principal, considerando además el número de cambios de fila que se hayan realizado en la estrategia de pivoteo.

Sean

A: matriz cuadrada
T: Matriz triangular superior obtenida con el algoritmo de Gauss, sin normalizar
$\mathbf{a_{i,i}}$ **:** Elementos en la diagonal de la matriz **T**. Son los divisores
k: Número de cambios de fila realizados
det(A): Determinante de la matriz **A**

Entonces

$$\mathbf{det(A) = (-1)^k \prod_{i=1}^{n} a_{i,i}}$$

4.4 Sistemas mal condicionados

Al resolver un sistema de ecuaciones lineales usando un método directo, es necesario analizar si el resultado calculado es confiable. En esta sección se estudia el caso especial de sistemas que son muy sensibles a los errores en los datos o en los cálculos y que al resolverlos producen resultados con mucha variabilidad. Para describir estos sistemas se considera el siguiente ejemplo:

Ejemplo. Un comerciante compra cuatro productos: **A, B, C, D** pero en las facturas únicamente consta la cantidad comprada en Kg. y el valor total de la compra en dólares. Se necesita determinar el precio unitario de cada producto. Se dispone de cuatro facturas con los siguientes datos:

Factura	Kg. de **A**	Kg. de **B**	Kg. de **C**	Kg. de **D**	Valor pagado
1	2.6	0.3	2.4	6.2	50.78
2	7.7	0.4	4.7	1.4	47.36
3	5.1	9.9	9.5	1.5	91.48
4	6.0	7.0	8.5	4.8	98.17

Sean x_1, x_2, x_3, x_4 variables que representan al precio unitario ($/kg) de cada producto. Entonces, se puede escribir:

$$\begin{aligned} 2.6x_1 + 0.3x_2 + 2.4x_3 + 6.2x_4 &= 50.78 \\ 7.7x_1 + 0.4x_2 + 4.7x_3 + 1.4x_4 &= 47.36 \\ 5.1x_1 + 9.9x_2 + 9.5x_3 + 1.5x_4 &= 91.48 \\ 6.0x_1 + 7.0x_2 + 8.5x_3 + 4.8x_4 &= 98.17 \end{aligned}$$

En notación matricial

$$\begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.0 & 7.0 & 8.5 & 4.8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 50.78 \\ 47.36 \\ 91.48 \\ 98.17 \end{bmatrix}$$

Si se resuelve este sistema con un método directo se obtiene:

```
>>> from numpy import*
>>> A=array([[2.6,0.3,2.4,6.2],[7.7,0.4,4.7,1.4],[5.1,9.9,9.5,1.5],
[6.0,7.0,8.5,4.8]],float)
>>> B=array([[50.78],[47.36],[91.48],[98.17]],float)
>>> X=linalg.solve(A,B)

X = [ 2.5]
    [ 3.2]
    [ 4.1]
    [ 5.4]
```

Supondremos ahora que el digitador se equivoca al ingresar los datos en la matriz y registra **6.1** en lugar del valor correcto **6.0**, de tal manera que el nuevo sistema es:

$$\begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.1 & 7.0 & 8.5 & 4.8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 50.78 \\ 47.36 \\ 91.48 \\ 98.17 \end{bmatrix}$$

Si se resuelve este sistema con el método directo se obtiene:

```
X = [ 0.1494]
    [ 0.4182]
    [ 8.3432]
    [ 4.8777]
```

Un cambio menor en un coeficiente produjo un cambio muy significativo en la solución. El resultado fue afectado fuertemente por este error. Esto es un indicio de que el sistema es de un tipo especial denominado **mal condicionado**.

En la siguiente prueba, se modifica el último coeficiente **4.8** y se lo sustituye por **4.7**

$$\begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.0 & 7.0 & 8.5 & 4.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 50.78 \\ 47.36 \\ 91.48 \\ 98.17 \end{bmatrix}$$

Si se resuelve este sistema nuevamente con un método directo se obtiene una solución incoherente:

```
X = [-31.5482]
    [-37.0941]
    [ 65.5636]
    [ -2.1643]
```

Los resultados obtenidos con este tipo de sistemas no son confiables para tomar decisiones.

Es necesario detectar si un sistema es mal condicionado. Para esto, se debe cambiar ligeramente el valor de algún coeficiente y observar el cambio en el vector solución. Si la solución cambia significativamente, entonces es un sistema mal condicionado y debe revisarse la elaboración del modelo matemático.

Esta situación se origina en el hecho de que algunas ecuaciones pueden depender de otras ecuaciones, lo cual puede afectar a la confianza en la solución calculada. Este hecho se puede asociar al valor del determinante de la matriz. En el ejemplo, si la matriz se normaliza dividiéndola para la magnitud del mayor elemento, el determinante es $\mathbf{0.0001555}$

Si el determinante fuera cero no habría una solución única, pero este valor muy pequeño es un indicio que algunas ecuaciones son “bastante dependientes” de otras ecuaciones.

En esta sección se establece una medida para cuantificar el nivel de mal condicionamiento de un sistema de ecuaciones lineales.

4.4.1 Definiciones

La norma de un vector o de una matriz es una manera de expresar la magnitud de sus componentes

Sean **X**: vector de n componentes
A: matriz de nxn componentes

Algunas definiciones comunes para la norma:

$$\|\mathbf{X}\| = \sum_{i=1}^{n} |x_i|$$

$$\|\mathbf{X}\| = \max |x_i|, \; i = 1,2,\ldots,n$$

$$\|\mathbf{X}\| = \left(\sum_{i=1}^{n} x^2\right)^{1/2}$$

$$\|\mathbf{A}\| = \max\left(\sum_{j=1}^{n} |a_{i,j}|, \; i = 1,2,\ldots,n\right)$$

$$\|\mathbf{A}\| = \max\left(\sum_{i=1}^{n} |a_{i,j}|, \; j = 1,2,\ldots,n\right)$$

$$\|\mathbf{A}\| = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2\right)^{1/2}$$

Las dos primeras, tanto para vectores como para matrices, se denominan **norma 1** y **norma infinito**. La tercera es la norma euclideana.

Ejemplo. Dada la siguiente matriz

$$\mathbf{A} = \begin{bmatrix} 5 & -3 & 2 \\ 4 & 8 & -4 \\ 2 & 6 & 1 \end{bmatrix}$$

Calcule la norma infinito (norma de fila).

Solución

Esta norma es el mayor valor de la suma de las magnitudes de los componentes de cada fila

Fila 1: |5| + |-3| + |2| = 10
Fila 2: |4| + |8| + |-4| = 16
Fila 3: |2| + |6| + |1| = 9

Por lo tanto, la norma por fila de la matriz es 16

4.4.2 Algunas propiedades de normas

Sea **A**: matriz de nxn componentes.

a) $\|\mathbf{A}\| \geq \mathbf{0}$

b) $\|\mathbf{kA}\| = |\mathbf{k}| \|\mathbf{A}\|$, $\mathbf{k} \in \Re$

c) $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$

d) $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$

e) $\mathbf{||(kA)^{-1}|| = ||1/k\ A^{-1}||}$, $\mathbf{k} \in \Re$

4.4.3 Número de condición

El número de condición de una matriz se usa para cuantificar su nivel de mal condicionamiento.

Definición: Número de condición

> Sea **AX = B** un sistema de ecuaciones lineales, entonces
> **cond(A) = || A || || A^{-1} ||** es el número de condición de la matriz **A**.

Cota para el número de condición:

$$\mathbf{cond(A) = ||\ A\ ||\ ||\ A^{-1}\ ||} \geq \mathbf{||\ A\ A^{-1}\ ||} = \mathbf{||\ I\ ||} = 1 \Rightarrow \mathbf{cond(A)} \geq 1$$

El número de condición no cambia si la matriz es multiplicada por alguna constante:

$$\mathbf{cond(kA) = ||kA||\ ||(kA)^{-1}|| = k\ ||A||\ ||1/k\ A^{-1}|| = ||A||\ ||A^{-1}||}$$

Ejemplo. Calcule la norma, determinante y número de condición de las matrices y analice los resultados

$$\mathbf{A} = \begin{bmatrix} \mathbf{0.010} & \mathbf{0.005} \\ \mathbf{0.025} & \mathbf{0.032} \end{bmatrix}; \quad \mathbf{B} = \mathbf{1000A} = \begin{bmatrix} \mathbf{10} & \mathbf{5} \\ \mathbf{25} & \mathbf{32} \end{bmatrix}$$

Solución

	A	**B**
Determinante	0.000195	195
Norma_1 de la matriz	0.0370	37
Norma_1 de la inversa	292.3077	0.2923
Número de condición	**10.8154**	**10.8154**

La matriz B es 1000 veces la matriz A. El determinante y la norma de ambas matrices y de su inversas son diferentes, pero el número de condición es igual.

Este resultado muestra que la norma no es suficiente para medir el nivel de mal condicionamiento de una matriz.

Si la matriz tiene filas “bastante dependientes” de otras filas, su determinante tomará un valor muy pequeño y su inversa tendrá valores muy grandes, siendo esto un indicio de que la matriz es mal condicionada o es “casi singular”. Este valor interviene en el número de condición de la matriz.

Por otra parte, si la matriz tiene valores muy pequeños, su determinante será muy pequeño y su inversa contendrá valores grandes aunque la matriz no sea mal condicionada.

Si el número de condición solo dependiera de la norma de la matriz inversa, esta norma tendría un valor grande en ambos casos. Por esto, y usando la propiedad anotada anteriormente, es necesario multiplicar la norma de la matriz inversa por la norma de la matriz original para que el número de condición sea grande únicamente si la matriz es mal condicionada.

Ejemplo. Calcule y analice el número de condición de la matriz del ejemplo inicial

$$\mathbf{A} = \begin{bmatrix} \mathbf{2.6} & \mathbf{0.3} & \mathbf{2.4} & \mathbf{6.2} \\ \mathbf{7.7} & \mathbf{0.4} & \mathbf{4.7} & \mathbf{1.4} \\ \mathbf{5.1} & \mathbf{9.9} & \mathbf{9.5} & \mathbf{1.5} \\ \mathbf{6.0} & \mathbf{7.0} & \mathbf{8.5} & \mathbf{4.8} \end{bmatrix}$$

$$\mathbf{A^{-1}} = \begin{bmatrix} \mathbf{-91.1455} & \mathbf{-20.3769} & \mathbf{-107.6454} & \mathbf{157.3120} \\ \mathbf{-107.8389} & \mathbf{-24.3503} & \mathbf{-127.2826} & \mathbf{186.1699} \\ \mathbf{164.4788} & \mathbf{37.0436} & \mathbf{194.3121} & \mathbf{-283.9787} \\ \mathbf{-20.0676} & \mathbf{-4.6161} & \mathbf{-23.9171} & \mathbf{34.9495} \end{bmatrix}$$

$$\text{cond}(\mathbf{A}) = \mathbf{\| A \| \| A^{-1} \| = 17879.09}$$

Es un valor muy alto, respecto al valor mínimo que es **1**

Una matriz puede considerarse mal condicionada si una ligera perturbación, error o cambio en la matriz de coeficientes produce un cambio muy significativo en el vector solución.

4.4.4 El número de condición y el error de redondeo

Dado un sistema de ecuaciones lineales $\mathbf{AX} = \mathbf{B}$ cuya solución existe y es $\mathbf{X}$

Suponer que debido a errores de medición, la matriz **A** de los coeficientes tiene un error $\mathbf{E}_A$
Sea $\overline{\mathbf{A}} = \mathbf{A} + \mathbf{E}_A$, la matriz con los errores de medición. Suponer que el vector **B** es exacto

Entonces, al resolver el sistema con la matriz $\overline{\mathbf{A}}$ se tendrá una solución $\overline{\mathbf{X}}$ diferente a la solución **X** del sistema con la matriz **A**. Esta solución $\overline{\mathbf{X}}$ satisface al sistema: $\overline{\mathbf{A}}\ \overline{\mathbf{X}} = \mathbf{B}$

Es importante determinar la magnitud de la diferencia entre ambas soluciones: $\mathbf{X} - \overline{\mathbf{X}}$

Sustituyendo $\overline{\mathbf{A}}\ \overline{\mathbf{X}} = \mathbf{B}$ en la solución del sistema original $\mathbf{AX} = \mathbf{B}$:

$$
\begin{aligned}
\mathbf{X} &= \mathbf{A}^{-1}\mathbf{B} \\
&= \mathbf{A}^{-1}(\overline{\mathbf{AX}}) \\
&= \mathbf{A}^{-1}(\mathbf{A} + \mathbf{E}_A)\overline{\mathbf{X}} \\
&= \mathbf{A}^{-1}\mathbf{A}\ \overline{\mathbf{X}} + \mathbf{A}^{-1}\mathbf{E}_A\overline{\mathbf{X}} \\
&= \mathbf{I}\ \overline{\mathbf{X}} + \mathbf{A}^{-1}\mathbf{E}_A\overline{\mathbf{X}} \\
&= \overline{\mathbf{X}} + \mathbf{A}^{-1}\mathbf{E}_A\overline{\mathbf{X}}
\end{aligned}
$$

$$
\Rightarrow \quad \mathbf{X} - \overline{\mathbf{X}} = \mathbf{A}^{-1}\mathbf{E}_A\overline{\mathbf{X}} \ \Rightarrow\ \left\|\mathbf{X} - \overline{\mathbf{X}}\right\| \le \left\|\mathbf{A}^{-1}\right\|\left\|\mathbf{E}_A\right\|\left\|\overline{\mathbf{X}}\right\| \quad \Rightarrow \quad \left\|\mathbf{E}_X\right\| \le \left\|\mathbf{A}^{-1}\right\|\left\|\mathbf{A}\right\|\frac{\left\|\mathbf{E}_A\right\|}{\left\|\mathbf{A}\right\|}\left\|\overline{\mathbf{X}}\right\|
$$

De donde se puede escribir, sustituyendo el número de condición de **A**:

Definición: Cota para el error relativo de la solución

$$
\frac{\|\mathbf{E}_X\|}{\|\overline{\mathbf{X}}\|} \le \mathbf{cond(A)}\frac{\|\mathbf{E}_A\|}{\|\mathbf{A}\|}
$$

$$
|\mathbf{e}_X| \le \mathbf{cond(A)}\,|(\mathbf{e}_A)|
$$

Cota para el error relativo de la solución

X es el vector solución calculado con la matriz inicial **A**
$\overline{\mathbf{X}}$ es el vector solución calculado con la matriz modificada $\overline{\mathbf{A}}$
$\mathbf{E}_A = \overline{\mathbf{A}} - \mathbf{A}$ es la matriz con las diferencias entre los componentes de las matrices.

$\mathbf{e}_X$ es el error relativo del vector solución
$\mathbf{e}_A$ es el error relativo de la matriz

La expresión establece que la magnitud del error relativo de la solución está relacionada con el error relativo de la matriz del sistema, ponderada por el número de condición. El número de condición es un factor que amplifica el error en la matriz **A** aumentando la dispersión y la incertidumbre de la solución calculada $\overline{\mathbf{X}}$

Ejemplo. Encuentre una cota para el error en la solución del ejemplo inicial

Matriz original $\mathbf{A} = \begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.0 & 7.0 & 8.5 & 4.8 \end{bmatrix}$

Matriz modificada $\bar{\mathbf{A}} = \begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.1 & 7.0 & 8.5 & 4.8 \end{bmatrix}$

Error en la matriz: $\mathbf{E_A} = \bar{\mathbf{A}} - \mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \end{bmatrix}$

Norma del error relativo de la matriz:

$$|e_A| = \frac{\| E_A \|}{\| A \|} = \frac{0.1}{26.3} = 0.0038 = 0.38\%$$

Número de condición:

$$\text{cond}(A) = 17879.09$$

Cota para el error relativo de la solución:

$$|e_x| \leq \text{cond}(A)\, |e_A| = 17879.09\ (0.0038) = 67.94 = 6794\%$$

Indica que la magnitud del error relativo de la solución puede variar hasta en **6794%**, por lo tanto no se puede confiar en ninguno de los dígitos de la respuesta calculada.

Ejemplo. Encuentre el error relativo de la solución en el ejemplo inicial y compare con el error relativo de la matriz de los coeficientes.

Sistema original:
$$\begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.0 & 7.0 & 8.5 & 4.8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 50.78 \\ 47.36 \\ 91.48 \\ 98.17 \end{bmatrix}$$
Solución: $\mathbf{X} = \begin{bmatrix} 2.5 \\ 3.2 \\ 4.1 \\ 5.4 \end{bmatrix}$

Sistema modificado:
$$\begin{bmatrix} 2.6 & 0.3 & 2.4 & 6.2 \\ 7.7 & 0.4 & 4.7 & 1.4 \\ 5.1 & 9.9 & 9.5 & 1.5 \\ 6.1 & 7.0 & 8.5 & 4.8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 50.78 \\ 47.36 \\ 91.48 \\ 98.17 \end{bmatrix}$$
Solución: $\bar{\mathbf{X}} = \begin{bmatrix} 0.1494 \\ 0.4182 \\ 8.3432 \\ 4.8778 \end{bmatrix}$

Error en la solución:
$$\mathbf{E_X} = \bar{\mathbf{X}} - \mathbf{X} = \begin{bmatrix} 0.1494 \\ 0.4182 \\ 8.3432 \\ 4.8778 \end{bmatrix} - \begin{bmatrix} 2.5 \\ 3.2 \\ 4.1 \\ 5.4 \end{bmatrix} = \begin{bmatrix} -2.3506 \\ -2.7818 \\ 4.2432 \\ -0.5222 \end{bmatrix}$$

Norma del error relativo de la solución:

$$|e_X| = \frac{\| E_X \|}{\| \bar{X} \|} = \frac{4.2432}{8.3432} = 0.5086 = 50.86\%$$

Norma del error relativo de la matriz:

$$|e_A| = \frac{\| E_A \|}{\| A \|} = \frac{0.1}{26.3} = 0.0038 = 0.38\%$$

La variación en el vector solución es muy superior a la variación de la matriz de coeficientes.

Se concluye que es un sistema mal condicionado.

Ejemplo. Una empresa compra tres materiales **A, B, C** en cantidades en kg. como se indica en el cuadro. Se dispone de dos facturas en las que consta el total pagado en dólares. Se desconoce el total pagado en la segunda factura:

Factura	A	B	C	Total
1	2	5	4	35
2	3	9	8	k
3	2	3	1	17

Solución

Sean x_1, x_2, x_3 variables que representan al precio unitario de cada producto. Entonces, se puede escribir:

$$\begin{aligned} 2x_1 + 5x_2 + 4x_3 &= 35 \\ 3x_1 + 9x_2 + 8x_3 &= k \\ 2x_1 + 3x_2 + x_3 &= 17 \end{aligned}$$

Con el método de **Gauss-Jordan** encuentre la solución en función de **k**

$$A|B = \begin{bmatrix} 2 & 5 & 4 & 35 \\ 3 & 9 & 8 & k \\ 2 & 3 & 1 & 17 \end{bmatrix} = \begin{bmatrix} 1 & 5/2 & 2 & 35/2 \\ 0 & 3/2 & 2 & k-105/2 \\ 0 & -2 & -3 & -18 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -4/3 & 105-5k/3 \\ 0 & 1 & 4/3 & 2k/3-35 \\ 0 & 0 & -1/3 & 4k/3-88 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 457-7k \\ 0 & 1 & 0 & 6k-387 \\ 0 & 0 & 1 & 264-4k \end{bmatrix}$$

$$X = \begin{bmatrix} 457-7k \\ 6k-387 \\ 264-4k \end{bmatrix}$$

Suponga que el valor pagado en la segunda factura es **65** dólares. Entonces la solución exacta

$$X = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

Para verificar que la solución es confiable, en la matriz de coeficientes se sustituye **5** por **5.1** y se obtiene nuevamente la solución con el método anterior con **k=65**.

$$A|B = \begin{bmatrix} 2 & 51/10 & 4 & 35 \\ 3 & 9 & 8 & 65 \\ 2 & 3 & 1 & 17 \end{bmatrix} = \begin{bmatrix} 1 & 51/20 & 2 & 35/2 \\ 0 & 27/20 & 2 & 25/2 \\ 0 & -21/10 & -3 & -18 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -16/9 & -55/9 \\ 0 & 1 & 40/27 & 250/27 \\ 0 & 0 & 1/9 & 13/9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 17 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & 13 \end{bmatrix}$$

$$X = \begin{bmatrix} 17 \\ -10 \\ 13 \end{bmatrix}$$

El error de **0.1** en un coeficiente distorsionó completamente la solución, por lo tanto la solución no es confiable.

Error relativo de la solución y error relativo de la matriz

$$|e_x| = \frac{\|X - \bar{X}\|}{\|X\|} = 3.75 = 375\%, \qquad |e_A| = \frac{\|A - \bar{A}\|}{\|A\|} = 0.005 = 0.5\%$$

0.5% de distorsión en la matriz produjo una distorsión de **375%** en la solución. Estos resultados confirman que el sistema es muy sensible a cambios o errores en los datos. Se concluye que es un sistema mal condicionado y no se puede tener confianza en ninguno de los dígitos de la solución calculada.

4.4.5 Funciones de Python para normas y número de condición

Cálculo de normas de vectores y matrices en Python

Sea **a** un vector o una matriz. El módulo linalg de la librería NumPy tiene varias definiciones para norma y número de condición. Se muestran algunas:

linalg.norm(a, 1)	para obtener la norma 1 (norma de columna)
linalg.norm(a, inf)	para obtener la norma infinito (norma de fila)
linalg.cond(a, 1)	número de condición con la norma 1
linalg.cond(a, inf)	número de condición con la norma infinito

Ejemplo. Calcule norma, inversa y número de condición de la matriz $A = \begin{bmatrix} 4 & 5 \\ 4.1 & 5 \end{bmatrix}$

Solución

Escribimos en la pantalla interacftiva de Python:

```
>>> from numpy import*
>>> a=array([[4,5],[4.1,5]],float)
>>> n=linalg.norm(a,inf)
>>> print(n)
9.0999999999999996
>>> d=linalg.inv(a)
>>> print(d)
[[-10. ,  10. ],
 [  8.2,  -8. ]]
>>> c=linalg.cond(a,inf)
>>> print(c)
182.00000000000
```

4.5 Sistemas singulares

En esta sección se describe un método directo para intentar resolver un sistema lineal propuesto de **n** ecuaciones con **m** variables, **n<m**. Estos sistemas también se obtienen como resultado de la reducción de un sistema dado originalmente con **m** ecuaciones y **m** variables en los que algunas ecuaciones no son independientes. En ambos casos la matriz de coeficientes contendrá una o más filas nulas y por lo tanto **no tienen inversa** y se dice que la **matriz es singular**, en este caso también diremos que el **sistema es singular**. Estos sistemas no admiten una solución única.

Sin embargo, si el sistema es un modelo que representa algún problema de interés, es importante detectar si el sistema es incompatible para el cual no existe solución, o se trata de un sistema incompleto para el cual existe infinidad de soluciones. Más aún, es útil reducirlo a una forma en la cual se facilite determinar las variables libres, a las que se pueden asignar valores arbitrarios para analizar las soluciones resultantes en términos de éstas variables y su relación con el problema.

Para facilitar el análisis de estos sistemas, es conveniente convertirlo en una forma más simple. La estrategia que usaremos es llevarlo a la forma de la matriz identidad hasta donde sea posible

4.5.1 Formulación matemática y algoritmo

Se desea resolver el sistema de **n** ecuaciones lineales con **m** variables, siendo **n≤m**

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_1 + \ldots + a_{1,m}x_m &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_1 + \ldots + a_{2,m}x_m &= b_2 \\ &\ldots \\ a_{n,1}x_1 + a_{n,2}x_1 + \ldots + a_{n,m}x_m &= b_n \end{aligned}$$

En donde

$a_{i,j} \in \Re$: Coeficientes

$b_i \in \Re$: Constantes

$x_i \in \Re$: Variables cuyo valor debe determinarse

En notación matricial:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,m} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,m} \\ \ldots & & & \ldots \\ a_{n,1} & a_{n,1} & \ldots & a_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix}$$

Simbólicamente: **AX = B**, en donde

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \cdots & & & \cdots \\ a_{n,1} & a_{n,1} & \cdots & a_{n,m} \end{bmatrix}; \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix}; \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}$$

Para unificar la descripción algorítmica, es conveniente aumentar la matriz **A** con el vector **B** pues en ambos deben realizarse simultáneamente las mismas operaciones:

$$A \mid B = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} & a_{1,m+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} & a_{2,m+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} & a_{n,m+1} \end{bmatrix}$$

Siendo $a_{i,m+1} = b_i$, $i = 1, 2, 3, \ldots, n$

El procedimiento consiste en transformar la matriz aumentada, de manera similar al método de Gauss-Jordan. El objetivo es reducir la matriz aumentada a una forma escalonada con **1's** en la diagonal mediante intercambios de filas, hasta donde sea posible hacerlo.

Si **n<m** la matriz transformada finalmente tendrá la siguiente forma

$$A \mid B = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} & a_{1,m+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} & a_{2,m+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} & a_{n,m+1} \end{bmatrix} \rightarrow \ldots \rightarrow \left[\begin{array}{cccccccc|c} 1 & 0 & \cdots & 0 & a_{1,n+1} & \cdots & a_{1,m} & a_{1,m+1} \\ 0 & 1 & \cdots & 0 & a_{2,n+1} & \cdots & a_{2,m} & a_{2,m+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & a_{n,n+1} & \cdots & a_{n,m} & a_{n,m+1} \end{array}\right]$$

En el sistema resultante, las variables $x_{n+1}, x_{n+2}, \ldots, x_m$ se denominan **variables libres** y pueden tomar valores arbitrarios, normalmente asociados al problema que se analiza, mientras que las otras variables $x_1, x_2, \ldots, x_n$ pueden tomar valores dependientes de las variables libres. Los valores $a_{i,j}$ que aparecen en la matriz reducida, son los valores resultantes de las transformaciones aplicadas. La forma final facilita asignar valores a estas variables.

En cada etapa, primero se hará que el elemento en la diagonal tome el valor 1. Luego se hará que los demás elementos de la columna correspondiente, tomen el valor 0. Realizando previamente intercambios de filas para colocar como divisor el elemento de mayor magnitud. Esta estrategia se denomina pivoteo parcial y se usa para determinar si el sistema es singular.

La formulación es similar al método de Gauss-Jordan descrito en una sección anterior, pero el algoritmo incluye el registro de las variables libres que se detectan como variables libres.

Algoritmo: Gauss-Jordan para sistemas singulares

Entra
a: matriz aumentada del sistema de n ecuaciones lineales
v: vector de variables libres (inicialmente nulo)
Sale
x: Solución calculada o un vector nulo si la solución no es única
a: Matriz reducida a la forma diagonal

Para e = 1, 2, . . ., n
Elegir el valor de mayor magnitud de la columna e en las filas e, e+1, ..., n
Si este valor es cero
agregar e al vector v de las variable libres
avanzar a la siguiente etapa e
Sino (continuar con la transformación matricial)
$t \leftarrow a_{e,e}$
Para j=e, e+1, ..., n+1
$a_{e,j} \leftarrow a_{e,j} / t$ **Normalizar la fila e** ($a_{e,e} \neq 0$)
Fin
Para i=1, 2, …, n; i ≠ e
$t \leftarrow a_{i,e}$
Para j=e, e+1, ..., n+1
$a_{i,j} \leftarrow a_{i,j} - t\ a_{e,j}$ **Reducir las otras filas**
Fin
Fin
Fin
Fin
Si el vector v no contiene variables libres
El vector solución x es la última columna de la matriz a
Sino
Entregar un vector x nulo
Entregar la matriz a reducida
Fin

Ejemplo. Una empresa produce cuatro productos: **P1, P2, P3, P4** usando tres tipos de materiales **M1, M2, M3**. Cada Kg. de producto requiere la siguiente cantidad de cada material en Kg.:

	P1	**P2**	**P3**	**P4**
M1	0.2	0.5	0.4	0.2
M2	0.3	0	0.5	0.6
M3	0.5	0.5	0.1	0.2

La cantidad disponible de cada material es: 10, 12, 15 Kg. respectivamente, los cuales deben usarse completamente. Se quiere analizar alguna estrategia de producción factible.

Solución

Sean x_1, x_2, x_3, x_4 cantidades en Kg. producidas de **P1, P2, P3, P4**, respectivamente. Se obtienen las ecuaciones:

$$\begin{aligned} 0.2x_1 + 0.5x_2 + 0.4x_3 + 0.2x_4 &= 10 \\ 0.3x_1 \quad\quad + 0.5x_3 + 0.6x_4 &= 12 \\ 0.5x_1 + 0.5x_2 + 0.1x_3 + 0.2x_4 &= 15 \end{aligned}$$

Si fuesen desigualdades tipo menor o igual, este problema se puede interpretar como un problema de búsqueda de la mejor solución y cae en otro ámbito de la matemática (programación lineal)

Es un sistema de tres ecuaciones y cuatro variables. En notación matricial

$$\begin{bmatrix} 0.2 & 0.5 & 0.4 & 0.2 \\ 0.3 & 0 & 0.5 & 0.6 \\ 0.5 & 0.5 & 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 15 \end{bmatrix}$$

Reducción con el método de Gauss-Jordan usando pivoteo

$$\begin{bmatrix} 1.00 & 1.00 & 0.20 & 0.40 & 30.00 \\ 0 & -0.30 & 0.44 & 0.48 & 3.00 \\ 0 & 0.30 & 0.36 & 0.12 & 4.00 \end{bmatrix}$$

$$\begin{bmatrix} 1.00 & 0 & 1.66 & 2.00 & 40.00 \\ 0 & 1.00 & -1.46 & -1.60 & -10.00 \\ 0 & 0 & 0.80 & 0.60 & 7.00 \end{bmatrix}$$

$$\begin{bmatrix} 1.00 & 0 & 0 & 0.75 & 25.41 \\ 0 & 1.00 & 0 & -0.50 & 2.83 \\ 0 & 0 & 1.00 & 0.75 & 8.75 \end{bmatrix}$$

Sistema equivalente reducido:

$$\begin{aligned} x_1 \quad\quad\quad\quad + 0.75x_4 &= 25.41 \\ x_2 \quad\quad - 0.50x_4 &= 2.83 \\ x_3 + 0.75x_4 &= 8.75 \end{aligned}$$

La variable x_4 queda como variable libre:

Sea $x_4 = t, t \geq 0, t \in \Re$ (variable libre)

Conjunto solución: $\mathbf{X} = [25.41 - 0.75\,t, 2.83 + 0.50\,t, 8.75 - 0.75\,t, t]^T$

Dependiendo del problema se pueden afinar los rangos para las variables. En este ejemplo, las variables no pueden tomar valores negativos:

Rango para la variable libre:

$x_3 = 8.75 - 0.75\,t \geq 0 \Rightarrow t \leq 11.66$

$x_2 = 2.83 + 0.50\,t \geq 0 \Rightarrow t \geq 0$

$x_1 = 25.41 - 0.75\,t \geq 0 \Rightarrow t \leq 33.88$

Se concluye que el rango factible para x_4 es: $0 \leq t \leq 11.66$

Rango para las otras variables

$0 \leq t \leq 11.66$

$x_3 = 8.75 - 0.75\,t \Rightarrow 0 \leq x_3 \leq 8.75$

$x_2 = 2.83 + 0.50\,t \Rightarrow 2.83 \leq x_2 \leq 8.66$

$x_1 = 25.41 - 0.75\,t \Rightarrow 16.66 \leq x_1 \leq 25.41$

Esta información puede ser útil para decidir la cantidad que debe producirse de cada artículo usando todos los recursos disponibles y usando cualquier artículo como referencia.

Ejemplo. Si se decide producir **10** Kg del producto P_4, entonces para que no sobren materiales, la producción será:

$x_4 = 10 \Rightarrow t = x_4 = 10,\ x_3 = 1.25,\ x_2 = 7.83,\ x_1 = 17.91$

Ejemplo. Si se decide producir **20** Kg del producto P_1, entonces para que no sobren materiales, la producción será:

$x_1 = 20 \Rightarrow t = x_4 = 7.21,\ x_3 = 3.34,\ x_2 = 6.43$

4.5.2 Instrumentación computacional para sistemas singulares

La instrumentación se realiza mediante una función con el nombre **slin**. La matriz es estandarizada dividiendo por el elemento de mayor magnitud para reducir el error de truncamiento y detectar en forma consistente si el sistema es singular.

Por los errores de redondeo se considerará que el elemento de la diagonal es nulo si su valor tiene una magnitud menor que 10^{-10}.

Parámetros de entrada

a: matriz de coeficientes

b: vector de constantes

Parámetros de salida

x: vector solución

c: matriz de coeficientes reducida a la forma escalonada

Uso de **slin**

[x, c] = slin(a, b) (se pueden puede omitir los corchetes)

```
#Sistemas singulares
import numpy as np
def slin(a,b):
    [n,m]=np.shape(a)                          #Dimensiones de A
    c=np.concatenate([a,b],axis=1)             #Matriz aumentada
    z=np.amax(abs(a))
    v=[]
    if m>n:
        for i in range(n,m):                   #Variables libres
            v=v+[i]
    if n>m:
        return [[],[]]
    for i in range(n):                         #Estandarizar sistema
        for j in range(m+1):
            c[i,j]=c[i,j]/z

    for e in range(n):
        p=e
        for i in range(e+1,n):                 #Pivoteo
            if abs(c[i,e])>abs(c[p,e]):
                p=i

        for j in range(e,m+1):                  #Intercambiar filas
            t=c[e,j]
            c[e,j]=c[p,j]
            c[p,j]=t
        t=c[e][e]
        if abs(t)<1e-10:                       #Detectar variable libre
            v=v+[e]
        else:
            for j in range(e,m+1):             #Normalizar fila e
                c[e,j]=c[e,j]/t
            for i in range(n):                 #Reducir otras filas
                if i!=e:
                    t=c[i,e]
                    for j in range(e,m+1):
                        c[i,j]=c[i,j]-t*c[e,j]
    x=[]
    if v==[] and n==m:                         #Solución
        x=np.zeros([n,1],float)
        for i in range(n):
            x[i]=c[i,n]
    return [x,c]
```

Ejemplo. El ejemplo anterior usando **slin**

```
>>> from numpy import*
>>> from slin import*
>>> a=array([[0.2,0.5,0.4,0.2],[0.3,0.0,0.5,0.6],[0.5,0.5,0.1,0.2]],float)
>>> b=array([[10],[12],[15]],float)
>>> x,c=slin(a,b)
>>> x
[]
>>> print(c)
[[ 1.  0.   0.   0.75   25.41666667]
 [ 0.  1.   0.  -0.5     2.83333333]
 [ 0.  0.   1.   0.75    8.75      ]]
```

En los siguientes ejemplos se utiliza una notación informal para identificar cada tipo de sistema que se resuelve. Los resultados obtenidos deben interpretarse según lo descrito en la instrumentación computacional de la función **slin.**

Sistema completo: Tiene **n** ecuaciones y **n** variables
Sistema incompleto: Tiene **n** ecuaciones y **m** variables, **n<m**. Puede ser dado inicialmente o puede ser resultado del proceso de transformación matricial en el que algunas ecuaciones desaparecen pues son linealmente dependientes.

Sistema consistente: Tiene una solución única
Sistema redundante: Tiene variables libres y por lo tanto, infinidad de soluciones
Sistema incompatible: Contiene ecuaciones incompatibles. La transformación matricial reduce el sistema a uno conteniendo proposiciones falsas

Las variables libres se reconocen porque no están diagonalizadas, es decir que no contienen 1 en la diagonal principal de la matriz transformada.

1) Sistema consistente

$$
\begin{aligned}
x_1 \quad &+ 2x_3 + 4x_4 = 1 \\
x_2 \; &+ 2x_3 \quad\;\; = 0 \\
x_1 + 2x_2 \; &+ x_3 \quad\;\; = 0 \\
x_1 + x_2 \quad &+ 2x_4 = 2
\end{aligned}
$$

```
>>> from numpy import*
>>> from slin import*
>>> a=array([[1,0,2,4],[0,1,2,0],[1,2,1,0],[1,1,0,2]],float)
>>> b=array([[1],[0],[0],[2]],float)
>>> [x,c]=slin(a,b)
```

```
>>> print(x)
[[-3. ]
 [ 2. ]
 [-1. ]
 [ 1.5]]
>>> print(c)
[[ 1.   0.   0.   0.  -3. ]
 [ 0.   1.   0.   0.   2. ]
 [ 0.   0.   1.   0.  -1. ]
 [ 0.   0.   0.   1.   1.5]]
```

2) Sistema completo reducido a incompleto redundante

```
>>> from numpy import*
>>> from slin import*
>>> a=array([[1,1,2,2],[1,2,2,4],[2,4,2,4],[1,3,0,2]],float)
>>> b=array([[1],[2],[2],[1]],float)
>>> [x,c]=slin(a,b)
>>> print(x)
[]
>>> print(c)
[[ 1.  0.  0. -4. -2.]
 [ 0.  1.  0.  2.  1.]
 [ 0.  0.  1.  2.  1.]
 [ 0.  0.  0.  0.  0.]]
```

Se obtiene un sistema equivalente. Las soluciones se asignan mediante la variable libre x_4 :

$$\begin{array}{rl} x_1 - 4x_4 = -2 & \\ x_2 - 2x_4 = 1 & \Rightarrow \\ x_3 + 2x_4 = 1 & \end{array} \qquad \begin{array}{l} x_1 = 4x_4 - 2 \\ x_2 = 2x_4 + 1 \\ x_3 = -2x_4 + 1 \end{array}$$

Sea $x_4 = t, t \in \mathbf{R}$, entonces el conjunto solución es **{4t-2, 2t+1, -2t +1}**

3) Sistema incompleto

```
>>> from numpy import*
>>> from slin import*
>>> a=array([[1,0,2,4],[4,1,2,4],[2,4,5,2]],float)
>>> b=array([[1],[0],[2]],float)
>>> [x,c]=slin(a,b)
>>> print(x)
[]
>>> print(c)
[[ 1.    0.    0.    0.64  0.09]
 [ 0.    1.    0.   -1.92 -1.02]
 [ 0.    0.    1.    1.68  0.83]]
```

Se obtiene un sistema equivalente. Las soluciones se asignan mediante la variable libre x_4:

$$\begin{aligned} x_1 &+ 0.64x_4 = -0.28 \\ x_2 &- 1.92x_4 = -0.16 \\ x_3 &+ 1.68x_4 = 0.64 \end{aligned}$$

4) Sistema completo reducido a incompleto incompatible

```
>>> from numpy import*
>>> from slin import*
>>> a=array([[1,1,2,2],[1,2,2,4],[2,4,2,4],[1,3,0,2]],float)
>>> b=array([[1],[2],[2],[4]],float)
>>> [x,c]=slin(a,b)
>>> print(x)
[]
>>> print(c)
[[ 1.    0.    0.   -4.   -2.  ]
 [ 0.    1.    0.    2.    1.  ]
 [ 0.    0.    1.    2.    1.  ]
 [ 0.    0.    0.    0.    0.75]]
```

Sistema equivalente:

$$\begin{aligned} x_1 &- 4x_4 = -2 \\ x_2 &- 2x_4 = 1 \\ x_3 &+ 2x_4 = 1 \\ 0x_4 &= 3 \end{aligned}$$

Sistema incompatible

5. Resolver el sistema incompleto

$$\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 2 \\ 30 \\ -4 \\ 10 \\ 50 \end{bmatrix}$$

```
>>> from numpy import*
>>> from slin import*
>>> a=array([[1,0,-1,0,0,0],[0,0,0,1,0,-1],[0,0,1,0,-1,0],
      [0,1,-1,0,0,0],[0,0,0,1,0,0]],float)
>>> b=array([[2],[30],[-4],[10],[50]],float)
>>> [x,c]=slin(a,b)
>>> print(x)
[]
```

```
>>> print(c)
[[  1.   0.   0.   0.  -1.   0.  -2.]
 [  0.   1.   0.   0.  -1.   0.   6.]
 [  0.   0.   1.   0.  -1.   0.  -4.]
 [  0.   0.   0.   1.   0.  -1.  30.]
 [  0.   0.   0.   0.   0.   1.  20.]]
```

La variable libre es x_5

Del sistema reducido se puede obtener:

$x_6 = 20$

$x_4 = 50$

$x_5 = t, \quad t \geq 0$ (variable libre)

$x_3 = -4 + t$

$x_2 = 6 + t$

$x_1 = -2 + t$

Los resultados obtenidos también establecen que $t \geq 4$

4.6 Sistemas tridiagonales

En un sistema tridiagonal la matriz de los coeficientes contiene todos sus componentes iguales a cero excepto en las tres diagonales principales. Estos sistemas se presentan en la aplicación de cierto tipo de métodos numéricos como el caso de los trazadores cúbicos y en la solución de ecuaciones diferenciales con diferencias finitas.

Se puede diseñar un método directo para resolver un sistema tridiagonal con eficiencia de primer orden: **T(n)=O(n)** lo cual representa una enorme mejora respecto a los métodos directos generales para resolver sistemas de ecuaciones lineales, cuya eficiencia es **$T(n)=O(n^3)$**.

4.6.1 Formulación matemática y algoritmo

Un sistema tridiagonal de **n** ecuaciones expresado en notación matricial:

$$\begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & c_3 & \\ & & \dots & \dots & \dots \\ & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_n \end{bmatrix}$$

Para obtener la formulación se puede considerar únicamente un sistema de tres ecuaciones para luego extenderla al caso general. Las transformaciones son aplicadas a la matriz aumentada:

$$\begin{bmatrix} b_1 & c_1 & & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ & a_3 & b_3 & d_3 \end{bmatrix}$$

1) Sea $w_1 = b_1$. Dividir la primera fila para w_1

$$\begin{bmatrix} 1 & \frac{c_1}{w_1} & & \frac{d_1}{w_1} \\ a_2 & b_2 & c_2 & d_2 \\ & a_3 & b_3 & d_3 \end{bmatrix}$$

2) Sea $g_1 = \frac{d_1}{w_1}$. Restar de la segunda fila, la primera multiplicada por a_2

$$\begin{bmatrix} 1 & \frac{c_1}{w_1} & & g_1 \\ 0 & b_2 - a_2\frac{c_1}{w_1} & c_2 & d_2 - a_2 g_1 \\ & a_3 & b_3 & d_3 \end{bmatrix}$$

3) Sea $w_2 = b_2 - a_2 \frac{c_1}{w_1}$. Dividir la segunda fila para w_2

$$\begin{bmatrix} 1 & \frac{c_1}{w_1} & & g_1 \\ 0 & 1 & \frac{c_2}{w_2} & \frac{d_2 - a_2 g_1}{w_2} \\ & a_3 & b_3 & d_3 \end{bmatrix}$$

4) Sea $g_2 = \frac{d_2 - a_2 g_1}{w_2}$. Restar de la tercera fila, la segunda multiplicada por a_3

$$\begin{bmatrix} 1 & \frac{c_1}{w_1} & & g_1 \\ 0 & 1 & \frac{c_2}{w_2} & g_2 \\ & 0 & b_3 - a_3 \frac{c_2}{w_2} & d_3 - a_3 g_2 \end{bmatrix}$$

5) Sea $w_3 = b_3 - a_3 \frac{c_2}{w_2}$. Dividir la tercera fila para w_3

$$\begin{bmatrix} 1 & \frac{c_1}{w_1} & & g_1 \\ 0 & 1 & \frac{c_2}{w_2} & g_2 \\ & 0 & 1 & \frac{d_3 - a_3 g_2}{w_3} \end{bmatrix}$$

6) Sea $g_3 = \frac{d_3 - a_3 g_2}{w_3}$. Finalmente se obtiene:

$$\begin{bmatrix} 1 & \frac{c_1}{w_1} & & g_1 \\ 0 & 1 & \frac{c_2}{w_2} & g_2 \\ & 0 & 1 & g_3 \end{bmatrix}$$

De donde se puede hallar directamente la solución

$$x_3 = g_3$$

$$x_2 = g_2 - \frac{c_2}{w_2} x_3$$

$$x_1 = g_1 - \frac{c_1}{w_1} x_2$$

La formulación se extiende al caso general [(2)]

Transformación matricial de un sistema tridiagonal de **n** ecuaciones lineales

$$w_1 = b_1$$

$$g_1 = \frac{d_1}{w_1}$$

$$w_i = b_i - \frac{a_i c_{i-1}}{w_{i-1}}, \qquad i = 2,3,\ldots,n$$

$$g_i = \frac{d_i - a_i g_{i-1}}{w_i}, \qquad i = 2,3,\ldots,n$$

Obtención de la solución

$$x_n = g_n$$

$$x_i = g_i - \frac{c_i x_{i+1}}{w_i}, \qquad i = n-1,\ n-2,\ldots,2,1$$

[(2)] Algoritmo de Thomas

El algoritmo incluye un ciclo dependiente del tamaño del problema **n** para reducir la matriz y otro ciclo separado para obtener la solución, ambos dependientes de **n**. Por lo tanto este algoritmo tiene eficiencia **T(n)=O(n).**

4.6.2 Instrumentación computacional del Método de Thomas

Con la formulación anterior se escribe una función para resolver un sistema tridiagonal de **n** ecuaciones lineales. La función recibe los coeficientes y las constantes en los vectores **a, b, c, d**. La solución es entregada en el vector **x**

```
def tridiagonal(a, b, c, d):
    n=len(d)
    w=[b[0]]
    g=[d[0]/w[0]]
    for i in range(1,n):
        w=w+[b[i]-a[i]*c[i-1]/w[i-1]]
        g=g+[(d[i]-a[i]*g[i-1])/w[i]]
    x=[]
    for i in range(n):
        x=x+[0]
    x[n-1]=g[n-1]
    for i in range(n-2,-1,-1):
        t=x[i+1]
        x[i]=g[i]-c[i]*t/w[i]
    return x
```

Ejemplo. Resuelva el siguiente sistema tridiagonal de ecuaciones lineales usando la función anterior en la ventana interactiva de Python

$$\begin{bmatrix} 7 & 5 & & \\ 2 & -8 & 1 & \\ & 6 & 4 & 3 \\ & & 9 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \\ 7 \\ 8 \end{bmatrix}$$

```
>>> from tridiagonal import*
>>> a=[0,2,6,9]                          Vectores con las diagonales (completas)
>>> b=[7,-8,4,8]
>>> c=[5,1,3,0]
>>> d=[6,5,7,8]
>>> x=tridiagonal(a,b,c,d)
>>> print(x)
[[ 0.74024024]
 [ 0.16366366]
 [ 4.82882883]
 [-4.43243243]]
```

5 Métodos iterativos para resolver sistemas de ecuaciones lineales

La resolución de sistemas de ecuaciones lineales también puede hacerse con fórmulas iterativas que permiten acercarse a la respuesta mediante aproximaciones sucesivas, sin embargo desde el punto de vista práctico es preferible usar métodos directos que con el soporte computacional actual resuelven grandes sistemas en forma eficiente y con mucha precisión, a diferencia de los sistemas de ecuaciones no-lineales cuya solución no se puede obtener mediante métodos directos.

Las fórmulas iterativas no siempre convergen. El análisis de convergencia puede ser complicado, la eficiencia es de primer orden y se requiere elegir algún vector inicial para comenzar el proceso iterativo. En la época actual el estudio de estos métodos iterativos se puede considerar principalmente como de interés teórico matemático, excepto para resolver grandes sistemas de ecuaciones lineales con matrices esparcidas y cuya convergencia se puede determinar.

Para definir un método iterativo, se expresa el sistema **AX = B** en la forma **X = G(X)** con el mismo fundamento descrito en el método del Punto Fijo para resolver ecuaciones no lineales.

5.1 Método de Jacobi

5.1.1 Formulación matemática

Dado un sistema de ecuaciones lineales

$$a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = b_2$$

. . . .

$$a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n = b_n$$

Expresado abreviadamente en notación matricial: **AX = B**

Un procedimientox simple para obtener la forma **X = G(X)** consiste en re-escribir el sistema despejando de la ecuación **i** la variable x_i a condición de que $a_{i,i}$ sea diferente de **0:**

$$x_1 = 1/a_{1,1}\,(b_1 - a_{1,2}x_2 - a_{1,3}x_3 - \ldots - a_{1,n}x_n)$$

$$x_2 = 1/a_{2,2}\,(b_2 - a_{2,1}x_1 - a_{2,3}x_3 - \ldots - a_{2,n}x_n)$$

. . . .

$$x_n = 1/a_{n,n}\,(b_n - a_{n,1}x_1 - a_{n,2}x_2 - \ldots - a_{n,n-1}x_{n-1})$$

El cual puede escribirse con la notación de sumatoria:

$$x_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j - \sum_{j=i+1}^{n} a_{i,j}x_j\right) = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{i,j}x_j\right); \quad i = 1, 2, \ldots, n;$$

El sistema está en la forma **X = G(X)** la cual sugiere su uso iterativo.

Utilizamos un índice arriba para indicar iteración:

$$X^{(k+1)} = G(X^{(k)}),\ k=0, 1, 2, \ldots.$$ (iteraciones)

Fórmula iterativa de Jacobi:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{i,j}x_j^{(k)}\right);\quad i = 1, 2, \ldots, n;\ k = 0, 1, 2, \ldots$$

$X^{(0)}$ es el vector inicial. A partir de este vector se obtienen los vectores $X^{(1)}, X^{(2)}, \ldots$

Si el método converge, entonces $X^{(k)}$ tiende a la solución X a medida que **k** crece:

$$X^{(k)} \underset{k\to\infty}{\to} X$$

Ejemplo. Dadas las ecuaciones:

$$5x_1 - 3x_2 + x_3 = 5$$
$$2x_1 + 4x_2 - x_3 = 6$$
$$2x_1 - 3x_2 + 8x_3 = 4$$

a) Formule un sistema iterativo con el método de Jacobi
b) Realice dos iteraciones, comenzando con los valores iniciales: $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 1$

Solución

$$x_1^{(k+1)} = 1/5\,(5 + 3x_2^{(k)} - x_3^{(k)})$$
$$x_2^{(k+1)} = 1/4\,(6 - 2x_1^{(k)} + x_3^{(k)})$$
$$x_3^{(k+1)} = 1/8\,(4 - 2x_1^{(k)} + 3x_2^{(k)}) \qquad k = 0, 1, 2, \ldots$$

k=0:
$$x_1^{(1)} = 1/5\,(5 + 3x_2^{(0)} - x_3^{(0)}) = 1/5\,(5 + 3(1) - (1)) = 1/5\,(7) = 1.4$$
$$x_2^{(1)} = 1/4\,(6 - 2x_1^{(0)} + x_3^{(0)}) = 1/4\,(6 - 2(1) + (1)) = 1/4\,(5) = 1.25$$
$$x_3^{(1)} = 1/8\,(4 - 2x_1^{(0)} + 3x_2^{(0)}) = 1/8\,(4 - 2(1) + 3(1)) = 1/8\,(5) = 0.625$$

k=1:
$$x_1^{(2)} = 1/5\,(5 + 3x_2^{(1)} - x_3^{(1)}) = 1/5\,(5 + 3(1.25) - (0.625)) = 1.6250$$
$$x_2^{(2)} = 1/4\,(6 - 2x_1^{(1)} + x_3^{(1)}) = 1/4\,(6 - 2(1.4) + (0.625)) = 0.9563$$
$$x_3^{(2)} = 1/8\,(4 - 2x_1^{(1)} + 3x_2^{(1)}) = 1/8\,(4 - 2(1.4) + 3(1.25)) = 0.6188$$

5.1.2 Manejo computacional de la fórmula de Jacobi

La siguiente función en Python recibe un vector **X** y entrega un nuevo vector **X** calculado con la fórmula iterativa

```
def jacobi(a,b,x):
    n=len(x)
    t=x.copy()
    for i in range(n):
        s=0
        for j in range(n):
            if i!=j:
                s=s+a[i,j]*t[j]
        x[i]=(b[i]-s)/a[i,i]
    return x
```

Nota: La función **copy()** asigna a **t** una copia de **x** en celdas diferentes

Uso de la función Jacobi para el ejemplo anterior:

```
>>> from numpy import*
>>> from jacobi import*
>>> a=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> b=array([[5],[6],[4]],float)
>>> x= array([[1],[1],[1]],float)          Vector inicial
>>> x=jacobi(a,b,x);print(x)               Repetir y observar la convergencia
[[ 1.625  ]
 [ 0.95625]
 [ 0.61875]]
>>> x=jacobi(a,b,x);print(x)
[[ 1.45      ]
 [ 0.8421875 ]
 [ 0.45234375]]
>>> x=jacobi(a,b,x);print(x)
[[ 1.41484375]
 [ 0.88808594]
 [ 0.45332031]]
>>> x=jacobi(a,b,x);print(x)
[[ 1.4421875 ]
 [ 0.9059082 ]
 [ 0.47932129]]
>>> x=jacobi(a,b,x);print(x)
[[ 1.44768066]
 [ 0.89873657]
 [ 0.4791687 ]]
```

```
>>> x=jacobi(a,b,x);print(x)
[[ 1.4434082 ]
 [ 0.89595184]
 [ 0.47510605]]
>>> x=jacobi(a,b,x);print(x)
[[ 1.4425499 ]
 [ 0.89707241]
 [ 0.47512989]]
```

5.1.3 Algoritmo de Jacobi

En este algoritmo se incluye el criterio de convergencia y un conteo de iteraciones. Si no converge entrega un vector nulo

Algoritmo: Jacobi
Entra
a: matriz de coeficientes,
b: vector de constantes del sistema de **n** ecuaciones lineales
e: estimación del error para la solución,
m: máximo de iteraciones permitidas
x: vector inicial para la solución, **k** es el conteo de iteraciones
Sale
x: vector solución
t ← x
Para k = 1, 2, . . ., m
Calcular el vector **x** con la fórmula de Jacobi
Si || **x – t** || < **e**
x es el vector solución con error **e**
Terminar
sino
t ← x
fin
fin

5.1.4 Instrumentación computacional del método de Jacobi

La siguiente función en Python recibe la matriz de coeficientes **a** y el vector de constantes **b** de un sistema lineal. Adicionalmente recibe un vector inicial **x**, la estimación del error **e** y el máximo de iteraciones permitidas **m**. Esta función utiliza dentro de un ciclo, la función **jacobi** definida anteriormente. Entrega el vector **x** calculado y el número de iteraciones realizadas **k**. Si el método no converge, **x** contendrá un vector nulo.

```
from jacobi import*
import numpy as np
def jacobim(a,b,x,e,m):
    n=len(x)
    t=x.copy()
    for k in range(m):
        x=jacobi(a,b,x)
        d=np.linalg.norm(array(x)-array(t),inf)
        if d<e:
            return [x,k]
        else:
            t=x.copy()
    return [[],m]
```

Nota: La función **copy()** asigna a **t** una copia de **x** en celdas diferentes

Ejemplo. Use la función **jacobim** para encontrar el vector solución del ejemplo anterior con un error de **0.0001** Determine cuántas iteraciones se realizaron. Comenzar con el vector inicial **x=[1; 1; 1]**

```
>>> from numpy import*
>>> from jacobim import*
>>> a=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> b=array([[5],[6],[4]],float)
>>> x=array([[1],[1],[1]],float)
>>> [x,k]=jacobim(a,b,x,0.0001,20)
>>> print(x)
[[ 1.44322632]
 [ 0.89729181]
 [ 0.47566235]]
>>> print(k)
11
```

5.1.5 Forma matricial del método de Jacobi

Dado el sistema de ecuaciones lineales

$$\mathbf{AX = B}$$

La matriz **A** se reescribe como la suma de tres matrices:

$$\mathbf{A = L + D + U}$$

D es una matriz diagonal con los elementos de la diagonal principal de **A**

L es una matriz triangular inferior con ceros en la diagonal principal. Debajo de la diagonal se copian los componentes respectivos de la matriz **A**

U es una matriz triangular superior con ceros en la diagonal principal. Arriba de la diagonal se copian los componentes respectivos de la matriz **A**

En forma desarrollada:

$$\mathbf{A}=\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} = \mathbf{L+D+U} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ a_{2,1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & 0 \end{bmatrix} + \begin{bmatrix} a_{1,1} & 0 & 0 & 0 \\ 0 & a_{2,2} & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & a_{n,n} \end{bmatrix} + \begin{bmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ 0 & 0 & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Sustituyendo en la ecuación:

$$\mathbf{(L + D + U)X = B}$$

$\mathbf{LX + DX + UX = B}$ Despejar **X** de la matriz diagonal

$\mathbf{X = D^{-1}B - D^{-1}(L + U)X}$ Siempre que $\mathbf{D^{-1}}$ exista

Ecuación matricial del método de Jacobi en formato del Punto Fijo **X = G(X)**

En donde

$$\mathbf{D^{-1}} = \left[\frac{1}{a_{i,i}}\right]_{nxn}$$

Se puede definir la fórmula recurrente matricial iterativa de Jacobi:

$$\mathbf{X^{(k+1)} = D^{-1}B - D^{-1}(L+U)X^{(k)}, \quad k = 0,1,2,...} \qquad \text{(iteraciones)}$$

Fórmula que tiene la forma general:

$$\mathbf{X^{(k+1)} = C + TX^{(k)}, \quad k = 0,1,2,...}$$ en donde **C** es un vector y **T** es una matriz

Fórmula iterativa con las matrices desarrolladas

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} b_1/a_{1,1} \\ b_2/a_{2,2} \\ \vdots \\ b_n/a_{n,n} \end{bmatrix} - \begin{bmatrix} 0 & a_{1,2}/a_{1,1} & a_{1,3}/a_{1,1} & \cdots & a_{1,n}/a_{1,1} \\ a_{2,1}/a_{2,2} & 0 & a_{2,3}/a_{2,2} & \cdots & a_{2,n}/a_{2,2} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1}/a_{n,n} & a_{n,2}/a_{n,n} & a_{n,3}/a_{n,n} & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}$$

$\mathbf{X}^{(0)}$ es el vector inicial. A partir de este vector se obtienen sucesivamente los vectores $\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots$ Si el método converge, entonces la sucesión $\{ \mathbf{X}^{(k)} \}_{k=0,1,2,..}$ tiende al vector solución $\mathbf{X}$

Ejemplo. Resuelva el sistema con el método de Jacobi en notación matricial:

$$\begin{aligned} 5x_1 - 3x_2 + x_3 &= 5 \\ 2x_1 + 4x_2 - x_3 &= 6 \\ 2x_1 - 3x_2 + 8x_3 &= 4 \end{aligned}$$

Solución

$$A = \begin{bmatrix} 5 & -3 & 1 \\ 2 & 4 & -1 \\ 2 & -3 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 5 \\ 6 \\ 4 \end{bmatrix}, \quad D = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 2 & -3 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -3 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \quad X^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$\mathbf{X}^{(k+1)} = \mathbf{C} + \mathbf{T}\mathbf{X}^{(k)} = \mathbf{D}^{-1}\mathbf{B} - \mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{X}^{(k)}, \quad k = 0,1,2,\ldots$ (Fórmula matricial de Jacobi iterativa)

$$\mathbf{C} = \mathbf{D}^{-1}\mathbf{B} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 5 \\ 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 1/5 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/8 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.5 \\ 0.5 \end{bmatrix}$$

$$\mathbf{T} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}) = -\begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 0 & -3 & 1 \\ 2 & 0 & -1 \\ 2 & -3 & 0 \end{bmatrix} = -\begin{bmatrix} 1/5 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/8 \end{bmatrix} \begin{bmatrix} 0 & -3 & 1 \\ 2 & 0 & -1 \\ 2 & -3 & 0 \end{bmatrix}$$

$$= -\begin{bmatrix} 0 & -0.6 & 0.2 \\ 0.5 & 0 & -0.25 \\ 0.25 & -0.375 & 0 \end{bmatrix}$$

$$\mathbf{X}^{(1)} = \mathbf{C} + \mathbf{T}\mathbf{X}^{(0)} = \begin{bmatrix} 1 \\ 1.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0 & -0.6 & 0.2 \\ 0.5 & 0 & -0.25 \\ 0.25 & -0.375 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.4 \\ 1.25 \\ 0.625 \end{bmatrix}$$

$$\mathbf{X}^{(2)} = \mathbf{C} + \mathbf{T}\mathbf{X}^{(1)} = \begin{bmatrix} 1 \\ 1.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0 & -0.6 & 0.2 \\ 0.5 & 0 & -0.25 \\ 0.25 & -0.375 & 0 \end{bmatrix} \begin{bmatrix} 1.4 \\ 1.25 \\ 0.625 \end{bmatrix} = \begin{bmatrix} 1.625 \\ 0.9562 \\ 0.6187 \end{bmatrix}$$

$$X^{(3)} = C + TX^{(2)} = \begin{bmatrix} 1 \\ 1.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0 & -0.6 & 0.2 \\ 0.5 & 0 & -0.25 \\ 0.25 & -0.375 & 0 \end{bmatrix} \begin{bmatrix} 1.625 \\ 0.9562 \\ 0.6187 \end{bmatrix} = \begin{bmatrix} 1.45 \\ 0.8422 \\ 0.4523 \end{bmatrix},$$

Etc.

Manejo matricial computacional en Python del método de Jacobi

```
>>> from numpy import*
>>> A=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> B=array([[5],[6],[4]],float)
>>> D=diag(diag(A))
>>> Di=linalg.inv(D)
>>> L=tril(A)-D
>>> U=triu(A)-D
>>> C=dot(Di,B)
>>> T=-dot(Di,L+U)
>>> X=array([[1],[1],[1]],float)
>>> X=C+dot(T,X);print(X)
[[ 1.4  ]
 [ 1.25 ]
 [ 0.625]]
>>> X=C+dot(T,X);print(X)
[[ 1.625  ]
 [ 0.95625]
 [ 0.61875]]
>>> X=C+dot(T,X);print(X)
[[ 1.45      ]
 [ 0.8421875 ]
 [ 0.45234375]]
>>> X=C+dot(T,X);print(X)
[[ 1.41484375]
 [ 0.88808594]
 [ 0.45332031]]
>>> X=C+dot(T,X);print(X)
[[ 1.4421875 ]
 [ 0.9059082 ]
 [ 0.47932129]]
```

5.2 Método de Gauss-Seidel

Este método calcula los nuevos valores del vector **X** usando los valores más recientes del vector **X**, es decir aquellos que ya han sido calculados en la misma iteración, los otros valores provienen de la iteración anterior. En el método de Jacobi se utilizan todos los valores de la iteración anterior. Por este motivo podemos suponer que el método de Gauss-Seidel en general converge o diverge más rápidamente que el método de Jacobi.

5.2.1 Formulación matemática

La fórmula de Gauss-Seidel se la obtiene directamente de la fórmula de Jacobi separando la sumatoria en dos partes: los componentes que aún no han sido calculados se los toma de la iteración anterior k, mientras que los ya calculados, se los toma de la iteración k+1:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)}\right); \quad i = 1, 2, ..., n; \quad k = 0, 1, 2, ...$$

En general, el método de Gauss-Seidel requiere menos iteraciones que el método de Jacobi en caso de que converja

Ejemplo. Dadas las ecuaciones:

$$5x_1 - 3x_2 + x_3 = 5$$
$$2x_1 + 4x_2 - x_3 = 6$$
$$2x_1 - 3x_2 + 8x_3 = 4$$

a) Formule un sistema iterativo con el método de Gauss-Seidel:

b) Comenzando con el vector inicial: $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 1$, realice dos iteraciones:

Solución

$$x_1 = 1/5\ (5 + 3x_2 - x_3)$$
$$x_2 = 1/4\ (6 - 2x_1 + x_3)$$
$$x_3 = 1/8\ (4 - 2x_1 + 3x_2)$$

Fórmula iterativa:

$$x_1^{(k+1)} = 1/5\ (5 + 3x_2^{(k)} - x_3^{(k)})$$
$$x_2^{(k+1)} = 1/4\ (6 - 2x_1^{(k+1)} + x_3^{(k)})$$
$$x_3^{(k+1)} = 1/8\ (4 - 2x_1^{(k+1)} + 3x_2^{(k+1)})$$

k=0:
$$x_1^{(1)} = 1/5\ (5 + 3x_2^{(0)} - x_3^{(0)}) = 1/5\ (5 + 3(1) - (1)) = 1.4$$
$$x_2^{(1)} = 1/4\ (6 - 2x_1^{(1)} + x_3^{(0)}) = 1/4\ (6 - 2(1.4) + (1)) = 1.05$$
$$x_3^{(1)} = 1/8\ (4 - 2x_1^{(1)} + 3x_2^{(1)}) = 1/8\ (4 - 2(1.4) + 3(1.05)) = 0.5438$$

k=1:
$$x_1^{(2)} = 1/5\ (5 + 3x_2^{(1)} - x_3^{(1)}) = 1/5\ (5 + 3(1.05) - (0.5438)) = 1.5212$$
$$x_2^{(2)} = 1/4\ (6 - 2x_1^{(2)} + x_3^{(1)}) = 1/4\ (6 - 2(1.5212) + (0.5438)) = 0.8753$$
$$x_3^{(2)} = 1/8\ (4 - 2x_1^{(2)} + 3x_2^{(2)}) = 1/8\ (4 - 2(1.5212) + 3(0.8753)) = 0.4479$$

5.2.2 Manejo computacional de la fórmula de Gauss-Seidel

La siguiente función en Python recibe un vector **X** y entrega un nuevo vector **X** calculado en cada iteración

```
def gaussseidel(a,b,x):
    n=len(x)
    for i in range(n):
        s=0
        for j in range(n):
            if i!=j:
                s=s+a[i,j]*x[j]
        x[i]=(b[i]-s)/a[i,i]
    return x
```

Resuelva el ejemplo anterior usando la función **gaussseidel**:

```
>>> from numpy import*
>>> from gaussseidel import*
>>> a=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> b=array([[5],[6],[4]],float)
>>> x=array([[1],[1],[1]],float)
>>> x=gaussseidel(a,b,x); print(x)
[[ 1.4    ]
 [ 1.05   ]
 [ 0.54375]]
>>> x=gaussseidel(a,b,x); print(x)
[[ 1.52125   ]
 [ 0.8753125 ]
 [ 0.44792969]]
>>> x=gaussseidel(a,b,x); print(x)
[[ 1.43560156]
 [ 0.89418164]
 [ 0.47641772]]
>>> x=gaussseidel(a,b,x); print(x)
[[ 1.44122544]
 [ 0.89849171]
 [ 0.47662803]]
>>> x=gaussseidel(a,b,x); print(x)
[[ 1.44376942]
 [ 0.8972723 ]
 [ 0.47553476]]                ....
```

En general, la convergencia es más rápida que con el método de Jacobi

5.2.3 Instrumentación computacional del método de Gauss-Seidel

Similar al método de Jacobi, conviene instrumentar el método incluyendo el control de iteraciones dentro de la función, suministrando los parámetros apropiados.

Los datos para la función son la matriz de coeficientes **a** y el vector de constantes **b** de un sistema lineal. Adicionalmente recibe un vector inicial **x**, el criterio de error **e** y el máximo de iteraciones permitidas **m**. Esta función utiliza dentro de un ciclo, la función **gaussseidel** definida anteriormente. Entrega el vector **x** calculado y el número de iteraciones realizadas **k**. Si el método no converge, **x** contendrá un vector nulo y el número de iteraciones **k** será igual al máximo **m**.

```
from gaussseidel import*
import numpy as np
def gaussseidelm(a,b,x,e,m):
    n=len(x)
    t=x.copy()
    for k in range(m):
        x=gaussseidel(a,b,x)
        d=np.linalg.norm(array(x)-array(t),inf)
        if d<e:
            return [x,k]
        else:
            t=x.copy()
    return [[],m]
```

Ejemplo. Use la función **gaussseidelm** para encontrar el vector solución del ejemplo anterior con **E=0.0001**. Determine cuántas iteraciones se realizaron. Use $\mathbf{X^{(0)}}$**: [1, 1, 1]**

```
>>> from numpy import*
>>> from gaussseidelm import*
>>> a=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> b=array([[5],[6],[4]],float)
>>> x=array([[1],[1],[1]],float)
>>> [x,k]=gaussseidelm(a,b,x,0.0001,20)
>>> print(x)
[[ 1.44322195]
 [ 0.8973032 ]
 [ 0.47568321]]
>>> print(k)
6
```

5.2.4 Forma matricial del método de Gauss-Seidel

Dado el sistema de ecuaciones lineales

$$\mathbf{AX} = \mathbf{B}$$

La matriz **A** se re-escribe como la suma de tres matrices:

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$$

D es una matriz diagonal con los elementos de la diagonal principal de **A**

L es una matriz triangular inferior con ceros en la diagonal principal. Debajo de la diagonal se copian los componentes respectivos de la matriz **A**

U es una matriz triangular superior con ceros en la diagonal principal. Arriba de la diagonal se copian los componentes respectivos de la matriz **A**

Sustituyendo en la ecuación:

$$\mathbf{(L + D + U)X} = \mathbf{B}$$

Conviene separar en dos bloques: **(L+D)** y **U**

$$\mathbf{(L + D)X + UX} = \mathbf{B}$$

Los componentes de **X** para el bloque **L+D** se asocian a la iteración actual **k+1**. Este bloque incluye a los componentes de **X** que se van a calcular en la iteración actual y los componentes de **X** que ya han sido calculados en la iteración actual y que ya se pueden utilizar.

Los componentes de **X** para el bloque **U** se asocian a la iteración anterior **k**. Este bloque incluye a los componentes de **X** que aun no han sido calculados en la iteración actual **k+1** por lo cual se deben usar los valores de la iteración anterior k.

$$\mathbf{(L + D)X^{(k+1)} + UX^{(k)}} = \mathbf{B},\ \mathbf{k=0,1,2,...}\ \text{(iteraciones)}$$

De donde se obtiene la fórmula matricial iterativa de Gauss-Seidel

$$\mathbf{X^{(k+1)}} = \mathbf{(L + D)^{-1}B - (L + D)^{-1}UX^{(k)}},\ \mathbf{k=0,1,2,...}\ \text{Siempre que } \mathbf{(L+D)^{-1}} \text{ exista}$$

Fórmula que tiene la forma general:

$$\mathbf{X^{(k+1)} = C + TX^{(k)}},\quad \mathbf{k = 0,1,2,...}\ \text{en donde } \mathbf{C} \text{ es un vector y } \mathbf{T} \text{ es una matriz}$$

$\mathbf{X^{(0)}}$ es el vector inicial. A partir de este vector se obtienen sucesivamente los vectores $\mathbf{X^{(1)}}$, $\mathbf{X^{(2)}}$, ...

Si el método converge, entonces la sucesión $\{ \mathbf{X^{(k)}} \}_{\mathbf{k=0,1,2,..}}$ tiende al vector solución **X**

Fórmula matricial iterativa desarrollada

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} a_{1,1} & 0 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix}^{-1} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} - \begin{bmatrix} a_{1,1} & 0 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix}^{-1} \begin{bmatrix} 0 & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ & 0 & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}$$

Ejemplo. Resuelva el sistema con el método de Gauss-Seidel en notación matricial:

$$\begin{aligned} 5x_1 - 3x_2 + x_3 &= 5 \\ 2x_1 + 4x_2 - x_3 &= 6 \\ 2x_1 - 3x_2 + 8x_3 &= 4 \end{aligned}$$

Solución

$$A = \begin{bmatrix} 5 & -3 & 1 \\ 2 & 4 & -1 \\ 2 & -3 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 5 \\ 6 \\ 4 \end{bmatrix}, \quad D = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 2 & -3 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -3 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \quad X^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Fórmula matricial de Gauss-Seidel iterativa

$$X^{(k+1)} = C + TX^{(k)} = (L + D)^{-1}B - (L + D)^{-1}UX^{(k)}, \; k=0,1,2,...$$

$$C = (L+D)^{-1}B = \begin{bmatrix} 5 & 0 & 0 \\ 2 & 4 & 0 \\ 2 & -3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 5 \\ 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0.6250 \end{bmatrix}$$

$$T = -(L+D)^{-1}U = -\begin{bmatrix} 5 & 0 & 0 \\ 2 & 4 & 0 \\ 2 & -3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 0 & -3 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.6 & -0.2 \\ 0 & -0.3 & 0.35 \\ 0 & -0.2625 & 0.18125 \end{bmatrix}$$

$$X^{(1)} = C + TX^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 0.625 \end{bmatrix} + \begin{bmatrix} 0 & 0.6 & -0.2 \\ 0 & -0.3 & 0.35 \\ 0 & -0.2625 & 0.18125 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.4 \\ 1.05 \\ 0.5437 \end{bmatrix}$$

$$X^{(2)} = C + TX^{(1)} = \begin{bmatrix} 1 \\ 1 \\ 0.625 \end{bmatrix} + \begin{bmatrix} 0 & 0.6 & -0.2 \\ 0 & -0.3 & 0.35 \\ 0 & -0.2625 & 0.18125 \end{bmatrix} \begin{bmatrix} 1.4 \\ 1.05 \\ 0.5437 \end{bmatrix} = \begin{bmatrix} 1.5213 \\ 0.8753 \\ 0.4479 \end{bmatrix}$$

$$X^{(3)} = C + TX^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 0.625 \end{bmatrix} + \begin{bmatrix} 0 & 0.6 & -0.2 \\ 0 & -0.3 & 0.35 \\ 0 & -0.2625 & 0.18125 \end{bmatrix} \begin{bmatrix} 1.5213 \\ 0.8753 \\ 0.4479 \end{bmatrix} = \begin{bmatrix} 1.4356 \\ 0.8942 \\ 0.4764 \end{bmatrix}, \text{ Etc.}$$

Manejo matricial computacional en Python del método de Gauss-Seidel

```
>>> from numpy import*
>>> A=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> B=array([[5],[6],[4]],float)
>>> LD=tril(A)
>>> D=diag(diag(A))
>>> U=triu(A)-D
>>> C=dot(linalg.inv(LD),B)
>>> T=-dot(linalg.inv(LD),U)
>>> X=array([[1],[1],[1]],float)
>>> X=C+dot(T,X);print(X)
[[ 1.4    ]
 [ 1.05   ]
 [ 0.54375]]
>>> X=C+dot(T,X);print(X)
[[ 1.52125   ]
 [ 0.8753125 ]
 [ 0.44792969]]
>>> X=C+dot(T,X);print(X)
[[ 1.43560156]
 [ 0.89418164]
 [ 0.47641772]]
>>> X=C+dot(T,X);print(X)
[[ 1.44122544]
 [ 0.89849171]
 [ 0.47662803]]
>>> X=C+dot(T,X);print(X)
[[ 1.44376942]
 [ 0.8972723 ]
 [ 0.47553476]]
>>>
```

5.3 Método de relajación

Es un dispositivo para acelerar la convergencia (o divergencia) de los métodos iterativos

5.3.1 Formulación matemática

Se la obtiene de la fórmula de Gauss-Seidel incluyendo un factor de convergencia

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i}^{n} a_{i,j} x_j^{(k)}\right); \quad i = 1, 2, ..., n; \quad k = 0, 1, 2, ...$$

$0 < \omega < 2$ es el factor de relajación

Si $\omega = 1$ la fórmula se reduce a la fórmula iterativa de Gauss-Seidel

Ejemplo. Dadas las ecuaciones:

$$5x_1 - 3x_2 + x_3 = 5$$
$$2x_1 + 4x_2 - x_3 = 6$$
$$2x_1 - 3x_2 + 8x_3 = 4$$

a) Formule un sistema iterativo con el método de Gauss-Seidel:

b) Comenzando con el vector inicial: $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 1$, realice una iteración con $\omega = 1.1$

Solución

$$x_1^{(k+1)} = x_1^{(k)} + \omega/5\ (5 - 5x_1^{(k)} + 3x_2^{(k)} - x_3^{(k)})$$
$$x_2^{(k+1)} = x_2^{(k)} + \omega/4\ (6 - 2x_1^{(k+1)} - 4x_2^{(k)} - x_3^{(k)})$$
$$x_3^{(k+1)} = x_3^{(k)} + \omega/8\ (4 - 2x_1^{(k+1)} + 3x_2^{(k+1)} - 8x_3^{(k)})$$

k=0:

$x_1^{(1)} = x_1^{(0)} + 1.1/5\ (5 - 5x_1^{(0)} + 3x_2^{(0)} - x_3^{(0)}) = 1 + 1.1/5\ (5 - 5(1) + 3(1) - 1)$
$= 1.4400;$

$x_2^{(1)} = x_2^{(0)} + 1.1/4\ (6 - 2x_1^{(1)} - 4x_2^{(0)} + x_3^{(0)}) = 1 + 1.1/4\ (6 - 2(1.6) - 4(1) + 1)$
$= 1.0330$

$x_3^{(1)} = x_3^{(0)} + 1.1/8\ (4 - 2x_1^{(1)} + 3x_2^{(1)} - 8x_3^{(0)}) = 1 + 1.1/8\ (4 - 2(1.6) - 3(0.925) - 8(1))$
$= 0.4801$

5.3.2 Manejo computacional de la fórmula de relajación

La siguiente función en Python recibe un vector **X** y el factor de relajación **w**. Entrega un nuevo vector **X** calculado en cada iteración

```
def relajacion(a,b,x,w):
    n=len(x)
    for i in range(n):
        s=0
        for j in range(n):
            s=s+a[i,j]*x[j]
        x[i]=x[i]+w*(b[i]-s)/a[i,i]
    return x
```

Resuelva el ejemplo anterior usando la función **relajación** con **k=1.2**:

```
>>> from relajacion import*
>>> from numpy import*
>>> a=array([[5,-3,1],[2,4,-1],[2,-3,8]],float)
>>> b=array([[5],[6],[4]],float)
>>> x=array([[1],[1],[1]],float)
>>> x=relajacion(a,b,x,1.2);print(x)
[[ 1.48  ]
 [ 1.012 ]
 [ 0.4114]]
>>> x=relajacion(a,b,x,1.2);print(x)
[[ 1.533904  ]
 [ 0.8006776 ]
 [ 0.41785372]]
...
```

5.3.3 Forma matricial del método de relajación

Dado el sistema de ecuaciones lineales

$$AX = B$$

La matriz **A** se re-escribe como la suma de las siguientes matrices:

$$A = L + D + S - D$$

D es una matriz diagonal con elementos iguales a los de la diagonal principal de **A**
L es una matriz triangular inferior con ceros en la diagonal principal y los otros elementos iguales a los elementos respectivos de la matriz **A**
S es una matriz triangular superior con todos sus elementos iguales a los elementos respectivos de la matriz **A**

Sustituyendo en la ecuación:

$$(L + D + S - D)X = B$$
$$LX + DX + SX - D X = B$$
$$X = D^{-1}B - D^{-1}LX - D^{-1}SX + D^{-1}DX, \quad \text{siempre que } D^{-1} \text{ exista}$$
$$X = X + D^{-1}B - D^{-1}LX - D^{-1}SX$$

Matrices desarrolladas

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1/a_{1,1} \\ b_2/a_{2,2} \\ \vdots \\ b_n/a_{n,n} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{2,1}/a_{2,2} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{n,1}/a_{n,n} & a_{n,2}/a_{n,n} & a_{n,3}/a_{n,n} & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} 1 & a_{1,2}/a_{1,1} & a_{1,3}/a_{1,1} & \dots & a_{1,n}/a_{1,1} \\ 0 & 1 & a_{2,3}/a_{2,2} & \dots & a_{2,n}/a_{2,2} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

El sistema está en la forma recurrente del punto fijo **X = G(X)** que sugiere su uso iterativo.

En el método de Relajación se agrega un factor ω

Fórmula iterativa en forma matricial

$$X^{(k+1)} = X^{(k)} + \omega(D^{-1}B - D^{-1}LX^{(k+1)} - D^{-1}SX^{(k)}), \quad k = 0,1,2,...$$

ω es el factor de relajación. Este factor modifica al residual tratando de reducirlo a cero con mayor rapidez que el método de Gauss-Seidel.

Si $\omega = 1$ la fórmula iterativa se reduce a la fórmula del método de Gauss-Seidel
Si $\omega \in (0,1)$ se denomina método de subrelajación.
Si $\omega \in (1,2)$ se denomina método de sobrerelajación.

$X^{(0)}$ es el vector inicial.
A partir de este vector se obtienen sucesivamente los vectores $X^{(1)}$, $X^{(2)}$, ...

5.4 Convergencia de los métodos iterativos para sistemas lineales

Dado el sistema de ecuaciones lineales

$$AX = B$$

SI la matriz A no es singular, se puede re-escribir en un sistema equivalente en la forma:

$$X = G(X) = C + TX$$

En donde **C** es un vector constante y **T** se denomina matriz de transición o matriz de iteración del método iterativo.

El sistema con la matriz de transición se puede usar para definir la forma recurrente del método iterativo:

$$X^{(k+1)} = C + TX^{(k)}, \quad k = 0,1,2,...$$

Estas dos ecuaciones se pueden restar para relacionar el error de truncamiento:

$$X - X^{(k+1)} = T\,(X - X^{(k)})$$

Estas diferencias constituyen el error de truncamiento vectorial entre dos iteraciones consecutivas:

$$E^{(k)} = X - X^{(k)}$$

$$E^{(k+1)} = X - X^{(k+1)}$$

Sustituyendo en la ecuación anterior

$$\mathbf{E^{(k+1)} = T\,E^{(k)}}$$

La relación muestra que la matriz de transición **T** interviene en la convergencia del método iterativo actuando como un factor.

La siguiente definición permite establecer una medida para la matriz de transición **T** como factor de convergencia:

Definición: Radio espectral

Si **A** es una matriz cuadrada, entonces su radio espectral se define como

$$\rho(\mathbf{A}) = \max_{1\le k\le n}\left\{|\lambda_k|,\ \lambda_k \text{ valor propio de } \mathbf{A}\right\}$$

Si λ_k es un número complejo en la forma a+bi, con a,b en **R**, entonces

$$|\lambda_k| = |a+bi| = \sqrt{a^2+b^2}$$

Teorema de convergencia para los métodos iterativos

La sucesión $\{X^{(k)}\}_{k=0,1,2,..}$ definida con la fórmula iterativa $X^{(k+1)} = C + TX^{(k)},\ k = 0,1,2,...$ converge con cualquier vector inicial $X^{(0)} \in R^n$ al vector solución **X** si y solo si $\rho(T) < 1$.

Es una **condición necesaria y suficiente** para la convergencia de los métodos iterativos.

5.4.1 Matriz de transición para los métodos iterativos

Forma general de la fórmula recurrente de los métodos iterativos

$$X^{(k+1)} = C + TX^{(k)},\quad k = 0,1,2,...$$

En donde **C** es un vector constante y **T** es la matriz de transición

Matriz de transición para el método de Jacobi

$$X^{(k+1)} = C + TX^{(k)} = D^{-1}B - D^{-1}(L+U)X^{(k)},\quad k = 0,1,2,...$$

$$T = -D^{-1}(L+U)$$

Matriz de transición para el método de Gauss-Seidel

$$X^{(k+1)} = C + TX^{(k)} = (L+D)^{-1}B - (L+D)^{-1}UX^{(k)},\ k=0,1,2,...$$

$$T = -(L+D)^{-1}U$$

Matriz de transición para el método de Relajación

$$X^{(k+1)} = X^{(k)} + \omega(D^{-1}B - D^{-1}LX^{(k+1)} - D^{-1}SX^{(k)}), \quad k = 0,1,2,...$$

De donde se obtiene:

$$T = -(I + \omega D^{-1}L)^{-1}(I - \omega D^{-1}S)$$

Una condición suficiente de convergencia

Con la norma de la matriz de transición **T** se puede establecer una condición suficiente de convergencia para los métodos iterativos, más simple pero más débil que la condición necesaria y suficiente de convergencia con el radio espectral.

Relación del error de truncamiento de los métodos iterativos:

$$E^{(k+1)} = T\,E^{(k)}$$

Su norma:

$$\| E^{(k+1)} \| \leq \| T \| \| E^{(k)} \|, \quad k = 0, 1, 2, ...$$

Esta relación define una condición suficiente para la convergencia. El método iterativo converge si se cumple que la norma de la matriz de transición es menor que 1:

$$\| T \| < 1$$

Se puede demostrar que para los métodos iterativos: $\rho(T) \leq \| T \|$

En el **caso del método iterativo de Jacobi**, se puede constatar directamente en el sistema de ecuaciones si se cumple esta condición suficiente de convergencia:

Matriz de transición para el método iterativo de Jacobi:

$$T = -D^{-1}(L+U) = -\begin{bmatrix} 0 & a_{1,2}/a_{1,1} & a_{1,3}/a_{1,1} & \cdots & a_{1,n}/a_{1,1} \\ a_{2,1}/a_{2,2} & 0 & a_{2,3}/a_{2,2} & \cdots & a_{2,n}/a_{2,2} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n,1}/a_{n,n} & a_{n,2}/a_{n,n} & a_{n,3}/a_{n,n} & \cdots & 0 \end{bmatrix}$$

Por la forma de la matriz **T** se puede establecer que si en cada fila de la matriz **A** el elemento en la diagonal es mayor en magnitud que la suma de los otros elementos de la fila respectiva, entonces $\| T \| < 1$ (norma de fila). Consecuentemente, el método converge.

Si la matriz **A** cumple esta propiedad se dice que es **"diagonal dominante"** y constituye una condición suficiente para la convergencia del método de Jacobi:

$$\forall i \left(|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}|\right) \Rightarrow \|T\| < 1$$

En resumen, si la matriz de coeficientes **A** es de tipo diagonal dominante, entonces el método de Jacobi converge con cualquier vector inicial $\mathbf{X}^{(0)} \in \mathbf{R}^n$

Ejemplo. Se plantea resolver el siguiente sistema con un método iterativo y se desea determinar previamente la propiedad de convergencia examinando la matriz de transición:

$$8x_1 + 9x_2 + 2x_3 = 69$$
$$2x_1 + 7x_2 + 2x_3 = 47$$
$$2x_1 + 8x_2 + 6x_3 = 68$$

Solución

Fórmula iterativa de Jacobi

$$\mathbf{X}^{(k+1)} = \mathbf{C} + \mathbf{T}\mathbf{X}^{(k)} = \mathbf{D}^{-1}\mathbf{B} - \mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{X}^{(k)}, \quad k = 0,1,2,...$$

Matriz de transición **T**

$$\mathbf{T} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}) = -\begin{bmatrix} 0 & 9/8 & 2/8 \\ 2/7 & 0 & 2/7 \\ 2/6 & 8/6 & 0 \end{bmatrix}$$

Norma de fila de T:

$$\|T\| = 5/3 > 1$$

Este resultado indica que no se puede afirmar que el método de Jacobi convergerá.

Usando el teorema de convergencia con el radio espectral:

$$\rho(\mathbf{A}) = \max_{1\le k\le n} \{|\lambda_k|, \ \lambda_k \text{ valor propio de } \mathbf{A}\}$$

Los valores propios λ de **T** se pueden calcular con las raíces del polinomio característico:

$$p(\lambda) = \det(T - \lambda I) = -\lambda^3 + 11/14\lambda - 17/84 = 0$$

Los valores propios son: **0.287968, 0.706615, -0.994583**

Por lo tanto $\rho(T) = 0.9945 < 1$ y se puede concluir que el método de Jacobi si convergerá. Sin embargo, la convergencia será muy lenta pues el radio espectral es cercano a **1**.

Cálculos con Python del radio espectral de la matriz de transición de Jacobi:

```
>>> from numpy import*
>>> T=-[[0,9/8,2/8],[2/7,0,2/7],[2/6,8/6,0]]      Matriz de transición de Jacobi
>>> val,vect=linalg.eig(T)                        Valores y vector propios
```

```
>>> print(val)
[ 0.99458399, -0.28796854, -0.70661544]
>>> ro=max(abs(val))                              Radio espectral
>>> ro
0.99458398589455355
```

Fórmula iterativa de Gauss-Seidel

$$X^{(k+1)} = C + TX^{(k)} = (L + D)^{-1}B - (L + D)^{-1}UX^{(k)},\ k=0,1,2,...$$

Matriz de transición **T**

$$T = -(L+D)^{-1}U = \begin{bmatrix} 0 & -1.125 & -0.25 \\ 0 & 0.3214 & -0.2145 \\ 0 & -0.0536 & 0.3690 \end{bmatrix}$$

Norma de fila de T:

$$\| T \| = 1.375 > 1$$

Este resultado indica que no se puede afirmar que el método de Gauss-Seidel convergerá.

Cálculos con Python para hallar el radio espectral de la matriz de transición de Gauss-Seidel

```
>>> from numpy import*
>>> T=[[0,-1.125,-0.25],[0,0.3214,-0.2143],[0,-0.0538,0.3690]]
>>> val,vect=linalg.eig(T)
>>> print(val)
[ 0.  0.23521918  0.45518082]
>>> ro=max(abs(val))
>>> print(ro)
0.455180816509
```

Por lo tanto $\rho(T) = 0.45518 < 1$ y se puede concluir que el método de Gauss-Seidel si convergerá y lo hará mucho más rápido que el método de Jacobi pues el radio espectral es significativamente menor que **1** comparado con el método de Jacobi.

5.5 Eficiencia de los métodos iterativos

La fórmula del error de truncamiento expresa que la convergencia de los métodos iterativos es de primer orden:

$$E^{(k+1)} = O(E^{(k)})$$

Cada iteración requiere multiplicar la matriz de transición por un vector, por lo tanto la cantidad de operaciones aritméticas realizadas $T(n)$ en cada iteración es de segundo orden: $T(n) = O(n^2)$

Si **k** representa la cantidad de iteraciones que se realizan hasta obtener la precisión requerida, entonces la eficiencia de cálculo de los métodos iterativos es: $T(n) = k\ O(n^2)$. Usualmente **k>>n**, por lo tanto los métodos iterativos son menos eficientes que los métodos directos.

5.6 Estimación del error de truncamiento en los métodos iterativos

Si la fórmula iterativa converge, se puede escribir:

$$X^{(k)} \to X, \text{ si } k \to \infty$$

$$X^{(k+1)} \to X, \text{ si } k \to \infty$$

$$\Rightarrow \quad \| X^{(k+1)} - X^{(k)} \| \to 0, \text{ si } k \to \infty$$

Entonces, si el método converge, se tendrá que para cualquier valor positivo **E** arbitrariamente pequeño, en alguna iteración **k**:

$$\| X^{(k+1)} - X^{(k)} \| < E$$

Se dice que el vector calculado tiene error **E** (es una estimación del error de truncamiento)

Para que el error sea independiente de la magnitud de los resultados se puede usar la definición de error relativo:

$$\frac{\| X^{(k+1)} - X^{(k)} \|}{\| X^{(k+1)} \|} < e,$$ en donde **e** puede expresarse como porcentaje

5.7 Instrumentación del método de Gauss-Seidel con el radio espectral

La siguiente instrumentación del método de Gauss-Seidel usa el radio espectral para determinar la convergencia. En caso de existir convergencia entrega la solución, la cantidad de iteraciones realizadas y el radio espectral. Caso contrario, entrega un vector nulo, el conteo nulo de iteraciones y el valor del radio espectral.

Nombre de la función: **gs**

Uso: **`X,i,ro = gs(A,B,E)`**

Entrada

- **A**: matriz de coeficientes
- **B**: vector de constantes
- **E**: error permitido

Salida

- **X**: vector solución
- **i**: conteo de iteraciones realizadas
- **ro**: radio espectral

```
from numpy import*
def gs(A,B,E):
    n=len(B)
    D=diag(diag(A))
    LD=tril(A)
    U=triu(A)-D
    C=dot(linalg.inv(LD),B)
    T=-dot(linalg.inv(LD),U)
    [val,vec]=linalg.eig(T)
    ro=max(abs(val))
    if ro>=1:                    #No converge
        return [[],0,ro]
    X0=ones([n,1],float)         #Vector inicial
    i=1
    while True:
        X1=C+dot(T,X0)
        if linalg.norm(X1-X0,inf)<E:
            return[X1,i,ro]
        i=i+1
        X0=X1.copy()
```

Ejemplo. Resolver el siguiente sistema de ecuaciones con el método iterativo de Gauss-Seidel matricial, con el radio espectral. En caso de que converja determine la cantidad de iteraciones realizadas hasta que los resultados tengan error $E=10^{-4}$

$$8x_1 + 9x_2 + 2x_3 = 69$$
$$2x_1 + 7x_2 + 2x_3 = 47$$
$$2x_1 + 8x_2 + 6x_3 = 68$$

```
>>> ========= RESTART =========
>>> from gs import*
>>> A=array([[8,9,2],[2,7,2],[2,8,6]],float)
>>> B=array([[69],[47],[68]],float)
>>> X,i,ro = gs(A,B,0.0001)
>>> print(X)
[[ 1.99995275]
 [ 5.00002219]
 [ 3.99998616]]
>>> print(i)
14
>>> print(ro)
0.454994576873
```

5.8 Práctica computacional con los métodos iterativos

Ejemplo La siguiente matriz es del tipo que aparece en algunas aplicaciones de los métodos numéricos:

$$a = \begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 \\ -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 \end{bmatrix}$$

Use el Teorema de convergencia: $\rho(T) < 1$, para determinar el método iterativo más favorable para realizar los cálculos con esta matriz.

```
>>> from numpy import*
>>> a=array([[4,-1,-1,-1],[-1,4,-1,-1],[-1,-1,4,-1],[-1,-1,-1,4]],float)
>>> d=diag(diag(a))
>>> l=tril(a)-d
>>> u=triu(a)-d
>>> s=triu(a)
>>> di=linalg.inv(d)
```

Matriz de transición del método de Jacobi: $T = -D^{-1}(L+U)$

```
>>> T=dot(-di,l+u)
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> print(ro)
0.75
```

Matriz de transición del método de Gauss-Seidel: $T = -(L + D)^{-1}U$

```
>>> T=-dot(linalg.inv(l+d),u)
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> print(ro)
0.569944948814
```

Matriz de transición del método de Relajación: $T = -(I + \omega D^{-1}L)^{-1}(I - \omega D^{-1}S)$

```
>>> w=0.8
>>> T=-dot(linalg.inv(identity(4)+w*dot(di,l)),(identity(4)-w*dot(di,s)))
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> ro
0.70510569776388976
```

```
>>> w=0.9
>>> T=-dot(linalg.inv(identity(4)+w*dot(di,l)),(identity(4)-w*dot(di,s)))
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> ro
0.64378493913940837

>>> w=1.1
>>> T=-dot(linalg.inv(identity(4)+w*dot(di,l)),(identity(4)-w*dot(di,s)))
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> print(ro)
0.475406826544

>>> w=1.2
>>> T=-dot(linalg.inv(identity(4)+w*dot(di,l)),(identity(4)-w*dot(di,s)))
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> print(ro)
0.331237712223

>>> w=1.3
>>> T=-dot(linalg.inv(identity(4)+w*dot(di,l)),(identity(4)-w*dot(di,s)))
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> print(ro)
0.374013357241

>>> w=1.4
>>> T=-dot(linalg.inv(identity(4)+w*dot(di,l)),(identity(4)-w*dot(di,s)))
>>> val,vect=linalg.eig(T)
>>> ro=max(abs(val))
>>> print(ro)
0.46816927085
```

Estos resultados muestran que para esta matriz, los tres métodos convergerían. Para los datos de estas pruebas se observa que la convergencia será más rápida si se usa el **método de relajación** con ω **= 1.2**. Se puede verificar realizando las iteraciones con las funciones respectivas escritas en Python

También se puede verificar que si ω **= 1**, el método de Relajación es igual al método de Gauss-Seidel.

5.9 Ejercicios y problemas con sistemas de ecuaciones lineales

1. Dado el sistema lineal de ecuaciones:

$$3x_1 - x_3 = 5$$
$$\alpha x_1 + 2x_2 - x_3 = 2$$
$$-x_1 + x_2 + (\alpha + 1)x_3 = 1$$

Indique para cuales valores de α el sistema tiene una solución

2. Dado el sistema $[a_{i,j}]\ x = [b_i],\ \ i, j = 1, 2, 3$
Siendo $a_{i,j} = i/(i + j),\ \ b_i = 2i$

a) Escriba el sistema de ecuaciones lineales correspondiente
b) Resuelva el sistema con el Método de Gauss-Jordan

3. Los puntos **(x,y): (1,3), (2,5), (4,2), (5,4)** pertenecen a la siguiente función:
$f(x) = a_1x^2 + a_2\ e^{0.1x} + a_3\ x + a_4$

a) Escriba el sistema de ecuaciones con los puntos dados,
b) Resuelva el sistema con el Método de Gauss usando la estrategia de pivoteo con 4 decimales

4. Demuestre mediante un conteo que la cantidad de multiplicaciones que requiere el método de directo de Gauss-Jordan para resolver un sistema de n ecuaciones lineales es $n^3/2 + O(n^2)$ y que para el Método de Gauss es $n^3/3 + O(n^2)$

5. En el método de Gauss con "pivoteo parcial", en cada etapa **e** de la transformación, se elige como divisor el coeficiente con mayor magnitud de los coeficiente ubicados en la columna **e** y en las filas **e, e+1, e+2, …, n**.

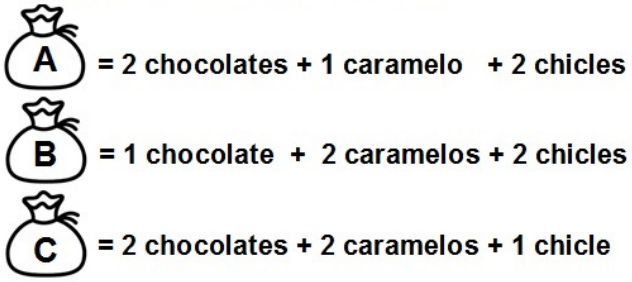
Describa en seudocódigo un algoritmo para realizar el "pivoteo total" que consiste en elegir como divisor el coeficiente de mayor magnitud en la submatriz ubicada desde la fila e hasta la fila n y desde la columna e hasta la columna n. Su algoritmo debe describir únicamente el intercambio de filas y columnas indicando los ciclos y los índices para comparar e intercambiar filas y columnas.

6. Describa, en notación simbólica o en código Python, un algoritmo que reciba una matriz **Anxn** y entregue como resultado las matrices **L, D, U** tales que **A = L+D+U**. Describa la descomposición matricial mediante ciclos e índices. **L**: sub matriz debajo de la diagonal, **D**: matriz diagonal, **U**: submatriz sobre la diagonal. No use **diag**, **tril**, **triu** de Python

7. Considere la matriz de los coeficientes del ejercicio 3 de la sección anterior
a) Use el método de Gauss-Jordan para encontrar la matriz inversa
b) Calcule el número de condición. ¿Es una matriz mal condicionada?

8. Ejercicio para práctica de planteamiento de sistemas de ecuaciones

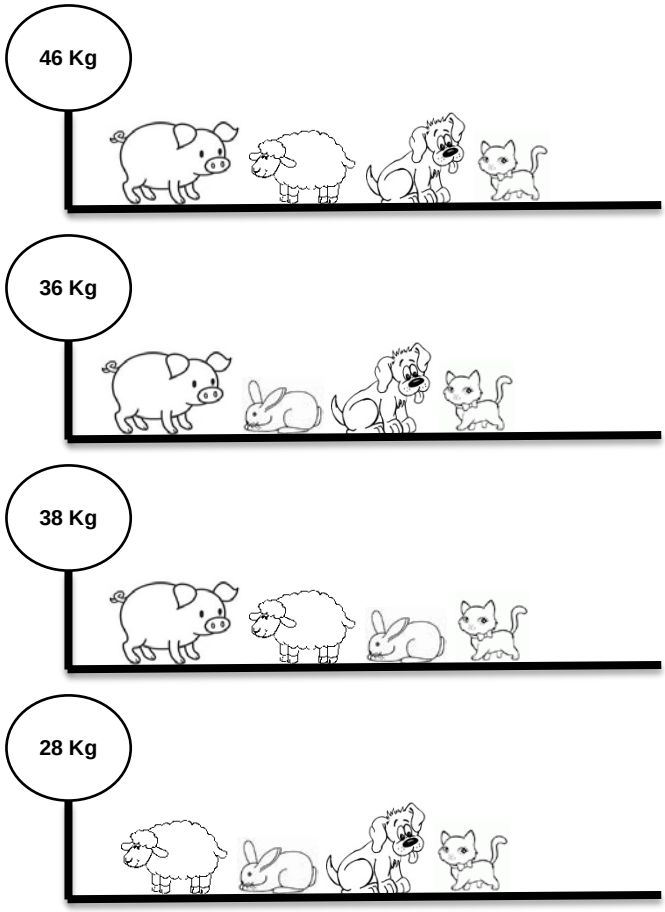
Carmen vende tres tipos de fundas:

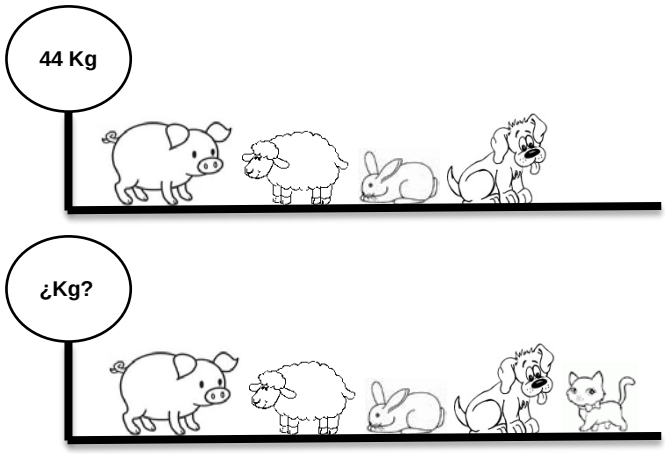


Cuantas fundas debo comprar para obtener en total, sin que falten o sobren, 16 chocolates + 14 caramelos + 15 chicles

9. El siguiente problema también es un ejercicio práctico de planteamiento de sistemas de ecuaciones. Formule el modelo matemático para representar esta situación y obtenga la solución con un método numérico directo.

El gráfico representa una balanza con el registro del peso en Kg.





10. Dado el siguiente sistema de ecuaciones

$$\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/4 & 1/5 & 1/6 \\ 1/7 & 1/8 & 1/9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

a) Resuelva el sistema usando el método de Gauss-Jordan. Simultáneamente encuentre la inversa de la matriz

b) Modifique la matriz de coeficientes sustituyendo el valor de elemento $\mathbf{a_{1,1}}$ con el valor 0.9 Resuelva nuevamente el sistema. Encuentre la variación en la solución calculada.

c) Obtenga el número de condición de la matriz original.

d) Suponga que el error en los coeficientes no excede a 0.01. Use la definición indicada para encontrar una cota para el error en la solución

11. Un comerciante compra tres productos: A, B, C. Estos productos se venden por peso en Kg. pero en las facturas únicamente consta el total que debe pagar. El valor incluye el impuesto a las ventas y supondremos, por simplicidad que es 10%. El comerciante desea conocer el precio unitario de cada artículo, para lo cual dispone de tres facturas con los datos indicados. Formule el modelo matemático y halle la solución con un método directo.

Factura	Kg. de A	Kg. de B	Kg. de C	Valor pagado
1	4	2	5	$19.80
2	2	5	8	$30.03
3	2	4	3	$17.82

12. El siguiente programa en Python se puede usar para construir una matriz Anxn mal condicionada con número de condición mayor a 1000. Los valores son enteros aleatorios entre 0 y 20.

```
import numpy as np
n=int(input('Ingrese la dimensión '))
a=np.zeros([n,n],float)
c=0
while c<=1000:
    for i in range(n):
        for j in range(n):
            a[i,j]=np.random.randint(0,20)
    c=np.linalg.cond(a)
print(c)
print(a)
```

a) Use este programa para construir una matriz mal condicionada de dimensión 5x5
b) Defina un vector de constantes y con la matriz anterior, defina un sistema de ecuaciones.
c) Use un método directo computacional para obtener la solución del sistema anterior.
d) Modifique ligeramente algún coeficiente de la matriz, recalcule la solución y verifique la sensibilidad de la solución a este cambio.
e) Use el número de condición para establecer una cota para el error relativo de la solución en términos del error relativo de la matriz de coeficientes.

13. Suponga que el siguiente modelo describe la cantidad de personas que son infectadas por un virus, en donde **x** es tiempo en días: $\mathbf{f(x) = k_1\, x + k_2\, x^2 + k_3\, e^{0.15X}}$. Se conoce además que la cantidad de personas infectadas fue **25, 130** y **650** en los días **10, 15** y **20** respectivamente. En el modelo, $\mathbf{k_1}$, $\mathbf{k_2}$ y $\mathbf{k_3}$ son coeficientes que deben determinarse.

a) Plantee el sistema de ecuaciones lineales que permitiría determinar los coeficientes
b) Verifique que el vector $\mathbf{K = [-17.325094, -2.242168, 94.265303]^T}$ es la solución
c) Suponga que el dato **650** realmente correspondía al día **21**. Al resolver nuevamente el sistema se obtiene el vector solución $\mathbf{K = [-14.427522, -0.897956, 57.8065182]^T}$.
¿Es confiable la solución calculada?
d) Determine la variación relativa de la solución con respecto a la variación relativa de la matriz de coeficientes, sabiendo que la inversa de la matriz original es:

$$\begin{bmatrix} 0.2576 & -0.0768 & -0.0212 \\ -0.0397 & 0.0396 & -0.0098 \\ 0.5336 & -0.7114 & 0.2668 \end{bmatrix}$$

14. Dado el siguiente sistema de ecuaciones

$$\begin{bmatrix} 2 & 5 & 4 \\ 3 & 9 & 8 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 35 \\ 65 \\ 17 \end{bmatrix}$$

a) Obtenga la solución con un método directo

b) En la matriz de coeficientes sustituya 5 por 5.1 y obtenga nuevamente la solución con un método directo y compare con la solución del sistema original
c) Encuentre el error relativo de la solución y compare con el error relativo de la matriz. Comente acerca del tipo de sistema

15. Use la función **slin** para resolver el siguiente sistema. Identifique las variables libres. Escriba el conjunto solución en términos de la variable libre. Asigne un valor a la variable libre y determine el valor de cada una de las otras variables:

$$\begin{bmatrix} 8 & 2 & 9 & 7 & 6 & 7 \\ 5 & 5 & 3 & 2 & 2 & 4 \\ 1 & 3 & 2 & 6 & 4 & 2 \\ 9 & 9 & 1 & 3 & 3 & 1 \\ 6 & 8 & 5 & 8 & 6 & 6 \\ 2 & 9 & 9 & 9 & 1 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 4 \\ 6 \\ 5 \end{bmatrix}$$

16. Determine si el sistema de ecuaciones lineales obtenido con la siguiente fórmula converge a la solución con el Método de Jacobi

$$3kX_{k-1} - 6kX_k + 2kX_{k+1} = 2k-1$$
$$X_0 = -3, \quad X_n = 4, \quad k=1, 2, 3, \ldots, n-1, \quad n>3$$

17. Sea el sistema lineal de ecuaciones: $\begin{cases} 2X_1 - X_3 = 1 \\ \beta X_1 + 2X_2 - X_3 = 2 \\ -X_1 + X_2 + \alpha X_3 = 1 \end{cases}$

a) ¿Para que valores de α y β se puede asegurar la convergencia con el método de Jacobi?

b) Realice 3 iteraciones del método de Jacobi, con $\mathbf{X^{(0)} = [1, 2, 3]^T}$ usando valores de α y β que aseguren la convergencia.

18. Una empresa produce semanalmente cuatro productos: **A, B, C, D** usando tres tipos de materiales **M1, M2, M3**. Cada Kg. de producto requiere la siguiente cantidad de cada material, en Kg.:

	P1	**P2**	**P3**	**P4**
M1	0.1	0.3	0.6	0.4
M2	0.2	0.6	0.3	0.4
M3	0.7	0.1	0.1	0.2

La cantidad disponible semanal de cada material es: **100, 120, 150** Kg. respectivamente, los cuales **deben usarse completamente**. Se quiere analizar la estrategia de producción.

a) Formule un sistema de ecuaciones lineales siendo $\mathbf{x_1, x_2, x_3, x_4}$ cantidades en Kg. producidas semanalmente de los productos **A, B, C, D** respectivamente
b) Obtenga una solución con la función **slin** y re-escriba el sistema de ecuaciones resultante.

c) Escriba el conjunto solución expresado mediante la variable libre.
d) Encuentre el rango factible para la variable libre
e) Encuentre el rango factible para las otras variables
f) Defina cuatro casos de producción eligiendo un valor para cada una de las cuatro variables y estableciendo el nivel de producción para las restantes variables.

19. Un ingeniero que tiene a cargo una obra de construcción requiere 60 m^3 de arena, 50 m^3 de grava fina y 40 m^3 de grava gruesa. Hay tres canteras de las que pueden obtenerse dichos materiales. La composición (fracción) de 1 m^3 de dichas canteras es:

	Arena	Grava fina	Grava gruesa
Cantera 1	0.52	0.30	0.18
Cantera 2	0.20	0.50	0.30
Cantera 3	0.25	0.20	0.55

Ejemplo. Un m^3 extraído de la Cantera 1 contiene 0.52 de arena (52%), 0.30 de grava fina (30%) y 0.18 de grava gruesa (18%).

Se desea determinar cuantos metros cúbicos tienen que extraerse de cada cantera para satisfacer las necesidades del ingeniero.

a) Formule el sistema de ecuaciones lineales
b) Determine la convergencia del método de Jacobi.
c) Aplicar el método de Gauss-Seidel comenzando con un vector cero. Estime el error absoluto y el error relativo en la quinta iteración

20. Dado el sistema siguiente. Reordene las ecuaciones tratando de acercarlo a la forma "diagonal dominante".

$$\mathbf{4x_1 + 2x_2 + 5x_3 = 18.00}$$
$$\mathbf{2x_1 + 5x_2 + x_3 = 27.30}$$
$$\mathbf{2x_1 + 4x_2 + 3x_3 = 16.20}$$

a) Escriba la matriz de transición del método de Jacobi y determine si se cumple la condición suficiente de convergencia al ser de tipo "diagonal dominante".

b) Determine si la matriz de transición de Jacobi cumple la condición general de convergencia. Puede calcular los valores característicos con la función **eig** de Python

c) Escriba la matriz de transición del método de Gauss-Seidel y verifique que cumple la condición general de convergencia. Puede calcular los valores característicos con la función **eig** de Python

b) Use la función Gaussseidel para realizar 15 iteraciones. Comente los resultados obtenidos.

21. Suponga un sistema biológico con 4 especies de animales (**1, 2, 3, 4**) y 3 tipos de alimentos (**A, B, C**). En el cuadro se muestra el **consumo diario promedio** de cada tipo de alimento por cada miembro de especie animal, y la **cantidad diaria de alimento disponible**. Por ejemplo, el animal de la especie **1**, cada día consume una unidad de alimento **A**, **1** unidad de alimento **B** y **cero** unidades de alimento **C**

Especies de animales

Tipo de alimento	1	2	3	4	Cantidad diaria de unidades de alimento disponible
A	**1**	**2**	**2**	**1**	**350**
B	**1**	**0**	**1**	**2**	**270**
C	**0**	**1**	**3**	**2**	**400**

Sea $\mathbf{x_j}$ el número de miembros de cada especie animal **j = 1, 2, 3, 4.**

a) Escriba un sistema de ecuaciones para determinar la cantidad de miembros de cada especie que pueden sustentarse cada día consumiendo toda la cantidad de alimentos disponibles.
b) Encuentre una solución con el método de Gauss-Jordan en la que la última variable quede libre. Escriba el conjunto de soluciones posibles en función de la variable libre.
c) Encuentre el rango para cada una de las cuatro especies de animales.

d) Si se extingue la especie animal **1**, ¿Qué cantidad de cada una de las otras tres especies podría soportarse con la cantidad diaria de alimento disponible?. Necesita reformular el modelo matemático.

22. Para probar las propiedades de tres nuevos productos (1, 2, 3) se mezclarán tres ingredientes (A, B, C) de los que se dispone respectivamente de 23.92, 20.68 y 31.40 Kg.
Cada Kg. del producto se obtiene mezclando los ingredientes de la siguiente forma:

1 Kg. de Producto 1: 0.70 Kg. de A, 0.25 Kg. de B, 0.05 Kg.de C
1 Kg. de Producto 2: 0.20 kg. de A, 0.60 Kg. de B, 0.20 Kg. de C
1 Kg. de Producto 3: 0.16 kg. de A, 0.04 Kg. de B, 0.80 Kg. de C

Se necesita conocer la cantidad de Kg. que se obtendrá de cada producto sin que hayan sobrantes de ingredientes

a) Formule el modelo matemático

b) Exprese el sistema en la forma matricial iterativa: $X^{(k+1)} = C + T\ X^{(k)}$ correspondiente al método de Gauss-Seidel.

c) Realice cinco iteraciones con la fórmula iterativa matricial. Comience con un vector inicial nulo

d) Escriba la matriz de transición del método de Gauss-Seidel y determine si cumple la condición suficiente de convergencia: $|| E^{(k+1)} || \leq || T || || E^{(k)} ||$

e) Estime el error absoluto y el error relativo en la cuarta iteración.

23. Con el propósito de analizar las preferencias de los consumidores una empresa ha dividido la ciudad en 16 cuadrículas. Los resultados de las mediciones se incluyen en el siguiente cuadro. Sin embargo, en algunas cuadrículas no se pudo realizar la medición y sus valores se los estimará mediante un promedio de las cuadrículas que están inmediatamente a su alrededor. En el cálculo de cada promedio incluya también los valores desconocidos adyacentes.

25	**30**		**20**
28		**40**	**26**
34	**32**		**43**
27		**35**	**50**

a) Plantee un sistema de ecuaciones para obtener la solución (valores de las casillas vacías).
b) Si se utiliza el método de Jacobi, determine si se cumple alguna condición de convergencia
c) Usando **notación matricial** y comenzando con un vector con sus cuatro valores iguales al promedio de los 12 datos conocidos, realice **tres iteraciones** con el método de Jacobi en notación matricial.

24. Considere el siguiente sistema $[a_{i,j}][x_i] = [b_i], \quad i = 1, 2, 3; \quad j = 1, 2, 3$
En donde $a_{i,j} = 1/(i+j), \quad b_i = i$
a) Sustituya los valores y formule un sistema de ecuaciones lineales
b) Encuentre la solución transformando la matriz de coeficientes a la forma de la matriz identidad. Al mismo tiempo aplique las transformaciones al vector de las constantes y a la matriz identidad, de tal manera que la matriz identidad se convierta en la inversa de la matriz de coeficientes.
c) Obtenga el número de condición de la matriz de coeficientes. Indique si es un sistema mal condicionado.
d) Encuentre una cota para el error relativo de la solución dependiente del error relativo de la matriz de coeficientes

25. Se tienen cuatro lingotes de 100 gr. cada uno compuestos del siguiente modo

Lingote	Oro (gramos)	Plata (gramos)	Cobre (gramos)	Estaño (gramos)
1	**20**	**50**	**20**	**10**
2	**30**	**40**	**10**	**20**
3	**20**	**40**	**10**	**30**
4	**50**	**20**	**20**	**10**

Se requiere determinar la fracción que debe tomarse de cada uno de los cuatro lingotes anteriores para formar un nuevo lingote de 100 gramos que contenga 27 gramos de oro, 39 gramos de plata, 15 gramos de cobre y 19 gramos de estaño.

a) Plantee un modelo matemático para describir este problema
b) Describa un método numérico directo para encontrar la solución. Muestre evidencia suficiente del uso del método numérico

c) Encuentre una cota para el error en la solución calculada y comente suponiendo que algún elemento de la matriz de coeficientes puede tener un error de **0.1**

26. Suponga que en el siguiente modelo **f(x)** describe la cantidad de personas que son infectadas por un virus, en donde **x** es tiempo en días:

$$f(x) = k_1 x + k_2 x^2 + k_3 e^{0.15x}$$

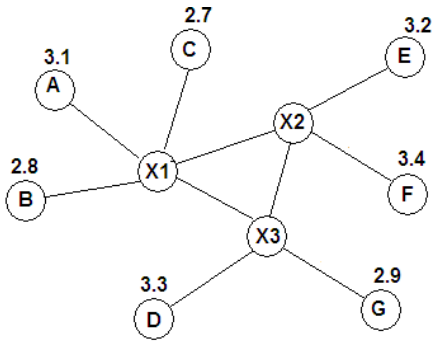
En el modelo k_1 , k_2 y k_3 son coeficientes que deben determinarse.
Se conoce la cantidad de personas infectadas en los días 10, 15 y 20:

$$f(10)=25,\ f(15)=130,\ f(20)=650$$

Plantee un sistema de ecuaciones lineales para determinar los coeficientes y use la solución para definir el modelo **f(x)** para determinar el día más cercano en el cual la cantidad de personas infectadas por el virus será mayor a **10000**. Muestre el gráfico de la ecuación y los valores intermedios calculados.

27. En una región se desean instalar tres nuevos distribuidores **X1, X2, X3** de un producto. En las cercanías ya existen otros distribuidores: **A, B, C, D, E, F, G** del mismo producto. En el gráfico, los círculos indican el precio de venta del producto en cada distribuidor. Las líneas indican con que otros distribuidores están directamente conectados.

Determine el precio de venta que deben establecer los distribuidores de tal manera que sean el promedio de los precios de los distribuidores con los que están directamente conectados.



28. La matriz insumo-producto propuesto por W. Leontief, es un modelo muy importante en Economía. En esta matriz se describe la producción de los diferentes sectores económicos y la demanda interna para satisfacer a estos mismos sectores, expresada como una fracción de su producción. Ejemplo. Suponga que hay tres sectores, A: agricultura, M: manufactura, y S: servicios y su demanda interna es:

Demanda / Producción	**A**	**M**	**S**
A	**0.42**	**0.03**	**0.02**
M	**0.06**	**0.37**	**0.10**
S	**0.12**	**0.15**	**0.19**

Sea **T** el nombre de esta matriz. Para los datos propuestos, en la primera columna de la matriz **T**, el sector **A** requiere 0.42 de su propia producción, 0.06 del sector **M**, y 0.12 del sector **S**, etc.

Sea **D** el vector de demanda externa de cada sector y **X** el vector de la producción total de cada sector requerida para satisfacer las demandas interna y externa:

$\mathbf{D} = \begin{bmatrix} 90 \\ 140 \\ 200 \end{bmatrix}$, $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, en donde x_1, x_2, x_3 representan la producción total de cada sector.

Entonces la ecuación **X = TX + D** proporciona la producción total **X** para satisfacer las demandas externa e interna.

a) Formule un método iterativo en notación vectorial para usar la ecuación anterior. Indique cual es el nombre de la matriz **T**. Determine si el método iterativo es convergente. Norma por filas.

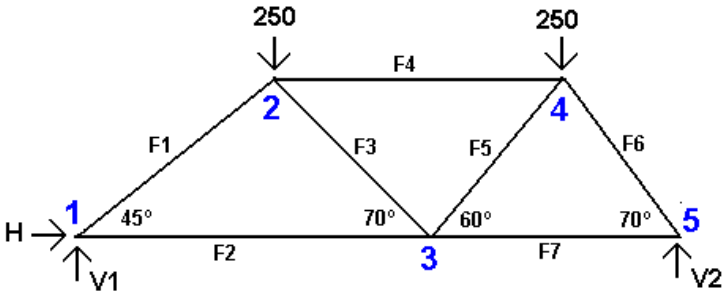
b) Comience con un vector inicial $\mathbf{X = [200, 300, 400]^T}$ realice las iteraciones necesarias hasta que la norma de la tolerancia sea menor a 1.

c) Resuelva el sistema con el método de eliminación de Gauss, para lo cual la ecuación inicial se la reescribe en la siguiente forma: **(I – T)X = D**, en donde **I** es la matriz identidad.

d) Calcule el número de condición de la matriz de coeficientes y comente al respecto.

29. Las cerchas son estructuras reticuladas con elementos triangulares, usualmente metálicas, que se utilizan para soportar grandes cargas. Es importante conocer las fuerzas que actúan en cada nodo. Para ello se plantean ecuaciones de equilibrio de fuerzas verticales y horizontales para cada nodo, las cuales conforman un sistema de ecuaciones lineales.

Determine las fuerzas que actúan en cada nodo en la siguiente cercha con las cargas verticales, en Kg, especificadas en los nodos 2 y 4. La cercha está sostenida en los nodos 1 y 5:



F1, F2, F3, F4, F5, F6, F7 son las fuerzas en los elementos que actúan para mantener la estructura unida, juntando los nodos y evitando que se separen. Sus valores son desconocidos. Se convendrá que si las fuerzas en cada nodo actúan hacia la derecha y hacia arriba tienen signo positivo. **H** es la fuerza horizontal y **V1, V2** son las fuerzas verticales que soportan la estructura. Sus valores son desconocidos.

Ecuaciones de equilibrio en cada nodo:

Nodo 1: $V1 + F1 \operatorname{sen}(45^o) = 0$

H + F1 cos(45º) + F2 = 0

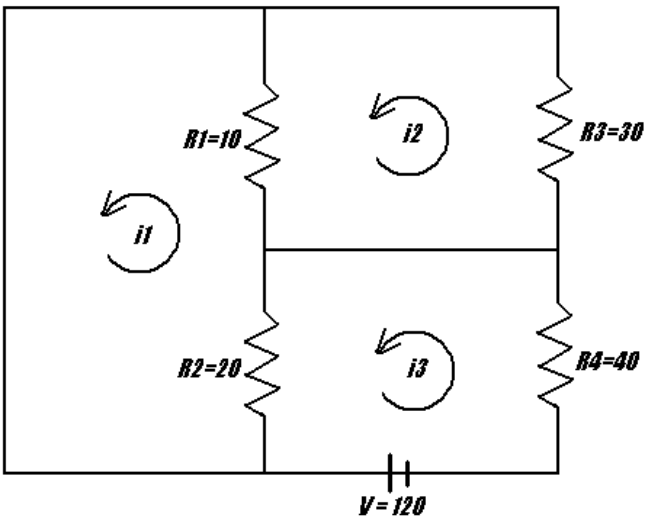
Nodo 2: -F1 sen(45º) – F3 sen(70º) – 250 = 0
-F1 cos(45º) + F4 + F3 cos(70º) = 0
Nodo 3: F3 sen(70º) + F5 sen(60º) = 0
-F2 – F3 cos(70º) + F5 cos(60º) + F7 = 0

Nodo 4: -F5 sen(60º) – F6 sen(70º) – 250 = 0
-F4 - F5 cos(60º) + F6 cos(70º) = 0

Nodo 5: V2 + F6 sen(70º) = 0
-F7 – F6 cos(70º) = 0

Use una de las funciones instrumentadas en Python para obtener la solución.

30. Se desea conocer la corriente i que fluye en un circuito que incluye una fuente de voltaje V y resistencias R como se indica en el gráfico:



El análisis se basa en leyes básicas de la física:
a) La suma de la caída de voltaje en un circuito cerrado es cero
b) El voltaje a través de una resistencia es el producto de su magnitud multiplicado por el valor de la corriente.

Para determinar el valor de la corriente en cada uno de los ciclos cerrados se conviene que la corriente es positiva si el sentido es opuesto al sentido del reloj.
Circuito cerrado a la izquierda: **$10\ i_1 + 20\ i_1 - 10\ i_2 - 20\ i_3 = 0$**
Circuito cerrado arriba a la derecha: **$30\ i_2 + 20\ i_2 - 10\ i_1 = 0$**
Circuito cerrado abajo a la derecha: **$20\ i_3 + 40\ i_3 - 20\ i_2 = 120$**

a) Obtenga la solución con el método de Gauss
b) Suponga que la resistencia con valor 10 se quemó y que debe reemplazarla con otra resistencia cuyo valor es 11. Determine como afecta este cambio a la respuesta.

6 INTERPOLACIÓN

Para introducir este capítulo se analiza un problema para el cual se requiere como modelo un polinomio de interpolación.

Problema. La inversión en la promoción para cierto producto en una región ha producido los siguientes resultados:

(x, f): (5, 1.2), (10, 2.5), (15, 4.3), (20, 4.0)

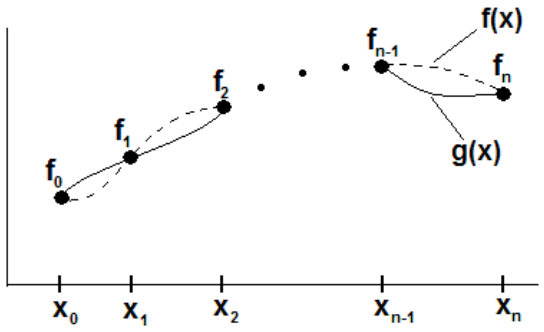
En donde **x:** tiempo invertido en la promoción del producto en horas
f: porcentaje de incremento en las ventas del producto

Se desea usar los datos para determinar la cantidad óptima de horas que deberían invertirse en regiones similares para maximizar la eficiencia de las ventas. Con los datos obtenidos se debe construir alguna función que permita obtener la respuesta y también estimar el error en el resultado.

6.1 El polinomio de interpolación

La interpolación es una técnica matemática que permite describir un conjunto de puntos mediante alguna función. También tiene utilidad cuando se desea aproximar una función por otra función matemáticamente más simple.

Dados los puntos (x_i, f_i), $i = 0, 1, 2, \ldots, n$ que pertenecen a alguna función **f** que supondremos desconocida pero diferenciable, se debe encontrar alguna función **g** para aproximarla.



La función **g** debe incluir a los puntos dados: $g(x_i) = f_i$, $i = 0, 1, 2, \ldots, n$

Por simplicidad se elegirá como función **g**, un polinomio de grado no mayor a n que incluya a todos los **n** puntos dados. Se requiere determinar los coeficientes:

$$g(x) = p_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

Pn(x) se denomina polinomio de interpolación. Los polinomios son funciones continuas, derivables e integrables.
En la siguiente secciones se revisan algunos fundamentos básicos muy conocidos relacionados con el polinomio de interpolación.

6.1.1 Existencia del polinomio de interpolación

El polinomio de interpolación **p** debe incluir a cada punto dado:

$$x=x_0: \quad p_n(x_0) = a_0x_0^n+a_1x_0^{n-1}+\ldots+a_{n-1}x_0+a_n = f_0$$

$$x=x_1: \quad p_n(x_1) = a_0x_1^n+a_1x_1^{n-1}+\ldots+a_{n-1}x_1+a_n = f_1$$

$$\ldots$$

$$x=x_n: \quad p_n(x_n) = a_0x_n^n+a_1x_n^{n-1}+\ldots+a_{n-1}x_n+a_n = f_n$$

Expresado en notación matricial

$$\begin{bmatrix} x_0^n & x_0^{n-1} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \ldots & x_1 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \ldots \\ f_n \end{bmatrix}$$

La matriz de los coeficientes es muy conocida y tiene nombre propio: Matriz de Vandermonde. Para calcular su determinante existe una fórmula específica. La demostración puede ser hecha con inducción matemática.

Sea $\mathbf{D} = \begin{bmatrix} x_0^n & x_0^{n-1} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \ldots & x_1 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \end{bmatrix}$ entonces $|D| = \prod_{j=0,\, j<i}^{n} (x_j - x_i),\ i = 1,2,\ldots,n$

Ejemplo. Formule el determinante de la matriz de Vandermonde para **n=2**

$$D = \begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix} \Rightarrow |D| = \prod_{j=0,j<i}^{2} (x_j - x_i) = (x_0 - x_1)(x_0 - x_2)(x_1 - x_2)$$

De la definición anterior, se concluye que el determinante de esta matriz será diferente de cero si los valores de **X** dados no están repetidos. Por lo tanto, una condición necesaria para la existencia del polinomio de interpolación es que las abscisas de los datos dados sean diferentes entre si.

Ejemplo. Dados los siguientes puntos: **(2, 5), (4, 6), (5, 3).** Halle la matriz de Vandermonde, calcule el número de condición y encuentre el polinomio de interpolación.

```
>>> from numpy import*
>>> x=array([2,4,5],float)
>>> f=array([5,6,3],float)
>>> d=vander(x)                    Matriz de Vandermonde
>>> print(d)
[[  4.   2.   1.]
 [ 16.   4.   1.]
 [ 25.   5.   1.]]
>>> c=linalg.cond(d,inf)           Número de condición
>>> print(c)
341.0
>>> a=linalg.solve(d,f)            Obtención de la solución
>>> print(a)
[-1.16666667, 7.5, -5.33333333]
```

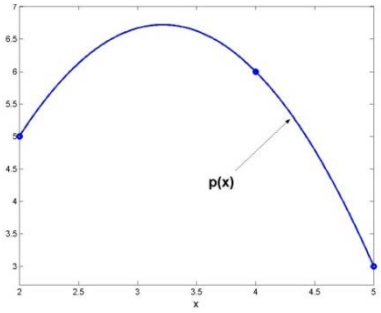
Los resultados son los coeficientes del polinomio de interpolación:

$$\begin{bmatrix} 2^2 & 2 & 1 \\ 4^2 & 4 & 1 \\ 5^2 & 5 & 1 \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}=\begin{bmatrix} 5 \\ 6 \\ 3 \end{bmatrix} \Rightarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}=\begin{bmatrix} -1.1666... \\ 7.5 \\ -5.3333... \end{bmatrix}$$

Polinomio de interpolación:

$$p(x) = -1.666x^2 + 7.5x - 5.3333$$

Su gráfico:



La matriz de Vandermonde es una matriz desbalanceada y su número de condición es alto, lo cual no necesariamente significa que el polinomio de interpolación no es un buen modelo para representar a los datos.

Es conveniente establecer algún criterio para estimar la sensibilidad del polinomio de interpolación a los errores en los datos.

Existen métodos para encontrar el polinomio de interpolación con mayor eficiencia y sin usar la matriz de Vandermonde.

Primero se confirmará de otra manera que el polinomio de interpolación es único.

6.1.2 Unicidad del polinomio de interpolación con diferentes métodos

Dados los datos (x_i, f_i), $i = 0, 1, 2, \ldots, n$, el polinomio de interpolación que incluye a todos los puntos es único.

Demostración
Suponer que usando los mismos datos y con métodos diferentes se han obtenido dos polinomios de interpolación: $p(x)$, y $q(x)$. Ambos polinomios deben incluir a los puntos dados:

$$p(x_i) = f_i \ , \ i=0, 1, 2, \ldots, n$$
$$q(x_i) = f_i \ , \ i=0, 1, 2, \ldots, n$$

Sea la función $h(x) = p(x) - q(x)$. Esta función también debe ser un polinomio y de grado no mayor a n. Al evaluar la función h en los puntos dados se tiene:

$$h(x_i) = p(x_i) - q(x_i) = f_i - f_i = 0, \ i=0, 1, 2, \ldots, n$$

Esto significaría que el polinomio h cuyo grado no es mayor a n tendría $n+1$ raíces, contradiciendo el teorema fundamental del álgebra. La única posibilidad es que la función h sea la función cero, por lo tanto, $\forall x \in \Re[h(x)=0] \Rightarrow \forall x \in \Re[p(x)-q(x)=0] \Rightarrow \forall x \in \Re[p(x) = q(x)]$

6.2 El polinomio de interpolación de Lagrange

Dados los datos (x_i, f_i), $i = 0, 1, 2, \ldots, n$. La siguiente fórmula permite obtener el polinomio de interpolación:

Definición: Polinomio de Interpolación de Lagrange

$$p_n(x) = \sum_{i=0}^{n} f_i \, L_i(x)$$

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}, \quad i = 0, 1, \ldots, n$$

Para probar que esta definición proporciona el polinomio de interpolación, la expresamos en forma desarrollada:

$$p_n(x) = f_0L_0(x) + f_1L_1(x) + \ldots + f_{i-1}L_{i-1}(x) + f_iL_i(x) + f_{i+1}L_{i+1}(x) + \ldots + f_nL_n(x) \qquad (1)$$

$$L_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}, \quad i = 0,1,\ldots,n \qquad (2)$$

De la definición (2):

$L_i(x_i) = 1$, $i=0,1,\ldots,n$ (Por simplificación directa)

$L_i(x_k) = 0$, $k=0,1,\ldots,n$; $k \neq i$, (Contiene un factor nulo para algún $j = 0, 1, \ldots, n$; $j \neq i$)

Sustituyendo en la definición (1) de $p_n(x)$ se obtiene

$$p_n(x_0) = f_0L_0(x_0) + f_1L_1(x_0) + \ldots + f_nL_n(x_0) = f_0(1) + f_1(0) + \ldots + f_n(0) = f_0$$
$$p_n(x_1) = f_0L_0(x_1) + f_1L_1(x_1) + \ldots + f_nL_n(x_1) = f_0(0) + f_1(1) + \ldots + f_n(0) = f_1$$
$$\ldots$$
$$p_n(x_n) = f_0L_0(x_n) + f_1L_1(x_n) + \ldots + f_nL_n(x_n) = f_0(0) + f_1(0) + \ldots + f_n(1) = f_n$$

En general:

$$p_n(x_i) = f_i \; ; \quad i = 0, 1, \ldots, n$$

Por otra parte, cada factor $L_i(x)$ es un polinomio de grado n, por lo tanto $p_n(x)$ también será un polinomio de grado n con lo cual la demostración está completa.

6.2.1 Algoritmo del polinomio de interpolación de Lagrange

Algoritmo: Lagrange
Restricción: No se verifica la existencia del polinomio de interpolación
Entra: x_i, f_i Puntos (datos)
Sale: p Polinomio de interpolación
n ← numeracíon del último punto
p ← 0
Para i ← 0, 1, …, n
L ← 1
Para j ← 0, 1, …, n; (j ≠ i)
$$L \leftarrow L \ \frac{x-x_j}{x_i-x_j}$$
Fin
p ← p + f_i L
Fin

Ejemplo. Dados los siguientes puntos: **(2, 5), (4, 6), (5, 3).** Con la fórmula de Lagrange, encuentre el polinomio de interpolación que incluye a estos puntos.

Solución

$$p_2(x) = \sum_{i=0}^{2} f_i L_i(x) = f_0L_0(x) + f_1L_1(x) + f_2L_2(x) = 5L_0(x) + 6L_1(x) + 3L_2(x)$$

$$L_i(x) = \prod_{j=0, j\neq i}^{2} \frac{(x-x_j)}{(x_i-x_j)}, \quad i=0, 1, 2$$

$$L_0(x) = \prod_{j=0, j\neq 0}^{2} \frac{(x-x_j)}{(x_0-x_j)} = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-4)(x-5)}{(2-4)(2-5)} = \frac{x^2-9x+20}{6}$$

$$L_1(x) = \prod_{j=0, j\neq 1}^{2} \frac{(x-x_j)}{(x_1-x_j)} = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x-2)(x-5)}{(4-2)(4-5)} = \frac{x^2-7x+10}{-2}$$

$$L_2(x) = \prod_{j=0, j\neq 2}^{2} \frac{(x-x_j)}{(x_2-x_j)} = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(x-2)(x-4)}{(5-2)(5-4)} = \frac{x^2-6x+8}{3}$$

Sustituir en el polinomio y simplificar:

$$p_2(x) = 5\left(\frac{x^2-9x+20}{6}\right) + 6\left(\frac{x^2-7x+10}{-2}\right) + 3\left(\frac{x^2-6x+8}{3}\right) = -\frac{7}{6}x^2 + \frac{15}{2}x - \frac{16}{3}$$

Se puede verificar que este polinomio incluye a los tres puntos dados.

Si únicamente se desea evaluar el polinomio de interpolación, entonces no es necesario obtener las expresiones algebraicas $L_i(x)$. Conviene sustituir desde el inicio el valor de **x** para obtener directamente el resultado numérico.

Ejemplo. Dados los siguientes puntos: **(2, 5), (4, 6), (5, 3).** Con la fórmula de Lagrange evalúe en x=3, con el polinomio de interpolación que incluye a estos tres puntos dados.

Solución

$$p_2(3) = \sum_{i=0}^{2} f_i L_i(3) = f_0 L_0(3) + f_1 L_1(3) + f_2 L_2(3) = 5L_0(3) + 6L_1(3) + 3L_2(3)$$

$$L_i(3) = \prod_{j=0, j\neq i}^{2} \frac{(3-x_j)}{(x_i-x_j)}, \quad i=0, 1, 2$$

$$L_0(3) = \prod_{j=0, j\neq 0}^{2} \frac{(3-x_j)}{(x_0-x_j)} = \frac{(3-x_1)(3-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(3-4)(3-5)}{(2-4)(2-5)} = 1/3$$

$$L_1(3) = \prod_{j=0, j\neq 1}^{2} \frac{(3-x_j)}{(x_1-x_j)} = \frac{(3-x_0)(3-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(3-2)(3-5)}{(4-2)(4-5)} = 1$$

$$L_2(3) = \prod_{j=0, j\neq 2}^{2} \frac{(3-x_j)}{(x_2-x_j)} = \frac{(3-x_0)(3-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(3-2)(3-4)}{(5-2)(5-4)} = -1/3$$

Finalmente se sustituyen los valores dados de **f**

$$p_2(3) = 5(1/3) + 6(1) + 3(-1/3) = 20/3$$

6.2.2 Eficiencia del método de Lagrange

La fórmula de Lagrange involucra dos ciclos anidados que dependen del número de datos **n**:

$$p_n(x) = \sum_{i=0}^{n} f_i \, L_i(x)$$

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}, \quad i = 0, 1, \ldots, n$$

La evaluación de esta fórmula es un método directo que incluye un ciclo para sumar **n+1** términos. Cada factor $L_i(x)$ de esta suma requiere un ciclo con **2n** multiplicaciones, por lo tanto, la eficiencia de este algoritmo es $T(n) = 2n(n+1) = O(n^2)$, significativamente mejor que el método matricial que requiere resolver un sistema lineal cuya eficiencia es $T(n) = O(n^3)$.

6.2.3 Instrumentación computacional del método de Lagrange

La siguiente función recibe un conjunto de puntos y entrega el polinomio de interpolación en forma algebraica usando la fórmula de Lagrange. Opcionalmente, si la función recibe como parámetro adicional un valor escalar para interpolar, el resultado entregado es un valor numérico, resultado de la interpolación.

x, y: puntos base para la interpolación
u: valor para interpolar (parámetro opcional).
t: variable simbólica para construir el polinomo de interpolación

```
from sympy import*
def lagrange(x,y,u=None):
    n=len(x)
    t=Symbol('t')
    p=0
    for i in range(n):
        L=1
        for j in range(n):
            if j!=i:
                L=L*(t-x[j])/(x[i]-x[j])
        p=p+y[i]*L
        p=expand(p)
    if u==None:
        return p
    elif type(u)==list:
        v=[]
        for i in range(len(u)):
            v=v+[p.subs(t,u[i])]
        return v
    else:
        return p.subs(t,u)
```

Esta instrumentación permite explorar algunas de las características de Python para manejo matemático simbólico:

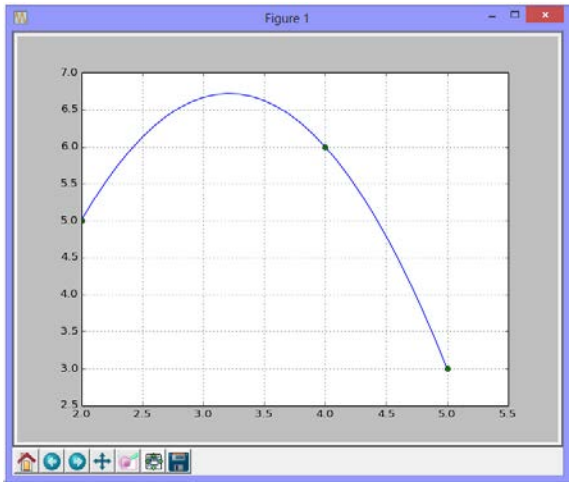
Ejemplo. Dados los puntos: **(2, 5), (4, 6), (5, 3)** use la función Lagrange para encontrar el polinomio de interpolación.

```
>>> from lagrange import*
>>> x=[2,4,5]
>>> y=[5,6,3]
>>> p=lagrange(x,y)                    Polinomio de interpolación
>>> print(p)
-7*t**2/6 + 15*t/2 - 16/3
>>> r=lagrange(x,y,4.25)               Evaluar el polinomio en otro punto
```

```
>>> print(r)
5.46875000000000
>>> r=lagrange(x,y,[3.15,3.74,4.25])
>>> print(r)
[6.71541666666667, 6.39780000000000, 5.46875000000000]
```

Gráfico del polinomio con la librería Pylab de Python:

```
>>> import pylab as pl                          Librería para graficar
>>> u=pl.arange(2,5.1,0.1)
>>> v=lagrange(x,y,list(u))                     Puntos de p(u)
>>> pl.plot(u,v)                                Gráfico del polinomio
>>> pl.plot(x,y,'o')                            Gráfico de los puntos
>>> pl.grid(True)
>>> pl.show()
```



Encuentre el valor de x para el cual p(x) = 4:

```
>>> from biseccion import*
>>> def g(t): return -7*t**2/6 + 15*t/2 - 28/3      Ecuación g(x) = p(x)-4 = 0
>>> s=biseccion(g,4,5,0.0001)                      Uso del método de la bisección
>>> print(s)
4.74127197265625
>>>
```

6.3 Interpolación múltiple

Se puede extender la interpolación a funciones de más variables. El procedimiento consiste en interpolar en una variable, fijando los valores de las otras variables y luego combinar los resultados. En esta sección se usará el polinomio de Lagrange en un ejemplo que contiene datos de una función que depende de dos variables. No es de interés encontrar la forma analítica del polinomio de interpolación que tendría términos con más de una variable.

Ejemplo. Se tienen tabulados los siguientes datos **f(x,y)** de una función **f** que depende de las variables independientes **x**, **y**. Usar **todos** los datos disponibles para estimar mediante interpolación polinomial el valor de **f(3,12)**

x \ y	5	10	15	20
2	3.7	4.2	5.8	7.1
4	4.1	5.3	6.1	7.9
6	5.6	6.7	7.4	8.2

Solución

Existen solo tres datos en la dirección X, por lo tanto conviene interpolar primero en X con los datos de cada columna. Se requiere un polinomio de grado dos. No es necesario costruir el polinomio. Se sustituye directamente el valor

$$p_2(x) = \sum_{i=0}^{2} f_i L_i(x) = f_0 L_0(x) + f_1 L_1(x) + f_2 L_2(x)\ ;$$

$$L_i(x) = \prod_{j=0, j\neq i}^{2} \frac{(x-x_j)}{(x_i-x_j)}\ ,\quad i=0,\ 1,\ 2$$

Interpolación en x=3:

$$L_0(x=3) = \prod_{j=0, j\neq 0}^{2} \frac{(3-x_j)}{(x_0-x_j)} = \frac{(3-x_1)(3-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(3-4)(3-6)}{(2-4))(2-6)} = 3/8$$

$$L_1(x=3) = \prod_{j=0, j\neq 1}^{2} \frac{(3-x_j)}{(x_1-x_j)} = \frac{(3-x_0)(3-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(3-2)(3-6)}{(4-2))(4-6)} = 3/4$$

$$L_2(x=3) = \prod_{j=0, j\neq 2}^{2} \frac{(3-x_j)}{(x_1-x_j)} = \frac{(3-x_0)(3-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(3-2)(3-4)}{(6-2))(6-4)} = -1/8$$

$$p_2(x=3) = f_0 L_0(x=3) + f_1 L_1(x=3) + f_2 L_2(x=3) = f_0(3/8) + f_1(3/4) + f_2(-1/8)$$

Para completar la interpolación en **x**, ahora se deben sustituir los datos de cada columna:

y=5: $p_2(x=3) = 3.7(3/8) + 4.1(3/4) + 5.6(-1/8) = 3.7625$

y=10: $p_2(x=3) = 4.2(3/8) + 5.3(3/4) + 6.7(-1/8) = 4.7125$

y=15: $p_2(x=3) = 5.8(3/8) + 6.1(3/4) + 7.4(-1/8) = 5.8250$

y=20: $p_2(x=3) = 7.1(3/8) + 7.9(3/4) + 8.2(-1/8) = 7.5625$

Interpolación en y=12:

Con los cuatro resultados se interpola en **y = 12** con un polinomio de tercer grado:

y	5	10	15	20
f(3, y)	3.7625	4.7125	5.8250	7.5625

$$p_3(y) = \sum_{i=0}^{3} f_i L_i(y) = f_0 L_0(y) + f_1 L_1(y) + f_2 L_2(y) + f_3 L_3(y);$$

$$L_i(y) = \prod_{j=0, j\neq i}^{3} \frac{(y-y_j)}{(y_i-y_j)}, \quad i=0, 1, 2, 3$$

Se sustituye directamente el valor para interpolar con la otra variable: **y = 12**

$$L_0(y=12) = \prod_{j=0, j\neq 0}^{3} \frac{(12-y_j)}{(y_0-y_j)} = \frac{(12-y_1)(12-y_2)(12-y_3)}{(y_0-y_1)(y_0-y_2)(y_0-y_3)} = \frac{(12-10)(12-15)(12-20)}{(5-10)(5-15)(5-20)} = -8/125$$

$$L_1(y=12) = \prod_{j=0, j\neq 1}^{3} \frac{(12-y_j)}{(y_1-y_j)} = \frac{(12-y_0)(12-y_2)(12-y_3)}{(y_1-y_0)(y_1-y_2)(y_1-y_3)} = \frac{(12-5)(12-15)(12-20)}{(10-5)(10-15)(10-20)} = 84/125$$

$$L_2(y=12) = \prod_{j=0, j\neq 2}^{3} \frac{(12-y_j)}{(y_2-y_j)} = \frac{(12-y_0)(12-y_1)(12-y_3)}{(y_2-y_0)(y_2-y_1)(y_0-y_3)} = \frac{(12-5)(12-10)(12-20)}{(15-5)(15-10)(15-20)} = 56/125$$

$$L_3(y=12) = \prod_{j=0, j\neq 3}^{3} \frac{(12-y_j)}{(y_2-y_j)} = \frac{(12-y_0)(12-y_1)(12-y_2)}{(y_3-y_0)(y_3-y_1)(y_3-y_2)} = \frac{(12-5)(12-10)(12-15)}{(20-5)(20-10)(20-15)} = -7/125$$

$$p_3(y=12) = f_0 L_0(y=12) + f_1 L_1(y=12) + f_2 L_2(y=12) + f_3 L_3(y=12)$$

$$= (3.7625)(-8/125) + (4.7125)(84/125) + (5.8250)(56/125) + (7.5625)(-7/125) = 5.1121$$

$$f(3, 12) \cong 5.1121$$

6.3.1 Instrumentación computacional del método de Lagange con dos variables

Para interpolar en dos o más variable, se puede usar la función **lagrange** para interpolar en una variable. Al aplicarla en cada dirección se obtienen los resultados parciales. Interpolando con estos resultados producirá el resultado final.

Para el ejemplo anterior:

```
>>> from lagrange import*
>>> x=[2,4,6]                    Interpolaciones parciales en x para cada columna de y
>>> f=[3.7,4.1,5.6]
>>> r1=lagrange(x,f,3)
>>> f=[4.2,5.3,6.7]
>>> r2=lagrange(x,f,3)
>>> f=[5.8,6.1,7.4]
>>> r3=lagrange(x,f,3)
>>> f=[7.1,7.9,8.2]
>>> r4=lagrange(x,f,3)

>>> y=[5,10,15,20]               Interpolación en  y  con los resultados parciales
>>> f=[r1,r2,r3,r4]
>>> p=lagrange(y,f,12)           Resultado final
>>> p
5.11210000000000
```

6.4 Error en la interpolación

Para entender este concepto usaremos un polinomio de interpolación para aproximar a una función conocida. Así puede determinarse en forma exacta el error en la interpolación.

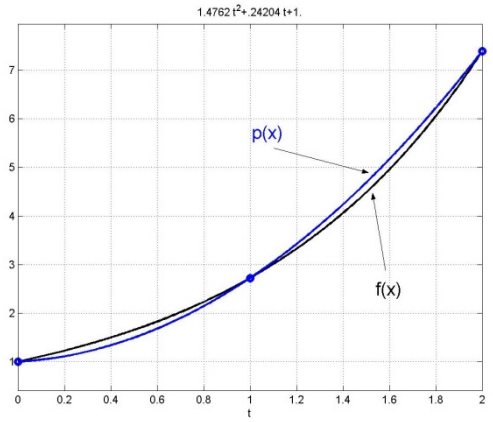
Ejemplo. Suponga que se desea aproximar la función $f(x)=e^x$, $0 \le x \le 2$, con un polinomio de segundo grado.

a) Encuentre el error en la aproximación cuando **x = 0.5**
b) Encuentre el máximo error en la aproximación

Solución

Para obtener este polinomio tomamos tres puntos de la función f: $(0, e^0), (1, e^1), (2, e^2)$ y usamos la fórmula de Lagrange para obtener el polinomio de interpolación, con cinco decimales:

$$p_2(x) = 1.4762x^2 + 0.24204x+1$$



Si se aproxima f(x) con $p_2(x)$ se introduce un error cuyo valor es $f(x) - p_2(x)$

$$f(0.5)-p_2(0.5) = e^{0.5}-1.4762(0.5)^2-0.24204(0.5)-1 = 0.1587$$

Si se desea conocer cual es el máximo error en la aproximación, se debe resolver la ecuación

$$\frac{d}{dx}(f(x)-p_2(x)) = 0 \Rightarrow e^x - 2.9524x - 0.2420 = 0$$

Con un método numérico se encuentra que el máximo ocurre cuando **x = 1.6064**.

Entonces el máximo valor del error es: $f(1.6064) - p_2(1.6064) = -0.2133$

En el caso general, se tienen los puntos **(x_i, f_i), i=0, 1, ..., n,** pero **f** es desconocida

Sea **$p_n(x)$** el polinomio de interpolación, tal que **$p_n(x_i) = f_i$, i=0, 1, ..., n**

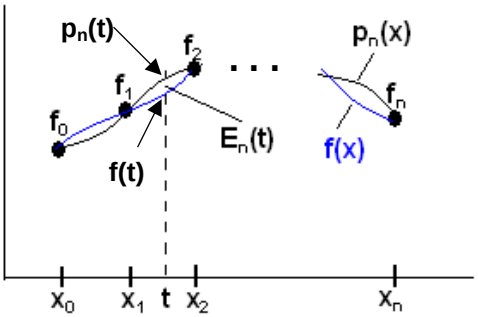
Suponer que se desea evaluar **f** en un punto **t** usando **$p_n(t)$** como una aproximación:

$f(t) \cong p_n(t)$, $t \neq x_i$, i=0, 1, ..., n

Definición. Error en la interpolación

$E_n(t) = f(t) - p_n(t)$

Representación gráfica del error en la interpolación



Siendo **f** desconocida, no es posible conocer el error pues el valor exacto **f(t)** es desconocido.

En los puntos dados el error es cero: **$E_n(x_i) = f(x_i) - p_n(x_i) = 0$, i=0,1,..., n**, pero es importante establecer una cota para el error en la interpolación en cualquier punto **t: $E_n(t) = f(t) - p_n(t)$**

6.4.1 Una fórmula para aproximar el error en la interpolación

En las aplicaciones normalmente se conocen puntos de una función desconocida **f** que se desea aproximar. Si con estos puntos se construye un polinomio de interpolación para aproximar a la función y se usa el polinomio para interpolar, es importante estimar la magnitud del error, es decir la diferencia entre el valor interpolado y el valor exacto de **f**.

A continuación se describe un conocido procedimiento para estimar este error.

Sea $\mathbf{g(x) = \prod_{i=0}^{n}(x - x_i) = (x-x_0)(x-x_1) \ldots (x-x_n),}$ un polinomio de grado **n+1**

Se define una función con algunas propiedades de interés:

$$\mathbf{h(x) = f(x) - p_n(x) - g(x)E_n(t)/g(t)}$$

1) h es diferenciable si suponemos que f es diferenciable
2) $h(t) = 0$
3) $h(x_i) = 0,\ i = 0, 1, \ldots, n$

Por lo tanto, **h** es diferenciable. La ecuación
$h(x) = 0$ tendrá n+2 ceros en el intervalo $[x_0, x_n]$

Aplicando sucesivamente el Teorema de Rolle:
$h^{(n+1)}(x) = 0$ tendrá al menos 1 cero en el intervalo $[x_0, x_n]$

Sea $z \in [x_0, x_n]$ el valor de x tal que $h^{(n+1)}(z) = 0$

La n+1 derivada de h:
$$h^{(n+1)}(x) = f^{(n+1)}(x) - 0 - (n+1)!\ E_n(t)/g(t)$$

Al evaluar esta derivada en z:
$$h^{(n+1)}(z) = 0 = f^{(n+1)}(z) - 0 - (n+1)!\ E_n(t)/g(t)$$

Se obtiene finalmente:
$$E_n(t) = g(t)\ f^{(n+1)}(z)/(n+1)!,\ \ t \neq x_i,\ \ z \in [x_0, x_n]$$

Definición. Fórmula para aproximar el error en el polinomio de interpolación

$$\mathbf{E_n(x) = g(x)\ f^{(n+1)}(z)/(n+1)!,\ \ x \neq x_i,\ \ z \in [x_0, x_n]}$$

Siendo $\mathbf{g(x) = \prod_{i=0}^{n}(x - x_i) = (x-x_0)(x-x_1) \ldots (x-x_n)}$

Para utilizar esta fórmula es necesario poder estimar el valor de $\mathbf{f^{(n+1)}(z)}$.

En otra sección se revisará una técnica para estimar $\mathbf{f^{(n+1)}(z)}$ con los puntos de **f**.

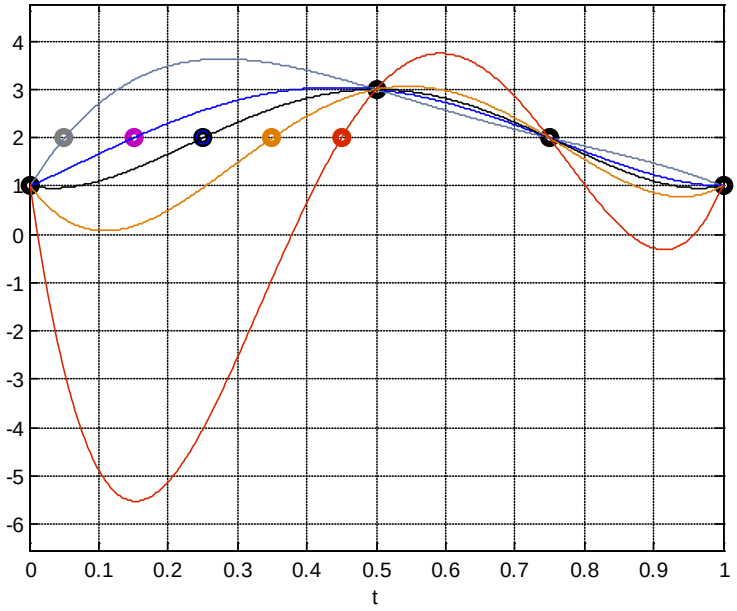
6.4.2 Elección de los puntos base para construir el polinomio de interpolación

La elección de los puntos para construir el polinomio de interpolación puede tener un efecto importante en el polinomio resultante.

Ejemplo. En el siguiente ejemplo se han escrito cinco conjuntos de datos $(x_i, f_k(x_i))$, $i = 0, 1, 2, 3, 4$ que pertenecen a las funciones $f_k(x)$, $k = 1, 2, 3, 4, 5$ respectivamente. Los valores tabulados de cada función tienen una tendencia creciente en el intervalo $x \in [0, 0.5]$ y decreciente en el intervalo $x \in [0.5, 1]$. Cada conjunto tiene cinco datos. Se diferencian únicamente en el valor de la abscisa del segundo punto, resaltado en color rojo:

```
1) (0, 1), (0.05, 2), (0.5, 3), (0.75, 2), (1, 1)
2) (0, 1), (0.15, 2), (0.5, 3), (0.75, 2), (1, 1)
3) (0, 1), (0.25, 2), (0.5, 3), (0.75, 2), (1, 1)
4) (0, 1), (0.35, 2), (0.5, 3), (0.75, 2), (1, 1)
5) (0, 1), (0.45, 2), (0.5, 3), (0.75, 2), (1, 1)
```

La siguiente figura muestra el gráfico de los cinco polinomios de grado cuatro construidos con los cinco puntos de cada conjunto de datos.



Se observa que a medida que la abscisa del segundo punto se acerca a alguno de los puntos adyacentes, el polinomio toma una forma distorsionada, cada vez más lejana de la tendencia creciente – decreciente que se había supuesto para cada función.

Los mejores resultados parecen corresponder a la elección de valores para las abscisas con una distribución regular, con la misma separación (puntos de color negro en el gráfico):

Sin embargo, existe una forma óptima de elegir los puntos base para construir el polinomio de interpolación. Estos puntos corresponden a las raíces del polinomio de Chebyshev.

Definición: Polinomio de Chebyshev de grado **n**. Construcción recursiva:

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad n = 2, 3, 4, \ldots$$

El matemático Chebyshev demostró que si los puntos $x_0, x_1, \ldots, x_n$ son las raìces del polinomio de Chebyshev de grado **n+1**, entonces el componente **g(x)** en la fórmula del error en la interpolación se minimiza. Para cualquier otra elección de los puntos base, **g(x)** tendrá un valor mayor. Esto permite fijar una cota para el error en el polinomio de interpolación de grado **n**:

$$g(x) = \prod_{i=0}^{n}(x - x_i) = (x-x_0)(x-x_1) \ldots (x-x_n)$$
$$E_n(x) = g(x)\, f^{(n+1)}(z)/(n+1)!, \quad x \neq x_i, \quad z \in (x_0, x_n)$$

Si se usan como puntos base las raices del polinomio de Chebysev de grado **n+1** en el intervalo $-1 \leq x \leq 1$, entonces **g(x)** tiene un valor máximo:

$$|g(x)| \leq \frac{1}{2^n}$$

Por ende, se puede acotar el error en el polinomio de interpolación de grado **n**, en el intervalo $-1 \leq x \leq 1$:

$$E_n(x) = |f(x) - p_n(x)| \leq \frac{1}{2^n(n+1)!} \max_{-1 \leq z \leq 1} |f^{(n+1)}(z)|, \quad -1 \leq x \leq 1$$

Cálculo de las raíces del polinomio de Chebyshev

La siguiente fórmula permite calcular las raíces del polinomio de Chebyshev de grado **n+1** para usarlos como puntos base para el polinomio de interpolación de grado **n** en el intervalo $-1 \leq x \leq 1$

$$x_j = \cos\left(\frac{2j+1}{2(n+1)}\pi\right), \quad j = 0,1,\ldots,n$$

Mediante una sustitución se pueden obtener las raíces del polinomio de Chebyshev de grado **n+1** para construir el polinomio de interpolación de grado **n** en un intervalo arbitrario $a \leq x \leq b$

$$x_j = \frac{b+a}{2} + \frac{b-a}{2}\cos\left(\frac{2j+1}{2(n+1)}\pi\right), \quad j = 0,1,\ldots,n$$

En este caso la cota para **g(x)** en el intervalo $a \leq x \leq b$, es:

$$|g(x)| \leq \frac{1}{2^{2n+1}}(b-a)^{n+1}, \quad a \leq x \leq b$$

Se puede acotar el error en el polinomio de interpolación de grado **n**, en el intervalo $a \leq x \leq b$:

$$E_n(x) = |f(x) - p_n(x)| \leq \frac{(b-a)^{n+1}}{2^{2n+1}(n+1)!} \max_{a \leq z \leq b} |f^{(n+1)}(z)|, \quad a \leq x \leq b$$

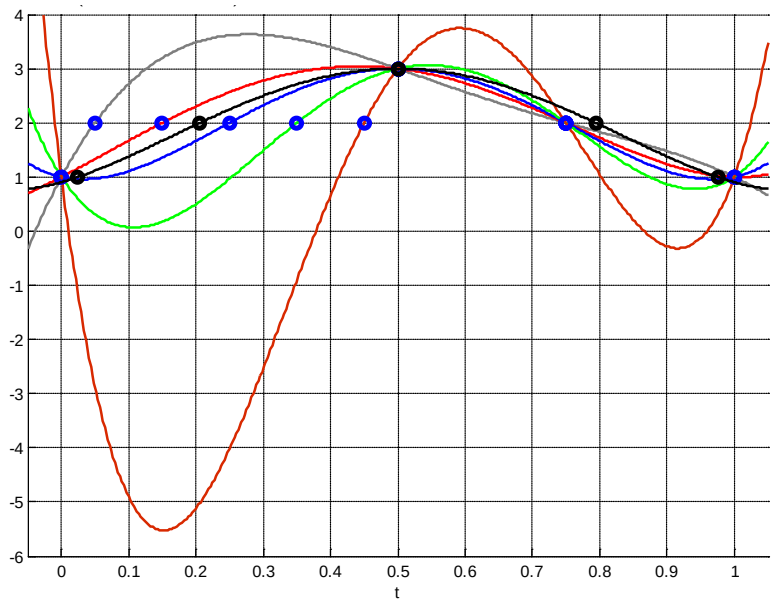
Ejemplo. Calcular las raíces del polinomio de Chebyshev de grado **5** para usarlas como los puntos base para construir el polinomio de interpolación de grado **n=4** en el intervalo $\mathbf{0 \le x \le 1}$

a=0, b=1, n=4: $\mathbf{x_j = \frac{1}{2} + \frac{1}{2}\cos(\frac{2j+1}{10}\pi), \quad j = 0,1,2,3,4}$

Las raíces calculadas, con 6 decimales:

x_0 = 0.024471
x_1 = 0.206107
x_2 = 0.5
x_3 = 0.793892
x_4 = 0.975528

En la siguiente figura se representa el polinomio construido con los puntos de las raíces del polinomio de Chebyshev, en color negro, junto con los polinomios de interpolación de la figura anterior.



Con los puntos de Chebyshev la distorsión del polinomio de interpolación es menor que con los otros conjuntos de puntos. La distorsión también se reduce en la región fuera del dominio del polinomio de interpolación. De paso, se observa el riesgo de usar el polinomio de interpolación para aproximar a una función, fuera del dominio de los puntos base.

Ejemplo. En el siguiente ejemplo se coloca un polinomio de grado **n** = **4** sobre cinco puntos tomados de una función especificada. Se prueban tres casos con diferentes valores para las abscisas. Los polinomios obtenidos se comparan con el gráfico de la función:

Dada la función: $f(x) = sen(x^2)+1,\ x \in [0, 2]$

Casos para comparar con la función especificada:

Caso 1. Puntos base espaciados de manera equidistante.

```
x = [0, 0.5, 1, 1.5, 2]
```

Caso 2. Puntos base espaciados de manera irregular, sesgada hacia puntos adyacentes

```
x = [0, 0.6, 1.6, 1.9, 2]
```

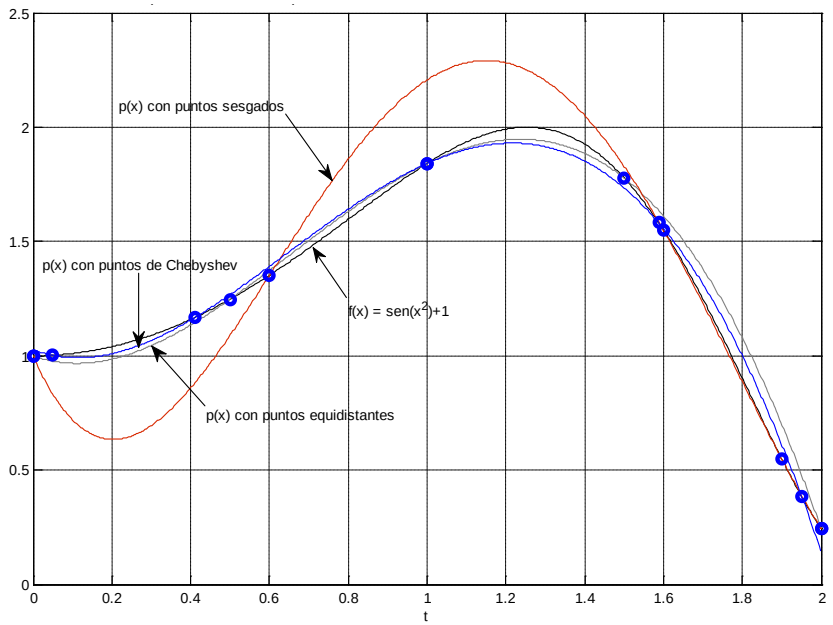
Caso 3. Puntos base correspondientes a las raíces del polinomio de Chebyshev de grado **5**

a=0, b=2, n=4: $x_j = 1 + \cos(\frac{2j+1}{10}\pi), \quad j = 0,1,2,3,4$

Las raíces calculadas, con 6 decimales:

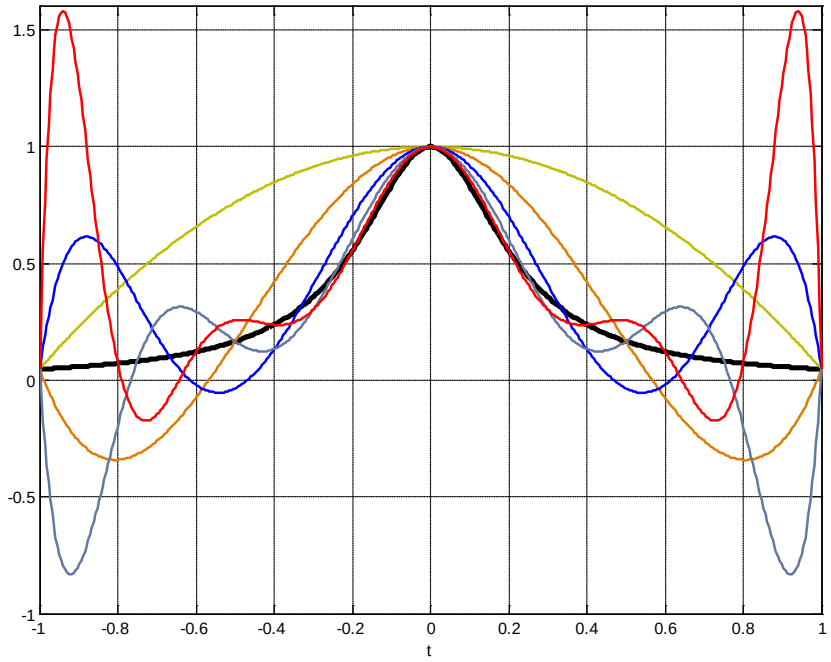
$x_0 = 0.048943$
$x_1 = 0.412214$
$x_2 = 1.0$
$x_3 = 1.587785$
$x_4 = 1.951056$

Gráfico de los tres polinomios de interpolación obtenidos y la función especificada



Por otra parte, no es conveniente utilizar muchos puntos para construir el polinomio de interpolación. El siguiente ejemplo es parecido al ejemplo propuesto por el matemático Runge para demostrar que NO es cierto que el polinomio de interpolación tiende en forma contínua a la función que se trata de aproximar, a medida que la cantidad de puntos crece.

Ejemplo. La función que se representa es $f(x) = 1/(1+20x^2)$, $x \in [-1, 1]$. Esta función se muestra en color negro. En colores están varios polinomios de interpolación y se observa que mientras más puntos de la función se utilizan para construir el polinomio, más se distorsiona, con lo cual el error de interpolación crece cada vez más.



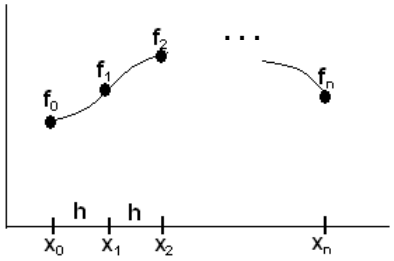
Sin embargo, es cierto que mientras más puntos de una función se utlicen para construir polinomios de interpolación por segmentos, estos polinomios pueden aproximar mejor al perfil de la función que representa.

El caso más simple es usar segmentos de rectas para conectar cada par de puntos pero entre segmentos se pierde la continuidad en pendiente y curvatura. Un método muy importante que evita esta pérdida de continuidad es el Trazador Cúbico, el cual será revisado en otro capítulo.

6.5 Diferencias finitas

Las siguientes definiciones establecen algunas relaciones simples entre los puntos dados. Estas definiciones tienen varias aplicaciones en análisis numérico.

Suponer que se tienen los puntos **(x_i, f_i), i=0, 1, ..., n,** tales que las abscisas están espaciadas regularmente en una distancia **h:** **$x_{i+1} - x_i = h$, i=0, 1, ..., n-1**



Definiciones

Primera diferencia finita avanzada: **$\Delta^1 f_i = f_{i+1} - f_i$, i = 0,1,2, ...**

$\Delta^1 f_0 = f_1 - f_0$
$\Delta^1 f_1 = f_2 - f_1$, etc

Segunda diferencia finita avanzada: **$\Delta^2 f_i = \Delta^1 f_{i+1} - \Delta^1 f_i$, i = 0,1,2, ...**

$\Delta^2 f_0 = \Delta^1 f_1 - \Delta^1 f_0$
$\Delta^2 f_1 = \Delta^1 f_2 - \Delta^1 f_1$, etc

Las diferencias finitas se pueden expresar con los puntos dados:

$$\Delta^2 f_0 = \Delta^1 f_1 - \Delta^1 f_0 = (f_2 - f_1) - (f_1 - f_0) = f_2 - 2f_1 + f_0$$

K-ésima diferencia finita avanzada:

$\Delta^k f_i = \Delta^{k-1} f_{i+1} - \Delta^{k-1} f_i$, i=0,1,2, ..., k=1, 2, 3, ... con **$\Delta^0 f_i = f_i$**

Es útil tabular las diferencias finitas en un cuadro como se muestra a continuación:

i	x_i	f_i	$\Delta^1 f_i$	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
0	x_0	f_0	$\Delta^1 f_0$	$\Delta^2 f_0$	$\Delta^3 f_0$	...
1	x_1	f_1	$\Delta^1 f_1$	$\Delta^2 f_1$	...	...
2	x_2	f_2	$\Delta^1 f_2$	...	...	...
3	x_3	f_3	...	...	...	...
...	...	...	...	...	...	...

Cada diferencia finita se obtiene restando los dos valores consecutivos de la columna anterior.

Ejemplo. Tabule las diferencias finitas correspondientes a los siguientes datos

(1.0, 5), (1.5, 7), (2.0, 10), (2.5, 8)

i	x_i	f_i	$\Delta^1 f_i$	$\Delta^2 f_i$	$\Delta^3 f_i$
0	1.0	5	2	1	-6
1	1.5	7	3	-5	
2	2.0	10	-2		
3	2.5	8			

6.5.1 Relación entre derivadas y diferencias finitas

Desarrollo de la serie de Taylor de una función que suponemos diferenciable, **f** alrededor de un punto x_0 a una distancia **h**, con un término:

$$f(x_0+h) = f_1 = f_0 + h\, f'(z), \text{ para algún } z \in [x_0, x_1]$$

De donde:

$$f'(z) = \frac{f_1 - f_0}{h} = \frac{\Delta^1 f_0}{h}, \text{ para algún } z \in [x_0, x_1]$$

Es el Teorema del Valor Medio. Este teorema de existencia, con alguna precaución, se usa como una aproximación para la primera derivada en el intervalo especificado:

$\frac{\Delta^1 f_0}{h}$ es una aproximación para **f'** en el intervalo $[x_0, x_1]$.

Desarrollo de la serie de Taylor de la función **f**, que suponemos diferenciable, alrededor del punto x_1 hacia ambos a una distancia **h**, con dos términos:

(1) $$f_2 = f_1 + hf'_1 + \frac{h^2}{2!} f''(z_1), \text{ para algún } z_1 \in [x_1, x_2]$$

(2) $$f_0 = f_1 - hf'_1 + \frac{h^2}{2!} f''(z_2), \text{ para algún } z_2 \in [x_0, x_1]$$

Sumando (1) y (2), sustituyendo la suma $f''(z_1) + f''(z_2)$ por un valor promedio $2f''(z)$ y despejando:

$$f''(z) = \frac{f_2 - 2f_1 + f_0}{h^2} = \frac{\Delta^2 f_0}{h^2}, \text{ para algún } z \in [x_0, x_2]$$

$\frac{\Delta^2 f_0}{h^2}$ es una aproximación para **f''** en el intervalo $[x_0, x_2]$

En general

Definición. Relación entre derivadas y diferencias finitas

$$f^{(n)}(z) = \frac{\Delta^n f_0}{h^n}, \text{ para algún } z \text{ en el intervalo } [x_0, x_n].$$

$\frac{\Delta^n f_0}{h^n}$ es una aproximación para $f^{(n)}$ en el intervalo $[x_0, x_n]$.

Si suponemos que **f** tiene derivadas cuyos valores no cambian significativamente en el intervalo considerado, esta fórmula puede usarse para aproximar sus derivadas. El valor de **h** debe ser pequeño, pero si es demasiado pequeño puede aparecer el error de redondeo al restar números muy cercanos y la estimación de las derivadas de orden alto no será aceptable.

Si se usa esta aproximación para acotar el error, se debería tomar el mayor valor de la diferencia finita tabulada respectiva. En general, esta aproximación es aceptable si los valores de las diferencias finitas en cada columna no cambian significactivamente y tienden a reducirse.

6.5.2 Diferencias finitas de un polinomio

Si la función **f** de donde provienen los datos es un polinomio de grado n, entonces la n-ésima diferencia finita será constante y las siguientes diferencias finitas se anularán.

Demostración:

Sea f un polinomio de grado n: $f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n$

Su n-ésima derivada es una constante: $f^{(n)}(x) = n!\, a_0$

Por lo tanto, la n-ésima diferencia finita también será constante: $\Delta^n f_i = h^n f^{(n)}(x) = h^n\, n!\, a_0$

Ejemplo. Tabule las diferencias finitas de **$f(x) = 2x^2 + x + 1$**, para **$x = -2, -1, 0, 1, 2, 3$**

I	x_i	f_i	$\Delta^1 f_i$	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
0	-2	7	-5	4	0	0
1	-1	2	-1	4	0	0
2	0	1	3	4	0	
3	1	4	7	4		
4	2	11	11			
5	3	22				

El polinomio de interpolación, por la propiedad de unicidad, coincidirá con el polinomio **f**.

Ejemplo. Verifique si la siguiente función puede expresarse mediante un polinomio en el mismo dominio.

$$f(x) = 2x + \sum_{i=1}^{x} i^2 \quad , x = 1, 2, 3, ...$$

Solución

La función con el sumatorio expresado en forma desarrollada:

$$f(x) = 2x + (1^2 + 2^2 + 3^2 + ... + x^2), \quad x = 1, 2, 3, \ldots$$

Tomando algunos puntos de esta función, tabulamos las diferencias finitas:

I	x_i	f_i	$\Delta^1 f_i$	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
0	1	3	6	5	2	0
1	2	9	11	7	2	0
2	3	20	18	9	2	
3	4	38	27	11		
4	5	65	38			
5	6	103				

Siendo f una expresión algebraica, y observado que la tercera diferencia finita es constante, se puede concluir que es un polinomio de grado tres. Para encontrar el polinomio de interpolación podemos usar la conocida fórmula de Lagrange tomando cuatro puntos tabulados:

```
>>> from lagrange import*
>>> x=[1,2,3,4]
>>> f=[3,9,20,38]
>>> p=lagrange(x,f)
>>> p                                   Polinomio que representa a  f
t**3/3 + t**2/2 + 13*t/6
>>> r=lagrange(x,f,100)                 Evaluar f(100)
>>> r
338550.000000000
```

6.6 El polinomio de interpolación de diferencias finitas avanzadas

Se puede obtener el polinomio de interpolación desde la tabla de diferencias finitas.

Dado un conjunto de puntos (x_i, f_i), **i=0, 1,…, n** espaciados en forma regular en una distancia **h** y que provienen de una función desconocida **f(x),** pero supuestamente diferenciable, se desea obtener el polinomio de interpolación. Si se tienen tabuladas las diferencias finitas se puede obtener el polinomio de interpolación mediante un procedimiento de generalización.

Suponer que se tienen dos puntos (x_0, f_0), (x_1, f_1) con los cuales se debe obtener el polinomio de primer grado, es decir, la ecuación de una recta:

$$p_1(x) = a_0 + a_1(x-x_0)$$

Sustituyendo los dos puntos y despejando a_0 y a_1 se obtienen, agregando la notación anterior:

$$a_0 = f_0$$
$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = \frac{\Delta f_0}{h}$$
$$p_1(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0)$$

En el caso de tener tres puntos (x_0, f_0), (x_1, f_1), (x_2, f_2), se debe obtener un polinomio de segundo grado. Se propone la siguiente forma algebraica para este polinomio:

$$p_2(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1)$$

Sustituyendo los tres puntos y despejando a_0, a_1 y a_2 se obtienen, incluyendo la notación anterior:

$$a_0 = f_0$$
$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = \frac{\Delta f_0}{h}$$
$$a_2 = \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = \frac{\frac{\Delta f_1}{h} - \frac{\Delta f_0}{h}}{2h} = \frac{\Delta f_1 - \Delta f_0}{2h^2} = \frac{\Delta^2 f_0}{2h^2}$$
$$p_2(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2h^2}(x - x_0)(x - x_1)$$

Se puede generalizar para un conjunto de puntos (x_i, f_i), **i=0, 1,…, n**:

Definición. Polinomio de diferencias finitas avanzadas o polinomio de Newton

$$p_n(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2!h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 f_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2) + \ldots + \frac{\Delta^n f_0}{n!h^n}(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

Si las diferencias finitas están tabuladas en forma de un triángulo, los coeficientes del polinomio de interpolación se toman directamente de la primera fila del cuadro de datos (fila sombreada):

i	x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$		$\Delta^n f_i$
0	x_0	f_0	$\Delta^1 f_0$	$\Delta^2 f_0$	$\Delta^3 f_0$	…	$\Delta^n f_0$
1	x_1	f_1	$\Delta^1 f_1$	$\Delta^2 f_1$	$\Delta^3 f_n$	…	…
2	x_2	f_2	$\Delta^1 f_2$	$\Delta^2 f_2$	…	…	…
3	x_3	f_3	$\Delta^1 f_3$	…	…	…	…
…	…	…	…	…	…	…	…
n	x_n	f_n	…	…	…	…	…

Ejemplo. Dados los siguientes puntos: **(2, 5), (3, 6), (4, 3), (5, 2),** encuentre el polinomio de interpolación que incluye a los cuatro datos usando el método de diferencias finitas

Solución

El método de diferencias finitas es aplicable pues **h** es constante e igual a **1**

Tabla de diferencias finitas:

i	x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$
0	2	5	1	-4	6
1	3	6	-3	2	
2	4	3	-1		
3	5	2			

Polinomio de tercer grado de diferencias finitas:

$$p_3(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2!h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 f_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2)$$

Reemplazando los coeficientes, tomados de la primera fila, y simplificando:

$$p_3(x) = 5 + \frac{1}{1}(x - 2) + \frac{-4}{2(1^2)}(x - 2)(x - 3) + \frac{6}{3!(1^3)}(x - 2)(x - 3)(x - 4)$$

$$= x^3 - 11x^2 + 37x - 33$$

6.6.1 Práctica computacional

Obtención del polinomio de diferencias finitas en la ventana interactiva de Python

Dados los puntos: **(2, 5), (3, 6), (4, 3), (5, 2)**

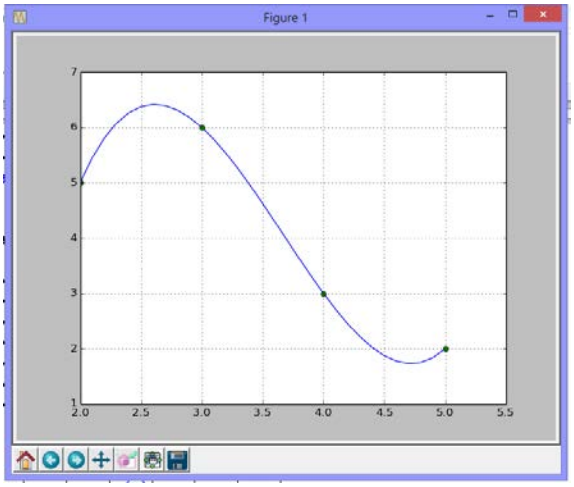
Obtenga el polinomio de interpolación en la ventana interactiva de Python mediante sustitución directa en la fórmula:

$$p_3(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2!h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 f_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2)$$

```
>>> from sympy import*
>>> t=Symbol('t')
>>> p=5+1/1*(t-2)+(-4)/(2*1**2)*(t-2)*(t-3)+6/(6*1**3)*(t-2)*(t-3)*(t-4)
>>> p
1.0*t + (-2.0*t + 4.0)*(t - 3) + (t - 4)*(t - 3)*(1.0*t - 2.0) + 3.0
>>> p=simplify(p)
>>> p
1.0*t**3 - 11.0*t**2 + 37.0*t - 33.0
```

Graficación del polinomio y los puntos

```
>>> import pylab as pl
>>> def f(t): return t**3 - 11*t**2 + 37*t - 33
>>> t=pl.arange(2,5.1,0.1)
>>> pl.plot(t,f(t))                          Gráfico del polinomio
>>> x=[2,3,4,5]
>>> y=[5,6,3,2]
>>> pl.plot(x,y,'o')                         Gráfico de los puntos
>>> pl.grid(True)
>>> pl.show()
```



6.6.2 Eficiencia del polinomio de interpolación de diferencias finitas

Forma original del polinomio de interpolación de diferencias finitas avanzadas:

$$p_n(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2!h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 f_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2) + \ldots$$
$$+ \frac{\Delta^n f_0}{n!h^n}(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

La evaluación de este polinomio es un método directo en el que se suman **n+1** términos. Cada término de esta suma requiere una cantidad variable de multiplicaciones en las que aparece el valor x que se interpola. Entonces la eficiencia de este algoritmo es: **$T(n) = O(n^2)$,** similar **a** la fórmula de Lagrange, y significativamente mejor que el método matricial que involucra la resolución de un sistema de ecuaciones lineales cuya eficiencia es **$T(n) = O(n^3)$**. El polinomio de diferencias finitas puede escribirse en forma recurrente:

$$p_n(x) = f_0 + \frac{(x - x_0)}{h}(\Delta f_0 + \frac{(x - x_1)}{2h}(\Delta^2 f_0 + \frac{(x - x_2)}{3h}(\Delta^3 f_0 + \ldots + \frac{(x - x_{n-2})}{(n-1)h}(\Delta^{n-1} f_0 + \frac{(x - x_{n-1})}{nh}\Delta^n f_0)\ldots)))$$

Entonces, el polinomio puede expresarse mediante un algoritmo recursivo:

$$p_0 = \Delta^n f_0$$

$$p_1 = \Delta^{n-1} f_0 + \frac{(x - x_{n-1})}{nh} p_0$$

$$p_2 = \Delta^{n-2} f_0 + \frac{(x - x_{n-2})}{(n-1)h} p_1$$

.

.

$$p_{n-1} = \Delta^1 f_0 + \frac{(x - x_1)}{2h} p_{n-2}$$

$$p_n = \Delta^0 f_0 + \frac{(x - x_0)}{h} p_{n-1}, \qquad \text{siendo } \Delta^0 f_0 = f_0$$

La evaluación del polinomio con este procedimiento requiere **2n** sumas y restas y **3n** multiplicaciones y divisiones: **$T(n) = O(n)$**. Esto constituye una mejora significativa con respecto a los métodos anteriores. Sin embargo, la tabulación de las diferencias finitas tiene eficiencia **$O(n^2)$** pero únicamente contiene restas y debe calcularse una sola vez para interpolar en otros puntos.

Ejemplo. Dados los siguientes puntos: **(1.2, 5), (1.4, 6), (1.6, 3), (1.8, 2),** use el algoritmo recursivo anterior para evaluar en **x=1.5** el polinomio de interpolación que incluye a los cuatro datos.

Solución

Tabla de diferencias finitas:

i	x_i	f_i	$\Delta^1 f_i$	$\Delta^2 f_i$	$\Delta^3 f_i$
0	1.2	5	1	-4	6
1	1.4	6	-3	2	
2	1.6	3	-1		
3	1.8	2			

$$p_0 = \Delta^3 f_0 \qquad\qquad p_0 = 6$$

$$p_1 = \Delta^2 f_0 + \frac{(x - x_2)}{3h} p_0 \qquad\qquad p_1 = -4 + \frac{(1.5 - 1.6)}{3(0.2)} 6 = -5$$

$$p_2 = \Delta^1 f_0 + \frac{(x - x_1)}{2h} p_1 \quad \Rightarrow \quad p_2 = 1 + \frac{(1.5 - 1.4)}{2(0.2)} (-5) = -0.25$$

$$p_3 = \Delta^0 f_0 + \frac{(x - x_0)}{h} p_2 \qquad\qquad p_3 = 5 + \frac{(1.5 - 1.2)}{0.2} (-0.25) = 4.625$$

6.6.3 El error en el polinomio de interpolación de diferencias finitas

Polinomio de interpolación de diferencias finitas avanzadas:

$$p_n(x) = f_0 + \frac{\Delta f_0}{h}(x - x_0) + \frac{\Delta^2 f_0}{2!h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 f_0}{3!h^3}(x - x_0)(x - x_1)(x - x_2) + \ldots$$
$$+ \frac{\Delta^n f_0}{n!h^n}(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

Se tiene el error en el polinomio de interpolación:

$$E_n(x) = g(x)\frac{f^{(n+1)}(z)}{(n+1)!}, \quad x \neq x_i, \quad z \in [x_0, x_n], \quad \text{siendo} \quad g(x) = (x-x_0)(x-x_1)\ldots(x-x_n)$$

Si los puntos están regularmente espaciados a una distancia **h**, se estableció una relación entre las derivadas de **f** y las diferencias finitas:

$$f^{(n)}(t) = \frac{\Delta^n f_0}{h^n}, \quad \text{para algún } t \text{ en el dominio de los datos dados}$$

Se usa como una aproximación para la derivada si no cambia significativamente y **h** es pequeño.

Extendiendo esta relación a la siguiente derivada y sustituyendo en la fórmula del error en la interpolación

$$E_n(x) = g(x)\frac{f^{(n+1)}(z)}{(n+1)!} \cong g(x)\frac{\Delta^{n+1}f_0}{h^{n+1}(n+1)!} = (x-x_0)(x-x_1)...(x-x_n)\frac{\Delta^{n+1}f_0}{h^{n+1}(n+1)!}$$

Si se compara con el polinomio de diferencias finitas avanzadas se observa que el error en la interpolación es aproximadamente igual al siguiente término que no es incluido en el polinomio de interpolación.

Definición. Estimación del error en el polinomio de interpolación de diferencias finitas

$$E_n(x) \cong \frac{\Delta^{n+1}f_0}{h^{n+1}(n+1)!}(x-x_0)(x-x_1)...(x-x_n)$$

Si se toman **n+1** puntos para construir el polinomio de interpolación de grado **n**, y los puntos provienen de un polinomio de grado **n**, entonces el $f^{(n+1)}(\ )$ es cero y también $\Delta^n f$, por lo tanto $E_n(x)$ también es cero, en forma consistente con la propiedad de unicidad del polinomio de interpolación.

Ejemplo. Aplicar la definición para estimar el error en la interpolación con los datos:

i	x_i	f_i	$\Delta^1 f_i$	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
0	0.0	1.000000	0.110517	0.023247	0.003666	0.000516
1	0.1	1.110517	0.133764	0.026913	0.004182	
2	0.2	1.244281	0.160677	0.031095		
3	0.3	1.404958	0.191772			
4	0.4	1.596730				

En la tabla puede observarse que las diferencias finitas tienden a reducir su valor, entonces un polinomio de interpolación es una aproximación adecuada para esta función. Adicionalmente, las diferencias finitas en cada columna tienen valores de similar magnitud, por lo tanto se pueden usar para estimar a las derivadas.

El grado del polinomio de interpolación depende del error que toleramos y su valor está relacionado directamente con el orden de la diferencia finita incluida.

Supongamos que deseamos evaluar el polinomio de interpolación en **x = 0.08** usando el polinomio de diferencias finitas avanzadas de segundo grado.

$$f(x) \cong p_2(x) = f_0 + \frac{\Delta f_0}{h}(x-x_0) + \frac{\Delta^2 f_0}{2!h^2}(x-x_0)(x-x_1)$$

$$f(0.08) \cong p_2(0.08) = 1 + \frac{0.110517}{0.1}(0.08-0) + \frac{0.023247}{2(0.1)^2}(0.08-0)(0.08-0.1) = 1.086554$$

Estimar el error en la interpolación:

$$E_2(x) \cong \frac{\Delta^3 f_0}{3!h^3}(x-x_0)(x-x_1)(x-x_2)$$

$$E_2(0.08) \cong \frac{0.003666}{3!0.1^3}(0.08-0)(0.08-0.1)(0.08-0.2) = 0.0001173$$

Para comparar, calculemos el valor exacto de **f(0.08)** con la función de la cual fueron tomados los datos: $\mathbf{f(x) = x\,e^x + 1}$

$$\mathbf{f(0.08) = 0.08\,e^{0.08} + 1 = 1.086663}$$

El error exacto es

$$\mathbf{1.083287 - 1.086554 = 0.0001089}$$

El valor calculado con el polinomio de interpolación de segundo grado concuerda muy bien con el valor exacto.

6.6.4 Forma estándar del polinomio de interpolación de diferencias finitas

El polinomio de interpolación de diferencias finitas avanzadas:

$$p_n(x) = f_0 + \frac{\Delta f_0}{h}(x-x_0) + \frac{\Delta^2 f_0}{2!h^2}(x-x_0)(x-x_1) + \frac{\Delta^3 f_0}{3!h^3}(x-x_0)(x-x_1)(x-x_2) + \ldots + \frac{\Delta^n f_0}{n!h^n}(x-x_0)(x-x_1)\ldots(x-x_{n-1})$$

Puede re-escribirse usando la siguiente sustitución: $S = \frac{x-x_0}{h}$

$$\frac{\Delta f_0}{h}(x-x_0) = \Delta f_0 S$$

$$\frac{\Delta^2 f_0}{2!h^2}(x-x_0)(x-x_1) = \frac{\Delta^2 f_0}{2!h^2}(x-x_0)(x-(x_0+h)) = \frac{\Delta^2 f_0}{2!h^2}\frac{(x-x_0)}{h}\frac{(x-x_0-h)}{h} = \frac{\Delta^2 f_0}{2!}S(S-1)$$

. .
. (sucesivamente)
. .

Mediante recurrencia se puede generalizar:

$$p_n(S) = f_0 + \Delta f_0 S + \frac{\Delta^2 f_0}{2!}S(S-1) + \frac{\Delta^3 f_0}{3!}S(S-1)(S-2) + \ldots + \frac{\Delta^n f_0}{n!}S(S-1)(S-2)\ldots(S-n+1)$$

Con la definición del coeficiente binomial

$$\binom{S}{i} = \frac{S(S-1)(S-2)\ldots(S-i+1)}{i!}$$

Se obtiene una forma compacta para el polinomio de interpolación de diferencias finitas:

Definición. Forma estándar del polinomio de interpolación de diferencias finitas

$$p_n(S) = f_0 + \binom{S}{1}\Delta f_0 + \binom{S}{2}\Delta^2 f_0 + \binom{S}{3}\Delta^3 f_0 + \ldots + \binom{S}{n}\Delta^n f_0 = \sum_{i=0}^{n}\binom{S}{i}\Delta^i f_0$$

También se puede expresar con esta notación el error en la interpolación:

$$E_n(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_n)}{(n+1)!}f^{(n+1)}(z) \cong \frac{(x-x_0)(x-x_1)\ldots(x-x_n)}{(n+1)!}\frac{\Delta^{n+1}f_0}{h^{n+1}}$$

Sustituyendo la definición $S = \frac{x-x_0}{h}$

$$E_n(S) \cong S(S-1)(S-2)\ldots(S-n)\frac{\Delta^{n+1}f_0}{(n+1)!}$$

Finalmente se puede expresar de la siguiente forma

Definición. Estimación del error en el polinomio de interpolación de diferencias finitas

$$E_n(S) \cong \binom{S}{n+1}\Delta^{n+1}f_0, \quad S = \frac{x-x_0}{h}, \quad x \neq x_i$$

6.6.5 Otras formas del polinomio de interpolación de diferencias finitas

Para algunas aplicaciones es de interés definir el polinomio de interpolación con referencia a un punto x_i con las diferencias finitas expresadas con puntos anteriores:

$$p_n(x) = p_n(S) = f_i + \binom{S}{1}\Delta f_{i-1} + \binom{S+1}{2}\Delta^2 f_{i-2} + \binom{S+2}{3}\Delta^3 f_{i-3} + \ldots + \binom{S+n-1}{n}\Delta^n f_{i-n}$$

$$E = \binom{S+n}{n+1}h^{n+1}y^{(n+1)}(z), \qquad s = \frac{x-x_i}{h}$$

Si el punto de referencia es x_{i+1} con las diferencias finitas expresadas con los puntos anteriores, el polinomio de interpolación toma la siguiente forma:

$$p_n(x) = p_n(S) = f_{i+1} + \binom{S}{1}\Delta f_i + \binom{S+1}{2}\Delta^2 f_{i-1} + \binom{S+2}{3}\Delta^3 f_{i-2} + \ldots + \binom{S+n-1}{n}\Delta^n f_{i-n+1}$$

$$E = \binom{S+n}{n+1}h^{n+1}y^{(n+1)}(z), \qquad s = \frac{x-x_{i+1}}{h}$$

6.7 El polinomio de interpolación de diferencias divididas

Dado un conjunto de puntos (x_i, f_i), $i=0, 1,\ldots, n$ espaciados en forma arbitraria y que provienen de una función desconocida $f(x)$ pero supuestamente diferenciable, se desea obtener el polinomio de interpolación.

Una forma alternativa al método de Lagrange es el polinomio de diferencias divididas cuya fórmula se la puede obtener mediante un procedimiento de recurrencia. La fórmula resultante es más eficiente que la fórmula de Lagrange pues en promedio requiere menos operaciones aritméticas.

Suponer que se tienen dos puntos $(x_0, f_0), (x_1, f_1)$ con los cuales se debe obtener el polinomio de primer grado, es decir, la ecuación de una recta:

$$p_1(x) = a_0 + a_1(x-x_0)$$

Sustituyendo los dos puntos, despejando a_0 y a_1 y adoptando una nueva notación se obtiene:

$$a_0 = f_0 = f[x_0]$$
$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_{0:1}]$$
$$p_1(x) = f[x_0] + f[x_{0:1}](x - x_0)$$

$f[x_{0:1}]$ denota la primera diferencia dividida en el rango $[x_0, x_1]$

La diferencia dividida $f[x_{0:1}]$ es una aproximación para $f'( )$ en el intervalo $[x_0, x_1]$

En el caso de tener tres puntos $(x_0, f_0), (x_1, f_1), (x_2, f_2),$ se debe obtener un polinomio de segundo grado. Se propone la siguiente forma para este polinomio:

$$p_2(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1)$$

Sustituyendo los tres puntos y despejando a_0, a_1 y a_2 se obtienen:

$$a_0 = f_0 = f[x_0]$$
$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_{0:1}]$$
$$a_2 = \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = \frac{f[x_{1:2}] - f[x_{0:1}]}{x_2 - x_0} = f[x_{0:2}]$$
$$p_2(x) = f[x_0] + f[x_{0:1}](x - x_0) + f[x_{0:2}](x - x_0)(x - x_1)$$

$f[x_{0:2}]$ denota la segunda diferencia dividida en el rango $[x_0, x_2]$

Al extender la recurrencia y generalizar al conjunto de puntos (x_i, f_i), **i=0, 1,…, n** se tiene:

$$f[x_{0:n}] = \frac{f[x_{1:n}] - f[x_{0:n-1}]}{x_n - x_0}$$

n-ésima diferencia dividida en el punto x_0 en el rango $[x_0, x_n]$

Definición. Polinomio de diferencias divididas

$$p_n(x) = f[x_0] + f[x_{0:1}](x - x_0) + f[x_{0:2}](x - x_0)(x - x_1) + f[x_{0:3}](x - x_0)(x - x_1)(x - x_2) + \ldots$$
$$+ f[x_{0:n}](x - x_0)(x - x_1) \ldots (x - x_{n-1})$$

Es conveniente tabular las diferencias divididas en forma de un triángulo en el que cada columna a la derecha se obtiene de la resta de los dos valores inmediatos de la columna anterior. Este resultado debe dividirse para la longitud del rango de los datos incluidos en el rango.

Los coeficientes del polinomio de interpolación serán los valores resultantes colocados en la primera fila del cuadro tabulado (fila sombreada):

Para la tabulación, la fórmula se puede extender a cada punto **i**:

$$f[x_{i:i+k}] = \frac{f[x_{i+1:i+k}] - f[x_{i:i+k-1}]}{x_{i+k} - x_i}$$

k-ésima diferencia dividida en el punto **i** en el rango $[x_i, x_{i+k}]$

i	x_i	$f[x_i]$	$f[x_{i:i+1}]$	$f[x_{i:i+2}]$	$f[x_{i:i+3}]$	$f[x_{i:i+4}]$
0	x_0	$f[x_0]$	$f[x_{0:1}]$	$f[x_{0:2}]$	$f[x_{0:3}]$	$f[x_{0:4}]$
1	x_1	$f[x_1]$	$f[x_{1:2}]$	$f[x_{1:3}]$	$f[x_{1:4}]$	
2	x_2	$f[x_2]$	$f[x_{2:3}]$	$f[x_{2:4}]$		
3	x_3	$f[x_3]$	$f[x_{3:4}]$			
4	x_4	$f[x_4]$				
…	…	…				

Ejemplo. Dados los siguientes puntos: **(2, 5), (4, 6), (5, 3), (7, 2),** encuentre y grafique el polinomio de interpolación que incluye a los cuatro datos, usando el método de diferencias divididas:

Tabla de diferencias divididas:

i	x_i	$f[x_i]$	$f[x_{i:i+1}]$	$f[x_{i:i+2}]$	$f[x_{i:i+3}]$
0	2	5	$\frac{6-5}{4-2}=\frac{1}{2}$	$\frac{-3-1/2}{5-2}=-\frac{7}{6}$	$\frac{5/6-(-7/6)}{7-2}=\frac{2}{5}$
1	4	6	$\frac{3-6}{5-4}=-3$	$\frac{-1/2-(-3)}{7-4}=\frac{5}{6}$	
2	5	3	$\frac{2-3}{7-5}=-\frac{1}{2}$		
3	7	2			

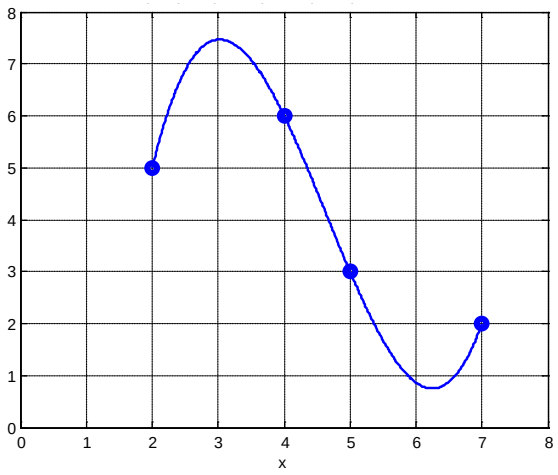
Polinomio de tercer grado de diferencias divididas:

$$p_3(x) = f[x_0] + f[x_{0:1}](x-x_0) + f[x_{0:2}](x-x_0)(x-x_1) + f[x_{0:3}](x-x_0)(x-x_1)(x-x_2)$$

Reemplazando los coeficientes, tomados de la primera fila, y simplificando:

$$p_3(x) = 5 + (1/2)(x-2) + (-7/6)(x-2)(x-4) + (2/5)(x-2)(x-4)(x-5)$$
$$= \frac{2}{5}x^3 - \frac{167}{30}x^2 + \frac{227}{10}x - \frac{64}{3}$$

Gráfico del polinomio:



6.7.1 El error en el polinomio de interpolación de diferencias divididas

Polinomio de diferencias divididas

$$p_n(x) = f[x_0] + f[x_{0:1}](x - x_0) + f[x_{0:2}](x - x_0)(x - x_1) + f[x_{0:3}](x - x_0)(x - x_1)(x - x_2) + \ldots + f[x_{0:n}](x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

Error en el polinomio de interpolación:

$$E_n(x) = g(x)\frac{f^{(n+1)}(z)}{(n+1)!},\ x \neq x_i,\ z \in [x_0, x_n],\ \text{siendo}\ g(x) = (x-x_0)(x-x_1)\ldots(x-x_n)$$

Si los puntos estuviesen igualmente espaciados en una distancia **h** se tendría

$$f[x_{0:n}] = \frac{f[x_{1:n}] - f[x_{0:n-1}]}{x_n - x_0} = \frac{\Delta^n f_0}{n!h^n} \cong \frac{f^{(n)}(z)}{n!}$$

En el polinomio de diferencias divididas, **h** sería el cociente promedio de las **n** distancias entre las abscisas de los datos.

Extendiendo esta relación a la siguiente derivada y sustituyendo en la fórmula del error en la interpolación

$$E_n(x) = g(x)\frac{f^{(n+1)}(z)}{(n+1)!} \cong g(x)f[x_{0:n+1}] = (x - x_0)(x - x_1)\ldots(x - x_n)f[x_{0:n+1}]$$

Si se compara con el polinomio de diferencias divididas se observa que el error en el polinomio de interpolación es aproximadamente igual al siguiente término del polinomio que no es incluido.

6.8 El polinomio de mínimos cuadrados

Dados los puntos (x_i, y_i), $i = 1, 2, ..., n$, que corresponden a observaciones o mediciones. Si se considera que estos datos no son exactos y que es de interés modelar únicamente su tendencia, entonces el polinomio de interpolación no es una buena opción. Una alternativa es el polinomio de mínimos cuadrados. Estos polinomios tienen mejores propiedades para realizar predicciones o extrapolaciones.

Si los datos tienen una tendencia lineal, la mejor recta será aquella que se coloca entre los puntos de tal manera que se minimizan las distancias de los puntos dados a esta recta, la cual se denomina recta de mínimos cuadrados. Se la obtiene con el siguiente procedimiento:

Sea $p(x) = a_1 + a_2 x$ la recta de mínimos cuadrados

Para cada valor x_i se tiene el dato y_i mientras que el valor $p(x_i)$ es obtenido con la recta de mínimos cuadrados.

Si $e_i = y_i - p(x_i)$ es la diferencia entre el dato y el punto de la recta de mínimos cuadrados, entonces el objetivo es minimizar e_i^2 para todos los puntos. El cuadrado es para considerar la distancia, no importa si el punto está sobre o debajo de la recta

Criterio de para obtener la recta de mínimos cuadrados

Minimizar $$S = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - p(x_i))^2 = \sum_{i=1}^{n} (y_i - a_1 - a_2 x)^2 \quad .$$

Para minimizar esta función S cuyas variables son a_1, a_2 se debe derivar con respecto a cada variable e igualar a cero:

$$\frac{\partial S}{\partial a_1} = 0 : \quad \frac{\partial}{\partial a_1} \sum_{i=1}^{n} (y_i - a_1 - a_2 x_i)^2 = 0 \Rightarrow n a_1 + a_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$\frac{\partial S}{\partial a_2} = 0 : \quad \frac{\partial}{\partial a_2} \sum_{i=1}^{n} (y_i - a_1 - a_2 x_i)^2 = 0 \Rightarrow a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i$$

De esta manera se obtiene el sistema de ecuaciones lineales:

$$a_1 n + a_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i$$

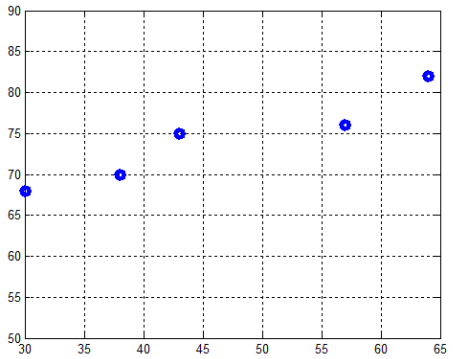
De donde se obtienen los coeficientes a_1, a_2 para la recta de mínimos cuadrados:

$$p(x) = a_1 + a_2 x$$

Ejemplo. Los siguientes datos corresponden a una muestra de 5 estudiantes que han tomado cierta materia. Los datos incluyen la calificación parcial y la calificación final. Se pretende encontrar un modelo que permita predecir la calificación final que obtendría un estudiante dada su calificación parcial.

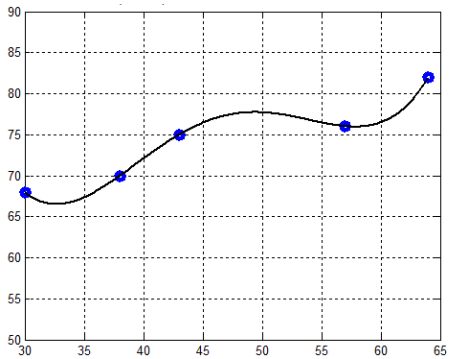
Estudiante (i)	Nota Parcial (x_i)	Nota final (y_i)
1	43	75
2	64	82
3	38	70
4	57	76
5	30	68

Representación de los datos en un diagrama



Se observa que los datos tienen aproximadamente una tendencia lineal creciente

Gráfico del polinomio de interpolación para representar a los cinco datos:



El polinomio de interpolación no es útil en este caso pues no muestra la tendencia de los datos que parece lineal creciente, es decir a mayor calificación en el examen parcial, mayor debería ser la calificación en el examen final. Se trata entonces de construir un modelo que exprese esta tendencia de manera simple.

Obtención de la recta de mínimos cuadrados

$$5a_1 + a_2\sum_{i=1}^{5} x_i = \sum_{i=1}^{5} y_i$$

$$a_1\sum_{i=1}^{5} x_i + a_2\sum_{i=1}^{5} x_i^2 = \sum_{i=1}^{5} x_i y_i$$

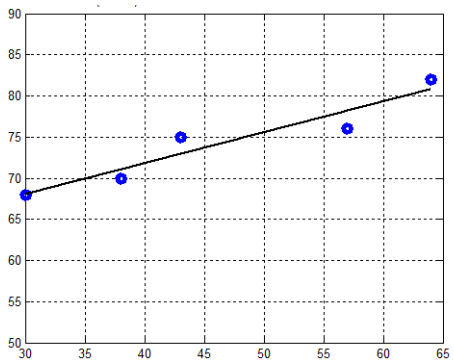
$$5a_1 + 232a_2 = 371$$

$$232a_1 + 11538a_2 = 17505$$

De donde se obtiene la recta de mínimos cuadrados

$$p(x) = a_1 + a_2 x = 56.7610 + 0.3758\, x$$

Representación gráfica de la recta de mínimos cuadrados:



La recta de mínimos cuadrados representa mejor la tendencia de los datos, considerando además que esta es una muestra y por lo tanto no representan de manera exacta a todo el conjunto de datos. Las interpolaciones o predicciones serán mejores con este modelo.

Ejemplo. Si x=50, el resultado que se puede predecir con esta recta es 75.551

El estudio detallado de este modelo se denomina Regresión Lineal y permite establecer criterios acerca de la calidad de la representación de los datos con la recta de mínimos cuadrados.

Adicionalmente, mediante una transformación se puede representar mediante la recta de mínimos cuadrados, algunos conjuntos de datos en cuya transformación se puede observar que tienen tendencia aproximadamente lineal.

6.8.1 La recta de mínimos cuadrados en notación matricial

Dado un conjunto de observaciones (x_i, y_i), $i=1,2,\ldots,n$, se propone el siguiente modelo para representarlas:

$$Y = a_1 + a_2X$$

De tal manera que cada dato pertenezca al modelo propuesto:

$$y_1 = a_1 + a_2x_1$$
$$y_2 = a_1 + a_2x_2$$
$$\ldots$$
$$y_n = a_1 + a_2x_n$$

Estas ecuaciones se pueden expresar en notación matricial:

$$\begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \ldots & \ldots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad \Rightarrow \quad Y = XA, \quad \text{con} \quad A = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Para calcular el vector **A**, la matriz **X** se la multiplica por su transpuesta para hacerla cuadrada. La matriz X se:

$X^TY = (X^TX)A$ de donde se puede obtener el vector solución: $A = (X^TX)^{-1}(X^TY)$

Solución del ejemplo anterior con la notación matricial y la librería NumPy de Python

```
>>> from numpy import*
>>> X=[[1,43],[1,64],[1,38],[1,57],[1,30]]
>>> Y=[[75],[82],[70],[76],[68]]
>>> A=dot(linalg.inv(dot(transpose(X),X)),dot(transpose(X),Y))
>>> print(A)
[[56.76099327]
 [ 0.37584066]]
```

El modelo descrito en notación matricial en esta sección es el mismo modelo de la recta de mínimos cuadrados de la sección anterior como se demuestra a continuación:

$$X^T X = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix}$$

$$X^T Y = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}$$

$$(X^T X)A = X^T Y \quad \Rightarrow \quad \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}$$

$$\Rightarrow \quad \begin{aligned} a_1 n + a_2 \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} y_i \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} x_i y_i \end{aligned}$$

6.9 Interpolación Paramétrica

En esta sección se describirá un procedimiento computacional con Python para graficar en el plano ecuaciones con variables que dependen de una tercera variable denominada parámetro, resaltando la ventaja de usar estas ecuaciones denominadas ecuaciones paramétricas. Se se incluye la interpolación paramétrica con el polinomio de Lagrange.

6.9.1 Curvas paramétricas

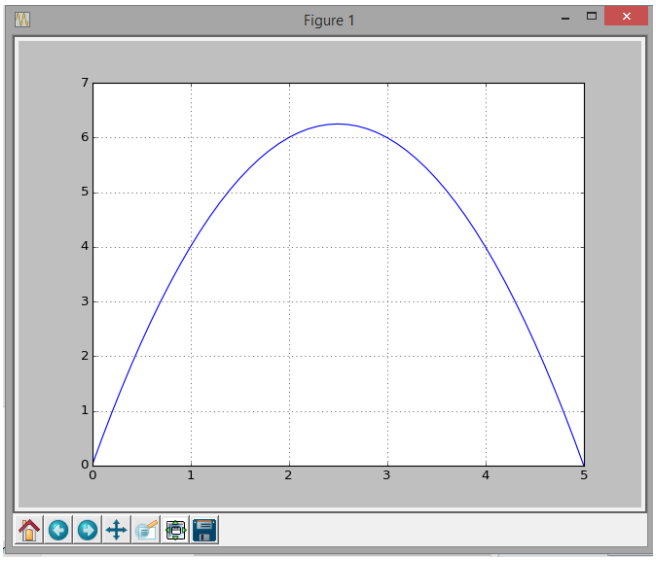
Las curvas paramétricas, se usan para expresar y graficar una relación entre dos variables que no es de tipo funcional utilizando otra variable denominada parámetro. Esta definición permite describir información adicional acerca de la relación entre las variables y que puede ser de interés conocer.

Ejemplo. Suponga que la trayectoria de un objeto está dada por la siguiente ecuación:

$$y(x) = 5x - x^2, \quad 0 \leq x \leq 5$$

Su gráfico aproximado en el plano se lo puede obtener con la librería PyLab de Python conectando con segmentos de recta muchos puntos espaciados a una distancia pequeña en el dominio de la variable independiente. Para este ejemplo se usó 0.01

```
>>> import pylab as pl                  Librería de Python para gráficación
>>> def y(x): return 5*x-x**2           Defición directa de una función en Python
>>> x=pl.arange(0,5,0.01)               Generación de puntos en un rango
>>> pl.plot(x,y(x),'-')                 Gráfico de puntos conectados con rectas
>>> pl.grid(True)                       Dibujo de cuadrículas
>>> pl.show()                           Mostrar el gráfico
```



El gráfico de esta ecuación de tipo funcional describe la trayectoria del objeto pero no permite conocer el instante en el cual se encuentra algún punto **(x, y(x))**. Para conocer esta importante información se puede usar otra variable, en este caso el tiempo **t**. Esta variable se denomina parámetro y las ecuaciones cuyas variables dependen de este parámetro se denominan ecuaciones paramétricas.

La trayectoria de la ecuación anterior puede describirse mediante las siguientes ecuaciones paramétricas en las que se expresan las variables **x, y** como funciones de **t**. El dominio del parámetro **t** se deduce directamente del dominio de la variable **x**

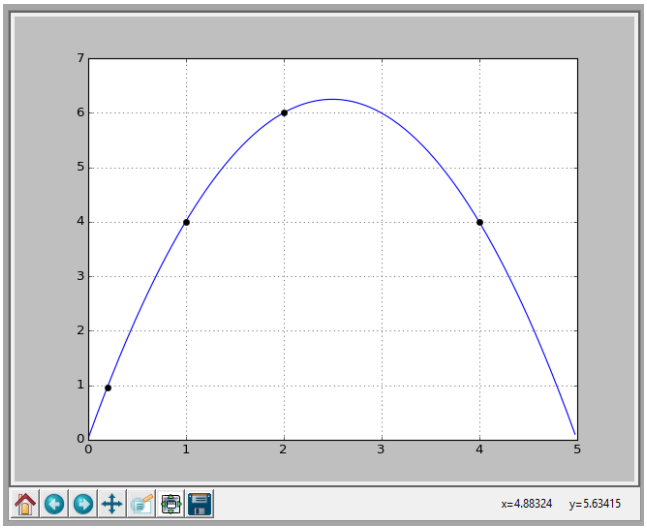
$$x(t) = 2t, \qquad 0 \le t \le 2.5$$
$$y(t) = 10t - 4t^2$$

Si se grafican puntos (x(t), y(t)) se obtiene la misma trayectoria como en el gráfico anterior, pero ahora es posible ver también la posición de puntos de esta curva en algún instante específico.

Se muestra la posición del objeto en los instantes t = 0.1, 0.5, 1, 2. La intervención del parámetro t permite adicionalmente darle una orientación al trazado de la curva.

```
>>> import pylab as pl
>>> def x(t): return 2*t
>>> def y(t): return 10*t-4*t**2
>>> t=pl.arange(0,2.5,0.01)
>>> pl.plot(x(t),y(t),'-')
>>> pl.plot(x(0.1),y(0.1),'ok')
>>> pl.plot(x(0.5),y(0.5),'ok')
>>> pl.plot(x(1),y(1),'ok')
>>> pl.plot(x(2),y(2),'ok')
>>> pl.grid(True)
>>> pl.show()
```

Gráfico de un punto, en color negro



Si en estas ecuaciones paramétricas se elimina mediante sustitución el partámetro t, se obtiene la ecuación anterior de la trayectoria.

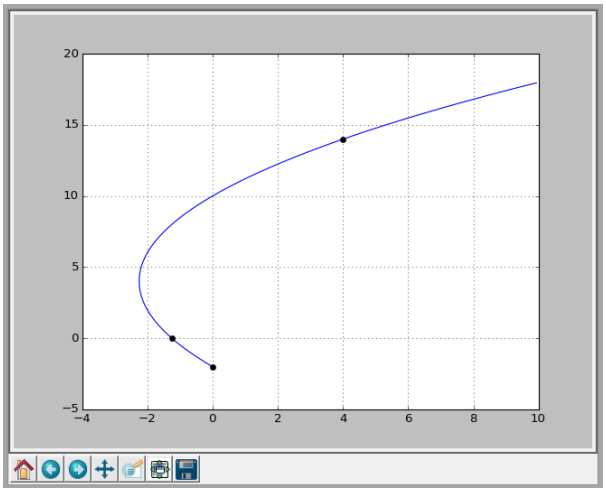
$$y(x) = 5x - x^2, \quad 0 \leq x \leq 5$$

Las ecuaciones paramétricas permiten trazar gráficos mas generales que los gráficos de funciones como se muestra en el siguiente ejemplo.

Ejemplo. Trazar la curva definida con las siguientes ecuaciones paramétricas e incluir algunos puntos que corresponden a valores específicos del parámetro: t = 0, 0.5, 4

$$x(t) = t^2 - 3t$$
$$y(t) = 4t - 2, \quad 0 \leq t \leq 5$$

```
>>> import pylab as pl
>>> def x(t): return t**2-3*t
>>> def y(t): return 4*t-2
>>> t=pl.arange(0,5,0.01)
>>> pl.plot(x(t),y(t),'-')
>>> pl.plot(x(0),y(0),'ok')
>>> pl.plot(x(0.5),y(0.5),'ok')
>>> pl.plot(x(4),y(4),'ok')
>>> pl.grid(True)
>>> pl.show()
```

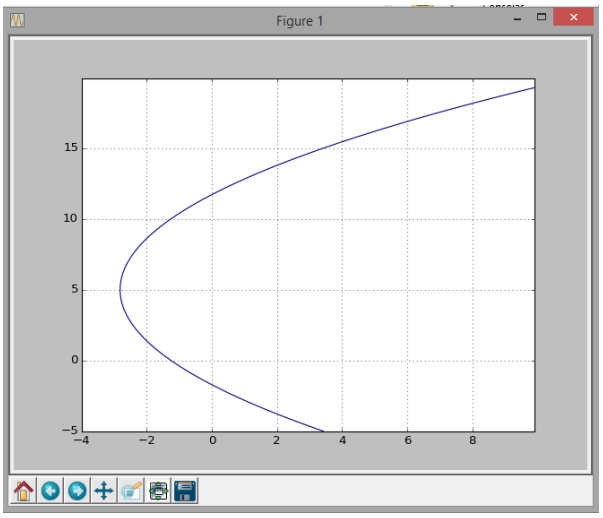


Si en las ecuaciones paramétricas de este ejemplo se elimina el parámetro **t** se obtiene una ecuación implícita con las variables **x, y**. Esta ecuación no se puede expresar como una función explícita **y(x)**:

$$y^2 - 10y - 16x - 20 = 0$$

El gráfico de esta ecuación puede obtenerse computacionalmente en Python como se ve a continuación. Igualmente, este gráfico solamente muestra la trayectoria, pero no los puntos que corresponden a instantes específicos del parámetro t. Adicionalmente, no es evidente el ajuste que debe hacerse al dominio de la variable x para que coincida con el dominio de la variable t.

```
>>> import pylab as pl
>>> xr = pl.arange(-4,10,0.1)
>>> yr = pl.arange(-5,20,0.1)
>>> x, y = pl.meshgrid(xr,yr)
>>> f=y**2-10*y-16*x-20
>>> pl.contour(x, y, f,[0])
>>> pl.grid(True)
>>> pl.show()
```



6.9.2 Interpolación paramétrica con el polinomio de interpolación

Los métodos de interpolación se usan para aproximar una función mediante un polinomio. Pero si los datos (x_i , y_i) no tienen una relación de tipo funcional $y(x)$, entonces no se pueden aplicar directamente los métodos de interpolación vistos en las secciones anteriores.

Sin embargo, si las coordenadas (x, y) se expresan como funciones de otra variable t denominada parámetro, entonces los puntos $x(t), y(t)$ si tienen relación funcional y con ellos se pueden construir polinomios de interpolación. Estos polinomios separados no son de interés, pero en cambio, las parejas de puntos $(x(t), y(t))$ definen a la curva que representa a los datos. Por lo tanto, esta curva puede tener una forma general.

Los valores del parámetro t permiten establecer numéricamente las relaciones funcionales x(t), y(t). Se pueden elegir estos valores como un subconjunto ordenado de los reales y pueden tener un significado práctico. Por ejemplo, el parámetro pueden ser tiempo, y las coordenadas x(t), y(t) representarían puntos de una trayectoria en instantes de t.

Los valores para el parámetro t deben ser los mismos para cada punto $x(t), y(t)$. El orden de estos valores le dan la orientación a la curva de la cual se conocen los puntos (x_i, y_i).

Después de construir los polinomios de interpolación, para que el gráfico se muestre como una curva contínua, se debe evaluar x(t) , y(t) con valores de t muy cercanos en el mismo dominio del parámetro. Estos puntos se conectan para trazar la curva.

En esta sección se utiliza la fórmula de Lagrange para realizar interpolación paramétrica aplicable a conjuntos de puntos $x(t_i), y(t_i)$ **q**ue no tienen una relación funcional.

Ejemplo. Las coordenadas $x(t), y(t)$ del recorrido de un cohete registradas en los instantes t: 0, 1, 2, 3, fueron respectivamente: x(t): 2,1,3,4, y(t): 0,4,5,0

Con esta información y usando polinomios de tercer grado

a) Estime la altura del cohete cuando t = 2.5
b) Estime la altura del cohete cuando x = 3.5

Solución

t: [0, 1, 2, 3]
x(t): [2, 1, 3, 4]
y(t): [0, 4, 5, 0]

Con el polinomio de Lagrange se obtienen los polinomios de interpolación paramétricos

$$p_x(t) = -\frac{2}{3}t^3 + \frac{7}{2}t^2 - \frac{23}{6}t + 2$$

$$p_y(t) = -\frac{1}{2}t^3 + \frac{9}{2}t$$

Respuestas

a) $p_y(2.5) = 3.4375$

b) $p_x(t) = -0.6666t^3 + 3.5t^2 - 3.8333t + 2 = 3.5 \Rightarrow t = 2.25$ (Ecuación no lineal)
$p_y(2.25) = 4.4297$

Cálculo computacional:

Obtención de polinomios paramétricos con la función Lagrange y gráfico de la trayectoria

```
>>> from lagrange import*
>>> x=[2,1,3,4]
>>> y=[0,4,5,0]
>>> t=[0,1,2,3]
>>> px=lagrange(t,x)
>>> print(px)
-2*t**3/3 + 7*t**2/2 - 23*t/6 + 2
>>> py=lagrange(t,y)
>>> print(py)
-t**3/2 + 9*t/2
>>> import pylab as pl
>>> u=pl.arange(0,3.01,0.01)
>>> vx=lagrange(t,x,list(u))
>>> vy=lagrange(t,y,list(u))
>>> pl.plot(vx,vy,'-')
>>> pl.plot(x,y,'o')
>>> pl.grid(True)
>>> pl.show()
```

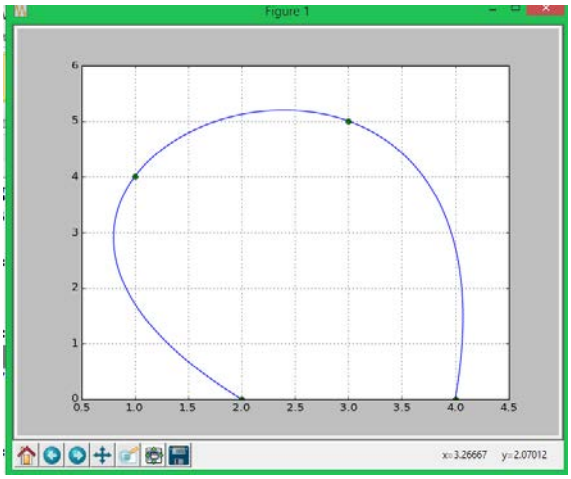


Gráfico aproximado de la trayectoria del cohete

6.10 Ejercicios y problemas con el polinomio de interpolación

1. Los siguientes datos son observaciones de los ingresos **f** en base al monto de inversión **x** realizada en cierto negocio (miles de dólares):

(5.5, 83.0), (8.2, 94.5), (12.4, 105.0), (19.0, 92.0)

a) Encuentre el polinomio de interpolación de tercer grado con la fórmula de Lagrange
Con este polinomio determine:
b) La ganancia que se obtiene si la inversión fuese 15.0
c) Cuanto habría que invertir si se desea una ganancia de 100.0
d) Para que valor de inversión se obtiene la máxima ganancia.

2. Encuentre un polinomio de interpolación para expresar en forma exacta la suma de los cubos de los primeros **k** números naturales:

$$s(k) = 1^3 + 2^3 + 3^3 + \ldots + k^3, \quad k = 1, 2, 3, 4, \ldots$$

Obtenga puntos de **s(k), k=1,2,3,4…** Construya sucesivamente el polinomio de interpolación con **2, 3, 4, …** puntos, hasta que el polinomio no cambie. Este polinomio será exacto pues **s(k)** es una expresión algebraica.

3. Los siguientes datos pertenecen a la curva de Lorentz, la cual relaciona el porcentaje de ingreso económico global de la población en función del porcentaje de la población:

% de población	% de ingreso global
25	**10**
50	**25**
75	**70**
100	**100**

Ej. El 25% de la población tiene el 10% del ingreso económico global.

a) Use los cuatro datos para construir un polinomio para expresar esta relación
b) Con el polinomio determine el porcentaje de ingreso económico que le corresponde al 60% de la población
c) Con el polinomio determine a que porcentaje de la población le corresponde el 60% de ingreso económico global. Resuelva la ecuación resultante con el método de Newton

4. Se han registrado los siguientes datos del costo total **c(x)**, en dólares, de fabricación de **x** unidades de cierto artículo:

(x, c(x)): (10, 358), (20, 1268), (30, 2778), (40, 4888), (50, 7598)

a) Mediante el polinomio de interpolación encuentre una función para expresar **c(x)**
b) El costo medio por unidad **q(x)** es el costo total **c(x)** dividido por el número de unidades producidas: **q(x) = c(x)/x**. Encuentre el nivel de producción **x** en el cual el costo medio por unidad es el menor

5. Los siguientes datos representan la medición de la demanda **f** de un producto durante cinco semanas consecutivas **t**:

t = [1, 2 , 3, 4, 5]
f = [24, 45, 62, 65, 58]

Use todos los datos dados para calcular los siguientes resultados y estimar el error en sus respuestas:

a) Encuentre la demanda en la semana **3.5**
b) Encuentre en que día la demanda fue **50**
c) Encuentre en que día se tuvo la mayor demanda

6. Suponga que en el siguiente modelo **f(x)** describe la cantidad de personas que son infectadas por un virus: $\mathbf{f(x) = a\,x + b\,x^2 + c\,e^{0.1x}}$, en donde **x** es tiempo en días.
Siendo **a, b, c** coeficientes que deben determinarse.

Se conoce que la cantidad de personas infectadas en los días **0**, **5** y **10** son respectivamente: **f(0)=1, f(5)=4, f(10)=20**

a) Plantee un sistema de ecuaciones lineales y resuélvalo para determinar los coeficientes.
b) Use el modelo **f(x)** para determinar en cual día la cantidad de personas infectadas por el virus será **1000**. Obtenga la solución con el método de la Bisección. Previamente encuentre un intervalo de convergencia y obtenga la respuesta con un decimal exacto. Muestre los valores intermedios calculados hasta llegar a la solución.

7. La función de variable real $\mathbf{f(x)=\cos(x)e^x + 1,}$ $\mathbf{0 \le x \le \pi}$, será aproximada con el polinomio de segundo grado p(x) que incluye a los tres puntos **f(0), f($\pi/2$), f(π).**

a) Determine el error en la aproximación si **x = π/4**
b) Encuentre la magnitud del máximo error **E(x)=f(x)-p(x)**, que se produciría al usar **p(x)** como una aproximación a **f(x).** Resuelva la ecuación no lineal resultante con la fórmula de Newton con un error máximo de **0.0001**

5. Dados los puntos que corresponden a una función diferenciable
(x, f(x)): (0.1, 0.501105), (0.2, 0.504885), (0.3, 0.512148), (0.4, 0.523869), (0.5, 0.541218)

a) Encuentre el valor aproximado de **f(0.12)** con un polinomio de tercer grado.
b) Estime el error en la interpolación con la fórmula del error en la interpolación.

8. Dados los puntos **(x,f(x))** de una función:

x :	1.0000	1.1000	1.2000	1.3000	1.4000	1.5000	1.6000
f(x) :	2.2874	2.7726	3.2768	3.7979	4.3327	4.8759	5.4209

a) Encuentre el valor de **f(1.55)** con un polinomio de tercer grado.
b) Estime el error en el resultado obtenido con la fórmula del error en la interpolación.

9. Luego de efectuarse un experimento se anotaron los resultados **f(x)** y se tabularon las diferencias finitas. Accidentalmente se borraron algunos valores quedando únicamente lo que se muestra a continuación:

x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$	$\Delta^4 f_i$
1.3	3.534	--------	0.192	0.053	0.002
1.5	--------	--------	--------	--------	
1.7	--------	--------	--------		
1.9	--------	--------			
2.1	--------				

También se había hecho una interpolación con un polinomio de primer grado en **x = 1.4** obteniéndose como resultado de la interpolación el valor **4.0755**

a) Con los datos suministrados reconstruya la tabla de diferencias finitas
b) Encuentre el valor de **f(1.62)** con un polinomio de interpolación de diferencias finitas de tercer grado, y estime el error en la interpolación.
c) Encuentre el valor de **x** tal que **f(x) = 5.4** con un polinomio de interpolación de diferencias finitas de tercer grado. Para obtener la respuesta debe resolver una ecuación cúbica. Use el método de Newton y obtenga el resultado con cuatro decimales exactos. Previamente encuentre un intervalo de convergencia.

10. La suma de los cuadrados de los primeros **k** números pares:

$$\mathbf{s(k) = 2^2 + 4^2 + 6^2 + \ldots + (2k)^2}$$

Se puede expresar exactamente mediante un polinomio de interpolación.

a) Encuentre el polinomio de interpolación con el polinomio de diferencias finitas
b) Calcule **s(100)** usando el polinomio.

11. Dados los siguientes puntos (x, f(x)):

x = 1.2000, 1.4000, 1.6000, 1.8000, 2.0000
f = 4.7830, 6.8147, 9.5058, 13.0672, 17.7387

a) Aproxime el valor de f(1.35) mediante el polinomio de diferencias finitas de tercer grado
b) Estime el error en el resultado de la interpolación

12. Un lago tiene forma aproximadamente rectangular de 80 m. por 150 m. Para un estudio ambiental se midió el nivel de contaminación en algunos puntos del lago. Los valores obtenidos y las coordenadas de los puntos en los que obtuvieron los datos se indican en la tabla siguiente:

Y / X	0	40	100	150
0	0.12	0.15	0.21	0.32
50	0.25	0.31	0.52	0.42
80	0.14	0.18	0.34	0.25

Use todos los datos tabulados y estime el nivel de contaminación en el centro del lago.

13. Se registraron los siguientes datos de la cantidad de producto obtenido experimentalmente en parcelas de cultivo en las que se suministraron tres cantidades diferentes de fertilizante tipo 1 y cuatro cantidades diferentes de fertilizante tipo 2:

Fertilizante 2

Fert. 1	**1.2**	**1.4**	**1.6**	**1.8**
1.0	7.2	7.8	7.5	7.3
1.5	8.2	8.6	9.2	9.0
2.0	9.5	9.6	9.3	8.6

Use todos los datos dados para determinar mediante una interpolación polinomial con el método de Lagrange, la cantidad de producto que se obtendría si se usaran **1.2** de fertilizante 1 y **1.5** de fertilizante 2.

14. Una empresa que vende cierto producto ha observado que su demanda depende del precio al que lo vende (P en $/unidad) y también del precio al que la competencia vende un producto de similares características (Q en $/unidad). Recopilando información histórica respecto a lo que ha sucedido en el pasado se observó que la demanda diaria (unidades vendidas por día) de este producto fueron de:

		P		
		1	1.1	1.2
Q	1	100	91	83
	1.1	110	100	92
	1.2	120	109	100
	1.3	130	118	108

Use **todos los datos dados** y el polinomio de interpolación de Lagrange para estimar los ingresos mensuales de la empresa por la venta de este producto si decide venderlo a $1.15 por unidad y conoce que la competencia estableció un precio de $1.25 por unidad.

15. El índice enfriador del viento es una función f que depende de dos factores: La temperatura real T y la velocidad del viento v, es decir f(T,v). La siguiente tabla registra los valores de f recogidos en cierto momento por un investigador en los páramos del Cotopaxi. Por ejemplo, cuando la temperatura real es 5 grados Celsius y el viento de 15 km/h, el índice f(5,15) = 2, lo cual significa que la temperatura que se siente en estas condiciones es de 2 grados, aunque la temperatura real sea de 5 grados.

T \ v	5	10	15	20
-5	-8	-10	-11	-12
0	-2	-3	-4	-5
5	4	2	2	1

Usando interpolación polinomial estime la temperatura que sentirá una persona situada en un lugar en el que la temperatura real es de 2 grados Celsius y la velocidad del viento es 18 km/h.

16. Se han registrado los siguientes datos del crecimiento demográfico **V** de los individuos en una región en donde **x** es tiempo en meses

x = **[5, 10, 15, 20]**

V = **[12, 25, 80, 200]**

Observando el crecimiento rápido de los valores de **V** se ha propuesto el siguiente modelo exponencial: $\mathbf{V = c\ (d)^x}$

a) Encuentre las constantes **c, d** del modelo propuesto con el siguiente procedimiento:
Grafique los puntos (**x, ln(V)**) y observe que estos puntos tienen una tendencia lineal, por lo tanto se puede usar el método de mínimos cuadrados para obtener la recta **Y** = **a + bx** con los puntos **(x, y)**, siendo **y** = **ln(V)**.
Para determinar **c, d** tome logaritmo natural a la ecuación $\mathbf{V = c\ (d)^x}$. Se obtendrá la ecuación de una recta. Compare esta recta con la recta de mínimos cuadrados que obtuvo anteriormente y deduzca finalmente el valor para las constantes **c, d**

b) Con el modelo propuesto, pronostique cuantos individuos habrán en el mes **25**

c) Con el modelo propuesto, determine en que mes habrán **150** individuos

17. Las coordenadas x(t), y(t) del recorrido de un cohete registradas en los instantes t: 0, 1, 2, 3, 4 fueron respectivamente: x(t): 3, 2, 4, 7, 8; y(t): 0, 4, 7, 4, 0

Con esta información y usando el polinomio de interpolación de Lagrange paramétrico:

a) Estime la altura del cohete cuando t = 2.5
b) Estime la altura del cohete cuando x = 4.5
c) Encuentre la altura máxima alcanzada por el cohete

6.11 El trazador cúbico

El polinomio de interpolación es útil si los datos tienen un comportamiento aproximadamente de tipo polinomial. En este caso un polinomio es una representación adecuada.

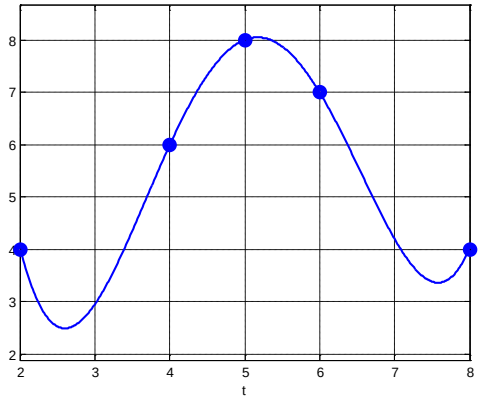
Dependiendo de la distribución de los datos, el polinomio puede tomar una forma distorsionada y en algunos casos puede ser inaceptable como un modelo para representarlos como se muestra en el siguiente ejemplo.

Ejemplo. Se tienen cinco observaciones **(2, 4), (4,6), (5,8), (6,7), (8,4)** y se desea construir con estos datos un modelo cuyo perfil tenga forma aproximadamente tipo campana

Con el método de Lagrange obtenemos el polinomio de interpolación

$$\mathbf{p(x) = 19/144\ x^4 - 389/144\ x^3 + 1375/72\ x^2} - 484\mathbf{/9\ x + 164/3}$$

El gráfico se muestra a continuación



Se observa que en los intervalos izquierdo y derecho la forma del polinomio no es apropiada para expresar la tendencia de los datos.

Una opción pudiera ser colocar polinomios de interpolación de menor grado incluyendo varios puntos consecutivos. Por ejemplo un polinomio de segundo grado con los puntos **(2, 4), (4, 6), (5, 8)**, y otro polinomio de segundo grado con los puntos **(5, 8), (6, 7), (8, 4)**. Sin embargo, en el punto intermedio **(5, 8)** en el que se unen ambos polinomios se perdería la continuidad de la pendiente y curvatura.

Una mejor opción consiste en colocar una función en cada subintervalo de dos puntos consecutivos, de tal manera que sigan la tendencia de los datos y se conecten adecuadamente con las funciones de intervalos adyacentes. Este concepto es la base del **trazador cúbico**. Este dispositivo matemático equivale a la regla flexible que usaban algunos dibujantes y que permitía flexionarla para seguir de una manera suave la trayectoria de los puntos sobre un plano.

6.11.1 El trazador cúbico natural

Sean (x_i, y_i), **i=0,1,2,3,...,n-1** puntos de una función real **u: R→R**, contínua y diferenciable en el dominio de los puntos dados.

Si se supone que **y(x)** es una función simple, se la puede representar aproximadamente mediante otra función **T(x)** definida en segmentos delimitados por los puntos dados:

$$T(x) = \begin{cases} T_0(x), & x_0 \le x \le x_1 \\ T_1(x), & x_1 \le x \le x_2 \\ & \cdots \\ T_{n-2}(x), & x_{n-2} \le x \le x_{n-1} \end{cases}$$

Es conveniente que las funciones T_i sean polinomios segmentados de tercer grado y se conecten manteniendo continuidad hasta la segunda derivada. Para la formulación, se les asignan la forma general:

$$T_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i; \quad x \in [x_i, x_{i+1}]; \quad i = 0,1,...,n-2$$

En los puntos interiores deben coincidir la pendiente y la curvatura de los polinomios de intervalos adyacentes:

$$T_{i-1}^{(k)}(t_i) = T_i^{(k)}(t_i); \quad i = 1,...,n-2; \quad k = 1,2$$

Formulación del trazador cúbico natural

Dados los puntos (x_i, y_i), **i=0,1,2,...,n-1**. Sea $h_i = x_{i+1} - x_i$ el espaciamiento entre puntos adyacentes

Los polinomios segmentados del trazador cúbico y sus dos primeras derivadas:

$$T_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i; \quad x \in [x_i, x_{i+1}]; \quad i = 0,1,...,n-2$$
$$T_i'(x) = 3a_i(x-x_i)^2 + 2b_i(x-x_i) + c_i$$
$$T_i''(x) = 6a_i(x-x_i) + 2b_i$$

Los polinomos del trazador cúbico y sus derivadas deben incluir a los puntos en cada intervalo:

$$T_i(x_i) = y(x_i) = y_i = a_i(x_i-x_i)^3 + b_i(x_i-x_i)^2 + c_i(x_i-x_i) + d_i = d_i \quad \Rightarrow \quad d_i = y_i \qquad (1)$$

$$T_i(x_{i+1}) = y(x_{i+1}) = y_{i+1} = a_i(x_{i+1}-x_i)^3 + b_i(x_{i+1}-x_i)^2 + c_i(x_{i+1}-x_i) + d_i$$
$$= a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i \quad \Rightarrow \quad y_{i+1} = a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i \qquad (2)$$

Conviene parametrizar la segunda derivada:

$$T_i''(x_i) = S_i = 6a_i(x_i - x_i) + 2b_i = 2b_i \quad \Rightarrow \quad b_i = \frac{S_i}{2} \tag{3}$$

$$T_i''(x_{i+1}) = S_{i+1} = 6a_i(x_{i+1} - x_i) + 2b_i = 6a_i h_i + 2b_i \quad \Rightarrow \quad a_i = \frac{S_{i+1} - S_i}{6h_i} \tag{4}$$

Sustituir **(1), (3),** y **(4)** en **(2)**

$$y_{i+1} = \frac{S_{i+1} - S_i}{6h_i} h_i^3 + \frac{S_i}{2} h_i^2 + c_i h_i + y_i \quad \Rightarrow \quad c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6} \tag{5}$$

Las fórmulas (1), (3), (4), (5) definen a los coeficientes $a_i, b_i, c_i, d_i, i=0,1,...,n-2$ de cada polinomio segmentado mediante valores de los datos x_i, y_i y de valores de S_i.

Coeficientes del trazador cúbico

$$a_i = \frac{S_{i+1} - S_i}{6h_i}$$

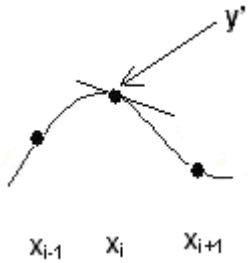
$$b_i = \frac{S_i}{2} \tag{6}$$

$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6}$$

$$d_i = y_i \qquad i = 0, 1, ..., n-2$$

En la siguiente sección se utiliza la continuidad de la primera derivada para hallar un dispositivo para encontrar los valores de S_i con los cuales se determinarán los coeficientes.

En el punto intermedio entre dos intervalos adyacentes, la pendiente de los polinomios debe coincidir:



Primera derivada en el intervalo $[x_i, x_{i+1}]$:

$$T_i'(x_i) = 3a_i(x_i - x_i)^2 + 2b_i(x_i - x_i) + c_i = c_i$$

Primera derivada en el intervalo $[x_{i-1}, x_i]$:

$$T'_{i-1}(x_i) = 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1} = 3a_{i-1}h_{i-1}^2 + 2b_{i-1}h_{i-1} + c_{i-1}$$

Continuidad en la primera derivada entre polinomios adyacentes, en el punto común $\mathbf{x_i}$:

$$T'_{i-1}(x_i) = T'_i(x_i) \quad \Rightarrow \quad 3a_{i-1}h_{i-1}^2 + 2b_{i-1}h_{i-1} + c_{i-1} = c_i \tag{7}$$

Sustituyendo las definiciones de (6) en (7) y simplificando:

$$h_{i-1}S_{i-1} + 2(h_{i-1} + h_i)S_i + h_iS_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right), \quad i = 1,2,3,\ldots,n-2 \tag{8}$$

Esta fórmula evaluada genera un sistema de **n-2** ecuaciones con variables $\mathbf{S_0, S_1, \ldots, S_{n-1}}$

En el trazador cúbico natural los segmentos cerca de los puntos extremos inicial y final se consideran libres o sueltos por lo que no tienen curvatura. Con esta suposición el valor de la segunda derivada tiene un valor nulo en los extremos, y se puede escribir:

$$\mathbf{S_0 = 0, \ S_{n-1} = 0} \tag{9}$$

La ecuación (8) desarrollada y con $\mathbf{S_0 = 0, \ S_{n-1} = 0}$ conforman un sistema completo de **n-2** ecuaciones y se pueden expresar en forma matricial mediante el sistema: **AS=B** en donde **S** es el vector de las variables: $\mathbf{S_1, S_2, \ldots, S_{n-2}}$, con la matriz **A** y el vector **B** definidos de la siguiente manera:

$$\mathbf{A} = \begin{bmatrix} 2(h_0+h_1) & h_1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & h_{n-5} & 2(h_{n-5}+h_{n-4}) & h_{n-4} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & h_{n-4} & 2(h_{n-4}+h_{n-3}) & h_{n-3} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & h_{n-3} & 2(h_{n-3}+h_{n-2}) \end{bmatrix}$$

$$S = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ . \\ . \\ S_{n-4} \\ S_{n-3} \\ S_{n-2} \end{bmatrix}, \qquad B = \begin{bmatrix} 6(\frac{y_2 - y_1}{h_1} - \frac{y_1 - y_0}{h_0}) \\ 6(\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1}) \\ 6(\frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2}) \\ . \\ . \\ 6(\frac{y_{n-3} - y_{n-4}}{h_{n-4}} - \frac{y_{n-4} - y_{n-5}}{h_{n-5}}) \\ 6(\frac{y_{n-2} - y_{n-3}}{h_{n-3}} - \frac{y_{n-3} - y_{n-4}}{h_{n-4}}) \\ 6(\frac{y_{n-1} - y_{n-2}}{h_{n-2}} - \frac{y_{n-2} - y_{n-3}}{h_{n-3}}) \end{bmatrix} \qquad (10)$$

La solución de este sistema entrega los valores de $S_1, S_2, ..., S_{n-2}$, que al sustituir en las definiciones (6) proporcionan los coeficientes de cada polinomio segmentado del trazador cúbico natural. En algunos coeficientes se hace referencia a S_{n-1} y S_0 los cuales con la definición establecida, deben sustituirse con el valor 0:

$$T_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i; \quad x \in [x_i, x_{i+1}]; \quad i = 0,1,...,n-2 \qquad (11)$$

6.11.2 Algoritmo del trazador cúbico natural

Dados los puntos: (x_i, y_i), $i = 0, 1, 2, ..., n-1$

1. Reemplace los datos en el sistema de ecuaciones **(10)**
2. Resuelva el sistema y obtenga la solución: $S_1, S_2, ..., S_{n-2}$. Agregue las definiciones dadas en **(9)**
3. Con las definiciones del cuadro **(8)** obtenga los coeficientes para el trazador cúbico.
4. Sustituya los coeficientes en **(11)** y obtenga el polinomio del trazador cúbico en cada uno de los intervalos.

Ejemplo. Encuentre el trazador cúbico natural usando la formulación anterior para los siguientes puntos: **(2, 4), (4, 6), (5, 8), (6, 7), (8, 4)**

Anotamos los datos en la terminología del trazador cúbico. **n = 5**

i	x_i	y_i	$h_i = x_{i+1} - x_i$
0	2	4	2
1	4	6	1
2	5	8	1
3	6	7	2
4	8	4	

$S_0 = 0$, $S_4 = 0$, de acuerdo a la definición para el trazador cúbico natural **(9)**

Sustituir los datos en el sistema de ecuaciones **(10)**:

$$\mathbf{AS = B:} \quad \begin{bmatrix} 2(h_0+h_1) & h_1 & 0 \\ h_1 & 2(h_1+h_2) & h_2 \\ 0 & h_2 & 2(h_2+h_3) \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 6(\frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0}) \\ 6(\frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1}) \\ 6(\frac{y_4-y_3}{h_3} - \frac{y_3-y_2}{h_2}) \end{bmatrix}$$

$$\mathbf{AS = B:} \quad \begin{bmatrix} 6 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 6 \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -18 \\ -3 \end{bmatrix}, \text{ cuya solución es: } \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 1.8409 \\ -5.0455 \\ 0.3409 \end{bmatrix}$$

Sustituyendo estos valores en las definiciones **(6)** se obtienen los coeficientes:

i	a_i	b_i	c_i	d_i
0	0.1534	0	0.3863	4
1	-1.1477	0.9204	2.2272	6
2	0.8977	-2.5227	0.6250	8
3	-0.0284	0.1704	-1.7272	7

Los coeficientes corresponden a los cuatro polinomios segmentarios del trazador cúbico:

$$\mathbf{T_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i; \quad x \in [x_i, x_{i+1}]; \quad i = 0,1,2,3}$$

$T_0(x) = a_0(x - x_i)^3 + b_0(x - x_i)^2 + c_0(x - x_i) + d_0$
$= 0.1534\,(x - 2)^3 + 0(x - 2)^2 - 0.3863(x - 2) + 4$
$= 0.1534\,x^3 - 0.9204\,x^2 + 2.2272\,x + 2.0,$ $\quad 2 \le x \le 4$

$T_1(x) = -1.1477(x - 4)^3 + 0.9204(x - 4)^2 + 2.2272(x - 4) + 6,$ $\quad 4 \le x \le 5$

$T_2(x) = 0.8977(x - 5)^3 - 2.5227(x - 5)^2 + 0.6250(x - 5) + 8,$ $\quad 5 \le x \le 6$

$T_3(x) = -0.0284(x - 8)^3 + 0.1704(x - 8)^2 - 1.7272\,(x - 8) + 7,$ $\quad 6 \le x \le 8$

El uso práctico de este dispositivo matemático, incluyendo su graficación, requiere una instrumentación computacional

6.11.3 Instrumentación computacional del trazador cúbico natural

La formulación del trazador cúbico natural se ha instrumentado en Python mediante una función denominada **trazador_natural.** Esta función entrega los polinomios segmentados almacenados en un vector de celdas. Opcionalmente se puede enviar un parámetro adicional conteniendo un valor o un vector. En este caso los resultados son valores de la evaluación del trazador cúbico.

Uso de la función **trazador_natural**

Entra: **x, y**: Puntos (por lo menos 3)
z: Parámetro adicional para evaluar el trazador (puede ser un vector)

Sale: Polinomios segmentarios del trazador natural o resultados de su evaluación

```
import numpy as np
from sympy import*
def trazador_natural(x,y,z=[]):
  n=len(x)
  h=np.zeros([n-1])
  A=np.zeros([n-2,n-2]);B=np.zeros([n-2]);S=np.zeros([n])
  a=np.zeros([n-1]);b=np.zeros([n-1]);c=np.zeros([n-1]);d=np.zeros([n-1])
  if n<3:
    T=[]
    return
  for i in range(n-1):
    h[i]=x[i+1]-x[i]
  A[0,0]=2*(h[0]+h[1])                                  #Armar el sistema
  A[0,1]=h[1]
  B[0]=6*((y[2]-y[1])/h[1]-(y[1]-y[0])/h[0])
  for i in range(1,n-3):
    A[i,i-1]=h[i]
    A[i,i]=2*(h[i]+h[i+1])
    A[i,i+1]=h[i+1]
    B[i]=6*((y[i+2]-y[i+1])/h[i+1]-(y[i+1]-y[i])/h[i])
  A[n-3,n-4]=h[n-3]
  A[n-3,n-3]=2*(h[n-3]+h[n-2])
  B[n-3]=6*((y[n-1]-y[n-2])/h[n-2]-(y[n-2]-y[n-3])/h[n-3])
  r=np.linalg.solve(A,B)                                #Resolver el sistema
  for i in range(1,n-1):
    S[i]=r[i-1]
  S[0]=0
  S[n-1]=0
```

```
for i in range(n-1):
  a[i]=(S[i+1]-S[i])/(6*h[i])
  b[i]=S[i]/2
  c[i]=(y[i+1]-y[i])/h[i]-(2*h[i]*S[i]+h[i]*S[i+1])/6
  d[i]=y[i]
try:
   if len(z)==0:                                    #Detecta si es un vector
      pass
except TypeError:
   z=[z]                                            #Vector con un número
if len(z)==0:                                       #Construir el trazador
  t=Symbol('t')
  T=[]
  for i in range(n-1):
     p=expand(a[i]*(t-x[i])**3+b[i]*(t-x[i])**2+c[i]*(t-x[i])+d[i])
     T=T+[p]
  return T
else:                                               #Evaluar el trazador
  m=len(z)
  q=np.zeros([m])
  for k in range(m):
     t=z[k]
     for i in range(n-1):
        if t>=x[i] and t<=x[i+1]:
           q[k]=a[i]*(t-x[i])**3+b[i]*(t-x[i])**2+c[i]*(t-x[i])+d[i]
  if m>2:
     k=m-1
     i=n-2
     q[k]=a[i]*(t-x[i])**3+b[i]*(t-x[i])**2+c[i]*(t-x[i])+d[i]
  if len(q)==1:
     return q[0]                                    #Retorna un valor
  else:
     return q                                       #Retorna un vector
```

Ejemplo. Encuentre el trazador cúbico natural usando la función anterior para los siguientes puntos: **(2, 4), (4, 6), (5, 8), (6, 7), (8, 4)**

```
>>> from trazador_natural import*
>>> x = [2,4,5,6,8]
>>> y = [4,6,8,7,4]
>>> T=trazador_natural(x,y)
>>> print(T[0])
0.1534*t**3 - 0.9204*t**2 + 2.2272*t + 2.0
>>> print(T[1])
-1.1477*t**3 + 14.6931*t**2 - 60.2272*t + 85.2727
>>> print(T[2])
0.8977*t**3 - 15.9886*t**2 + 93.1818*t - 170.4090
>>> print(T[3])
-0.0284*t**3 + 0.6818*t**2 - 6.8409*t + 29.6363
```

Se puede evaluar el trazador en puntos específicos. **Ejemplo.** Evaluar con x = 3:

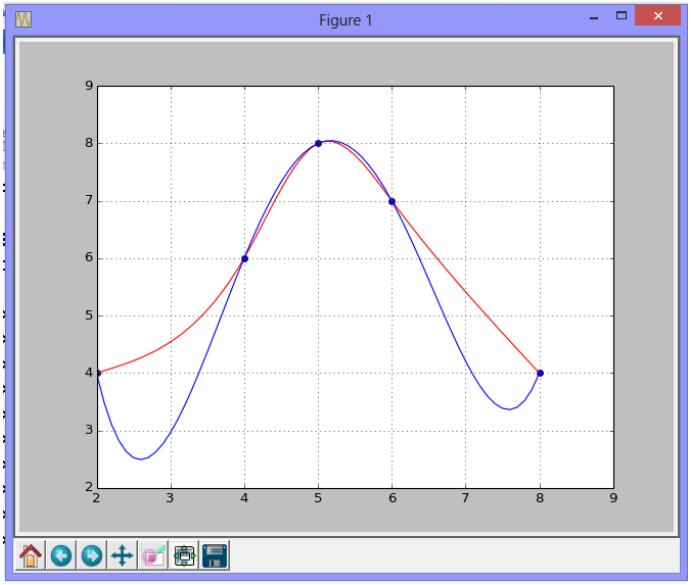
```
>>> r=trazador_natural(x,y,3)
>>> print(r)
4.5397
```

En los resultados, por simplicidad se muestran únicamente los cuatro primeros decimales.

Para observar el perfil del trazador, se pueden deben obtener más puntos y conectarlos con segmentos de recta con la función **plot** de la librería **Pylab**.

Ejemplo. Dibujar el perfil del trazador cúbico natural sobre los puntos dados y el polinomio de interpolación

```
>>> from trazador_natural import*
>>> from lagrange import*
>>> import pylab as pl
>>> x=[2,4,5,6,8]
>>> y=[4,6,8,7,4]
>>> u=pl.arange(2,8.1,0.1)
>>> v=trazador_natural(x,y,u)
>>> vp=lagrange(x,y,list(u))
>>> pl.plot(x,y,'o')
>>> pl.plot(u,v,'-r')
>>> pl.plot(u,vp,'-b')
>>> pl.grid(True)
>>> pl.show()
```



Puede observarse la mejora en el modelo para representar a los datos dados.

6.11.4 El trazador cúbico sujeto

Puede ser de interés asignar alguna pendiente específica a los extremos inicial y final del trazador cúbico. Esta versión se denomina trazador cúbico sujeto o fijo. Por lo tanto, ya no se aplica la definición del trazador cúbico natural en el que se supone que los extremos están sueltos o libres con pendiente constante, y sus segundas derivadas con un valor nulo.

Formulación del trazador cúbico sujeto

Dados los puntos (x_i, y_i), $i=0,1,2,...,n-1$. Sea $h_i = x_{i+1}-x_i$ el espaciamiento entre puntos adyacentes

Se aplica el mismo procedimiento del trazador cúbico natural para obtener las fórmulas para los coeficientes (6) y los polinomios segmentados del trazador cúbico (11):

$$T_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i; \quad x \in [x_i, x_{i+1}]; \quad i = 0,1,...,n-2$$

El procedimiento requiere usar la siguiente fórmula, deducida para el trazador cúbico natural **(8)**, para generar **n-2** ecuaciones siendo las incógnitas los valores de: $S_0, S_1, ..., S_{n-1}$

$$h_{i-1}S_{i-1} + 2(h_{i-1}+h_i)S_i + h_iS_{i+1} = 6\left(\frac{y_{i+1}-y_i}{h_i} - \frac{y_i-y_{i-1}}{h_{i-1}}\right), \quad i = 1,2,3,...,n-2$$

Para el trazador sujeto se obtienen dos ecuaciones adicionales considerando condiciones especiales para las pendientes de los polinomios segmentados en los intervalos extremos
Para el trazador cúbico sujeto se especifican como datos, las pendientes en los extremos:

$y'(x_0) = u$
$y'(x_{n-1}) = v$

Utilizamos la expresión para la primera derivada del trazador cúbico natural de la sección anterior:

$$T_i'(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i$$

Sustituimos los datos dados para los polinomios en el primero y último intervalo:

En el primer intervalo:

$$T_0'(x_0) = u = 3a_0(x_0 - x_0)^2 + 2b_0(x_0 - x_0) + c_0 = c_0$$

Se sustituye la definición de c_0

$$u = \frac{y_1 - y_0}{h_0} - \frac{2h_0S_0 + h_0S_1}{6}$$

De donde se obtiene:

$$-\frac{1}{3}h_0S_0 - \frac{1}{6}h_0S_1 = u - \frac{y_1 - y_0}{h_0} \qquad \textbf{(12)}$$

En el último intervalo:

$$T'(x_{n-1}) = v = 3a_{n-2}(x_{n-1} - x_{n-2})^2 + 2b_{n-2}(x_{n-1} - x_{n-2}) + c_{n-2}$$
$$= 3a_{n-2}h_{n-2}^2 + 2b_{n-2}h_{n-2} + c_{n-2}$$

Se sustituyen las definiciones de los coeficientes:

$$v = 3(\frac{S_{n-1} - S_{n-2}}{6h_{n-2}})h_{n-2}^2 + 2(\frac{S_{n-2}}{2})h_{n-2} + \frac{y_{n-1} - y_{n-2}}{h_{n-2}} - \frac{2h_{n-2}S_{n-2} + h_{n-2}S_{n-1}}{6}$$

De donde se obtiene

$$\frac{1}{6}h_{n-2}S_{n-2} + \frac{1}{3}h_{n-2}S_{n-1} = v - \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \qquad \textbf{(13)}$$

Las ecuaciones **(12) y (13)** junto con las ecuaciones que se obtienen de **(8)** conforman un sistema completo de **n** ecuaciones lineales con las **n** variables $S_0, S_1, ..., S_{n-1}$. En forma matricial se pueden expresar mediante el sistema: **AS=B** en donde **S** es el vector de las variables: $S_0, S_1, ..., S_{n-1}$, con la matriz **A** y el vector **B** definidos de la siguiente manera:

$$A=\begin{bmatrix} -\frac{1}{3}h_0 & -\frac{1}{6}h_0 & 0 & 0 & . & . & 0 & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & . & . & 0 & 0 & 0 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & . & . & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & h_{n-4} & 2(h_{n-4}+h_{n-3}) & h_{n-3} & 0 \\ 0 & 0 & 0 & 0 & . & . & 0 & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & 0 & 0 & 0 & . & . & 0 & 0 & \frac{1}{6}h_{n-2} & \frac{1}{3}h_{n-2} \end{bmatrix}$$

$$S=\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ . \\ . \\ S_{n-3} \\ S_{n-2} \\ S_{n-1} \end{bmatrix}, \qquad B=\begin{bmatrix} u-\frac{y_1-y_0}{h_0} \\ 6(\frac{y_2-y_1}{h_1}-\frac{y_1-y_0}{h_0}) \\ 6(\frac{y_3-y_2}{h_2}-\frac{y_2-y_1}{h_1}) \\ . \\ . \\ 6(\frac{y_{n-2}-y_{n-3}}{h_{n-3}}-\frac{y_{n-3}-y_{n-4}}{h_{n-4}}) \\ 6(\frac{y_{n-1}-y_{n-2}}{h_{n-2}}-\frac{y_{n-2}-y_{n-3}}{h_{n-3}}) \\ v-\frac{y_{n-1}-y_{n-2}}{h_{n-2}} \end{bmatrix} \qquad (14)$$

La solución de este sistema entrega los valores de $S_0, S_1, ..., S_{n-1}$ que al sustituir en las definiciones **(6)** proporcionan los coeficientes de cada polinomio segmentado del trazador cúbico sujeto **(11)**:

Coeficientes del trazador cúbico

$$a_i = \frac{S_{i+1} - S_i}{6h_i}$$

$$b_i = \frac{S_i}{2} \qquad (6)$$

$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_iS_i + h_iS_{i+1}}{6}$$

$$d_i = y_i \qquad i = 0, 1, ..., n-2$$

$$T_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i; \quad x \in [x_i, x_{i+1}]; \quad i = 0,1,...,n-2 \qquad (11)$$

6.11.5 Algoritmo del trazador cúbico sujeto

Dados los puntos: **(x_i, y_i), i = 0, 1, 2, ..., n-1**

1. Reemplace los datos en el sistema de ecuaciones **(14)**
2. Resuelva el sistema y obtenga la solución: **$S_0, S_1, ..., S_{n-1}$**.
3. Con las definiciones del cuadro **(8)** obtenga los coeficientes para el trazador cúbico.
4. Sustituya los coeficientes en **(11)** y obtenga el polinomio del trazador cúbico en cada uno de los intervalos.

Ejemplo. Utilizar el trazador cúbico sujeto con los datos del ejemplo anterior de tal manera que la curva comience y termine con **pendiente nula**. **(2, 4), (4, 6), (5, 8), (6, 7), (8, 4)**

Pendientes en los extremos de los segmentos inicial y final del trazador cúbico: **u = 0, v = 0**

Anotamos los datos en la terminología del trazador cúbico. **n = 5**

i	x_i	y_i	$h_i = x_{i+1} - x_i$
0	2	4	2
1	4	6	1
2	5	8	1
3	6	7	2
4	8	4	

Sustituir los datos en el sistema de ecuaciones:

$$\mathbf{AS=B:}\quad \begin{bmatrix} -\frac{1}{3}h_0 & -\frac{1}{6}h_0 & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & 0 \\ 0 & 0 & h_2 & 2(h_2+h_3) & h_3 \\ 0 & 0 & 0 & \frac{1}{6}h_3 & \frac{1}{3}h_3 \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} u-\frac{y_1-y_0}{h_0} \\ 6(\frac{y_2-y_1}{h_1}-\frac{y_1-y_0}{h_0}) \\ 6(\frac{y_3-y_2}{h_2}-\frac{y_2-y_1}{h_1}) \\ 6(\frac{y_4-y_3}{h_3}-\frac{y_3-y_2}{h_2}) \\ v-\frac{y_4-y_3}{h_3} \end{bmatrix}$$

$$\mathbf{AS=B:}\quad \begin{bmatrix} -2/3 & -1/3 & 0 & 0 & 0 \\ 2 & 6 & 1 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 1 & 6 & 2 \\ 0 & 0 & 0 & 1/3 & 2/3 \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 6 \\ -18 \\ -3 \\ 1.5 \end{bmatrix}, \text{ cuya solución es: } \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 0.725 \\ 1.55 \\ -4.75 \\ -0.55 \\ 2.525 \end{bmatrix}$$

Sustituyendo estos valores en las definiciones **(6)** se obtienen los coeficientes:

i	a_i	b_i	c_i	d_i
0	0.06875	0.3625	0	4
1	-1.05	0.775	2.275	6
2	0.7	-2.375	0.675	8
3	0.25626	-0.275	-1.975	7

Los coeficientes corresponden a los cuatro polinomios segmentarios del trazador cúbico:

$$\mathbf{T_i(x)=a_i(x-x_i)^3+b_i(x-x_i)^2+c_i(x-x_i)+d_i;\quad x\in[x_i,x_{i+1}];\quad i=0,1,2,3}$$

$T_0(x) = a_0(x - x_i)^3 + b_0(x - x_i)^2 + c_0(x - x_i) + d_0$
$= 0.06875(x - 2)^3 + 0.3625(x - 2)^2 - 0(x - 2) + 4,$
$= 0.0687t^3 - 0.05t^2 - 0.625t + 4.9 \qquad 2 \le x \le 4$
$T_1(x) = -1.05(x - 4)^3 + 0.775(x - 4)^2 + 2.275(x - 4) + 6, \qquad 4 \le x \le 5$
$T_2(x) = 0.7(x - 5)^3 - 2.375(x - 5)^2 + 0.675(x - 5) + 8, \qquad 5 \le x \le 6$
$T_3(x) = 0.25626(x - 8)^3 - 0.275(x - 8)^2 - 1.975(x - 8) + 7, \qquad 6 \le x \le 8$

Igual que en el trazador cúbico natural, el uso práctico de este dispositivo matemático, incluyendo su graficación, requiere la instrumentación computacional.

6.11.6 Instrumentación computacional del trazador cúbico sujeto

La formulación del trazador cúbico natural se ha instrumentado en Python mediante una función denominada **trazadorsujeto.** La versión instrumentada entrega los polinomios segmentarios almacenados en los componentes de un vector de celdas. Opcionalmente se puede enviar un parámetro adicional conteniendo un valor o un vector para evaluar el trazador cúbico.

```
import numpy as np
from sympy import*
def trazador_sujeto(x,y,u,v,z=[]):
    n=len(x)
    h=np.zeros([n-1])
    A=np.zeros([n,n]);B=np.zeros([n]);S=np.zeros([n-1])
    a=np.zeros([n-1]);b=np.zeros([n-1]);c=np.zeros([n-1]);d=np.zeros([n-1])
    if n<3:
        T=[]
        return
    for i in range(n-1):
        h[i]=x[i+1]-x[i]
    A[0,0]=-h[0]/3
    A[0,1]=-h[0]/6
    B[0]=u-(y[1]-y[0])/h[0]
    for i in range(1,n-1):
        A[i,i-1]=h[i-1]
        A[i,i]=2*(h[i-1]+h[i])
        A[i,i+1]=h[i]
        B[i]=6*((y[i+1]-y[i])/h[i]-(y[i]-y[i-1])/h[i-1])
    A[n-1,n-2]=h[n-2]/6
    A[n-1,n-1]=h[n-2]/3
    B[n-1]=v-(y[n-1]-y[n-2])/h[n-2]
    S=np.linalg.solve(A,B)
    for i in range(n-1):
      a[i]=(S[i+1]-S[i])/(6*h[i])
      b[i]=S[i]/2
      c[i]=(y[i+1]-y[i])/h[i]-(2*h[i]*S[i]+h[i]*S[i+1])/6
      d[i]=y[i]
    try:
      if  len(z)==0:
        pass
    except TypeError:
      z=[z]
    if  len(z)==0:
      t=Symbol('t')
```

```
        T=[]
        for i in range(n-1):
            p=expand(a[i]*(t-x[i])**3+b[i]*(t-x[i])**2+c[i]*(t-x[i])+d[i])
            T=T+[p]
        return T
    else:
        m=len(z)
        q=np.zeros([m])
        for k in range(m):
            t=z[k]
            for i in range(n-1):
                if t>=x[i] and t<=x[i+1]:
                    q[k]=a[i]*(t-x[i])**3+b[i]*(t-x[i])**2+c[i]*(t-x[i])+d[i]
        if m>2:
            k=m-1
            i=n-2
            q[k]=a[i]*(t-x[i])**3+b[i]*(t-x[i])**2+c[i]*(t-x[i])+d[i]
        if len(q)==1:
            return q[0]
        else:
            return q
```

Ejemplo. Corregir el perfil del trazador cúbico del ejemplo anterior para que la curva comience y termine con **pendiente nula**. De esta manera se acercará más al perfil de una distribución tipo campana, lo cual fue el objetivo inicial. Usar los mismos datos:

(2, 4), (4, 6), (5, 8), (6, 7), (8, 4)

```
>>> from trazador_sujeto import*
>>> x = [2,4,5,6,8]
>>> y = [4,6,8,7,4]
>>> T=trazador_sujeto(x,y,0,0)
>>> print(T[0])
0.0687*t**3 - 0.0500*t**2 - 0.6249*t + 4.9
>>> print(T[1])
-1.05*t**3 + 13.375*t**2 - 54.325*t + 76.5
>>> print(T[2])
0.7*t**3 - 12.875*t**2 + 76.925*t - 142.25
>>> print(T[3])
0.25625*t**3 - 4.8875*t**2 + 29.0*t - 46.4
```

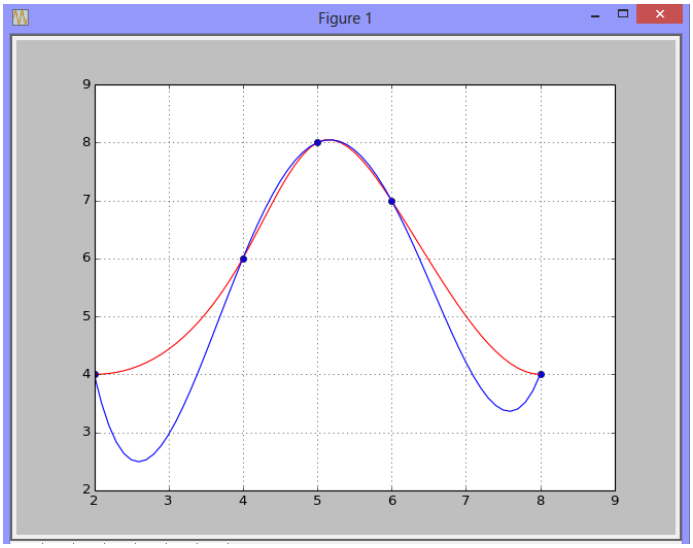
Se puede evaluar el trazador en puntos específicos. Ejemplo. Evaluar con x = 3:

```
>>> r=trazador_sujeto(x,y,0,0,3)
>>> print(r)
4.43125
```

Igual que en el trazador cúbico natural, para observar el perfil del trazador sujeto, es mejor generar más puntos y conectarlos con segmentos de recta usando la función **plot** de **Pylab**

Ejemplo. Dibujar el perfil del trazador cúbico sujeto sobre los puntos dados y el polinomio de interpolación.

```
>>> from trazador_sujeto import*
>>> from lagrange import*
>>> import pylab as pl
>>> x=[2,4,5,6,8]
>>> y=[4,6,8,7,4]
>>> u=pl.arange(2,8.1,0.1)
>>> v=trazador_sujeto(x,y,0,0,u)
>>> vp=lagrange(x,y,list(u))
>>> pl.plot(x,y,'o')
>>> pl.plot(u,v,'-r')
>>> pl.plot(u,vp,'-b')
>>> pl.grid(True)
>>> pl.show()
```



El perfil del trazador cúbico sujeto generalmente suaviza los bordes y proporciona un modelo que se acopla mejor en los extremos.

Ejemplo. Dos segmentos del trazador cúbico sujeto de la función f están definidos con:

$$f(x) \cong T(x) = \begin{cases} 1 + b_1 x + c_1 x^2 - 2x^3, & 0 \le x \le 1 \\ 1 + b_2 (x-1) - 4(x-1)^2 + 7(x-1)^3, & 1 \le x \le 2 \end{cases}$$

Estime la pendiente de **f** en **x=0**, **x=2**

Solución

$$f'(x) \cong T'(x) = \begin{cases} b_1 + 2c_1 x - 6x^2, & 0 \le x \le 1 \\ b_2 - 8(x-1) + 21(x-1)^2, & 1 \le x \le 2 \end{cases}$$

$$f''(x) \cong T''(x) = \begin{cases} 2c_1 - 12x, & 0 \le x \le 1 \\ -8x + 42(x-1), & 1 \le x \le 2 \end{cases}$$

En el punto intermedio los segmentos comparten la ordenada, la pendiente y la curvatura**:**

$$f(1) = T(1) = \begin{cases} 1 + b_1 + c_1 - 2, & 0 \le x \le 1 \\ 1, & 1 \le x \le 2 \end{cases} \quad \Rightarrow \quad 1 + b_1 + c_1 - 2 = 1 \quad \Rightarrow \quad b_1 + c_1 = 2$$

$$f'(1) = T'(1) = \begin{cases} b_1 + 2c_1 - 6, & 0 \le x \le 1 \\ b_2, & 1 \le x \le 2 \end{cases} \quad \Rightarrow \quad b_1 + 2c_1 - 6 = b_2$$

$$f''(1) = T''(1) = \begin{cases} 2c_1 - 12, & 0 \le x \le 1 \\ -8, & 1 \le x \le 2 \end{cases} \quad \Rightarrow \quad 2c_1 - 12 = -8 \quad \Rightarrow \quad c_1 = 2$$

Se obtiene: $c_1 = 2, \quad b_1 = 0, \quad b_2 = 6$

$$f'(x) \cong T'(x) = \begin{cases} 4x - 6x^2, & 0 \le x \le 1 \\ 6 - 8(x-1) + 21(x-1)^2, & 1 \le x \le 2 \end{cases}$$

Finalmente:

$$f'(0) \cong T'(0) = 0$$
$$f'(2) \cong T'(2) = 19$$

6.11.7 Interpolación paramétrica con el trazador cúbico

El procedimiento de interpolación paramétrica con el polinomio de Lagrange se puede sustituir con el Trazador Cúbico y obtener una aproximación mas cercana a la curva propuesta.

Normalmente el gráfico que se obtiene con el trazador cúbico se aproxima mejor al perfil de la curva que el gráfico del polinomio de interpolación. Debido al trabajo manual laborioso requerido, es adecuado un tratamiento computacional.

Ejemplo. Las coordenadas **x(t), y(t)** del recorrido de un cohete registradas en los instantes **t=0, 1, 2, 3** fueron respectivamente: **x(t)=2,1,3,4, y(t)=0,4,5,0**

Con esta información y usando el Trazador Cúbico Natural en forma paramétrica grafique la trayectoria del cohete y estime la altura de la trayectoria cuando **t = 2.5**

Cálculo computacional:

Obtención de polinomios paramétricos con la función Trazador_natural y gráfico de la trayectoria

```
>>> from trazador_natural import*
>>> x=[2,1,3,4]
>>> y=[0,4,5,0]
>>> t=[0,1,2,3]
>>> Tx=trazador_natural(t,x)
>>> print(Tx)
[0.866666666666667*t**3 - 1.86666666666667*t + 2.0,
-1.33333333333333*t**3 + 6.6*t**2 - 8.46666666666667*t + 4.2,
 0.466666666666667*t**3 - 4.2*t**2 + 13.1333333333333*t - 10.2]
>>> Ty=trazador_natural(t,y)
>>> print(Ty)
[-0.4*t**3 + 4.4*t,
 -1.0*t**3 + 1.8*t**2 + 2.6*t + 0.6,
  1.4*t**3 - 12.6*t**2 + 31.4*t - 18.6]
>>> vTy=trazador_natural(t,x,2.5)
>>> print(vTy)
3.675                    Respuesta
>>> import pylab as pl
>>> s=pl.arange(0,3.01,0.01)
>>> Txs=trazador_natural(t,x,s)
>>> Tys=trazador_natural(t,y,s)
>>> pl.plot(Txs,Tys,'-')
>>> pl.plot(x,y,'o')
>>> pl.grid(True)
>>> pl.show()
```

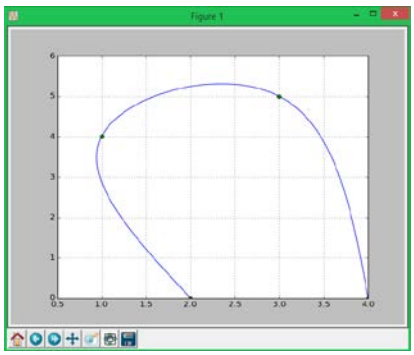


Gráfico aproximado de la trayectoria

6.11.8 El trazador cúbico paramétrico cerrado

En esta sección se propone una forma especial del trazador cúbico que permitirá construir el trazador cúbico paramétrico para modelar curvas planas cerradas.

Sean (t_i, u_i), $i=0,1,2,3,...,n-1$ puntos de una función real $u: R \rightarrow R$, contínua y diferenciable en el dominio de los puntos dados.

Si se supone que **u(t)** es una función simple, se la puede representar aproximadamente mediante otra función **T(t)** definida en segmentos delimitados por los puntos dados:

$$T(t) = \begin{cases} T_0(t), & t_0 \le t \le t_1 \\ T_1(t), & t_1 \le t \le t_2 \\ & \cdots \\ T_{n-2}(t), & t_{n-2} \le t \le t_{n-1} \end{cases}$$

Por las propiedades de continuidad es conveniente que las funciones T_i sean polinomios segmentados de tercer grado. Para la formulación, se les asignan la forma general:

$$T_i(t) = a_i(t-t_i)^3 + b_i(t-t_i)^2 + c_i(t-t_i) + d_i; \quad t \in [t_i, t_{i+1}]; \quad i = 0,1,...,n-2$$

En los puntos interiores deben coincidir la pendiente y la curvatura de los polinomios de intervalos adyacentes:

$$T_{i-1}^{(k)}(t_i) = T_i^{(k)}(t_i); \quad i = 1,...,n-2; \quad k = 1,2$$

Adicionalmente, para instrumentar la forma cerrada del trazador cúbico, es necesario que la pendiente y curvatura en el punto final del polinomio en el último intervalo, coincidan con la pendiente y curvatura en el primer punto del polinomio en el primer intervalo.

$$T_{n-2}^{(k)}(t_{n-1}) = T_0^{(k)}(t_0); \quad k = 1,2$$

Esta condición especial define a este conjunto de polinomios con el nombre de **trazador cúbico cerrado.**

6.11.9 Formulación del trazador cúbico cerrado

Dados los puntos (t_i, u_i), $i=0,1,2,...,n-1$. Sea $h_i = t_{i+1}-t_i$ el espaciamiento entre puntos adyacentes

La formulación de los los polinomios segmentados del trazador cúbico es similar a la que se obtuvo para el trazador cúbico paramétrico:

Polinomios segmentados

$$T_i(t) = a_i(t-t_i)^3 + b_i(t-t_i)^2 + c_i(t-t_i) + d_i; \quad t \in [t_i, t_{i+1}]; \quad i = 0,1,...,n-2$$

Coeficientes del trazador cúbico

$$a_i = \frac{S_{i+1} - S_i}{6h_i}$$

$$b_i = \frac{S_i}{2} \qquad (6)$$

$$c_i = \frac{u_{i+1} - u_i}{h_i} - \frac{2h_iS_i + h_iS_{i+1}}{6}$$

$$d_i = u_i \qquad i = 0, 1, ..., n-2$$

Fórmula para generar un sistema de **n-3** ecuaciones con las variables $S_0, S_1, ..., S_{n-2}$

$$h_{i-1}S_{i-1} + 2(h_{i-1}+h_i)S_i + h_iS_{i+1} = 6\left(\frac{u_{i+1}-u_i}{h_i} - \frac{u_i - u_{i-1}}{h_{i-1}}\right), \quad i = 1,2,3,...,n-3 \qquad (7)$$

Para el trazador cúbico cerrado se obtienen dos ecuaciones adicionales de las condiciones especiales de esta versión del trazador cúbico:

Condición de continuidad de la primera derivada entre los segmentos inicial y final:

$$T'_{n-2}(t_{n-1}) = T'_0(t_0)$$

Primera derivada del primer segmento evaluada en el punto inicial:

$$T'_0(t_0) = 3a_0(t_0-t_0)^2 + 2b_0(t_0-t_0) + c_0 = c_0 = \frac{u_1-u_0}{h_0} - \frac{2h_0S_0 + h_0S_1}{6}$$

Primera derivada del último segmento evaluada en el punto final:

$$T'_{n-2}(t_{n-1}) = 3a_{n-2}(t_{n-1}-t_{n-2})^2 + 2b_{n-2}(t_{n-1}-t_{n-2}) + c_{n-2} = 3a_{n-2}h_{n-2}^2 + 2b_{n-2}h_{n-2} + c_{n-2}$$
$$= 3(\frac{S_{n-1}-S_{n-2}}{6h_{n-2}})h_{n-2}^2 + 2(\frac{S_{n-2}}{2})h_{n-2} + \frac{u_{n-1}-u_{n-2}}{h_{n-2}} - \frac{2h_{n-2}S_{n-2}+h_{n-2}S_{n-1}}{6}$$

Igualando derivadas y simplificando, con $S_{n-1} = S_0$:

$$-\frac{1}{3}(h_0+h_{n-2})S_0 - \frac{1}{6}h_0S_1 - \frac{1}{6}h_{n-2}S_{n-2} = -\frac{u_1-u_0}{h_0} + \frac{u_{n-1}-u_{n-2}}{h_{n-2}} \qquad (8)$$

Condición de continuidad de la segunda derivada entre los segmentos inicial y final:

$$T''_{n-2}(t_{n-1}) = T''_0(t_0)$$

La ecuación (7) correspondiente al intervalo final **n-2,** con $S_{n-1} = S_0$

$$h_{n-3}S_{n-3} + 2(h_{n-3}+h_{n-2})S_{n-2} + h_{n-2}S_0 = 6(\frac{u_{n-1}-u_{n-2}}{h_{n-2}} - \frac{u_{n-2}-u_{n-3}}{h_{n-3}}) \qquad (9)$$

Las ecuaciones (7), (8), (9) conforman un sistema completo de **n-1** ecuaciones. Estas ecuaciones se pueden desarrollar y expresar en forma matricial mediante el sistema: **AS=B** en donde **S** es el vector de las variables: $S_0, S_1, ..., S_{n-2}$, mientras que la matriz **A** y el vector **B** se definen de la siguiente manera:

$$A = \begin{bmatrix} -\frac{1}{3}(h_0+h_{n-2}) & -\frac{1}{6}h_0 & 0 & 0 & . & . & 0 & 0 & 0 & -\frac{1}{6}h_{n-2} \\ h_0 & 2(h_0+h_1) & h_1 & 0 & . & . & 0 & 0 & 0 & 0 \\ 0 & h_1 & 2(h_0+h_1) & h_2 & . & . & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & h_{n-5} & 2(h_{n-5}+h_{n-4}) & h_{n-4} & 0 \\ 0 & 0 & 0 & 0 & . & . & 0 & h_{n-4} & 2(h_{n-4}+h_{n-3}) & h_{n-3} \\ h_{n-2} & 0 & 0 & 0 & . & . & 0 & 0 & h_{n-3} & 2(h_{n-3}+h_{n-2}) \end{bmatrix}$$

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ . \\ . \\ S_{n-4} \\ S_{n-3} \\ S_{n-2} \end{bmatrix}, \qquad B = \begin{bmatrix} -\frac{u_1 - u_0}{h_0} + \frac{u_{n-1} - u_{n-2}}{h_{n-2}} \\ 6(\frac{u_2 - u_1}{h_1} - \frac{u_1 - u_0}{h_0}) \\ 6(\frac{u_3 - u_2}{h_2} - \frac{u_2 - u_1}{h_1}) \\ . \\ . \\ 6(\frac{u_{n-3} - u_{n-4}}{h_{n-4}} - \frac{u_{n-4} - u_{n-5}}{h_{n-5}}) \\ 6(\frac{u_{n-2} - u_{n-3}}{h_{n-3}} - \frac{u_{n-3} - u_{n-4}}{h_{n-4}}) \\ 6(\frac{u_{n-1} - u_{n-2}}{h_{n-2}} - \frac{u_{n-2} - u_{n-3}}{h_{n-3}}) \end{bmatrix}$$

La solución de este sistema entrega los valores de $S_0, S_1, ..., S_{n-2}$, que al sustituir en las definiciones (1), (3), (4), y (5) proporcionan los coeficientes de cada polinomio segmentado del trazador cúbico cerrado:

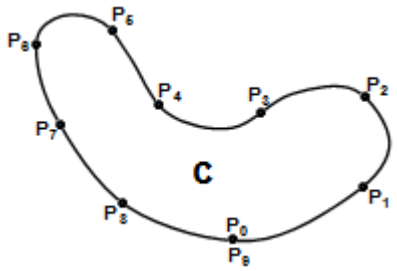
$$T_i(t) = a_i(t - t_i)^3 + b_i(t - t_i)^2 + c_i(t - t_i) + d_i; \quad t \in [t_i, t_{i+1}]; \quad i = 0,1,...,n-2$$

En algunos coeficientes se hace referencia al valor de S_{n-1}, el cual con la definición establecida, debe sustituirse con el valor de S_0

6.11.10 Modelado de curvas cerradas planas con el trazador cúbico paramétrico cerrado

Esta es una aplicación importante del trazador cúbico cerrado como dispositivo para modelar figuras cerradas en el plano y para calcular el valor del área, como se verá en una sección posterior.

Sean $P_0, P_1, P_2, ..., P_{n-1}$ puntos tomados en sentido antihorario de una figura cerrada **C** en el plano, con $P_{n-1}=P_0$, con coordenadas (x_i, y_i), $i=0, 1, 2,..., n-1$,



Estos puntos ya no pueden expresarse mediante una relación funcional **y(x)**, por lo que se debe usar un enfoque paramétrico.

Si se supone que **C** es una figura simple y regular por segmentos, se la puede representar aproximadamente en forma paramétrica mediante funciones **x(t), y(t)** definidas en segmentos delimitados por los puntos expresados en forma paramétrica con el parámetro **t**

$$P_i = \begin{cases} (t_i, x_i), & i = 0,1,2,...,n-1 \\ (t_i, y_i) \end{cases}$$

En donde los valores del parámetro t_i, **i=0, 1, 2, ...,n-1** pertenecen a algún subconjunto ordenado de **R**

Para aproximar la curva **C** en forma paramétrica una opción es elegir como funciones **x(t), y(t)**, los polinomios del trazador cúbico cerrado. Siendo estos polinomios de tercer grado, la continuidad entre los segmentos puede llegar hasta la segunda derivada y el gráfico final será simple y suave:

$$C = \begin{cases} T_x(t) \\ T_y(t) \end{cases}$$

$T_x(t)$ se lo construye con los puntos base (t_i, x_i), **i=0, 1, 2, ...,n-1**

$$T_x(t) = \begin{cases} T_{x,0}(t), & t_0 \le t \le t_1 \\ T_{x,1}(t), & t_1 \le t \le t_2 \\ & \cdots \\ T_{x,n-2}(t), & t_{n-2} \le t \le t_{n-1} \end{cases}$$

$T_y(t)$ se lo construye con los puntos base (t_i, y_i), **i=0, 1, 2, ...,n-1**

$$T_y(t) = \begin{cases} T_{y,0}(t), & t_0 \le t \le t_1 \\ T_{y,1}(t), & t_1 \le t \le t_2 \\ & \cdots \\ T_{y,n-2}(t), & t_{n-2} \le t \le t_{n-1} \end{cases}$$

El gráfico de puntos **(Tx(t), Ty(t))**, $t \in [t_0, t_{n-1}]$, será una aproximación para la curva **C**.

Una aplicación de los polinomios segmentados del trazador paramétrico cerrado es el cálculo del área de la región que encierra. Esta aplicación de interés será tratada en una siguiente sección.

El uso manual de estos dispositivos matemáticos involucra muchos cálculos y sería muy laborioso y susceptible a errores numéricos, especialmente si la cantidad de puntos es grande, por esto se requiere un tratamiento computacional. Para este artículo se usó como soporte el lenguaje computacional **Python** disponible como software libre y por ofrecer facilidades para este tipo de aplicaciones.

6.11.11 Instrumentación computacional del trazador cúbico paramétrico cerrado

La función **trazador_cerrado** desarrollada en lenguaje **Python** recibe separadamente los vectores **x, y** conteniendo las coordenadas de los puntos de la curva **C** que se desea modelar y los valores del parámetro **t**, y entrega cada uno de los polinomios segmentados del trazador cúbico paramétrico cerrado $\mathbf{T_x(t)}$ y $\mathbf{T_x(t)}$.

Si se incluye un vector adicional con puntos del parámetro, el resultado que entrega es un vector con puntos evaluados con los polinomios paramétricos segmentados. Si estos puntos son muy cercanos, se pueden usar para la graficación. Los polinomios que entrega $\mathbf{T_x(t)}$ y $\mathbf{T_x(t)}$ también pueden ser usados para calcular el área de la región que encierra, como se verá en el capítulo de integración numérica.

```
import numpy as np
from sympy import*
def trazador_cerrado(z,u,s=[]):
    n=len(z)
    h=np.zeros([n-1])
    A=np.zeros([n-1,n-1]);B=np.zeros([n-1]);S=np.zeros([n])
    a=np.zeros([n-1]);b=np.zeros([n-1]);c=np.zeros([n-1]);d=np.zeros([n-1])
    if n<3:
        T=[]
        return
    for i in range(n-1):
        h[i]=z[i+1]-z[i]
    A[0,0]=-1/3*(h[0]+h[n-2])              #Construir el sistema de ecuaciones
    A[0,1]=-1/6*h[0]
    A[0,n-2]=-1/6*h[n-2]
    B[0]=-(u[1]-u[0])/h[0]+(u[n-1]-u[n-2])/h[n-2]
    for i in range(1,n-2):
        A[i,i-1]=h[i-1]
        A[i,i]=2*(h[i-1]+h[i])
        A[i,i+1]=h[i]
        B[i]=6*((u[i+1]-u[i])/h[i]-(u[i]-u[i-1])/h[i-1])
    A[n-2,0]=h[n-2]
    A[n-2,n-3]=h[n-3]
    A[n-2,n-2]=2*(h[n-3]+h[n-2])
    B[n-2]=6*((u[n-1]-u[n-2])/h[n-2]-(u[n-2]-u[n-3])/h[n-3])
    r=np.linalg.solve(A,B)                 #Resolver el sistema
```

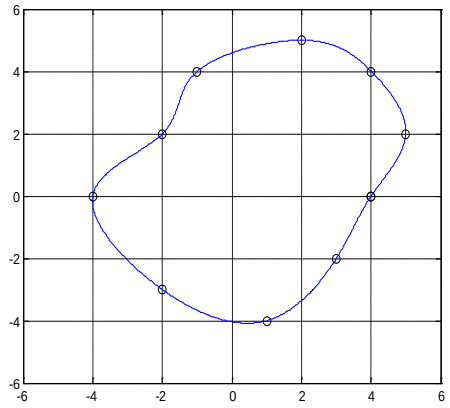
```
for i in range(n-1):
    S[i]=r[i]
S[n-1]=r[0]
for i in range(n-1):                          #Coeficientes de los polinomios
    a[i]=(S[i+1]-S[i])/(6*h[i])
    b[i]=S[i]/2
    c[i]=(u[i+1]-u[i])/h[i]-(2*h[i]*S[i]+h[i]*S[i+1])/6
    d[i]=u[i]
try:
    if len(s)==0:                             #Detecta si es un vector
        pass
except TypeError:
    s=[s]
if len(s)==0:                                 #Construir el trazador
    t=Symbol('t')
    T=[]
    for i in range(n-1):
        p=expand(a[i]*(t-z[i])**3+b[i]*(t-z[i])**2+c[i]*(t-z[i])+d[i])
        T=T+[p]
    return T                                  #Retorna los polinomios
else:                                         #Evaluar el trazador
    m=len(s)
    q=np.zeros([m])
    for k in range(m):
        t=s[k]
        for i in range(n-1):
            if t>=z[i] and t<=z[i+1]:
                q[k]=a[i]*(t-z[i])**3+b[i]*(t-z[i])**2+c[i]*(t-z[i])+d[i]
    if m>2:
      k=m-1
      i=n-2
      q[k]=a[i]*(t-z[i])**3+b[i]*(t-z[i])**2+c[i]*(t-z[i])+d[i]
    if len(q)==1:
      return q[0]                             #Retorna un valor
    else:
      return q                                #Retorna un vector
```

Ejemplo. Colocar el trazador cúbico paramétrico cerrado sobre los siguientes diez puntos:

$P_0(4,0)$, $P_1(5,2)$, $P_2(4,4)$, $P_3(2,5)$, $P_4(-1,4)$, $P_5(-2,2)$, $P_6(-4,1)$, $P_7(-2,-3)$, $P_8(1,-4)$, $P_9(3,-2)$

El punto inicial se debe repetir al final para cerrar la figura

```
>>> from trazador_cerrado import*
>>> x=[4,5,4,2,-1,-2,-4,-2,1,3,4]
>>> y=[0,2,4,5,4,2,0,-3,-4,-2,0]
>>> t=[1,2,3,4,5,6,7,8,9,10,11]
>>> import pylab as pl
>>> s=pl.arange(1,11,0.01)
>>> Txs=trazador_cerrado(t,x,s)
>>> Tys=trazador_cerrado(t,y,s)
>>> pl.plot(x,y,'o')
>>> pl.plot(Txs,Tys,'-')
>>> pl.show()
```



6.11.12 Procedimiento en Python para tomar puntos de una imagen

En las siguientes líneas se describe mediante instrucciones en la ventana interactiva, un procedimiento para de subir figuras a la pantalla gráfica de Python de la cual, utilizando el mouse, se toman puntos cuyas coordenadas se pueden usar para construir el trazador cúbico.

1. Subir la imagen a **Paint** y guardar la imagen con formato png: **nombre.png**

2. En la ventana interactiva de Python desplegar la imagen y tomar puntos con la función **ginput.** Las coordenadas de los puntos se almacenan en una lista **pts**

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import matplotlib.image as mpimg

>>> img=mpimg.imread('h:nombre.png')
>>> imgplot = plt.imshow(img)

>>> n=int(input('Cuantos puntos: '))
>>> pts = np.array(plt.ginput(n,timeout=0))
>>> print('Puntos: \n',pts)                    Listar los puntos
```

3. Cerrar el gráfico de la pantalla

```
>>> plt.plot(pts[:,0],pts[:,1],'o')
>>> plt.plot(pts[:,0],pts[:,1],'-')
>>> plt.show()
```

Graficar los puntos

4. Definir las coordenadas de los puntos para el trazador cúbico

```
x=pts[:,0]
y=pts[:,1]
```

5. Colocar el polinomio de interpolación o el trazador cúbico (directo, paramétrico, o cerrado)

6.12 Ejercicios con el trazador cúbico

1. Con los siguientes datos **(x, y):**
(1.2, 4.6), (1.5, 5.3), (2.4, 6.0), (3.0, 4.8), (3.8, 3.1)

a) Encuentre el trazador cúbico natural
b) Encuentre el valor interpolado para **x=2.25**

2. Con los siguientes datos (x, y):
y'(1.2) = 1, (1.5, 5.3), (2.4, 6.0), (3.0, 4.8), y'(3.8) = -1
Suponga que los puntos extremos están en el eje horizontal.

a) Encuentre el trazador cúbico sujeto
b) Encuentre el valor interpolado para **x=2.25**

Use las funciones instrumentadas en Python para comprobar sus resultados

3. Dados los siguientes datos obtenidos mediante observación:

(2, 3), (4, 5), (5, 8), (7, 3), (8, 1), (9, 1)

a) Obtenga y grafique un polinomio de interpolación incluyendo todos los datos. Use la función Lagrange instrumentada en este curso.

b) Obtenga y grafique el trazador cúbico natural con los mismos datos, usando la función trazador desarrollada en este curso. Observe y evalúe los resultados obtenidos.

4. Conocidos los siguientes datos (x, y):
`(1.0, 0), (1.2, 5.1), (2.5, 6.2), (3.2, 4.8), 3.8, 0)`
El tramo izquierdo debe comenzar con una inclinación de 45º y el tramo final debe terminar horizontal.

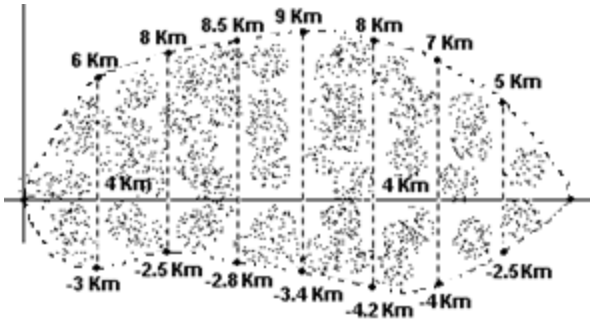
a) Encuentre y grafique aproximadamente el **trazador cúbico sujeto**
b) Encuentre el valor interpolado con el trazador cúbico para **x=3.0**

5. Las coordenadas x(t), y(t) del recorrido de un cohete registradas en los instantes t: 0, 1, 2, 3, 4, 5, 6 fueron respectivamente: x(t): 3, 2, 2, 4, 5, 7, 8; y(t): 0, 2, 4, 7, 7, 4, 0

Con esta información y usando el **trazador cúbico natural paramétrico**

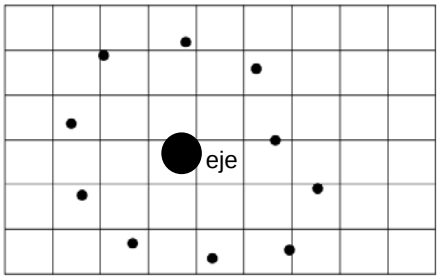
a) Estime la altura del cohete cuando t = 2.5
b) Estime la altura del cohete cuando x = 4.5

6. En el siguiente gráfico se muestra la zona de un derrame de petróleo ocurrido en cierta región. Los puntos referenciales han sido tomados a una distancia horizontal de 4 Km.



Con el **trazador cúbico paramétrico cerrado**, conecte y grafique el perfil de la zona afectada por el derrame de petróleo. Incluya los puntos extremos izquierdo y derecho.

7. En el siguiente gráfico se muestran 10 puntos del perfil de una leva (mecanismo que al girar alrededor del eje mueve verticalmente un dispositivo acoplado). Para el diseño se requiere dibujar el perfil. Las distancias son cm.



Tome aproximadamente las coordenadas de los 10 puntos (un entero y un decimal). El lado de cada cuadrado mide un cm. Use el **trazador cúbico paramétrico cerrado** para conectar y graficar el perfil del dispositivo.

7 INTEGRACIÓN NUMÉRICA

Introducimos este capítulo mediante un problema de interés práctico en el que el modelo matemático resultante es la evaluación de un integral. El objetivo es evaluar numéricamente un integral y estimar la precisión del resultado.

Problema.

La siguiente función es básica en estudios estadísticos y se denomina función de densidad de la distribución Normal Estándar: Esta función permite calcular la probabilidad que la variable **Z** pueda tomar algún valor en un intervalo **[a, b]** según la siguiente definición:

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} \qquad z \in \Re$$

$$\mathbf{P(a \le Z \le b)} = \int_a^b f(z)dz$$

Debido a que esta función no es integrable analíticamente, es necesario utilizar métodos numéricos para estimar el valor de **P**.

Sea **f** una función integrable y acotada definida en un intervalo **[a, b]** con **a<b∈R**.

Es de interés calcular el valor de **A**:

$$\mathbf{A} = \int_a^b \mathbf{f(x)dx}$$

En general hay dos situaciones en las que son útiles los métodos numéricos:

1) El integral existe pero es muy difícil o no se puede evaluar analíticamente.

2) Únicamente se conocen puntos de **f(x)** pero se requiere calcular en forma aproximada el integral debajo de la curva que incluye a los puntos dados

En ambos casos se trata de sustituir **f(x)** por alguna función más simple, siendo importante además estimar la precisión del resultado obtenido.

7.1 Fórmulas de Newton-Cotes

El enfoque básico para obtener fórmulas de integración numérica consiste en aproximar la función a ser integrada por el polinomio de interpolación. Las fórmulas así obtenidas se denominan de Newton-Cotes.

Si los puntos están espaciados regularmente, se puede usar el conocido polinomio de diferencias finitas o de Newton y para estimar el error se incluye el término del error del polinomio de interpolación:

$$f(x) = p_n(s) + E_n(s), \quad s = \frac{x - x_0}{h}$$

$$p_n(s) = f_0 + \Delta f_0 s + \frac{1}{2!}\Delta^2 f_0\, s(s-1) + \frac{1}{3!}\Delta^3 f_0\, s(s-1)(s-2) + \ldots + \frac{1}{n!}\Delta^n f_0 s(s-1)(s-2)\ldots(s-n+1)$$

$$E_n(s) = \binom{s}{n+1} h^{n+1} f^{(n+1)}(z),\ x_0 < x < x_n$$

El uso de polinomios de diferente grado para aproximar a **f** genera diferentes fórmulas de integración numérica

7.1.1 Fórmula de los trapecios

Esta fórmula usa como aproximación para **f** un polinomio de primer grado:

$$f(x) \cong p_1(s) = f_0 + \Delta f_0\, s$$

En general, la aproximación mediante una sola recta en el intervalo **[a, b]** tendría poca precisión por lo que conviene dividir el intervalo **[a, b]** en **m** sub-intervalos y colocar en cada uno, una recta cuyos extremos coinciden con **f(x)**. Por simplicidad se usarán puntos regularmente espaciados a una distancia **h** contante: **h=(b-a)/m**

La figura geométrica en cada intervalo es un trapecio. Sea A_i el área del trapecio **i** y sea T_i el error de truncamiento respectivo, es decir la diferencia entre el área debajo de **f(x)** y el área de cada trapecio **i, i = 1, 2, 3, …, m**. El área se puede aproximar con:

$$A = \int_a^b f(x)dx \approx A_1 + A_2 + A_3 + \ldots + A_m = \sum_{i=1}^{m} A_i$$

Mientras que el error de truncamiento total será:

$$T = T_1 + T_2 + T_3 + \ldots + T_m$$

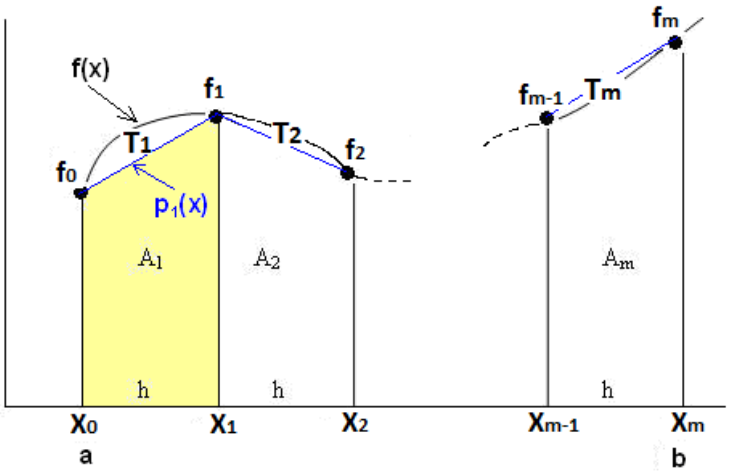
Si no hay discontinuidades en el intervalo **[a, b]** , es claro que

$$m \to \infty \ \Rightarrow\ T \to 0 \Rightarrow \sum_{i=1}^{m} A_i \to A$$

$$A = \int_a^b f(x)dx \approx \int_{x_0}^{x_1} p_1(s)dx + \int_{x_1}^{x_2} p_1(s)dx + \ldots. + \int_{x_{m-1}}^{x_m} p_1(s)dx$$

$$A = A_1 + A_2 + \ldots + A_m$$

m: cantidad de franjas espaciadas regularmente en **h,** siendo $h = \frac{b-a}{m}$



Aproximación del integral con el área de trapecios

Para obtener la fórmula es suficiente encontrar el valor del área de un trapecio y luego extender este resultado a las demás, como se indica a continuación.

Área del primer trapecio:

$$A_1 = \int_{x_0}^{x_1} p_1(s)dx = \int_{x_0}^{x_1} (f_0 + \Delta f_0 s)dx$$

Mediante las sustituciones:

$s = (x - x_0)/h$

$x = x_0 \Rightarrow S = 0$

$x = x_1 \Rightarrow S = 1$

$dx = h\, ds$

$$A_1 = \int_0^1 (f_0 + \Delta f_0 s)h\,ds = h\left[\, f_0 s + \frac{1}{2}\Delta f_0 s^2 \,\right]_0^1 = h\left[f_0 + \frac{1}{2}(f_1 - f_0)\right]$$

$A_1 = \frac{h}{2}[f_0 + f_1]$, es la conocida fórmula de la geometría para el área de un trapecio

El resultado anterior se extiende directamente a los restantes intervalos:

$$A \cong A_1 + A_2 + \ldots. + A_m = \frac{h}{2}[f_0 + f_1] + \frac{h}{2}[f_1 + f_2] + \ldots.. + \frac{h}{2}[f_{m-1} + f_m]$$

$$A \cong \frac{h}{2}[f_0 + 2f_1 + 2f_2 + \ldots.. + 2f_{m-1} + f_m]$$

Definición: Fórmula de los trapecios compuesta

$$A \cong \frac{h}{2}\,[f_0 + 2f_1 + 2f_2 + \ldots.. + 2f_{m-1} + f_m] = \frac{h}{2}[f_0 + 2\sum_{i=1}^{m-1} f_i + f_m],$$

m es la cantidad de trapecios.

Ejemplo. La siguiente función no es integrable analíticamente. $\mathbf{f(x) = \sqrt{x}\ sen(x)}$, $\mathbf{0 \le x \le 2}$
Use la fórmula de los trapecios con **m = 4**, para obtener una respuesta aproximada del integral

Solución

$$A = \int_0^2 f(x)dx = \int_0^2 \sqrt{x}\,sen(x)dx$$

$$A \cong \frac{h}{2}\,[f_0 + 2f_1 + 2f_2 + 2f_3 + f_4], \quad h = \frac{b-a}{m} = \frac{2-0}{4} = 0.5$$

$$= \frac{0.5}{2}[f(0) + 2f(0.5) + 2f(1) + 2f(1.5) + f(2)] = 1.5225$$

Sin embargo, es necesario poder contestar una pregunta fundamental: ¿Cuál es la precisión del resultado calculado?

7.1.2 Error de truncamiento en la fórmula de los trapecios

Es necesario estimar el error en el resultado obtenido con los métodos numéricos

Error al aproximar **f(x)** mediante $\mathbf{p_1(x)}$ en el primer sub-intervalo:

$$E_1(s) = \binom{s}{2} h^2 f^{(2)}(z_1),\ x_0 < z_1 < x_1$$

Cálculo del área correspondiente al error en el uso del polinomio para aproximar a **f**

$$T_1 = \int_{x_0}^{x_1} E_1(s)dx = \int_{x_0}^{x_1} \binom{s}{2} h^2 f^{(2)}(z_1)dx = \int_{x_0}^{x_1} \frac{1}{2} s(s-1) h^2 f^{(2)}(z_1)dx$$

Usando las sustituciones anteriores:

$s = (x - x_0)/h$
$x = x_0 \Rightarrow S = 0$
$x = x_1 \Rightarrow S = 1$
$dx = h\,ds$

$$T_1 = \frac{h^3}{2} \int_0^1 s(s-1) f''(z_1)ds\ ,$$

Con el teorema del valor medio para integrales, puesto que **s(s-1)** no cambia de signo en el intervalo **[0,1]**, se puede sacar del integral la función **f''** evaluada en algún punto $\mathbf{z_1}$, desconocido, en el mismo intervalo.

$$T_1 = \frac{h^3}{2} f''(z_1) \int_0^1 s(s-1)ds, \quad x_0 < z_1 < x_1$$

Luego de integrar se obtiene

$$T_1 = -\frac{h^3}{12} f''(z_1), \quad x_0 < z_1 < x_1$$

Este resultado se extiende a los **m** sub-intervalos en la integración

$$T = T_1 + T_2 + \ldots + T_m$$

$$T = -\frac{h^3}{12} f''(z_1) - \frac{h^3}{12} f''(z_2) - \ldots - \frac{h^3}{12} f''(z_m)$$

$$T = -\frac{h^3}{12} [f''(z_1) + f''(z_2) + \ldots + f''(z_m)]$$

$$T = -\frac{h^3}{12} m f''(z),$$ siendo **z** algún valor en el intervalo **(a, b)**

Mediante la sustitución $h = \frac{b-a}{m}$, la fórmula se puede expresar de la siguiente forma

Definición. Fórmula del error de truncamiento en la fórmula de los trapecios

$$T = -\frac{h^2}{12} (b-a) f''(z), \quad a \le z \le b$$

Esta fórmula se utiliza para acotar el error de truncamiento.

Siendo **z** desconocido, para acotar el error se puede usar un criterio conservador tomando el mayor valor de $|f''(z)|, a \le z \le b$. Este criterio no proporciona una medida muy precisa para el error y su aplicación puede ser un problema más complicado que la misma integración, por lo cual se puede intentar usar como criterio para estimar el error, la definición de convergencia indicada al inicio de esta sección, siempre que **f** sea una función integrable y acotada:

$$m \to \infty \Rightarrow \sum_{i=1}^{m} A_i \to A$$

Sean $A_m = \sum_{i=1}^{m} A_i$, $A_{m'} = \sum_{i=1}^{m'} A_i$ dos aproximaciones sucesivas con $m' > m$ trapecios

Entonces, se puede estimar el error de truncamiento absoluto del resultado con:

$$T \cong |A_m - A_{m'}|$$

Mientras que el error de truncamiento relativo se puede estimar con:

$$t \cong \frac{|A_m - A_{m'}|}{|A_{m'}|}$$

Hay que tener la precaución de no usar valores muy grandes para **m** por el efecto del error de redondeo acumulado en las operaciones aritméticas y que pudiera reducir la precisión en el resultado.

En caso de conocer únicamente puntos de **f**, al no disponer de más información para estimar el error de truncamiento, un criterio simple puede ser tomar el mayor valor de las segundas diferencias finitas como una aproximación para la segunda derivada en la fórmula del error, siempre que no cambien significativamente:

$$T = -\frac{h^2}{12}(b-a)\,f''(z), \quad a \le z \le b$$

$$f''(z) \cong \frac{\Delta^2 f_i}{h^2}$$

$$T \cong -\frac{(b-a)}{12}\Delta^2 f_i$$

Esta fórmula también pudiera usarse para estimar el error de truncamiento en el caso de que **f(x)** se conozca explícitamente y **m** haya sido especificado como un valor fijo. Habría que tabular las diferencias finitas para los puntos usados en la integración numérica y estimar el error con la fórmula anterior.

Ejemplo. Estime cuantos trapecios deben usarse para integrar **f(x) = sen(x)** en el intervalo **[0,2]** de tal manera que la respuesta tenga el error absoluto menor a 0.0001

Solución

Se requiere que error de truncamiento cumpla la condición:

$$|T| < 0.0001$$

$$\left| -\frac{h^2}{12}(b-a)\,f''(z) \right| < 0.0001$$

Siendo el valor de **z** desconocido se debe usar el máximo valor de **f''(z) = -sen(z), 0<z<2**

$$\max |f''(z)| = 1$$

$$\left| -\frac{h^2}{12}(2-0)(1) \right| < 0.0001$$

De donde $h^2 < 0.0006$

$h < 0.0245$

$$\frac{b-a}{m} < 0.0245$$

Entonces $m > (2-0)/0.0245$

$m > 81.63 \Rightarrow m = 82$ trapecios

Ejemplo. Estime cuantos trapecios deben usarse para integrar $\mathbf{f(x) = \sqrt{x}\ sen(x)}$ en el intervalo **[0,2]** de tal manera que la respuesta tenga el error absoluto menor a 0.0001

Solución

Se requiere que error de truncamiento cumpla la condición:

$$\mathbf{|\ T\ | < 0.0001}$$

$$\mathbf{\left| -\frac{h^2}{12}(b-a)\ f''(z)\right| < 0.0001}$$

Siendo el valor de **z** desconocido se debe usar el máximo valor de

$$\mathbf{f''(z) = \frac{cos(z)}{\sqrt{z}} - \sqrt{z}sen(z) - \frac{sen(z)}{4\sqrt{z^3}}},\quad 0 < z < 2$$

Problema demasiado complicado pues se requeriría derivar para estimar el mayor valor de la derivada y acotar el error.

Para usar la estimación del error con la aproximación de diferencias finitas, debe especificarse el valor de **m**.

Otra opción para estimar el error es comparar resultados con valores sucesivos de **m** hasta que la diferencia sea suficientemente pequeña. Para los cálculos conviene instrumentar una función en Python.

Ejemplo. Estime el error de truncamiento en la integración de $\mathbf{f(x) = \sqrt{x}\ sen(x)}$, **0≤x≤2** con la fórmula de los trapecios con **m = 4**

Solución

Se tabulan los cinco puntos de f(x) para estimar el error, aproximando la derivada con la diferencia finita respectiva. Con el criterio usual, se toma el mayor valor.

X	f	Δf	Δ^2f
0.0	0.0000	0.3390	0.1635
0.5	0.3390	0.5025	-0.1223
1.0	0.8415	0.3802	-0.3159
1.5	1.2217	0.0643	
2.0	1.2859		

$$\mathbf{T \cong -\frac{(b-a)}{12}\ \Delta^2 f_i = -\frac{(2-0)}{12}(-0.3159) = 0.0527}$$

7.1.3 Instrumentación computacional de la fórmula de los trapecios

Si se quiere integrar debajo de una función dada en forma explícita, conviene definir una función de Python para evaluar el integral dejando como dato el número de trapecios m. Los resultados calculados pueden usarse como criterio para estimar el error de truncamiento.

En la siguiente instrumentación debe suministrarse la función **f**, el intervalo de integración **a, b** y la cantidad de franjas o trapecios **m**

```
def trapecios(f,a,b,m):
    h=(b-a)/m
    s=0
    for i in range(1,m):
        s=s+f(a+i*h)
    r=h/2*(f(a)+2*s+f(b))
    return r
```

Ejemplo. Probar la función trapecios para integrar **f(x)=sen(x), 0≤x≤2** con 5 trapecios

```
>>> from trapecios import*
>>> from math import*
>>> def f(x): return sin(x)
>>> r=trapecios(f,0,2,5)
>>> r
1.3972143342518983
```

Se puede probar con más trapecios para mejorar la aproximación

```
>>> r=trapecios(f,0,2,100)
>>> r
1.4160996313378889
>>> r=trapecios(f,0,2,200)
>>> r
1.4161350353038356
```

Los resultados tienden hacia el valor exacto a medida que se incrementa el número de trapecios. Estos resultados pueden usarse como criterio de convergencia y para estimar la precisión.

Compare con el resultado calculado con la función **integrate** de la librería **SymPy** de Python

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=sin(x)
```

```
>>> r=integrate(f,(x,0,2))
>>> r
-cos(2) + 1                                   Respuesta analítica simbólica
>>> r.evalf(6)
1.41615                                       Respuesta numérica con 6 dígitos
```

Ejemplo. La siguiente función no es integrable analíticamente: $S = \int_0^2 \sqrt{x}\ sen(x)dx$

Use la fórmula de los trapecios computacionalmente para obtener la respuesta aproximada y estimar el error.

```
>>> from trapecios import*
>>> from math import*
>>> def f(x):return sqrt(x)*sin(x)
>>> r=trapecios(f,0,2,20);print(r)
1.5323419014333037
>>> r=trapecios(f,0,2,40);print(r)
1.5325751387611677
>>> r=trapecios(f,0,2,60);print(r)
1.5326151217909427
>>> r=trapecios(f,0,2,80);print(r)
1.5326285905297556
>>> r=trapecios(f,0,2,100);print(r)
1.5326346742219736
>>> r=trapecios(f,0,2,120);print(r)
1.5326379217488284
```

Estos resultados pueden usarse como criterio de convergencia y para estimar la precisión. El último resultado tiene cinco decimales que no cambian y se pueden considerar correctos.

Compare con el valor calculado con la función **integrate** de la librería **SymPy** de Python

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=sqrt(x)*sin(x)
>>> r=integrate(f,(x,0,2))
>>> r
-5*sqrt(2)*cos(2)*gamma(5/4)/(4*gamma(9/4))+5*sqrt(2)*sqrt(pi)*
fresnelc(2/sqrt(pi))*gamma(5/4)/(8*gamma(9/4))
>>> r.evalf(6)
1.53264
```

La respuesta analítica en el método de Python es aproximada mediante funciones especiales que requieren desarrollos en series de potencias.

Ejemplo. Dados puntos de una función **f:**

(0.1, 1.8), (0.2, 2.6), (0.3, 3.0), (0.4, 2.8), (0.5, 1.9)

Calcule el área **A** debajo de **f** aproximada mediante cuatro trapecios y estime el error en el resultado obtenido.

Solución

$$A = \frac{0.1}{2}[1.8 + 2(2.6) + 2(3.0) + 2(2.8) + 1.9] = 1.0250$$

Al no disponer de más información, se usarán las diferencias finitas para estimar el error

X	f	Δf	$\Delta^2 f$
0.1	1.8	0.8	-0.4
0.2	2.6	0.4	-0.6
0.3	3.0	-0.2	-0.7
0.4	2.8	-0.9	
0.5	1.9		

$$T \cong -\frac{(b-a)}{12}\Delta^2 f_i = -\frac{(0.5-0.1)}{12}(-0.7) = 0.0233$$

Esto indicaría que solamente podemos tener confianza en el primer decimal

7.1.4 Fórmula de Simpson

Esta fórmula usa como aproximación para **f** un polinomio de segundo grado, o parábola:

$$f(x) \cong p_2(s) = f_0 + \Delta f_0\, s + \frac{1}{2}\Delta^2 f_0\, s(s-1)$$

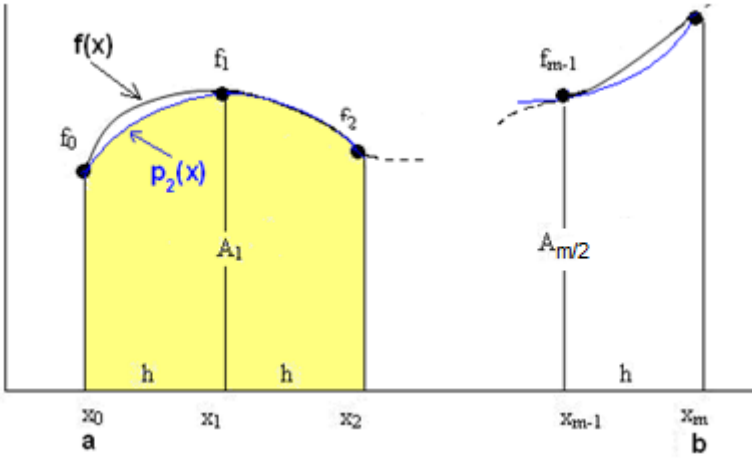
La integración se realiza dividiendo el intervalo de integración **[a, b]**, en subintervalos, incluyendo tres puntos en cada uno para colocar una parábola.

Por simplicidad se usarán puntos regularmente espaciados a una distancia **h**

$$A = \int_a^b f(x)dx \cong \int_{x_0}^{x_2} p_2(s)dx + \int_{x_2}^{x_4} p_2(s)dx + \ldots + \int_{x_{m-2}}^{x_m} p_2(s)dx$$

$$A \cong A_1 + A_2 + \ldots + A_{m/2}$$

El área de dos intervalos consecutivos es aproximada mediante el área debajo de parábolas. Los puntos son numerados desde cero: $x_0, x_1, \ldots, x_m$, e donde **m** debe ser un número par, así la cantidad de parábolas es **m/2**.



h=(b-a)/m

Para obtener la fórmula se debe encontrar el valor del área para una parábola:

$$A_1 = \int_{x_0}^{x_2} p_2(s)dx = \int_{x_0}^{x_2} [f_0 + \Delta f_s s + \frac{1}{2}\Delta^2 f_0 s(s-1)]\, dx$$

Mediante las sustituciones:

$s = (x - x_0)/h$

$x = x_0 \Rightarrow s = 0$

$x = x_2 \Rightarrow s = 2$

$dx = h\, ds$

$$A_1 = \int_0^2 (f_0 + \Delta f_0 s + \frac{1}{2}\Delta^2 f_0 s(s-1)hds$$

Luego de integrar, sustituir las diferencias finitas y simplificar se tiene

$A_1 = \frac{h}{3}[f_0 + 4f_1 + f_2]$ es el área debajo de la parábola en la primera franja

Por lo tanto, habiendo **m/2** franjas, el área total es la suma:

$$A = A_1 + A_2 + \ldots. + A_{m/2}$$

Después de sustituir y simplificar se obtiene la fórmula de integración

Definición. Fórmula de Simpson compuesta (fórmula de las parábolas)

$$A = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + \ldots.. + 2f_{m-2} + 4f_{m-1} + f_m]$$

m es un parámetro para la fórmula (debe ser un número par)

7.1.5 Error de truncamiento en la fórmula de Simpson

Del análisis del error se llega a

$$T = -\frac{h^4}{180}(b-a)\, f^{(4)}(z), \quad a<z<b$$

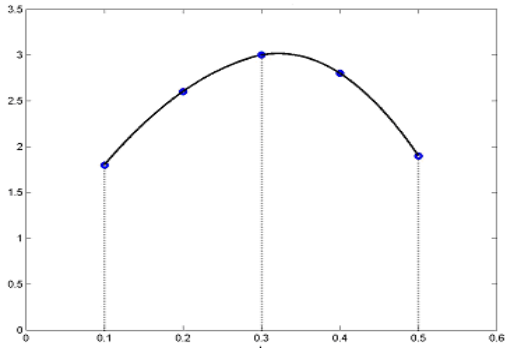
Esta fórmula puede usarse para acotar el error de truncamiento. Bajo ciertas consideraciones y si se fija m, se puede estimar la derivada con la diferencia finita correspondiente, con el criterio usual del mayor valor:

$$f^{(4)}(z) \cong \frac{\Delta^4 f_i}{h^4}, \quad T \cong -\frac{(b-a)}{180}\Delta^4 f_i$$

Ejemplo. Dados puntos de una función **f: (0.1, 1.8), (0.2, 2.6), (0.3, 3.0), (0.4, 2.8), (0.5, 1.9)**

Calcule el área debajo de **f** mediante una aproximación con parábolas.

Solución



La suma del área debajo de las dos parábolas, con la fórmula anterior:

$$A = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + f_4]$$

$$A = \frac{0.1}{3}(1.8 + 4(2.6) + 2(3.0) + 4(2.8) + 1.9) \quad = \quad 1.0433$$

Este resultado es aproximadamente igual al área debajo de **f.**

Estimar el error en el resultado obtenido

Al no disponer de más información, se usarán las diferencias finitas para estimar el error

X	f	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
0.1	1.8	0.8	-0.4	-0.2	0.1
0.2	2.6	0.4	-0.6	-0.1	
0.3	3.0	-0.2	-0.7		
0.4	2.8	-0.9			
0.5	1.9				

$$T = -\frac{h^4}{180}(b-a)\, f^{(4)}(z), \quad a<z<b$$

$$T \cong -\frac{(b-a)}{180}\Delta^4 f_i = -\frac{(0.5-0.1)}{180}(0.1) = -0.00022$$

Esto indicaría que podemos tener confianza en la respuesta hasta el tercer decimal. Resultado mejor que el obtenido con la Regla de los Trapecios

Ejemplo. Si **g** es una función diferenciable en el intervalo **[a,b]**, la longitud del arco de la curva **g** en ese intervalo se puede calcular con el integral $S = \int_a^b \sqrt{1+[g'(x)]^2}\,dx$

Calcular la longitud del arco de la curva **g(x)=sen(x), x∈[0, 2]** usando 2 parábolas **(m = 4)** y estimar el error en el resultado:

Solución

Longitud del arco: $S = \int_a^b \sqrt{1+[g'(x)]^2}\,dx = \int_a^b \sqrt{1+\cos^2(x)}\,dx$

$$S = \int_0^2 f(x)dx \quad \text{con} \quad f(x) = \sqrt{1+\cos^2(x)}$$

$$h = \frac{b-a}{m} = \frac{2-0}{4} = 0.5$$

$$S = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + f_4] = \frac{0.5}{3}[f(0) + 4f(0.5) + 2f(1) + 4f(1.5) + f(2)]$$

$$S = 2.3504$$

Estimación del error

Dado que **m** está especificado, se usa una aproximación de diferencias finitas para aproximar la derivada en la fórmula del error de truncamiento

$$T = -\frac{h^4}{180}(b-a)\, f^{(4)}(z), \quad a<z<b$$

X	f	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
0.0	1.4142	-0.0837	-0.1101	0.1698	-0.0148
0.5	1.3305	-0.1938	0.0597	0.1551	
1.0	1.1366	-0.1341	0.2148		
1.5	1.0025	0.0806			
2.0	1.0831				

$$T \cong -\frac{(b-a)}{180}\Delta^4 f_i = -\frac{(2-0)}{180}(-0.0148) = 0.00016$$

Es una estimación de la cota del error de truncamiento. La estimación del error utilizando en la cota el máximo de $f^{(4)}(x)$, $0<x<2$, sería muy laborioso.

7.1.6 Instrumentación computacional de la fórmula de Simpson

La función recibirá a la función **f** definida en forma simbólica, el intervalo de integración **a, b** y la cantidad de franjas **m**

```
def simpson(f, a, b, m):
    h=(b-a)/m
    s=0
    x=a
    for i in range (1,m):
        s=s+2*(i%2+1)*f(x+i*h)          #Coeficientes 4, 2, 4, 2, ...
    s=h/3*(f(a)+s+f(b))
    return s
```

Ejemplo. Integrar $f(x)=\sqrt{1+\cos^2(x)}$ $0\leq x\leq 2$, con la fórmula de Simpson iterativamente hasta que el error de truncamiento sea menor que **0.0001**

```
>>> from math import*
>>> from simpson import*
>>> def f(x): return sqrt(1+cos(x)**2)
>>> r=simpson(f,0,2,4)
>>> r
2.3504136916156644
>>> r=simpson(f,0,2,8)
>>> r
2.351646207253357
>>> r=simpson(f,0,2,12)
>>> r
2.3516805638227076
>>> r=simpson(f,0,2,16)
>>> r
2.3516862293406477
```

Respuesta con la precisión requerida, mediante la estimación numérica del error de truncamiento.

7.1.7 Error de truncamiento vs. Error de redondeo

En las fórmulas de integración numérica el error de truncamiento depende de **h**

Fórmula de los Trapecios: $\mathbf{T = -\frac{h^2}{12}(b - a)\, f''(z) = O(h^2)}$

Fórmula de Simpson: $\mathbf{T = -\frac{h^4}{180}(b - a)\, f^{(4)}(z) = O(h^4)}$

Además $\mathbf{h = \frac{b - a}{m}}$

Para ambas fórmulas, cuando $\mathbf{h \to 0 \Rightarrow T \to 0}$

Está claro que la fórmula de Simpson converge más rápido, supuesto que **h<1**

Por otra parte, al evaluar cada operación aritmética se puede introducir un error de redondeo $\mathbf{R_i}$ si no se conservan todos los dígitos decimales en los cálculos numéricos. La suma de estos errores es el error de redondeo acumulado **R**. Mientras más sumas se realicen, mayor es la cantidad de términos que acumulan error de redondeo.

$$\mathbf{R = R_1 + R_2 + R_3 + ... + R_m}$$

Estos errores de redondeo pueden tener signos diferentes y anularse, pero también puede ocurrir que tengan igual signo, por lo tanto el valor puede crecer

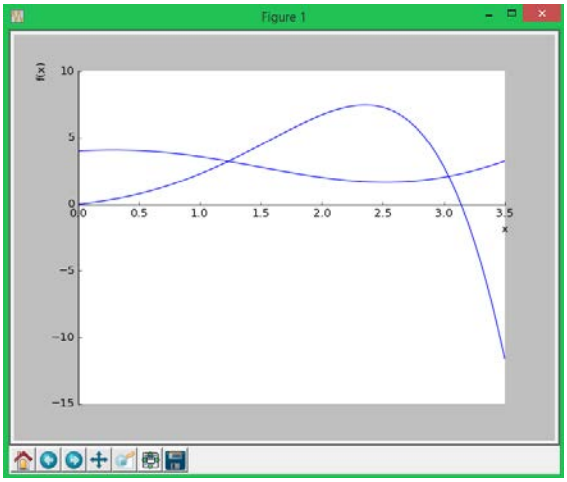
Siendo $\mathbf{m = \frac{b - a}{h}}$, cuando $\mathbf{h \to 0 \Rightarrow m \to \infty \Rightarrow}$ **R** puede crecer

En conclusión, para prevenir el crecimiento de **R**, es preferible usar fórmulas que no requieran que **m** sea muy grande para obtener el resultado con una precisión especificada. Este es el motivo para preferir la fórmula de Simpson sobre la fórmula de los Trapecios.

Ejemplo. Encontrar el área entre **f(x) = 4 + cos(x+1)**, y **g(x)=e^x sen(x)**, que incluya el área entre las intersecciones de **f** y **g** en el primer cuadrante. Use la Regla de Simpson, **m=10.**

Graficación de las funciones con la librería SymPy

```
>>> ========= RESTART =========
>>> from sympy import*
>>> x=Symbol('x')
>>> f=4+x*cos(x+1)
>>> g=exp(x)*sin(x)
>>> h=g-f
>>> print(h)
-x*cos(x + 1) + exp(x)*sin(x) - 4
>>> plot(f,g,(x,0,3.5))
```



Cálculo numérico de las intersecciones de **f** y **g** con el método Bisección y el área de **g - f** con el método Simpson

```
>>> ========= RESTART =========
>>> from biseccion import*
>>> from math import*
>>> def h(x):return -x*cos(x + 1) + exp(x)*sin(x) - 4
>>> a=biseccion(h,1,1.5,0.0001);print(a)
1.23370361328125
>>> b=biseccion(h,2.5,3.5,0.0001);print(b)
3.04071044921875
>>> from simpson import*
>>> s=simpson(h,a,b,10);print(s)
6.53910536745223
>>> s=simpson(h,a,b,20);print(s)
6.5393200536340546
```

7.1.8 Fórmula de Simpson 3/8

En esta fórmula se usan polinomios de interpolación de tercer grado. La fórmula básica requiere 4 puntos, con tres subintervalos espaciados en una distancia h. La fórmula compuesta utiliza **m** subintervalos espaciados en una distancia h con **h = (b-a)/m**

Definición. Fórmula de Simpson 3/8 compuesta

$$A = \frac{3h}{8}[f_0 + 3f_1 + 3f_2 + 2f_3 + 3f_4 + 3f_5 + 2f_6 + 3f_7 + 3f_8 + 2f_9 + \ldots + f_m]$$

m es un parámetro para la fórmula (**m** debe ser múltiplo de 3)

7.1.9 Error de truncamiento en la fórmula de Simpson 3/8 compuesta

Del análisis del error se llega a

$$T = \frac{h^4}{80}(b-a)\, f^{(4)}(z), \quad a<z<b$$

Esta fórmula puede usarse para acotar el error de truncamiento. Bajo ciertas consideraciones y si se fija m, se puede estimar la derivada con la diferencia finita correspondiente, con el criterio usual del mayor valor:

$$f^{(4)}(z) \cong \frac{\Delta^4 f_i}{h^4}, \quad T \cong \frac{(b-a)}{80}\Delta^4 f_i$$

7.1.10 Cálculo de la longitud de un arco definido con ecuaciones paramétricas

Una curva **C** puede darse en forma paramétrica con las ecuaciones:

x = f(t)
y = g(t), t ∈ [a, b]

Si no hay intersecciones entre **f** y **g**, entonces, la longitud **L** del arco **C** se puede calcular con la integral:

$$L = \int_a^b \sqrt{(f'(t))^2 + (g'(t))^2}\,dt$$

7.1.11 Cálculo aproximado de la longitud de un arco con el polinomio de interpolación paramétrico

Ejemplo. Las coordenadas **x(t), y(t)** del recorrido de un cohete registradas en los instantes **t=0, 1, 2, 3** fueron respectivamente: **x(t)=2, 1, 3, 4, y(t)=0, 4, 5, 0**

Con esta información, use polinomios de interpolación de tercer grado construido con los cuatro puntos para aproximar **f** y **g** . Con estos polinomios estime la longitud de la trayectoria del cohete.

t: [0, 1, 2, 3],
x(t): [2, 1, 3, 4],
y(t): [0, 4, 5, 0]

Con el polinomio de Lagrange se obtienen los polinomios de interpolación paramétricos

$$p_x(t) = f(t) = -\frac{2}{3}t^3 + \frac{7}{2}t^2 - \frac{23}{6}t + 2$$
$$p_y(t) = g(t) = -\frac{1}{2}t^3 + \frac{9}{2}t$$

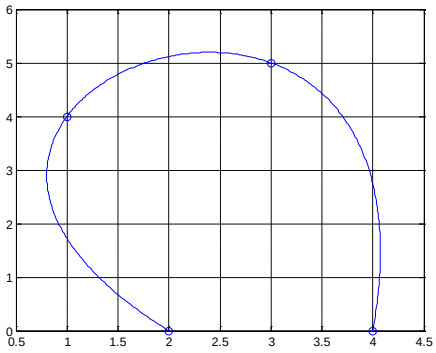


Gráfico aproximado de la trayectoria del cohete: **(x(t), y(t)),t∈[0,3]**

Longitud aproximada del recorrido del cohete

$\mathbf{f'(t) = -2t^2 + 7t - 23/6}$
$\mathbf{g'(t) = -1.5t^2 + 4.5}$

$$\mathbf{L = \int_a^b \sqrt{(f'(t))^2 + (g'(t))^2}\,dt = \int_0^3 \sqrt{(-2t^2 + 7t - 23/6)^2 + (-1.5t^2 + 4.5)^2}\,dt}$$

$$\mathbf{u(t) = \sqrt{(-2t^2 + 7t - 23/6)^2 + (-1.5t^2 + 4.5)^2}}$$

Con la fórmula de Simpson, $\mathbf{m = 4}$, $\Rightarrow$ $\mathbf{h = \frac{3-0}{4} = 0.75}$

$$\mathbf{L = \frac{h}{3}[u_0 + 4u_1 + 2u_2 + 4u_3 + u_4] = \frac{0.75}{3}[u(0)+4u(0.75)+2u(1.5)+4u(2.25)+u(3)] = 12.2011}$$

Cálculo con el integrador simbólico de SymPy de Python

```
>>> from sympy import*
>>> t=Symbol('t')
>>> px=-2/3*t**3+7/2*t**2-23/6*t+2
>>> f=-2/3*t**3+7/2*t**2-23/6*t+2
>>> g=-1/2*t**3+9/2*t
>>> df=diff(f)
>>> dg=diff(g)
>>> u=sqrt(df**2+dg**2)
>>> L=integrate(u,(t,0,3))
>>> float(L)
12.2047
```

7.1.12 Cálculo de la longitud del arco usando el Trazador Cúbico paramétrico

El trazador cúbico paramétrico puede ser usado para calcular en forma aproximada la longitud de un arco definido paramétricamente.

Una curva **C** puede aproximarse mediante el Trazador Cúbico paramétrico:

$$\mathbf{C = \begin{cases} T_x(t) \\ T_y(t) \end{cases}, \quad t \in [a, b]}$$

Si no hay intersecciones entre los segmentos del Trazador Cúbico, entonces, la longitud del arco **C** se puede calcular con la integral:

$$\mathbf{L = \int_a^b \sqrt{(T_x'(t))^2 + (T_y'(t))^2}\,dt}$$

Debido a la mayor cantidad de cálculos y la posibilidad de errores numéricos, conviene instrumentar un método computacional

Instrumentación computacional usando el integrador simbólico de SymPy

```
# Cálculo de la longitud de un arco con
# ecuaciones paramétricas con variable v, parámetro s
from sympy import*
def arco(Tx,Ty,v,s):
    n=len(Tx)
    r=0
    for i in range(n):
        x=Tx[i]
        dx=diff(x,v)
        y=Ty[i]
        dy=diff(y,v)
        r=r+integrate(sqrt(dx**2+dy**2),(v,s[i],s[i+1]))
    return float(r)
```

El tiempo de ejecución del integrador simbólico fue excesivamente alto. En este caso, una mejor opción es usar un método numérico para aproximar la integración. En la siguiente instrumentación se adaptó la fórmula de Simpson para manejo simbólico. Para esta instrumentación, la función debe especificarse y también la variable independiente. Los resultados fueron muy precisos y el tiempo de ejecución se redujo significativamente.

Instrumentación computacional usando un método numérico para integrar

```
# Fórmula de Simpson con f simbólica, variable v, y m franjas
def simpsons(f, v, a, b, m):
    h=(b-a)/m
    s=0
    for i in range (1,m):
        s=s+2*(i%2+1)*f.subs(v,a+i*h)
    s=h/3*(f.subs(v,a)+s+f.subs(v,b))
    return s


# Cálculo de la longitud de un arco con ecuaciones
# paramétricas Tx, Ty, con variable v, parámetro s, y m franjas
from simpsons import*
from sympy import*
def arco(Tx,Ty,v,s,m):
    n=len(Tx)
    r=0
```

```
    for i in range(n):
        x=Tx[i]
        dx=diff(x,v)
        y=Ty[i]
        dy=diff(y,v)
        f=sqrt(dx**2+dy**2)
        r=r+simpsons(f,v,s[i],s[i+1],m)
    return r
```

Ejemplo. Las coordenadas **x(s), y(s)** del recorrido de un cohete registradas en los instantes **s=0, 1, 2, 3** fueron respectivamente: **x(s)=2,1,3,4, y(s)=0,4,5,0**

Con esta información y usando el Trazador Cúbico Natural paramétrico estime la longitud de la trayectoria del cohete

Solución computacional

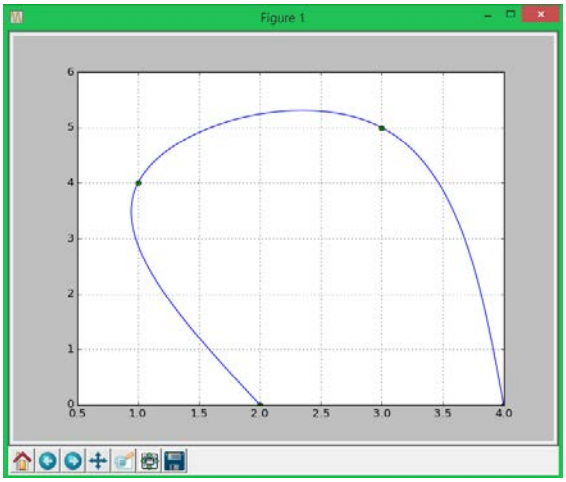


Gráfico aproximado de la trayectoria del cohete con el Trazador Cúbico Natural paramétrico

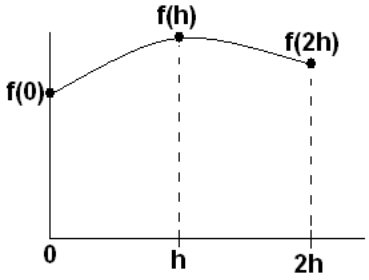
```
>>> from trazador_natural import*
>>> x=[2,1,3,4]
>>> y=[0,4,5,0]
>>> s=[0,1,2,3]
>>> Tx=trazador_natural(s,x)
>>> Ty=trazador_natural(s,y)
>>> from sympy import*
>>> t=Symbol('t')
>>> from arco import*
>>> s=arco(Tx,Ty,t,s,20)
>>> print(s)
12.0940
```

7.2 Obtención de fórmulas de integración numérica con el método de coeficientes indeterminados

En esta sección se describe una técnica para obtener fórmulas de integración numérica.

El procedimiento consiste en proponer una fórmula conteniendo algunas incógnitas. Esta fórmula es aplicada a casos conocidos con el propósito de obtener ecuaciones, de las cuales se determinan finalmente los valores para las incógnitas.

El ejemplo usa este método para obtener una fórmula de tres puntos espaciados en h:



Fórmula propuesta

$$\mathbf{A = \int_0^{2h} f(x)dx = c_0 f(0) + c_1 f(h) + c_2 f(2h)}$$

Deben determinarse los coeficientes $\mathbf{c_0, c_1, c_2}$. Para obtenerlos, se usarán tres casos con polinomios de grado 0, 1 y 2 con los cuales queremos que se cumpla la fórmula. Es suficiente considerar la forma más simple de cada caso:

1) f(x) = 1,

$$\mathbf{A = \int_0^{2h} (1)dx = 2h = c_0 f(0) + c_1 f(h) + c_2 f(2h) = c_0(1) + c_1(1) + c_2(1) \Rightarrow c_0 + c_1 + c_2 = 2h}$$

2) f(x) = x,

$$\mathbf{A = \int_0^{2h} xdx = 2h^2 = c_0 f(0) + c_1 f(h) + c_2 f(2h) = c_0(0) + c_1(h) + c_2(2h) \Rightarrow c_1 + 2c_2 = 2h}$$

3) $f(x) = x^2$,

$$\mathbf{A = \int_0^{2h} x^2 dx = \frac{8}{3}h^3 = c_0 f(0) + c_1 f(h) + c_2 f(2h) = c_0(0) + c_1(h^2) + c_2(4h^2) \Rightarrow c_1 + 4c_2 = \frac{8}{3}h}$$

Resolviendo las tres ecuaciones resultantes se obtienen: $\mathbf{c_0 = \frac{h}{3},\ c_1 = \frac{4h}{3},\ c_2 = \frac{h}{3}}$

Reemplazando en la fórmula propuesta se llega a la conocida fórmula de Simpson

$$\mathbf{A = \frac{h}{3}(f(0) + 4f(h) + f(2h))}$$

La obtención de la fórmula implica que es exacta si **f** es un polinomio de grado menor o igual a dos. Para otra **f**, será una aproximación equivalente a sustituir **f** por un polinomio de grado dos.

7.3 Cuadratura de Gauss

Las fórmulas de Newton-Cotes estudiadas utilizan polinomios de interpolación construidos con puntos fijos equidistantes. Estas fórmulas son exactas si la función es un polinomio de grado menor o igual al polinomio de interpolación respectivo.

Si se elimina la restricción de que los puntos sean fijos y equidistantes, entonces las fórmulas de integración contendrán incógnitas adicionales.

La cuadratura de Gauss propone una fórmula general en la que los puntos incluidos no son fijos como en las fórmulas de Newton-Cotes:

$$A = \int_a^b f(x)dx = c_0 f(t_0) + c_1 f(t_1) + \ldots + c_m f(t_m)$$

Los puntos $t_0, t_1, \ldots, t_m$, son desconocidos. Adicionalmente también deben determinarse los coeficientes $c_0, c_1, \ldots, c_m$

El caso simple es la fórmula de dos puntos. Se usa el método de los coeficientes indeterminados para determinar las cuatro incógnitas

7.3.1 Fórmula de la cuadratura de Gauss con dos puntos

Fórmula propuesta

$$A = \int_a^b f(x)dx = c_0 f(t_0) + c_1 f(t_1)$$

Por simplicidad se usará el intervalo **[-1, 1]** para integrar. Mediante una sustitución será extendido al caso general:

$$A = \int_{-1}^{1} f(t)dt = c_0 f(t_0) + c_1 f(t_1)$$

Habiendo cuatro incógnitas se tomarán cuatro casos en los que la fórmula sea exacta. Se usarán polinomios de grado 0, 1, 2, 3. Es suficiente considerarlos en su forma más simple:

1) f(t)=1, $A = \int_{-1}^{1} (1)dt = 2 = c_0 f(t_0) + c_1 f(t_1) = c_0(1) + c_1(1) \Rightarrow 2 = c_0 + c_1$

2) f(t)=t, $A = \int_{-1}^{1} t dt = 0 = c_0 f(t_0) + c_1 f(t_1) = c_0 t_0 + c_1 t_1 \Rightarrow 0 = c_0 t_0 + c_1 t_1$

3) f(t)=t^2, $A = \int_{-1}^{1} t^2 dt = \frac{2}{3} = c_0 f(t_0) + c_1 f(t_1) = c_0 t_0^2 + c_1 t_1^2 \Rightarrow \frac{2}{3} = c_0 t_0^2 + c_1 t_1^2$

4) f(t)=t^3, $A = \int_{-1}^{1} t^3 dt = 0 = c_0 f(t_0) + c_1 f(t_1) = c_0 t_0^3 + c_1 t_1^3 \Rightarrow 0 = c_0 t_0^3 + c_1 t_1^3$

Se genera un sistema de cuatro ecuaciones no-lineales. Una solución para este sistema se obtiene con facilidad mediante simple sustitución:

Los valores $c_0 = c_1 = 1$ satisfacen a la ecuación 1.

De la ecuación 2 se tiene $t_0 = -t_1$. Esto satisface también a la ecuación 4.

Finalmente, sustituyendo en la ecuación 3: $2/3 = (1)(-t_1)^2 + (1)(t_1)^2$ se obtiene:

$t_1 = \frac{1}{\sqrt{3}}$, entonces, $t_0 = -\frac{1}{\sqrt{3}}$ y se reemplazan en la fórmula propuesta:

Definición: Fórmula de cuadratura de Gauss con dos puntos

$$A = \int_{-1}^{1} f(t)dt = c_0 f(t_0) + c_1 f(t_1) = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}})$$

Esta simple fórmula es exacta si **f** es un polinomio de grado menor o igual a tres. Para otra **f** es una aproximación equivalente a sustituir **f** con un polinomio de grado tres.

Ejemplo. Calcule $A = \int_{-1}^{1} (2t^3 + t^2 - 1)dt$ con la Cuadratura de Gauss

Solución

$$A = \int_{-1}^{1} f(t)dt = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}}) = [2(-\frac{1}{\sqrt{3}})^3 + (-\frac{1}{\sqrt{3}})^2 - 1] + [2(\frac{1}{\sqrt{3}})^3 + (\frac{1}{\sqrt{3}})^2 - 1] = -4/3$$

La respuesta es exacta pues **f** es un polinomio de grado 3

Mediante un cambio de variable se extiende la fórmula al caso general:

$$A = \int_{a}^{b} f(x)dx = c_0 f(t_0) + c_1 f(t_1)$$

Sea $x = \frac{b-a}{2}t + \frac{b+a}{2}$

Se tiene que $t = 1 \Rightarrow x = b$, $t = -1 \Rightarrow x = a$, $dx = \frac{b-a}{2}dt$

Sustituyendo se tiene

Definición: Fórmula general de Cuadratura de Gauss con dos puntos

$$A = \int_{a}^{b} f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}t + \frac{b+a}{2})dt = \frac{b-a}{2}\left[f(-\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}) + f(\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2})\right]$$

Ejemplo. Calcule $A = \int_1^2 xe^x dx$ con la fórmula de la Cuadratura de Gauss con dos puntos

$$A = \int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f(\frac{b-a}{2}t + \frac{b+a}{2})dt \cong \frac{b-a}{2}[f(t_0) + f(t_1)],$$

$$t_0 = -\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}, \quad t_1 = \frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}$$

$$t_0 = -\frac{2-1}{2}\frac{1}{\sqrt{3}} + \frac{2+1}{2} = -\frac{1}{2\sqrt{3}} + \frac{3}{2} = 1.2113$$

$$t_1 = \frac{2-1}{2}\frac{1}{\sqrt{3}} + \frac{2+1}{2} = \frac{1}{2\sqrt{3}} + \frac{3}{2} = 1.7886$$

$$A \cong \frac{1}{2}[f(1.2113) + f(1.7886)] = 7.3825$$

La respuesta exacta con cuatro decimales es **7.3890.** Se observa que usando únicamente dos puntos se tiene una precisión mejor que usando la fórmula de Simpson con tres puntos.

Ejemplo. Calcule el valor del integral $A = \int_0^{1/2} \frac{5}{1+5u+u^2} du$

Aplique la cuadratura de Gauss con **m = 1, 2, 3.** Con estos resultados haga una estimación de la precisión de la respuesta.

$$A = \int_0^{1/2} \frac{5}{1+5u+u^2} du, \quad f(u) = \frac{5}{1+5u+u^2}$$

$$A = \int_a^b f(x)dx = \frac{b-a}{2}\left[f(-\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}) + f(\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2})\right]$$

m=1: $A \cong \int_0^{1/2} f(u)du = \frac{1/2-0}{2}\left[f(-\frac{1/2-0}{2}\frac{1}{\sqrt{3}} + \frac{1/2+0}{2}) + f(\frac{1/2-0}{2}\frac{1}{\sqrt{3}} + \frac{1/2+0}{2})\right]$

$= 0.25(3.2479 + 1.598) = 1.2117$

m=2 : $A \cong \int_0^{1/4} f(u)du + \int_{1/4}^{1/2} f(u)du$

$$= \frac{1/4-0}{2}\left[f(-\frac{1/4-0}{2}\frac{1}{\sqrt{3}} + \frac{1/4+0}{2}) + f(\frac{1/4-0}{2}\frac{1}{\sqrt{3}} + \frac{1/4+0}{2})\right]$$
$$+ \frac{1/2-0}{2}\left[f(-\frac{1/2-1/4}{2}\frac{1}{\sqrt{3}} + \frac{1/2+1/4}{2}) + f(\frac{1/2-1/4}{2}\frac{1}{\sqrt{3}} + \frac{1/2+1/4}{2})\right]$$

$= 1.2237$

$$m=3 : A \cong \int_0^{1/6} f(u)du + \int_{1/6}^{2/6} f(u)du + \int_{2/6}^{3/6} f(u)du$$

$$= 1.2251$$

$E_1 = 1.2237 - 1.2117 = 0.0120$
$E_2 = 1.2251 - 1.2237 = 0.0014$

La respuesta tiene el error en el orden 10^{-3} aproximadamente

7.3.2 Instrumentación computacional de la cuadratura de Gauss

```
from math import*
def cgauss(f,a,b):
    t0=-(b-a)/2*1/sqrt(3)+(b+a)/2
    t1= (b-a)/2*1/sqrt(3)+(b+a)/2
    s = (b-a)/2*(f(t0) + f(t1))
    return s
```

Ejemplo. Use la función cgauss para calcular $A = \int_1^2 xe^x dx$

```
>>> from cgauss import*
>>> from math import*
>>> def f(x): return x*exp(x)
>>> s=cgauss(f,1,2)
>>> s
7.3832726466092975
```

Para mejorar la precisión de ésta fórmula se la puede aplicar más de una vez dividiendo el intervalo de integración en sub-intervalos.

Ejemplo. Aplique dos veces la cuadratura de Gauss en el ejemplo anterior

$$A = \int_1^2 xe^x dx = A_1 + A_2 = \int_1^{1.5} xe^x dx + \int_{1.5}^2 xe^x dx$$

En cada subintervalo se aplica la fórmula de la Cuadratura de Gauss:

```
>>> s=cgauss(f,1,1.5)+cgauss(f,1.5,2)
>>> s
7.388682930054991
```

7.3.3 Instrumentación extendida de la cuadratura de Gauss

Se puede dividir el intervalo en más sub-intervalos para obtener mayor precisión. Conviene definir una función en Python con **m** como parámetro. **m** es la cantidad de sub-intervalos Para estimar la precisión del resultado, se compararán valores consecutivos, observando la convergencia del integral.

```
from cgauss import*
def cgaussm(f,a,b,m):
    h=(b-a)/m
    s=0
    x=a
    for i in range(m):
        a=x+i*h
        b=x+(i+1)*h
        s=s+cgauss(f,a,b)
    return s
```

Ejemplo. Aplicar sucesivamente la Cuadratura de Gauss incrementando el número de sub-intervalos, hasta que la respuesta tenga cuatro decimales exactos

```
>>> from cgaussm import*
>>> from math import*
>>> def f(x): return x*exp(x)
>>> s=cgaussm(f,1,2,1);print(s)
7.3832726466092975
>>> s=cgaussm(f,1,2,2);print(s)
7.388682930054991
>>> s=cgaussm(f,1,2,3);print(s)
7.388981945691469
>>> s=cgaussm(f,1,2,4);print(s)
7.389032587252556
>>> s=cgaussm(f,1,2,5);print(s)
7.389046459210758
```

En el último cálculo se han usado 5 sub-intervalos. El valor obtenido tiene cuatro decimales fijos

Para obtener fórmulas de cuadratura de Gauss con más puntos no es práctico usar el método de coeficientes indeterminados. Se puede usar un procedimiento general basado en la teoría de polinomios ortogonales. En la bibliografía se pueden encontrar estas fórmulas así como expresiones para estimar el error de truncamiento pero imprácticas para su uso.

7.4 Integrales con dominio no acotado

Estos integrales se denominan integrales impropios del Tipo I. Adicionalmente debe verificarse si el integral definido converge a un valor finito.

Ocasionalmente puede ser de interés calcular integrales cuyos límites no están acotados, por lo tanto no se pueden aplicar directamente las fórmulas de integración . Mediante alguna sustitución deben reducirse a una forma con límites acotados.

Ejemplo. Calcule $A = \int_0^{\infty} \frac{dx}{(1+x^2)^3}$ con la Cuadratura de Gauss, **m = 1, 2, 4**

Antes de la sustitución conviene separar el integral en dos sub-intervalos

$$A = \int_0^{\infty} \frac{dx}{(1+x^2)^3} = \int_0^{1} \frac{dx}{(1+x^2)^3} + \int_1^{\infty} \frac{dx}{(1+x^2)^3} = A_1 + A_2$$

$\mathbf{A_1}$ se puede calcular inmediatamente con la Cuadratura de Gauss

Para $\mathbf{A_2}$ se hace la sustitución

$$x = 1/t$$

$$x \to \infty \Rightarrow t \to 0,\ \ x = 1 \Rightarrow t = 1,\ \ dx = -1/t^2\, dt$$

$$A_2 = \int_1^{\infty} \frac{dx}{(1+x^2)^3} = \int_1^{0} \frac{1}{(1+1/t^2)^3}\left(-\frac{dt}{t^2}\right) = \int_0^{1} \frac{t^4}{(1+t^2)^3}\, dt$$

Ahora se puede aplicar también la Cuadratura de Gauss

Resultados calculados:

m=1: $A = A_1+A_2 = 0.6019$
m=2: $A = A_1+A_2 = 0.5891$
m=4: $A = A_1+A_2 = 0.5890$

El último resultado tiene un error en el orden de 0.0001

La librería **Sympy** de **Python** incorpora métodos numéricos para resolver algunos integrales con límites no acotados:

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=1/(1+x**2)**3
>>> r=integrate(f,(x,0,oo))
>>> float(r)
0.5890486225480862
```

El símbolo ∞ son dos letras **o** juntas

7.5 Integrales con rango no acotado

Estos integrales se denominan integrales impropios de Tipo II. Adicionalmente debe verificarse si el integral definido converge a un valor finito. Mediante alguna sustitución deben reducirse a una forma integrable eliminando los puntos singulares.

Ejemplo. Calcule $A = \int_0^1 \frac{sen(x)}{(x-1)^{2/3}} dx$ con la fórmula de Simpson, **m = 4**, y estime el error

No se puede aplicar directamente la fórmula de Simpson debido al punto singular. Se debe realizar una sustitución adecuada para eliminarlo:

$(x-1)^{1/3} = u$

$\Rightarrow x = u^3 + 1$: $x = 1 \Rightarrow u = 0$; $x = 0 \Rightarrow u = -1$; $dx = 3u^2 du$

$$A = \int_0^1 \frac{sen(x)}{(x-1)^{2/3}} dx = \int_{-1}^0 \frac{sen(u^3+1)}{u^2}(3u^2 du) = \int_{-1}^0 3sen(u^3+1)du = \int_{-1}^0 f(u)du$$

Integral bien definido en **[-1,0]**

Regla de Simpson, m = 4

$A = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + f_4]$

$m = 4 \Rightarrow h = 0.25$

A = 0.25/3(f(-1)+4f(-0.75)+2f(-0.5)+4f(-0.25)+f(0)) = 1.9735...

Estimación del error

Al no disponer de más información, se usarán las diferencias finitas para estimar el error:

X	f	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
-1	0	1.6394	-0.9761	0.5090	-0.2124
-0.75	1.6394	0.6633	-0.4671	0.2966	
-0.5	2.3026	0.1961	-0.1705		
-0.25	2.4988	0.0256			
0	2.5244				

$$T \cong -\frac{(b-a)}{180}\Delta^4 f_i = -\frac{(0-(-1))}{180}(-0.2124) = 0.0012$$

Nota. Este integral no puede evaluarse analíticamente. Si se usa la fórmula de Simpson incrementando el número de franjas, el resultado tiende a **1.9730...** consistente con el error calculado con la fórmula anterior.

Python no resuelve directamente el integral con el punto singular:

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=sin(x)/(x-1)**(2/3)
>>> r=integrate(f,(x,0,1))
...                                    Produce una respuesta compleja
```

Respuesta obtenida con la librería Sympy de Python luego de la sustitución:

```
>>> from sympy import*
>>> u=Symbol('u')
>>> f=3*sin(u**3+1)
>>> r=integrate(f,(u,-1,0))
>>> float(r)
1.9729653832914493
```

Ejemplo. Calcule $A = \int_0^1 \frac{sen(x)}{x} dx$ con la Cuadratura de Gauss con **m = 1, 2**

La fórmula de la Cuadratura de Gauss no requiere evaluar **f** en los extremos, por lo tanto se puede aplicar directamente:

$$A = \int_a^b f(x)dx = \frac{b-a}{2}\left[f(-\frac{b-a}{2}\frac{1}{\sqrt{3}}+\frac{b+a}{2}) + f(\frac{b-a}{2}\frac{1}{\sqrt{3}}+\frac{b+a}{2}) \right]$$

Aplicando la fórmula en el intervalo [0, 1]:

A = 0.94604

Aplicando la fórmula una vez en cada uno de los intervalos [0, 0.5] y [0.5, 1] y sumando:

A = 0.94608

Comparando ambos resultados se puede estimar el error en el quinto decimal.

NOTA: Para calcular el integral no se puede aplicar la fórmula de Simpson pues ésta requiere evaluar **f(x)** en los extremos.

Python tiene una función especial para resolver este tipo de integrales

```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=sin(x)/x
>>> r=integrate(f,(x,0,1))
>>> r
Si(1)
>>> float(r)
0.946083070367183
```

Ejemplo. Calcule $S = \int_2^{\infty} \frac{e^{1/x}}{x^{5/3}} dx$ con la fórmula de Simpson, **m=4.**

Solución

Esta expresión no es integrable ni analítica, ni numéricamente con las fórmulas anteriores.

La solución requiere sustituciones para ambos tipos de integrales impropios

Resolver el integral impropio tipo I

Sustitución: $u=1/x$: $x\to\infty \Rightarrow u\to 0$, $x=2 \Rightarrow u=1/2$, $dx=-du/u^2$

$$S = \int_2^{\infty} \frac{e^{1/x}}{x^{5/3}} dx = \int_{1/2}^{0} \frac{e^u}{1/u^{5/3}} (-du/u^2) = \int_0^{1/2} \frac{e^u}{u^{1/3}} du$$

Se obtiene un integral impropio de tipo II, no integrable ni analítica ni numéricamente con las fórmulas anteriores.

Resolver el integral impropio tipo II

Sustitución: $t=u^{1/3}$: $u=0 \Rightarrow t=0$, $u=1/2 \Rightarrow t=(1/2)^{1/3}$, $du=3t^2dt$

$$S = \int_0^{1/2} \frac{e^u}{u^{1/3}} du = \int_0^{(1/2)^{1/3}} \frac{e^{t^3}}{t} 3t^2 dt = \int_0^{(1/2)^{1/3}} 3te^{t^3} dt$$

La expresión final no es integrable analíticamente pero si numéricamente:

Regla de Simpson, m=4

$$S \cong \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + f_4], \quad f(t) = 3te^{t^3}$$

$$m = 4 \Rightarrow h = \frac{(1/2)^{1/3}}{4} = 0.198425$$

$$S \cong 0.066141\,(f(0)+4f(0.198425)+2f(0.396850)+4f(0.595275)+f(0.793700)) = 1.169439$$

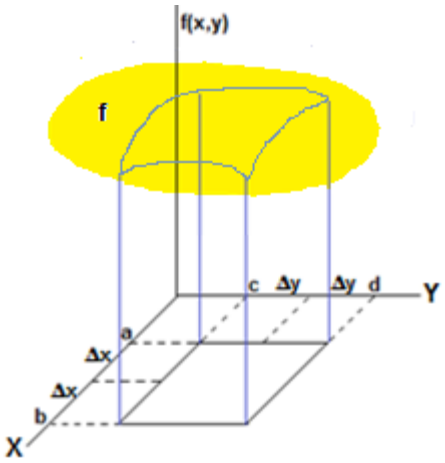
Python tiene métodos para integrar numéricamente este integral. Pero es necesario, como conocimiento formativo, entender e instrumentar nuestra versión de estos métodos.

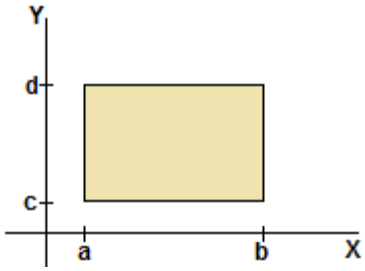
```
>>> from sympy import*
>>> x=Symbol('x')
>>> f=exp(1/x)/x**(5/3)
>>> r=integrate(f,(x,2,oo))
>>> float(r)
1.1674203419719458
```

7.6 Integrales múltiples

Para evaluar integrales múltiples se pueden usar las reglas de integración numérica anteriores integrando separadamente con cada variable.

Suponer que se desea calcular el integral $\int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy$, usando la fórmula de Simpson con una parábola (**m = 2**) en cada dirección. El intervalo de integración en cada dirección **X** e **Y** debe dividirse en dos subintervalos con espaciamiento Δx y Δy respectivamente. La regla se puede aplicar integrando en la dirección **X** fijando **Y** en cada subintervalo:





Región **X-Y** sobre la que se integra **f(x,y)**

Cálculo del integral

$$m = 2 \Rightarrow \Delta x = \frac{b-a}{2}, \quad \Delta y = \frac{c-d}{2}$$

$$S = \int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy \cong \frac{\Delta y}{3}\left[\int_{x_0}^{x_2} f(x,y_0)dx + 4\int_{x_0}^{x_2} f(x,y_1)dx + \int_{x_0}^{x_2} f(x,y_2)dx\right]$$

$$\int_{x_0}^{x_2} f(x,y_0)dx \cong \frac{\Delta x}{3}\left(f(x_0,y_0) + 4f(x_1,y_0) + f(x_2,y_0)\right)$$

$$\int_{x_0}^{x_2} f(x,y_1)dx \cong \frac{\Delta x}{3}\left(f(x_0,y_1) + 4f(x_1,y_1) + f(x_2,y_1)\right)$$

$$\int_{x_0}^{x_2} f(x,y_2)dx \cong \frac{\Delta x}{3}\left(f(x_0,y_2) + 4f(x_1,y_2) + f(x_2,y_2)\right)$$

Estimación del error de truncamiento compuesto

$$T = \left|-\frac{(\Delta x)^4}{180}(b-a)\frac{\partial f^{(4)}}{\partial x^4}(z_x)\right| + \left|-\frac{(\Delta y)^4}{180}(d-c)\frac{\partial f^{(4)}}{\partial y^4}(z_y)\right|, \quad a<z_x<b,\ c<z_y<d$$

Las derivadas se las puede estimar mediante diferencias finitas

Ejemplo. Integrar $f(x,y)=\text{sen}(x+y),\ 0\le x\le 1,\ 2\le y\le 3$ con la fórmula de Simpson, m=2, en cada dirección

$$S=\int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy=\int_{2}^{3}\int_{0}^{1}\text{sen}(x+y)dxdy$$

$$m=2 \Rightarrow \Delta x=\frac{1-0}{2}=0.5,\quad \Delta y=\frac{3-2}{2}=0.5$$

$$S\cong\frac{\Delta y}{3}\left[\int_0^1 f(x,y_0)dx+4\int_0^1 f(x,y_1)dx+\int_0^1 f(x,y_2)dx\right]$$

$$\int_0^1 f(x,y_0)dx=\int_0^1 \text{sen}(x+2)dx\cong\frac{0.5}{3}\left(\text{sen}(0+2)+4\text{sen}(0.5+2)+\text{sen}(1+2)\right)=0.5741$$

$$\int_0^1 f(x,y_1)dx=\int_0^1 \text{sen}(x+2.5)dx\cong\frac{0.5}{3}\left(\text{sen}(0+2.5)+4\text{sen}(0.5+2.5)+\text{sen}(1+2.5)\right)=0.1354$$

$$\int_0^1 f(x,y_2)dx=\int_0^1 \text{sen}(x+3)dx\cong\frac{0.5}{3}\left(\text{sen}(0+3)+4\text{sen}(0.5+3)+\text{sen}(1+3)\right)=-0.3365$$

$$S\cong\frac{0.5}{3}\left[0.5741+4(0.1354)+(-0.3365)\right]=0.1299$$

En general, suponer que se desea integrar $S=\int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy$ con la regla de Simpson con **m** subintervalos en ambas direcciones. La regla se puede aplicar integrando en la dirección **X** fijando **Y** en cada intervalo**.** Los símbolos Δx y Δy denotan la distancia entre los puntos en las direcciones **X, Y** respectivamente.

$$S=\int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy=\int_c^d\left(\int_a^b f(x,y)dx\right)dy$$

$$S\cong\frac{\Delta y}{3}\left[\int_a^b f(x,y_0)dx+4\int_a^b f(x,y_1)dx+2\int_a^b f(x,y_2)dx+4\int_a^b f(x,y_3)dx+\ldots+\int_a^b f(x,y_m)dx\right]$$

$$\int_a^b f(x,y_0)dx\cong\frac{\Delta x}{3}\left(f(x_0,y_0)+4f(x_1,y_0)+2f(x_2,y_0)+4f(x_3,y_0)+\ldots+f(x_m,y_0)\right)$$

$$\int_a^b f(x,y_1)dx\cong\frac{\Delta x}{3}\left(f(x_0,y_1)+4f(x_1,y_1)+2f(x_2,y_1)+4f(x_3,y_1)+\ldots+f(x_m,y_1)\right)$$

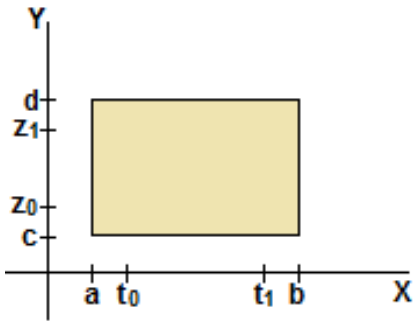
$$\int_a^b f(x,y_2)dx\cong\frac{\Delta x}{3}\left(f(x_0,y_2)+4f(x_1,y_2)+2f(x_2,y_2)+4f(x_3,y_2)+\ldots+f(x_m,y_2)\right)$$

$$\int_a^b f(x,y_3)dx\cong\frac{\Delta x}{3}\left(f(x_0,y_3)+4f(x_1,y_3)+2f(x_2,y_3)+4f(x_3,y_3)+\ldots+f(x_m,y_3)\right)$$

. . .

$$\int_a^b f(x,y_m)dx\cong\frac{\Delta x}{3}\left(f(x_0,y_m)+4f(x_1,y_m)+2f(x_2,y_m)+4f(x_3,y_m)+\ldots+f(x_m,y_m)\right)$$

Suponer que se desea calcular el integral $\int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy$, usando la fórmula de Cuadratura de Gauss con dos puntos en cada dirección. En cada subintervalo en X e Y, deben determinarse los puntos de cuadratura: t_0, t_1, z_0, z_1 respectivamente. La regla se puede aplicar integrando en la dirección **X** fijando **Y** en cada subintervalo:



$$t_0 = -\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}, \quad t_1 = \frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}$$

$$z_0 = -\frac{d-c}{2}\frac{1}{\sqrt{3}} + \frac{d+c}{2}, \quad z_1 = \frac{d-c}{2}\frac{1}{\sqrt{3}} + \frac{d+c}{2}$$

$$S = \int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy = \frac{d-c}{2}\left[\int_{x=a}^{b} f(x,z_0)dx + \int_{x=a}^{b} f(x,z_1)dx\right]$$

$$\int_{x=a}^{b} f(x,z_0)dx = \frac{b-a}{2}\left[f(t_0,z_0) + f(t_1,z_0)\right]$$

$$\int_{x=a}^{b} f(x,z_1)dx = \frac{b-a}{2}\left[f(t_0,z_1) + f(t_1,z_1)\right]$$

Ejemplo. Integrar $f(x,y) = e^{x^2+y^2}$, $0 \le x \le 1$, $0 \le y \le 1$ con la Cuadratura de Gauss con dos puntos en cada dirección

$$S = \int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy = \int_{0}^{1}\int_{0}^{1} e^{x^2+y^2}dxdy$$

$$t_0 = -\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2} = -\frac{1-0}{2}\frac{1}{\sqrt{3}} + \frac{1+0}{2} = 0.7886$$

$$t_1 = \frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2} = \frac{1-0}{2}\frac{1}{\sqrt{3}} + \frac{1+0}{2} = 0.2113$$

$$z_0 = -\frac{d-c}{2}\frac{1}{\sqrt{3}} + \frac{d+c}{2} = -\frac{1-0}{2}\frac{1}{\sqrt{3}} + \frac{1+0}{2} = 0.7886$$

$$z_1 = \frac{d-c}{2}\frac{1}{\sqrt{3}} + \frac{d+c}{2} = \frac{1-0}{2}\frac{1}{\sqrt{3}} + \frac{1+0}{2} = 0.2113$$

$$S = \int_{y=c}^{d}\int_{x=a}^{b} f(x,y)dxdy = \frac{d-c}{2}\left[\int_{x=a}^{b} f(x,z_0)dx + \int_{x=a}^{b} f(x,z_1)dx\right]$$

$$\int_{x=a}^{b} f(x,z_0)dx = \frac{b-a}{2}\left[f(t_0,z_0) + f(t_1,z_0)\right] = \frac{1-0}{2}\left[f(0.7886,0.7866) + f(0.2113,0.7886)\right] = 2.7086$$

$$\int_{x=a}^{b} f(x,z_1)dx = \frac{b-a}{2}\left[f(t_0,z_1) + f(t_1,z_1)\right] = \frac{1-0}{2}\left[f(0.7886,0.2113) + f(0.2113,0.2113)\right] = 1.5205$$

$$S = \frac{1-0}{2}\left[2.7086 + 1.5205\right] = 2.1145$$

Ejemplo. Un lago tiene forma aproximadamente rectangular de 200 m x 400 m. Se ha trazado un cuadriculado y se ha medido la profundidad en metros en cada cuadrícula de la malla como se indica en la tabla siguiente:

X / Y	0	100	200	300	400
0	**0**	**0**	**4**	**6**	**0**
50	**0**	**3**	**5**	**7**	**3**
100	**1**	**5**	**6**	**9**	**5**
150	**0**	**2**	**3**	**5**	**1**
200	**0**	**0**	**1**	**2**	**0**

Con todos los datos de la tabla estime el volumen aproximado de agua que contiene el lago. Utilice la **fórmula de Simpson** en ambas direcciones.

$$V = \int_{x=0}^{400}\int_{y=0}^{200} f(x,y)dydx$$

$$V = \frac{\Delta x}{3}\left\{\frac{\Delta y}{3}\left[f(x_0,y_0)+4f(x_0,y_1)+2f(x_0,y_2)+4f(x_0,y_3)+f(x_0,y_4)\right]\right.$$
$$+4\frac{\Delta y}{3}\left[f(x_1,y_0)+4f(x_1,y_1)+2f(x_1,y_2)+4f(x_1,y_3)+f(x_1,y_3)\right]$$
$$+2\frac{\Delta y}{3}\left[f(x_2,y_0)+4f(x_2,y_1)+2f(x_2,y_2)+4f(x_2,y_3)+f(x_2,y_4)\right]$$
$$+4\frac{\Delta y}{3}\left[f(x_3,y_0)+4f(x_3,y_1)+2f(x_3,y_2)+4f(x_3,y_3)+f(x_3,y_4)\right]$$
$$\left.+\frac{\Delta y}{3}\left[f(x_4,y_0)+4f(x_4,y_1)+2f(x_4,y_2)+4f(x_4,y_3)+f(x_4,y_4)\right]\right\}$$

$$V = \frac{100}{3}\left\{\frac{50}{3}\left[0+4(0)+2(1)+4(0)+0\right]\right.$$
$$+4\frac{50}{3}\left[0+4(3)+2(5)+4(2)+0\right]$$
$$+2\frac{50}{3}\left[4+4(5)+2(6)+4(3)+1\right]$$
$$+4\frac{50}{3}\left[6+4(7)+2(9)+4(5)+2\right]$$
$$\left.+\frac{50}{3}\left[0+4(3)+2(5)+4(1)+0\right]\right\}$$

$$V = 287777.78 \ \ m^3$$

7.6.1 Instrumentación computacional de la fórmula de Simpson en dos direcciones

Se instrumenta el método de Simpson para una función de dos variables **f(x,y)**. Primero se integra en la dirección **X** y después, con los resultados obtenidos, se aplica nuevamente la fórmula de Simpson en la dirección **Y**.

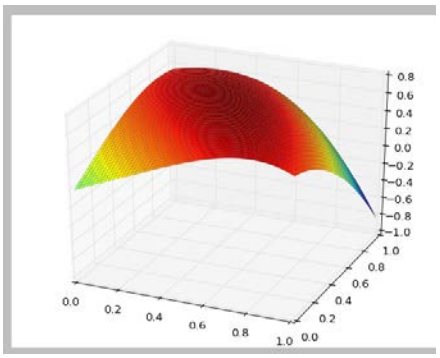
f(x,y): función de dos variables
ax, bx, ay, by: límites de integración en las direcciones **x, y** respectivamente
mx, my: cantidad de franjas en cada dirección

```
from sympy import*
from simpson import*
def simpson2(f,ax,bx,ay,by,mx,my):
    x=Symbol('x')
    dy=(by-ay)/my
    v=ay
    r=[]
    for i in range (0,my+1):
        def g(x): return f(x,v)
        u=simpson(g,ax,bx,mx)
        r=r+[u]
        v=v+dy
    s=0
    for i in range(1,my):
        s=s+2*(2-(i+1)%2)*r[i]
    s=dy/3*(r[0]+s+r[my])
    return s
```

Ejemplo. Calcule $V = \int_{y=0}^{1}\int_{x=0}^{1} \cos(x^2+y)(x+y)dxdy$ con la regla de Simpson instrumentada en la función anterior. Use **m = 4,8,12,…** en ambas variables para verificar la convergencia.

Gráfico de la superficie con las funciones de la librería **SymPy** de **Python**

```
>>> from sympy import*
>>> from sympy.plotting import*
>>> x,y=symbols('x,y')
>>> z=cos(x**2+y)*(x+y)
>>> plot3d(z,(x,0,1),(y,0,1))
```



Se observa que es una función bien definida y acotada

Solución con la función **simpson2**

```
>>> from math import*
>>> from simpson2 import*
>>> def f(x,y): return cos(x**2+y)*(x+y)
>>> r=simpson2(f,0,1,0,1,4,4);print(float(r))
0.5004153166122363
>>> r=simpson2(f,0,1,0,1,8,8);print(float(r))
0.500269266271819
>>> r=simpson2(f,0,1,0,1,12,12);print(float(r))
0.500258596816339
>>> r=simpson2(f,0,1,0,1,16,16);print(float(r))
0.5002567143287567
>>> r=simpson2(f,0,1,0,1,20,20);print(float(r))
0.5002561913044187
```

Si se incrementa el número de franjas, el resultado tiende a un valor fijo que esperamos sea el valor del integral.

Nota. Este integral no se puede resolver por métodos analíticos:

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> f=cos(x**2+y)*(x+y)
>>> r=integrate(f,(x,0,1),(y,0,1))
>>> r
Integral(-x*sin(x**2) + x*sin(x**2 + 1) + sin(x**2 + 1) - cos(x**2) +
cos(x**2 + 1), (x, 0, 1))
```

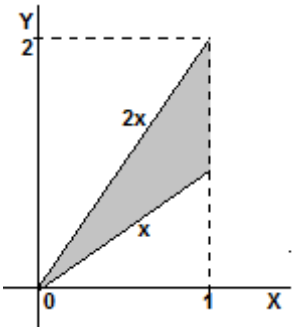
La librería Sympy de Python en algunos casos usa métodos numéricos para proporcionar una respuesta aproximada y se la puede encontrar evaluando como en este otro ejemplo:

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> f=cos(x+y)+exp(x-y)
>>> r=integrate(f,(x,0,1),(y,0,1))
>>> r
-3 + exp(-1) - cos(2) + 2*cos(1) + E
>>> float(r)
1.5829127179139093
```

7.7 Casos especiales

7.7.1 Integrales dobles con límites variables

Ejemplo. Calcule el integral de $f(x,y)=x^2+y^3$, $0\le x\le 1$, $x\le y\le 2x$. Use la fórmula de Simpson con **m = 2** en cada dirección.



$$S=\int_{x=0}^{1}\int_{y=x}^{2x} f(x,y)dydx=\int_{x=0}^{1}\int_{y=x}^{2x}(x^2+y^3)dydx\,,$$

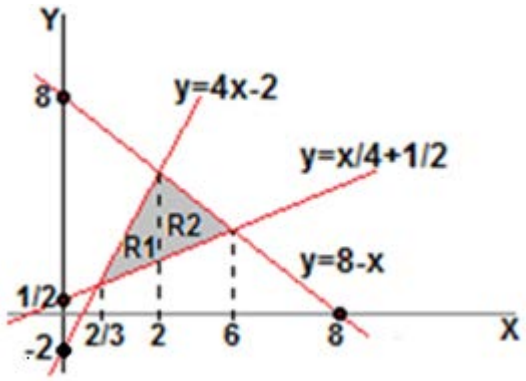
Fórmula de Simpson con una parábola en cada dirección (m=2): $\Delta x = 0.5$: $x = 0,\ 0.5,\ 1$

$$S\cong\frac{\Delta x}{3}\left[\int_{0}^{2(0)} f(0,y)dy+4\int_{0.5}^{2(0.5)} f(0.5,y)dy+\int_{1}^{2(1)} f(1,y)dy\right]$$

Los límites para integrar en **Y** así como la longitud Δy de los intervalos, cambian según **X**:

$$S=\frac{0.5}{3}\{\frac{0}{3}\left[f(0,0)+4f(0,0)+f(0,0)\right]+$$
$$4\frac{0.25}{3}\left[f(0.5,0.5)+4f(0.5,0.75)+f(0.5,1)\right]+\frac{0.5}{3}\left[f(1,1)+4f(1,1.5)+f(1,2)\right]\}=1.03125$$

Ejemplo. Calcule el valor del siguiente integral usando la Regla de Simpson con dos franjas en cada dirección $\mathbf{S = \iint_R \sqrt{xy + y^2}\, dA}$ en donde **R** es la región del plano acotada por las rectas **x+y=8, 4y-x=2, 4x-y=2**



$$\mathbf{R1} = \int_{2/3}^{2} \int_{x/4+1/2}^{4x-2} \mathbf{f(x,y)dydx} = \int_{2/3}^{2} \int_{x/4+1/2}^{4x-2} \sqrt{xy + y^2}\, dydx$$

$$\mathbf{R2} = \int_{2}^{6} \int_{x/4+1/2}^{8-x} \mathbf{f(x,y)dydx} = \int_{2}^{6} \int_{x/4+1/2}^{8-x} \sqrt{xy + y^2}\, dydx$$

S = R1 + R2

$$\mathbf{R1} \cong \frac{\Delta x}{3}\left[\int_{2/3}^{2/3} \mathbf{f(2/3,y)dy} + 4\int_{7/6}^{26/3} \mathbf{f(8/3,y)dy} + \int_{1}^{6} \mathbf{f(2,y)dy}\right], \quad \text{con} \quad \Delta x = \frac{2 - 2/3}{2} = 2/3$$

etc

7.7.2 Integrales dobles con puntos singulares

Ejemplo. Calcule el valor del siguiente integral con la fórmula de Simpson con **m = 4** en cada dirección.

$$S = \int_{y=0}^{0.5} \int_{x=0}^{0.5} \frac{e^{y-x}}{x^{1/3}}\, dx\, dy$$

Es un integral impropio no integrable analíticamente. Para aplicar la fórmula de Simpson se debe transformar

Sea $u = x^{1/3} \Rightarrow x = u^3,\ dx = 3u^2 du,\ x = 0 \Rightarrow u = 0,\ x = 0.5 \Rightarrow u = 0.5^{1/3}$

$$S = \int_{0}^{0.5} \int_{0}^{0.5^{1/3}} 3u\, e^{y-u^3}\, du\, dy$$

Esta función aunque es acotada, no es integrable analíticamente. Ahora se puede integrar numéricamente

```
>>> from math import*
>>> from simpson2 import*
>>> def f(x,y): return 3*x*exp(y-x**3)
>>> r=simpson2(f,0,0.5**(1/3),0,0.5,4,4);print(float(r))
0.5075301074265403
>>> r=simpson2(f,0,0.5**(1/3),0,0.5,8,8);print(float(r))
0.5074524699204189
>>> r=simpson2(f,0,0.5**(1/3),0,0.5,16,16);print(float(r))
0.5074477527225879
```

Comparar con el resultado de Python con la librería Sympy aproximado mediante funciones especiales:

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> f=3*x*exp(y-x**3)
>>> r=integrate(f,(x,0,0.5**(1/3)),(y,0,0.5))
>>> r
0.216240423566709*(-2*gamma(2/3)*lowergamma(2/3,0)+
2*gamma(2/3)*lowergamma(2/3, 0.5))/gamma(5/3)
>>> float(r)
0.5074474416154765
```

Python no puede resolver directamente este integral debido al punto singular:

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> f=exp(y-x)/x**(1/3)
>>> r=integrate(f,(x, 0, 0.5),(y, 0, 0.5))
Traceback (most recent call last):
  File "<pyshell#236>", line 1, in <module>
    r=integrate(f,(x,0,0.5),(y,0,0.5))
  File  "C:\Python34\lib\site-packages\sympy\utilities\decorator.py",  line
35, in threaded_func
  . . .
```

7.7.3 Integrales dobles con puntos singulares y límites variables

Ejemplo. Calcule el valor del integral $\int_{y=1}^{2}\int_{x=1}^{y} f(x,y)\,dx\,dy$ con $f(x,y)=\dfrac{\operatorname{sen}(x+y)}{\sqrt{2-x}}$ aproximada con una parábola en cada dirección (fórmula de Simpson con **m = 2**).

Es un integral impropio no integrable analíticamente. Para aplicar la fórmula de Simpson se debe transformar

Sea $u=\sqrt{2-x}$: $x = 1 \Rightarrow u = 1$, $x = y \Rightarrow u=\sqrt{2-y}$, $x = 2 - u^2$, $dx = -2udu$

$$S=\int_{1}^{2}\int_{1}^{\sqrt{2-y}} \frac{\operatorname{sen}(2-u^2+y)}{u}(-2udu)\,dy=\int_{1}^{2}\int_{\sqrt{2-y}}^{1} 2\operatorname{sen}(2-u^2+y)du\,dy$$

Ahora se puede aplicar el método numérico de integración, considerando límites variables:

$$S\cong\frac{\Delta y}{3}\left[\int_{\sqrt{2-1}}^{1} 2\operatorname{sen}(2-u^2+1)du + 4\int_{\sqrt{2-1.5}}^{1} 2\operatorname{sen}(2-u^2+1.5)du + \int_{\sqrt{2-2}}^{1} 2\operatorname{sen}(2-u^2+2)du\right]$$

$$S\cong\frac{0.5}{3}\left[\int_{1}^{1} 2\operatorname{sen}(2-u^2+1)du + 4\int_{\sqrt{0.5}}^{1} 2\operatorname{sen}(2-u^2+1.5)du + \int_{0}^{1} 2\operatorname{sen}(2-u^2+2)du\right]$$

$$\int_{1}^{1} 2\operatorname{sen}(2-u^2+1)du = 0$$

$$\int_{0.5}^{1} 2\operatorname{sen}(2-u^2+1.5)du=$$

$$\frac{0.1464}{3}\left[2\operatorname{sen}(2-0.7071^2+1.5)+4(2\operatorname{sen}(2-0.8535^2+1.5))+2\operatorname{sen}(2-1^2+1.5)\right]= 0.2133$$

$$\int_{0}^{1} 2\operatorname{sen}(2-u^2+2)du=\frac{0.5}{3}\left[2\operatorname{sen}(2-0^2+2)+4(2\operatorname{sen}(2-0.5^2+2))+2\operatorname{sen}(2-1^2+2)\right]=-0.9673$$

$$S\cong\frac{0.5}{3}\left[0+4(0.2133)+(-0.9673)\right]=-0.0190$$

7.8 Cálculo aproximado del área de figuras cerradas en el plano

El trazador cúbico paramétrico cerrado es un dispositivo para modelar figuras cerradas en el plano y puede ser usado para calcular en forma aproximada el valor del área de la región encerrada.

El teorema de Green puede utilizarse para calcular un integral de línea mediante un integral doble o para realizar el cálculo del área de una región mediante el integral de línea. Esta última aplicación es de interés.

Sea **C** una curva parametrizada en el plano, cerrada y simple. Sea **S** la región del plano determinada por **C** y su interior. Si $\mathbf{M, N : R^2 \rightarrow R}$ son funciones reales continuas y que tienen derivadas parciales continuas en toda la región **S**, entonces el siguiente enunciado es el teorema de Green:

$$\oint_C M(x,y)dx + N(x,y)dy = \iint_S \left(\frac{\partial N(x,y)}{\partial x} - \frac{\partial M(x,y)}{\partial y}\right)dA$$

Para aplicar este teorema en el cálculo del área de una región cerrada se toman las funciones simples: $M(x,y) = -y,\ N(x,y) = x$ en el enunciado del teorema. Entonces se obtienen:

$$\frac{\partial M(x,y)}{\partial y} = -1, \qquad \frac{\partial N(x,y)}{\partial x} = 1$$

Reemplazando en el teorema de Green se obtiene una fórmula para calcular el área de una región en el plano mediante el integral de línea de la curva parametrizada que la encierra:

$$A = \frac{1}{2}\oint_C x dy - y dx$$

Si **C** está definida con el trazador cúbico paramétrico cerrado:

$$C = \begin{cases} T_x(t) \\ T_y(t) \end{cases}$$

Entonces se puede calcular en forma aproximada el área de la región cerrada delimitada por el trazador cúbico paramétrico cerrado, con la siguiente fórmula:

$$A = \frac{1}{2}\oint_C T_x(t)\frac{dT_y(t)}{dt} - T_y(t)\frac{dT_x(t)}{dt}$$

$\mathbf{T_x(t)}$ y $\mathbf{T_y(t)}$ son los polinomios segmentados del trazador cúbico paramétrico cerrado:

$$T_x(t)=\begin{cases} T_{x,0}(t), & t_0 \le t \le t_1 \\ T_{x,1}(t), & t_1 \le t \le t_2 \\ \quad\ldots & \\ T_{x,n-2}(t), & t_{n-2} \le t \le t_{n-1} \end{cases}$$

$$T_y(t)=\begin{cases} T_{y,0}(t), & t_0 \le t \le t_1 \\ T_{y,1}(t), & t_1 \le t \le t_2 \\ \quad\ldots & \\ T_{y,n-2}(t), & t_{n-2} \le t \le t_{n-1} \end{cases}$$

Cálculo computacional con el teorema de Green del área de una región cerrada

La función **green** recibe los polinomios segmentados $\mathbf{T_x(t)}$ y $\mathbf{T_y(t)}$ del trazador cúbico paramétrico cerrado y los puntos del parámetro **t** que definen los extremos del intervalo de cada polinomio segmentado y entrega el área de la región cerrada calculada con la fórmula de Green.

```
from sympy import*
def green(Tx,Ty,s):
# Cálculo del área de una región cerrada
# con el teorema de Green
    t=Symbol('t')
    n=len(Tx)
    r=0
    for i in range(n):
        x=Tx[i]
        dx=diff(x,t)
        y=Ty[i]
        dy=diff(y,t)
        r=r+integrate(x*dy-y*dx,(t,s[i],s[i+1]))
    return 0.5*r
```

En el siguiente ejemplo se compara el resultado del método propuesto, con el valor exacto de una figura conocida.

Ejemplo. Calcular aproximadamente con la función **green** el área de un círculo unitario centrado en el origen tomando **16** puntos equidistantes de la circunferencia en sentido antihorario. Para cerrar la figura se agrega al final el primer punto.

```
>>> from math import*
>>> from trazador_cerrado import*
>>> a1=cos(3*pi/8); a2=cos(pi/4); a3=cos(pi/8)
>>> b1=sin(pi/8); b2=sin(pi/4); b3=sin(3*pi/8)
>>> x=[0,a1,a2,a3,1,a3,a2,a1,0,-a1,-a2,-a3,-1,-a3,-a2,-a1,0]
```

```
>>> y=[-1,-b3,-b2,-b1,0,b1,b2,b3,1,b3,b2,b1,0,-b1,-b2,-b3,-1]
>>> t=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
>>> Tx=trazador_cerrado(t,x)
>>> Ty=trazador_cerrado(t,y)
>>> from green import*
>>> A=green(Tx,Ty,t)
>>> A
3.14137
```

Este resultado difiere del valor exacto **π**, en aproximadamente 0.0002

Si los puntos provienen de una curva cerrada arbitraria, entonces el resultado de la aplicación de la fórmula de Green dependerá de la buena aproximación a la curva que proporciona el trazador cúbico paramétrico cerrado.

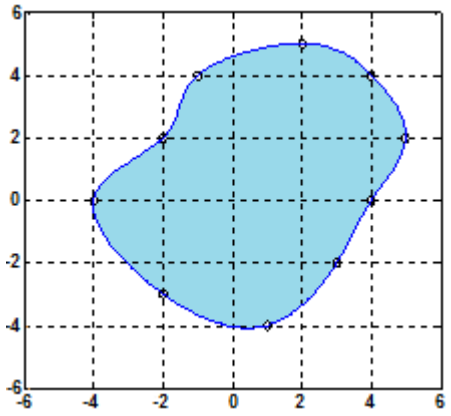
En la práctica, el área de una curva cerrada puede hacerse mediante un dispositivo denominado **planímetro**, pero este método no se puede aplicar si solamente se conocen puntos de la curva. En este caso primero debe construirse la curva para lo cual el trazador cúbico paramétrico cerrado es una opción adecuada.

Ejemplo. Colocar el trazador cúbico paramétrico cerrado sobre los siguientes diez puntos y calcular con el teorema de Green el valor del área de la región encerrada:

$P_0(4,0)$, $P_1(5,2)$, $P_2(4,4)$, $P_3(2,5)$, $P_4(-1,4)$, $P_5(-2,2)$, $P_6(-4,1)$, $P_7(-2,-3)$, $P_8(1,-4)$, $P_9(3,-2)$

El punto inicial se debe repetir al final para cerrar la figura

```
>>> from trazador_cerrado import*
>>> x=[4,5,4,2,-1,-2,-4,-2,1,3,4]
>>> y=[0,2,4,5,4,2,0,-3,-4,-2,0]
>>> t=[1,2,3,4,5,6,7,8,9,10,11]
>>> import pylab as pl
>>> s=pl.arange(1,11,0.01)
>>> Txs=trazador_cerrado(t,x,s)
>>> Tys=trazador_cerrado(t,y,s)
>>> pl.plot(x,y,'o')
>>> pl.plot(Txs,Tys,'-')
>>> pl.show()
>>> from green import*
>>> Tx=trazador_cerrado(t,x)
>>> Ty=trazador_cerrado(t,y)
>>> A=green(Tx,Ty,t)
>>> A
54.0402              Valor del área de la región encerrada
```



7.9 Cálculo aproximado de la longitud del perfil de una figura cerrada en el plano con el trazador cúbico paramétrico cerrado

El trazador cúbico paramétrico cerrado es un dispositivo para modelar figuras cerradas en el plano. La longitud del perfil de la figura se puede calcular en forma aproximada con la conocida fórmula de la longitud del arco.

Una curva **C** puede darse en forma paramétrica con los polinomios del Trazador cúbico paramétrico cerrado:

$$\mathbf{C} = \begin{cases} T_x(t) \\ T_y(t) \end{cases}, \quad t \in [a, b]$$

Si no hay intersecciones entre los polinomios del Trazador, entonces la longitud del arco **C** se puede calcular con la integral:

$$L = \int_a^b \sqrt{(T_x'(t))^2 + (T_y'(t))^2}\, dt$$

La instrumentación computacional para calcular la longitud del arco fue desarrollada en una sección anterior para el Trazador Cúbico en general.

Ejemplo. Para comparar resultados, calcular la longitud de una circunferencia con radio unitario.

Para modelar la circunferencia se toman 16 puntos de la circunferencia centrada en el origen. Los valores del parámetro deben coincidir con el rango para la integración.

Se usa la función **linspace** de **NumPy** para generar 17 puntos equidistantes en el intervalo de integración del arco: **[0, 2π]**

```
>>> from trazador_cerrado import*
>>> from math import*
>>> import numpy as np
>>> a1=cos(3*pi/8); a2=cos(pi/4); a3=cos(pi/8)
>>> b1=sin(pi/8); b2=sin(pi/4); b3=sin(3*pi/8)
>>> x=[0,a1,a2,a3,1,a3,a2,a1,0,-a1,-a2,-a3,-1,-a3,-a2,-a1,0]
>>> y=[-1,-b3,-b2,-b1,0,b1,b2,b3,1,b3,b2,b1,0,-b1,-b2,-b3,-1]
>>> t=np.linspace(0,2*pi,17)
>>> Tx=trazador_cerrado(t,x)
>>> Ty=trazador_cerrado(t,y)
>>> from arco import*
>>> s=arco(Tx,Ty,t)
>>> print(s)
6.282970
```

Este resultado difiere del valor exacto **2π**, en aproximadamente 0.0002

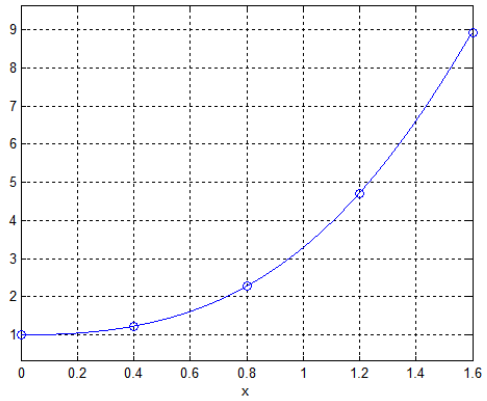
7.10 Ejercicios y problemas de integración numérica

1. Se desea calcular numéricamente $\mathbf{A = \int_1^2 ln(x)dx}$ con la Regla de los Trapecios. Use la fórmula del error de truncamiento respectiva y sin realizar la integración, estime la cantidad de trapecios que tendría que usar para que la respuesta tenga cinco decimales exactos.

2. Se desea integrar $\mathbf{f(x) = exp(x) + 5x^3}$, $\mathbf{x \in [0, 2]}$

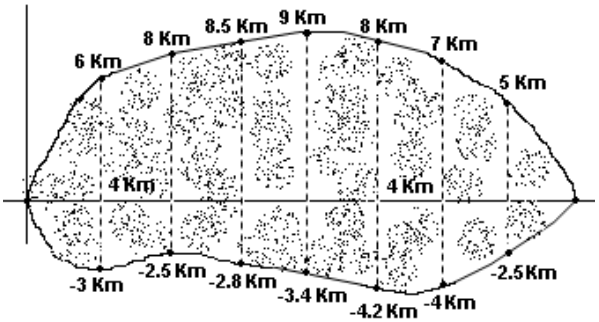
a) Use la fórmula del error para determinar la cantidad de trapecios que se deberían usar para obtener la respuesta con 2 decimales exactos.
b) Calcule la respuesta usando la cantidad de trapecios especificada

3. a) Encuentre en forma aproximada el área debajo de $\mathbf{f(x), 0 \leq x \leq 1.6}$ Use la fórmula de Simpson incluyendo los **cinco puntos** tomados mediante aproximación visual del gráfico.



b) Estime el error en el resultado obtenido con una aproximación de diferencias finitas.

4. En el siguiente gráfico se muestra delineada la zona de un derrame de petróleo ocurrido en cierta región. Las mediciones han sido obtenidas a distancias de 4 Km.



Con la fórmula de Simpson, encuentre en forma aproximada el área total cubierta por el derrame de petróleo.

5. La siguiente definición se denomina función error: $\mathbf{erf(x)} = \frac{2}{\sqrt{\pi}}\int_0^x \mathbf{e}^{-t^2}\mathbf{dt}$

Esta función se usa para definir la función de densidad de probabilidad de la distribución normal estándar. Calcule **erf(1)** con la fórmula de Simpson con **m=2,4,6**. Con estos resultados estime la precisión en la respuesta.

6. Una empresa está vendiendo una licencia para manejo de un nuevo punto de venta. La experiencia indica que dentro de **t** años, la licencia generará utilidades según **f(t) = 14000+490t** dólares por año. Si la tasa de interés **r** permanece fija durante los próximos **n** años, entonces el valor presente de la licencia se puede calcular con

$$\mathbf{V} = \int_0^n \mathbf{f(t)e^{-rt}dt}$$

Calcule **V** si **n=5** años y **r=0.07**. Use la fórmula de Simpson con **m=2,4,6.** Con estos resultados estime el error de truncamiento

7. Un comerciante usa el siguiente modelo para describir la distribución de la proporción x del total de su mercadería que se vende semanalmente:

$\mathbf{f(x)=20x^3(1-x),\ 0\le x\le 1}$

El área debajo de f(x) representa entonces la probabilidad que venda una cantidad x en cualquier semana.
Se desea conocer la probabilidad que en una semana venda más de la mitad de su mercadería (**debe integrar f entre 0.5 y 1**). Use la Fórmula de Simpson con m=4

8. Según la teoría de Kepler, el recorrido de los planetas es una elipse con el Sol en uno de sus focos. La longitud del recorrido total de la órbita esta dada por

$$\mathbf{s} = 4\int_0^{\pi/2} \mathbf{a}\sqrt{\mathbf{1-k^2sen^2t}}\ \mathbf{dt},$$

Siendo **k:** excentricidad de la órbita y **a:** longitud del semieje mayor
Calcule el recorrido del planeta Mercurio sabiendo que **k=0.206**, **a=0.387** UA
(1 UA = 150 millones de km). Use la fórmula de Simpson con **m=4** (cantidad de franjas)

9. Una placa rectangular metálica de 0.45 m por 0.60 m pesa 5 Kg. Se necesita recortar este material para obtener una placa de forma elíptica, con eje mayor igual a 50 cm, y eje menor igual a 40 cm. Calcule el área de la elipse y determine el peso que tendrá esta placa. Para calcular el área de la elipse use la fórmula de Simpson con m = 4. Finalmente, estime cual es el el error de truncamiento en el valor del área calculada.

10. Use la fórmula de Simpson, **m=4**, para calcular la longitud del arco de la curva dada con las siguientes ecuaciones paramétricas

$$\mathbf{x = f(t) = 2\cos(t),\ \ y = g(t) = \sqrt{3}sen(t),\ \ t \in [0, \pi/2]}$$

11. Calcule el integral $\mathbf{A} = \int_2^\infty \frac{1}{3x^2+2}\mathbf{dx}$. Use la Cuadratura de Gauss con dos puntos.

12. Calcule $A = \int_0^{\infty} \frac{dx}{1+x^4}$. Use la regla de Simpson, **m=2,4,6**. Con estos resultados estime el error.

13. Calcule $A = \int_1^{\infty} x^{-2} e^{-x^2} dx$. Aplique dos veces la cuadratura de Gauss con m=1, m=2 y estime el error de truncamiento.

14. Calcule $A = \int_2^{\infty} \frac{dx}{x^2 + x - 2}$. Use la regla de Simpson con **m=4**. Estime la precisión con la expresión para el error aproximando la derivada con una diferencia finita

15. De una función contínua y diferenciable se conocen los siguientes puntos:

(x, f(x)): (3, 5.1), (4, 6.4), (6, 6.8), (7, 6.9), (8, 6.7)

En donde **x** representa tiempo y **f(x)** es ganancia (miles de dólares)

Coloque el trazador cúbico sobre los puntos y calcule el área en el primer intervalo

a) Analíticamente con la función **integrate( )** de Python
b) Con la función **simpson** desarrollada en este curso, **m=4**
c) Con la función **cgaussm** desarrollada en este curso, **m=2**
d) Explique la precisión de los resultados obtenidos

16) Una curva **C** puede darse en forma paramétrica con ecuaciones: **x=f(t), y=g(t), t∈[a, b]** Si no hay intersecciones entre **f** y **g**, entonces, la longitud **L** del arco **C** se puede calcular con

$$L = \int_a^b \sqrt{(f'(t))^2 + (g'(t))^2}\, dt$$

Si las coordenadas **x(t), y(t)** del recorrido de un cohete registradas en los instantes **t=0, 1, 2, 3** fueron respectivamente: **x(t)=2, 1, 3, 4, y(t)=0, 4, 6, 0.** Use polinomios de interpolación de tercer grado construido con los cuatro puntos para aproximar **f** y **g** . Con estos polinomios estime con un método numérico la longitud de la trayectoria del cohete.

17. El siguiente integral $\int_1^{\infty} x^2 e^{-x^2} dx$ es acotado pero no se puede resolver analíticamente por los métodos conocidos. Si se desea resolverlo numéricamente con la fórmula de Simpson se elimina el límite superior por una sustitucion **u=1/x** pero se obtiene una función que no se puede evaluar, sin embargo, se puede usar la cuadratura de Gauss pues no requiere evaluar los extremos del intervalo.

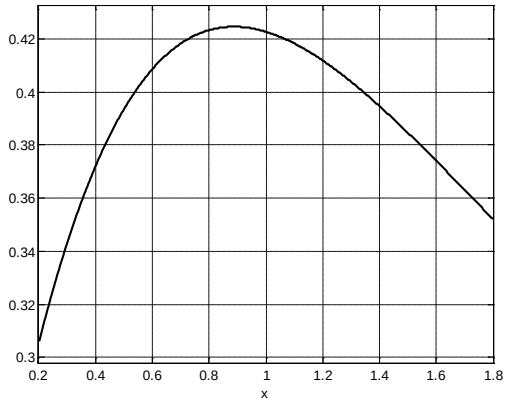
a) Use este método manualmente con **m=1,2,3,4**. Con estos resultados estime la precisión de la respuesta.

b) Use el método computacional **cgaussm** desarrollado en este curso con m = 4, 8, 12, 16,.. hasta obtener la respuesta con 4 decimales exactos.

18. La función $\Gamma(\alpha) = \int_0^{\infty} x^{\alpha-1} e^{-x} dx$ se denomina Función Gamma y es usada en algunos modelos de probabilidad. Encuentre un valor aproximado de $\Gamma(2)$ aplicando la Cuadratura de Gauss.

Sugerencia: Separe el intervalo de integración en dos intervalos: **[0,1], [1, ∞]**. Luego, para el segundo intervalo use un cambio de variable para eliminar el límite superior ∞. Finalmente aplique la Cuadratura de Gauss una vez en cada intervalo.

19. Escriba una tabla con los **cinco puntos** del gráfico, para **x = 0.2, 0.6, 1.0, 1.4, 1.8** aproximando visualmente con una precisión de tres decimales.



a) Encuentre en forma aproximada el área debajo de **f(x), 0.2≤x≤1.8** con la fórmula de Simpson incluyendo los **cinco puntos** tabulados. Estime el error de truncamiento.

b) Encuentre aproximadamente el área debajo de **f(x), 0.2≤x≤1.8** aplicando la cuadratura de Gauss con un intervalo y después con dos intervalos. Con estos dos resultados estime el error de truncamiento. Tome los valores de **f** del gráfico mediante una aproximación visual con tres decimales

20. Calcule $A = \int_1^{1.8} \int_{0.5}^{2.5} (x+y)\sin(x)dxdy$. Use la regla de Simpson en ambas direcciones, con **m=4**. Estime el error de truncamiento.

21. Calcular la siguiente integral, con el algoritmo de la integral doble de Simpson:

$$S = \iint_R x^2 \sqrt{9 - y^2}\, dA$$

Donde **R** es la región acotada por: $x^2 + y^2 = 9$. Usar **m=4** en ambas direcciones

22. El siguiente cuadro contiene valores de una función **f(x,y).** Use la fórmula de Simpson para calcular el volumen entre el plano **X-Y** y la superficie **f(x,y),** $\mathbf{0.1 \le x \le 0.5,\ \ 0.2 \le y \le 1.0}$. Use todos los datos disponibles.

Y / X	0.2	0.4	0.6	0.8	1.0
0.1	0.04	0.08	0.12	0.16	0.20
0.2	0.41	0.47	0.53	0.58	0.62
0.3	0.81	0.83	0.84	0.83	0.82
0.4	0.76	0.70	0.62	0.53	0.42
0.5	0.06	0.02	0.01	0.01	0.02

23. Cuando un cuerpo de masa m se desplaza verticalmente hacia arriba desde la superficie de la tierra, si prescindimos de toda fuerza de resistencia (excepto la fuerza de la gravedad), la velocidad de escape **v** está dada por: $\mathbf{v^2 = 2gR\int_1^{\infty} z^{-2}dz;\quad z = \frac{x}{R}}$

Donde **R = 6371** km. es el radio promedio de la tierra, **g = 9.81** m/s^2 es la constante gravitacional. Utilice cuadratura Gaussiana para hallar **v**.

24. El siguiente integral no puede resolverse analíticamente. Aplique la fórmula de Simpson con **2, 4, 6, 8** subintervalos. Con estos resultados estime la precisión del resultado de la integración.

$$\mathbf{A = \int_0^2 e^{sen(x)}dx}$$

25. El siguiente integral no puede resolverse analíticamente y tampoco pueden aplicarse las fórmulas comunes de integración numérica.:

$$\mathbf{\int_0^1 \frac{sen(x)}{\sqrt{x}}dx}$$

Aplique la Cuadratura de Gauss con uno, dos y tres subintervalos. Con estos resultados estime la precisión del resultado de la integración.

26. Encuentre un valor aproximado del siguiente integral $\mathbf{\int_0^1 \frac{sin(x)}{\sqrt{1-x}}dx}$

a) Use una sustitución para eliminar el punto singular y aplique la regla de Simpson, **m=4.**
b) Estime el error en el resultado obtenido

27. Evalúe de forma aproximada el siguiente integral impropio usando la fórmula de Simpson con 5 puntos: $\int_0^1 \frac{\cos(x)-1}{\sqrt{x}}dx$. Estime el error en la aproximación.

28. Se requiere calcular el valor del integral $\int_1^2 \int_1^y \frac{\text{sen}(x-y)}{\sqrt{2-x}} dxdy$ usando dos parábolas en cada dirección para aproximar a **f**

a) Escriba la formulación matemática del método numérico que utilizará
b) Realice los cálculos con cuatro decimales.

Transforme previamente el integral con alguna sustitución que permita eliminar el punto singular que se produce al evaluar **f(x,y)** en el límite superior.

29. En el techado de las casas se utilizan planchas corrugadas con perfil ondulado. Cada onda tiene la forma **f(x) = sen(x),** con un periodo de **2π** pulgadas

El perfil de la plancha tiene **8** ondas y la longitud L de cada onda se la puede calcular con la siguiente integral: $L = \int_0^{2\pi} \sqrt{1 + (f'(x))^2}\, dx$
Este integral no puede ser calculado por métodos analíticos.

a) Use la fórmula de Simpson con **m = 4, 6, 8, 10** para calcular L y estime el error en el último resultado
b) Con el último resultado encuentre la longitud del perfil de la plancha.

30. Dado el siguiente integral: $A = \int_2^3 \frac{e^x}{\sqrt[4]{3-x}} dx$

a) Realice alguna sustitución para eliminar la singularidad en x=3
b) Use la función **simpson** de este curso para calcular el resultado. Encuentre el menor valor de **m** tal que **E=0.0001** Calcule este resultado computacionalmente probando valores de **m**.

31. Para que una función pueda usarse como un modelo matemático para cálcular probabilidad debe cumplir que el valor del integral en el dominio de la función sea igual a **1**

Considere la función bivariada continua y diferenciable en el intervalo especificado:

$$f(x,y) = k(x+y)\sqrt{x^2+y},\quad 0 \le x \le y,\quad 0 \le y \le 1$$

Determine el valor de la constante **k** para que esta función cumpla la definición anterior. Obtenga la respuesta con la fórmula de Simpson usando **una parábola** en cada dirección.

32. Con la librería Pylab de Python graficar la función: $f(x) = e^x \cos(x^2+1) + 5$, $0 \le x \le 1.2$
a) Del gráfico aproxime cinco puntos de f(x) equidistantes
b) Calcule el área debajo de f(x) usando los cinco puntos **(x,f(x))** con la fórmula de Simpson.
c) Intente calcular la respuesta exacta con la función de Python: **integrate( )**
d) Calcule la respuesta con la función **simpson** desarrollada en este curso, **m=4**
e) Con este último resultado encuentre el error en el resultado obtenido en el literal b)

8 DIFERENCIACIÓN NUMÉRICA

En este capítulo se describen algunas fórmulas para evaluar derivadas. Estas fórmulas son de especial interés para algunos métodos usados en la resolución de ecuaciones diferenciales y que se estudiarán posteriormente.

8.1 Obtención de fórmulas de diferenciación numérica

Dados los puntos (x_i, f_i), $i=0, 1, 2, ..., n$, siendo **f** desconocida pero supuestamente diferenciable. Es de interés evaluar alguna derivada de **f** en alguno de los puntos x_i.

Un procedimiento para obtener fórmulas aproximadas para las derivadas de **f** consiste en usar los puntos dados para construir el polinomio de interpolación y luego derivar el polinomio y evaluar la derivada en el punto de interés:

$$f(x) \cong p_n(x) \Rightarrow f^{(k)}(x_i) \cong \frac{d^k}{dx^k}[p_n(x)]_{x=x_i}$$

Adicionalmente, con la fórmula del error en la interpolación se puede obtener una fórmula para estimar el error en las fórmulas de las derivadas.

Un procedimiento más simple consiste en usar la serie de Taylor, bajo la suposición de que la función **f** se puede expresar mediante este desarrollo. Mediante artificios algebraicos se pueden obtener fórmulas para aproximar algunas derivadas y su error de truncamiento:

Desarrollo de la serie de Taylor a una distancia **h** a la derecha y a la izquierda del punto x_i :

$$(1) \qquad f_{i+1} = f_i + hf'_i + \frac{h^2}{2!}f''_i + ... + \frac{h^n}{n!}f^{(n)}_i + \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(z), \quad x_i \le z \le x_{i+1}$$

$$(2) \qquad f_{i-1} = f_i - hf'_i + \frac{h^2}{2!}f''_i - ... + \frac{h^n}{n!}f^{(n)}_i - \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(z), \quad x_{i-1} \le z \le x_i$$

Por simplicidad se usa la notación $f(x_i) \equiv f_i$,

8.2 Una fórmula para la primera derivada

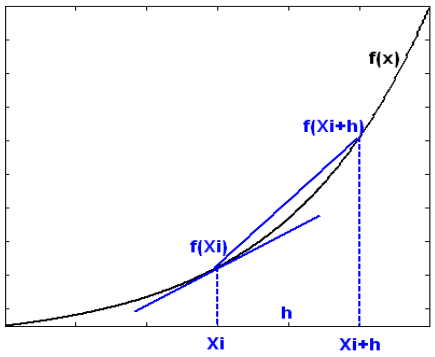
Tomando tres términos de (1) y despejando **f'** se obtiene una aproximación para la primera derivada en el punto x_i, y el error de truncamiento respectivo:

$$f_{i+1} = f_i + hf'_i + \frac{h^2}{2!}f''(z) \Rightarrow f'_i = \frac{f_{i+1} - f_i}{h} - \frac{h}{2}f''(z), \quad x_i \le z \le x_{i+1}$$

Definición: Una fórmula para la primera derivada con el error de truncamiento

$$f'_i \cong \frac{f_{i+1} - f_i}{h} = \frac{\Delta f_i}{h}$$

$$E = -\frac{h}{2}f''(z) = O(h), \quad x_i \le z \le x_{i+1}$$



Esta aproximación significa que la pendiente de la tangente a **f** en el punto x_i es aproximada mediante la pendiente de la recta que incluye a los puntos x_i y x_{i+1}, como se muestra en la figura.

Para que la aproximación sea aceptable, **h** debe ser suficientemente pequeño:

$$h \to 0 \Rightarrow E = -\frac{h}{2} f''(z) \to 0$$

Sin embargo, si **h** es demasiado pequeño, puede introducirse el error de redondeo como se describe a continuación. Este error se debe a la imprecisión en los cálculos aritméticos o a la limitada capacidad de representación de los dispositivos de memoria de almacenamiento de los números reales.

Suponer que se debe evaluar **f** en un punto x_i. Por lo mencionado anteriormente no se obtendrá exactamente f_i sino una aproximación $\bar{f}_i$. Sea R_i el error de redondeo: $f_i = \bar{f}_i + R_i$

Si se sustituye en la fórmula para la derivada se tiene:

$$f'_i = \frac{f_{i+1} - f_i}{h} - \frac{h}{2} f''(z) = \frac{\bar{f}_{i+1} + R_{i+1} - (\bar{f}_i + R_i)}{h} - \frac{h}{2} f''(z) = \frac{\bar{f}_{i+1} - \bar{f}_i}{h} + \frac{R_{i+1} - R_i}{h} - \frac{h}{2} f''(z)$$

Debido a que el error de redondeo solamente depende del dispositivo de almacenamiento y no de **h** entonces, mientras el error de truncamiento $E = -\frac{h}{2} f''(z)$ se reduce si $h \to 0$, puede ocurrir que el componente del error de redondeo: R_{i+1} y R_i crezca significativamente anulando la precisión que se obtiene al reducir el error de truncamiento **E**.

El siguiente programa escrito en Python incluye la fórmula para estimar la primera derivada de $f(x) = x\,e^x$ para $x = 1$ con valores de $h = 0.1, 0.01, 0.001, 0.0001, \ldots$, y su comparación con el valor exacto $f'(1) = 5.436563656918091\ldots$

```
#Comportamiento del error en diferenciación numérica
from math import*
def f(x):return x*exp(x)
r=5.436563656918091
h=0.1
for i in range(15):
    d=(f(1+h)-f(1))/h
    e=abs(r-d)
    print('%18.15f %18.15f %18.15f %18.15f'%(h,r,d,e))
    h=h/10
```

Resultados

h	Valor exacto	Valor aproximado	Error
0.100000000000000	5.436563656918091	5.863007978820320	0.426444321902229
0.010000000000000	5.436563656918091	5.477519670804032	0.040956013885941
0.001000000000000	5.436563656918091	5.440642892414527	0.004079235496436
0.000100000000000	5.436563656918091	5.436971417318581	0.000407760400490
0.000010000000000	5.436563656918091	5.436604431396929	0.000040774478838
0.000001000000000	5.436563656918091	5.436567733774210	0.000004076856119
0.000000100000000	5.436563656918091	5.436564070038229	0.000000413120138
0.000000010000000	**5.436563656918091**	**5.436563643712588**	**0.000000013205503**
0.000000001000000	5.436563656918091	5.436564087801797	0.000000430883706
0.000000000100000	5.436563656918091	5.436566752337056	0.000003095418965
0.000000000010000	5.436563656918091	5.436584515905450	0.000020858987359
0.000000000001000	5.436563656918091	5.437428285404166	0.000864628486075
0.000000000000100	5.436563656918091	5.435651928564766	0.000911728353326
0.000000000000010	5.436563656918091	5.417888360170763	0.018675296747328
0.000000000000001	5.436563656918091	6.217248937900876	0.780685280982785

Se observa que la precisión mejora cuando se reduce **h**, pero a partir de cierto valor de **h** el resultado pierde precisión.

La aproximación propuesta para $\mathbf{f'_i}$ es una fórmula de primer orden cuyo error de truncamiento es **E=O(h)**. Por lo tanto, si se desea una aproximación con alta precisión, se debe elegir para **h** un valor muy pequeño, pero ya hemos mencionado que esto puede hacer que el error de redondeo sea significativo.

Adicionalmente, si únicamente se tienen puntos de **f**, no se puede elegir **h**. Entonces es preferible usar fórmulas con mayor precisión usando los puntos dados.

8.3 Una fórmula de segundo orden para la primera derivada

Una fórmula más precisa para la primera derivada se obtiene restando (1) y (2) con cuatro términos:

$$(1) \qquad f_{i+1} = f_i + hf'_i + \frac{h^2}{2!}f''_i + \frac{h^3}{3!}f'''(z_1), \quad x_i \le z_1 \le x_{i+1}$$

$$(2) \qquad f_{i-1} = f_i - hf'_i + \frac{h^2}{2!}f''_i - \frac{h^3}{3!}f'''(z_2), \quad x_{i-1} \le z_2 \le x_i$$

$$(1) - (2): \qquad f_{i+1} - f_{i-1} = 2hf'_i + \frac{h^3}{3!}f'''(z_1) + \frac{h^3}{3!}f'''(z_2)$$

$$\Rightarrow f'_i = \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h^2}{12}(f'''(z_1) + f'''(z_2)) = \frac{f_{i+1} - f_{i-1}}{2h} - \frac{h^2}{12}(2f'''(z)), \quad x_{i-1} \le z \le x_{i+1}$$

Definición: Una fórmula de segundo orden para la primera derivada

$$f'_i \cong \frac{f_{i+1} - f_{i-1}}{2h}, \qquad E = -\frac{h^2}{6}f'''(z) = O(h^2), \quad x_{i-1} \le z \le x_{i+1}$$

Ejemplo. Usar las fórmulas para evaluar **f'(1.1)** dados los siguientes datos:
(x, f(x)): (1.0, 2.7183), (1.1, 3.3046), (1.2, 3.9841), (1.3, 4.7701)

$$f'_1 \cong \frac{f_2 - f_1}{h} = \frac{3.9841 - 3.3046}{0.1} = 6.7950, \qquad E=O(h)=O(0.1)$$

$$f'_1 \cong \frac{f_2 - f_0}{2h} = \frac{3.9841 - 2.7183}{2(0.1)} = 6.3293, \qquad E=O(h^2)=O(0.01)$$

Para comparar, estos datos son tomados de la función $\mathbf{f(x) = x\,e^x}$. El valor exacto es $\mathbf{f'(1.1) = 6.3087}$. El error en la primera fórmula es **-0.4863** y para la segunda es **-0.0206**

En la realidad, únicamente se conocen puntos de **f(x)** con los cuales se debe intentar estimar el error en los resultados de las fórmulas mediante las aproximaciones de diferencias finitas, recordando que esta aproximación es aceptable si los valores de las diferencias finitas en cada columna no cambian significactivamente y tienden a reducirse.

Ejemplo. Estime el error en los resultados del ejemplo anterior, con los dados dados:
(x, f(x)): (1.0, 2.7183), (1.1, 3.3046), (1.2, 3.9841), (1.3, 4.7701)

Tabulación de las diferencias finitas:

i	x_i	f_i	Δf_i	$\Delta^2 f_i$	$\Delta^3 f_i$
0	1.0	2.7183	0.5863	0.0932	0.0133
1	1.1	3.3046	0.6795	0.1065	
2	1.2	3.9841	0.7860		
3	1.3	4.7701			

$$f'_1 \cong \frac{f_2 - f_1}{h} = \frac{3.9841 - 3.3046}{0.1} = 6.7950,$$

$$E = -\frac{h}{2} f''(z) \cong -\frac{h}{2}\frac{\Delta^2 f_i}{h^2} = -\frac{\Delta^2 f_i}{2h} = -\frac{0.1066}{2(0.1)} = -0.5325$$

$$f'_1 \cong \frac{f_2 - f_0}{2h} = \frac{3.9841 - 2.7183}{2(0.1)} = 6.3293,$$

$$E = -\frac{h^2}{6} f'''(z) \cong -\frac{h^2}{6}\frac{\Delta^3 f_i}{h^3} = -\frac{\Delta^3 f_i}{6h} = -\frac{0.0133}{6(0.1)} = -0.0222$$

En el primer resultado, el error está en el primer decimal. En el segundo resultado, el error está en el segundo decimal. Esto coincide bien con los valores calculados.

8.4 Una fórmula para la segunda derivada

Una fórmula para la segunda derivada se obtiene sumando (1) y (2) con 5 términos:

$$(1)\qquad f_{i+1} = f_i + hf'_i + \frac{h^2}{2!}f''_i + \frac{h^3}{3!}f'''_i + \frac{h^4}{4!}f^{iv}(z_1), \quad x_i \le z_1 \le x_{i+1}$$

$$(2)\qquad f_{i-1} = f_i - hf'_i + \frac{h^2}{2!}f''_i - \frac{h^3}{3!}f'''_i + \frac{h^4}{4!}f^{iv}(z_2), \quad x_{i-1} \le z_2 \le x_i$$

$$(1)+(2):\qquad f_{i+1} + f_{i-1} = 2f_i + 2\left(\frac{h^2}{2!}f''_i\right) + \frac{h^4}{4!}\left(f^{iv}(z_1) + f^{iv}(z_2)\right)$$

$$\Rightarrow\ f''_i = \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} - \frac{h^2}{4!}(2f^{iv}(z)), \quad x_{i-1} \le z \le x_{i+1}$$

Definición: Una fórmula para la segunda derivada con el error de truncamiento

$$f''_i \cong \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2}$$

$$E = -\frac{h^2}{12}f^{iv}(z) = O(h^2), \quad x_{i-1} \le z \le x_{i+1}$$

Ejemplo. Use la fórmula para calcular **f''(1.1)** dados los siguientes datos:
(x, f(x)): (1.0, 2.7183), (1.1, 3.3046), (1.2, 3.9841), (1.3, 4.7701)

$$f''_1 \cong \frac{f_0 - 2f_1 + f_2}{h^2} = \frac{2.7183 - 2(3.3046) + 3.9841}{0.1^2} = 9.32, E = O(h^2) = O(0.01)$$

Estos datos son tomados de la función **$f(x) = x\,e^x$**. El valor exacto es **f''(1.1) = 9.3129**. El error de truncamiento es **-0.0071**. Si se tuviese un punto adicional, se pudiera estimar el error con la fórmula, usando los datos y una aproximación de diferencias finitas para la cuarta derivada.

8.5 Obtención de fórmulas de diferenciación numérica con el método de coeficientes indeterminados

Este método permite obtener fórmulas para derivadas usando como base cualquier grupo de puntos.

Dados los puntos **(x_i, f_i), i= 0, 1, 2, ..., n**, encontrar una fórmula para la **k-ésima derivada** de **f(x)** con la siguiente forma:

$$f^{(k)}(x_j) = c_o f_i + c_1 f_{i+1} + c_2 f_{i+2} + \ldots + c_m f_{i+m} + E, \qquad j \in \{i, i+1, i+2, \ldots, i+m\}$$

La derivada debe evaluarse en el punto **j** que debe ser alguno de los puntos que intervienen en la fórmula.

Para determinar los coeficientes y el error de truncamiento se sigue el procedimiento:

1) Desarrollar cada término alrededor del punto x_j con la serie de Taylor
2) Comparar los términos en ambos lados de la ecuación
3) Resolver el sistema resultante y obtener los coeficientes y la fórmula para **E**

Ejemplo. Con el Método de Coeficientes Indeterminados obtenga una fórmula para **f'** en el punto x_i usando los puntos x_i y x_{i+1} :

$$f'_i = c_o f_i + c_1 f_{i+1} + E$$

1) Desarrollar cada término alrededor del punto x_i

$$f'_i = c_o f_i + c_1 (f_i + hf'_i + \frac{h^2}{2!} f''(z)) + E$$

$$f'_i = (c_0 + c_1) f_i + c_1 h f'_i + c_1 \frac{h^2}{2!} f''(z) + E$$

2) Comparar términos

$$0 = c_0 + c_1$$
$$1 = c_1 h$$
$$0 = c_1 \frac{h^2}{2!} f''(z) + E$$

3) Con las dos primeras ecuaciones se obtiene:

$$c_1 = 1/h, \quad c_0 = -1/h$$

Con la tercera ecuación se obtiene:

$$E = -(1/h)\frac{h^2}{2!} f''(z)) = -\frac{h}{2} f''(z)$$

Sustituyendo en la fórmula propuesta:

$$f'_i = (-1/h) f_i + (1/h) f_{i+1} + E = \frac{f_{i+1} - f_i}{h} - \frac{h}{2} f''(z), \quad x_i \le z \le x_{i+1}$$

Igual a la fórmula que se obtuvo directamente con la serie de Taylor.

8.6 Algunas otras fórmulas de interés para evaluar derivadas

Fórmulas para la primera derivada

$$f'(x_0) = \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h} + O(h^2)$$

$$f'(x_n) = \frac{3f(x_{n-2}) - 4f(x_{n-1}) + f(x_n)}{2h} + O(h^2)$$

$$f'(x_i) = \frac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12h} + O(h^4)$$

....

8.7 Extrapolación para diferenciación numérica

Esta técnica se puede aplicar a la diferenciación numérica para mejorar la exactitud del valor de una derivada usando resultados previos de menor precisión.

Examinamos el caso de la primera derivada, comenzando con una fórmula conocida:

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} + O(h^2)$$ El error de truncamiento es de segundo orden

Si se define el operador **D**:

$$D(h) = \frac{f(x_i + h) - f(x_i - h)}{2h}$$

$$f'(x_i) = D(h) + O(h^2)$$

Reduciendo **h,** la aproximación mejora pero el error sigue de segundo orden:

$$f'(x_i) = D(h/2) + O(h^2)$$

La técnica de extrapolación permite combinar estas aproximaciones para obtener un resultado más preciso, es decir con un error de truncamiento de mayor orden

Para aplicar la extrapolación se utiliza la serie de Taylor con más términos:

$$f(x_i + h) = f_i + hf_i^{(1)} + \frac{h^2}{2} f_i^{(2)} + \frac{h^3}{6} f_i^{(3)} + \frac{h^4}{6} f_i^{(4)} + O(h^5)$$

$$f(x_i - h) = f_i - hf_i^{(1)} + \frac{h^2}{2} f_i^{(2)} - \frac{h^3}{6} f_i^{(3)} + \frac{h^4}{6} f_i^{(4)} + O(h^5)$$

Restando y dividiendo para 2h

$$D(h) = f_i^{(1)} + \frac{h^2}{6} f_i^{(3)} + O(h^4)$$

Definiendo

$$A = \frac{f_i^{(3)}}{6} \Rightarrow D(h) = f_i^{(1)} + Ah^2 + O(h^4)$$

Si se puede evaluar en h/2:

$$D(h/2) = f_i^{(1)} + A\frac{h^2}{4} + O(h^4)$$

Para eliminar A se combinan estas dos expresiones:

$$D(h) - 4D(h/2) = f_i^{(1)} + Ah^2 + O(h^4) - 4f_i^{(1)} - Ah^2 + O(h^2) = -3f_i^{(1)} + O(h^2)$$

De donde se obtiene

$$f_i^{(1)} = f'(x_i) = \frac{4D(h/2) - D(h)}{3} + O(h^4)$$

Es una fórmula mayor precisión que la fórmula inicial, para evaluar la primera derivada.

Este procedimiento puede continuar para encontrar fórmulas de mayor orden.

Ejemplo. Dados los puntos **(0.1, 0.1105), (0.2, 0.2442), (0.3, 0.4049), (0.4, 0.5967), (0.5, 0.8243)** de una función **f(x)**, calcule **f'(0.3)** usando extrapolación en la diferenciación

$$h = 0.2, \quad x_i = 0.3$$

$$f'(x_i) \cong D(h) = 1.7845;$$
$$f'(x_i) \cong D(h/2) = 1.7625;$$
$$f'(x_i) \cong \frac{4D(h/2) - D(h)}{3} = 1.7551$$

Para comparación, estos datos fueron tomados de la función $\mathbf{f(x)=x\,e^x}$

Valor exacto **f'(0.3) = 1.7548....** Se verifica que usando resultados con precisión limitada, se obtuvo un resultado con mayor precisión.

8.8 Ejercicios de diferenciación numérica

1. Se tomaron los siguientes datos en Km. para las coordenadas del recorrido de un cohete: **(50, 3.5), (80, 4.2), (110, 5.7), (140, 3.8), (170, 1.2).** Mediante aproximaciones de **segundo orden** determine

a) Velocidad en el centro de la trayectoria
b) Aceleración en el centro de la trayectoria

2. La fórmula de segundo orden $f'_i \cong \frac{f_{i+1} - f_{i-1}}{2h}$ para aproximar la primera derivada no puede aplicarse en los puntos extremos del conjunto de datos pues se requiere un punto a cada lado. Use el método de coeficientes indeterminados para encontrar fórmulas de segundo orden para la primera derivada con los siguientes puntos:

a) $f'_0 = C_0f_0 + C_1f_1 + C_2f_2 + E_T$
b) $f'_n = C_nf_n + C_{n-1}f_{n-1} + C_{n-2}f_{n-2} + E_T$

3. Con el método de los coeficientes indeterminados demuestre la siguiente fórmula que relaciona la segunda derivada con la segunda diferencia finita:

$f''(z) = \frac{\Delta^2 f_i}{h^2}$, para algún $z \in (x_i, x_{i+2})$

9 MÉTODOS NUMÉRICOS PARA RESOLVER ECUACIONES DIFERENCIALES ORDINARIAS

El análisis matemático de muchos problemas en ciencias e ingeniería conduce a la obtención de ecuaciones diferenciales ordinarias (EDO). El tratamiento clásico de estas ecuaciones ha enfatizado el estudio de métodos analíticos para resolverlas pero que no son aplicables a muchas EDO, especialmente si son de tipo no-lineal.

Los métodos numéricos son una opción importante para resolver estas ecuaciones cuando la solución analítica es muy complicada o no se puede aplicar. Estos métodos instrumentados computacionalmente proporcionan soluciones aproximadas que permiten analizar el comportamiento de la solución. Adicionalmente se pueden usar para experimentar numéricamente con aspectos de convergencia y estabilidad de la solución calculada, o con datos diferentes.

Por otra parte, programas como Python disponen de procedimientos simbólicos y numéricos para obtener y graficar las soluciones de este tipo de ecuaciones. Pero es un tema formativo el conocimiento de los métodos numéricos propios y de la instrumentación computacional.

Un problema importante es determinar las condiciones para que la solución exista y sea única, y conocer el dominio en el que la solución tiene validez. Otros temas relacionados son la sensibilidad de la solución a los cambios en la ecuación o en la condición inicial, y la estabilidad de la solución calculada, es decir el estudio de la propagación de los errores en el cálculo numérico. Sin embargo, el objetivo principal es resolver la ecuación.

Una ecuación diferencial ordinaria de orden n es una ecuación del tipo:

$$\mathbf{F(x, y, y', y'', ..., y^{(n-1)}, y^{(n)}) = 0}$$

En donde **y** es una función de una variable independiente, **x**. El orden de la ecuación diferencial es el orden de su derivada más alta.

Si es que es posible expresar la ecuación diferencial en la forma:

$$\mathbf{y^{(n)} + a_1 y^{(n-1)} + \ldots + a_{n-1} y' + a_n y = b}$$

en donde los coeficientes $\mathbf{a_1, a_2, \ldots a_n, b}$ son constantes o términos que contienen la variable independiente. Esta ecuación se denomina ecuación diferencial lineal explícita de orden **n**

En una EDO es de interés encontrar la función **y(x)** que satisface a la ecuación en cierto dominio y que incluye a las condiciones que normalmente se suministran para particularizar la ecuación. Los métodos numéricos proporcionan puntos de esta función como una aproximación a la solución analítica, y además deben permitir estimar la precisión.

Ejemplo. Un cuerpo de masa **m** sujeto a un extremo de un resorte con constante de amortiguación **k**, con el otro extremo fijo, se desliza sobre una mesa con un coeficiente de fricción **c**. A partir de un estado inicial, las oscilaciones decrecen hasta que se detiene. La ecuación del movimiento, en un sistema de unidades consistente, es

$$\mathbf{ma} = \sum \mathbf{F_s} = -\mathbf{cv} - \mathbf{ks}$$

En donde **a** es la aceleración, **v** es la velocidad, **s** es desplazamiento, **t** es tiempo.

Del planteamiento físico del problema se obtiene la ecuación:

$$\frac{\mathbf{d^2s}}{\mathbf{dt^2}} + \frac{\mathbf{c}}{\mathbf{m}}\frac{\mathbf{ds}}{\mathbf{dt}} + \frac{\mathbf{k}}{\mathbf{m}}\mathbf{s} = \mathbf{0}$$

Es una ecuación diferencial ordinaria lineal de segundo orden. Su solución describe el desplazamiento **s** en función del tiempo **t**, partiendo de condiciones iniciales.

9.1 Ecuaciones diferenciales ordinarias lineales de primer orden con la condición en el inicio

Estas ecuaciones tienen la forma general

$\mathbf{F(x, y, y')} = \mathbf{0}$, con la condición inicial $\mathbf{y(x_0)} = \mathbf{y_0}$

Si es una ecuación diferencial explícita, se puede escribir de la siguiente manera:

$\mathbf{y'(x)} = \mathbf{f(x, y)}, \quad \mathbf{y(x_0)} = \mathbf{y_0}$

En la notación de derivada:

$$\frac{\mathbf{dy}}{\mathbf{dx}} = \mathbf{f(x,y)}, \quad \mathbf{y(x_0)} = \mathbf{y_0}$$

Su solución es una función **y(x)**, en algún intervalo, que satisface a la ecuación e incluye a la condición inicial. La solución de esta ecuación se puede expresar integrando:

$$\int_{x_0}^{x} \mathbf{dy} = \int_{x_0}^{x} \mathbf{f(x,y)dx} \quad \Rightarrow \quad \mathbf{y(x) = y(x_0) + \int_{x_0}^{x} y'(x)dx}$$

Los métodos numéricos permiten obtener una solución aproximada cuando no es posible o es muy complicado obtenerla en forma explícita.

Para adquirir confianza al resolver ecuaciones diferenciales para las cuales no se puede obtener la solución analítica, es conveniente comenzar con ecuaciones que si tengan solución analítica y comparar la solución numérica con esta solución conocida. También se puede comparar con los resultados de los métodos numéricos incluidos en las librerías del lenguaje computacional utilizado.

Existencia de la solución

Condición de Lipschitz

Sea la función $\mathbf{f: A \to R, A \subseteq R}$. Si existe una constante $\mathbf{k \in R^+}$ tal que
$\mathbf{|f(a) - f(b)| \leq k|a - b|}$, para $\mathbf{\forall a,b \in A,}$ entonces **f** satisface la condición de Lipschitz en **A**

La condición de Lipschitz se puede interpretar geométricamente re-escribiéndola:

$$\left|\frac{\mathbf{f(a) - f(b)}}{\mathbf{a - b}}\right| \leq \mathbf{k}, \text{ para } \mathbf{\forall a,b \in A, \ a \neq b}$$

Entonces, una función **f** satisface la condición de Lipschitz si y solo si las pendientes de todos los segmentos de recta que unen dos puntos de la gráfica de **y=f(x)** en **A**, están acotadas por algún número positivo **k**

Teorema.- Si $\mathbf{f: A \to R, A \subseteq R}$ es una función de Lipschitz, entonces el dominio y el rango de **f** son conjuntos acotados.

Sea una ecuación diferencial de primer orden con una condición inicial:

$$\mathbf{y'(x) = f(x,y), \ y(x_0) = y_0, \ x_0 \leq x \leq x_n}$$

Teorema.- Si **f** es continua en $\mathbf{x_0 \leq x \leq x_n}$, $\mathbf{-\infty < y < \infty}$. Si f satisface la condición de Lipschitz en la variable $\mathbf{-\infty < y < \infty}$, entonces la ecuación diferencial $\mathbf{y'(x) = f(x,y), \ y(x_0) = y_0}$ tiene una solución única **y(x)** en $\mathbf{x_0 \leq x \leq x_n}$

Es decir que **f** debe ser continua en el rectángulo $\mathbf{x_0 \leq x \leq x_n, \ -\infty < y < \infty}$, y el valor de $\mathbf{y'(x) = f(x,y)}$ para los diferentes valores de **y**, debe estar acotado.

Adicionalmente, es importante verificar si la solución calculada es muy sensible a los errores en la formulación de la ecuación diferencial o en la condición inicial. Se puede detectar esta situación calculando numéricamente el problema original y el problema modificado con alguna perturbación.

9.1.1 Método de la serie de Taylor

Dada la ecuación diferencial de primer orden con la condición en el inicio:

$$y'(x) = f(x, y), \quad y(x_0) = y_0$$

El desarrollo de la Serie de Taylor se usa para obtener puntos de la solución a una distancia elegida **h**, a partir de la condición inicial conocida, y para estimar el error de truncamiento.

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y''_i + \ldots + \frac{h^n}{n!}y^{(n)}_i + \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(z), \quad x_i \le z \le x_{i+1}$$

$$= y_i + hf(x_i, y_i) + \frac{h^2}{2!}f^{(1)}(x_i, y_i) + \ldots + \frac{h^n}{n!}f^{(n)}(x_i, y_i) + \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(z), \quad x_i \le z \le x_{i+1}$$

h es el parámetro con el que se define la distancia para obtener los puntos de la solución

Esta fórmula permite mejorar la precisión incluyendo más términos de la serie. En la expresión resultante se usan las derivadas de **y(x)** por lo que las fórmulas resultantes son aplicables únicamente para la ecuación especificada.

El objetivo de los métodos numéricos es proporcionar fórmulas y algoritmos generales.

Ejemplo. Obtenga dos puntos de la solución de la siguiente ecuación diferencial utilizando los tres primeros términos de la serie de Taylor con **h = 0.1**

$$y' - y - x + x^2 - 1 = 0, \quad y(0) = 1$$

Solución: Desarrollo de la Serie de Taylor en el punto x_i

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y''_i, \quad E = \frac{h^3}{3!}y'''(z) = O(h^3) = O(0.001)$$ (Error de truncamiento)

$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Obtención de las derivadas

$$y' = f(x, y) = y - x^2 + x + 1,$$
$$y'' = f'(x, y) = y' - 2x + 1 = (y - x^2 + x + 1) - 2x + 1 = y - x^2 - x + 2$$

$$x_0 = 0, \; y_0 = 1, \; h = 0.1$$

Al sustituir las derivadas en el desarrollo de la Serie de Taylor se obtiene una fórmula para obtener puntos de la solución para la ecuación diferencial propuesta:

$$y_{i+1} = y_i + h(y_i - x^2_i + x_i + 1) + \frac{h^2}{2}(y_i - x^2_i - x_i + 2)$$

$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Puntos calculados:

i=0: $y_1 = y_0 + h(y_0 - x^2_0 + x_0 + 1) + \frac{h^2}{2}(y_0 - x^2_0 - x_0 + 2)$

$= 1 + 0.1(1 - 0^2 + 0 + 1) + \frac{0.1^2}{2}(1 - 0^2 - 0 + 2) = 1.2150$

$x_1 = x_0 + h = 0 + 0.1 = 0.1$

i=1: $y_2 = y_1 + h(y_1 - x^2_1 + x_1 + 1) + \frac{h^2}{2}(y_1 - x^2_1 - x_1 + 2)$

$= 1.2150 + 0.1(1.2150 - 0.1^2 + 0.1 + 1) + \frac{0.1^2}{2}(1.2150 - 0.1^2 - 0.1 + 2) = 1.4610$

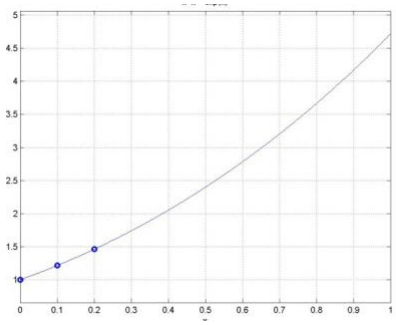
$x_2 = x_1 + h = 0.1 + 0.1 = 0.2$

Para comprobar la exactitud comparamos con la solución exacta**:** $y(x) = e^x + x + x^2$

y(0.1) = 1.2152
y(0.2) = 1.4614

El error concuerda con el orden del error de truncamiento para esta fórmula

En el siguiente gráfico se muestran los dos puntos obtenidos junto con el gráfico de la solución analítica exacta. La concordancia es muy buena.



Instrumentación computacional del método de la Serie de Taylor

La siguiente función programada en Python permite obtener puntos de la solución de una EDO de primer orden usando los primeros tres términos de la Serie de Taylor. La función requiere especificar **f(x,y), f'(x,y)**, el punto inicial **(x_0, y_0)** y los parámetros **h** (paso o distancia entre puntos), y **m** (cantidad de puntos).

La función entrega el resultado en **u** un arreglo de **NumPy** en el cual la primera columna contiene los valores de la variable independiente, **x**, mientras que la segunda columna contiene los valores de los resultados, **y**. El uso de arreglos de **NumPy** permite usar otras funciones de la librería, por ejemplo el formateo de los resultados.

Ecuación diferencial: $y'(x) = f(x,y),\ y(x_0) = y_0,\ x_0 \le x \le x_n$

Serie de Taylor: $y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y''_i = y_i + hf(x_i,y_i) + \frac{h^2}{2!}f'(x_i,y_i)$

$x_{i+1} = x_i + h,\ i = 0, 1, 2, \ldots$

```
import numpy as np
def taylor3(f,df,x,y,h,m):
    u=np.zeros([m,2])                #Iniciar la matriz u con mx2 ceros
    for i in range(m):
        y=y+h*f(x,y)+h**2/2*df(x,y)
        x=x+h
        u[i,0]=x                     #La primera columna almacena x
        u[i,1]=y                     #La segunda columna almacena y
    return u
```

Ejemplo. Con la función **taylor3**, obtenga 5 puntos de la solución de la siguiente ecuación diferencial. Tabule y grafique los puntos. Use **h = 0.1**

$$y' - y - x + x^2 - 1 = 0,\quad y(0) = 1$$

Solución

$y'(x) = f(x, y) = y - x^2 + x + 1$
$y''(x) = f'(x, y) = y - x^2 - x + 2$
$x_0 = 0,\ y_0 = 1,$
$h = 0.1,$
$m = 5$

```
>>> import numpy as np
>>> import pylab as pl
>>> from taylor3 import*
>>> def f(x,y):return y-x**2+x+1
>>> def df(x,y):return y-x**2-x+2
>>> u=taylor3(f,df,0,1,0.1,5)
```

```
>>> print(u)
[[ 0.1         1.215     ]
 [ 0.2         1.461025  ]
 [ 0.3         1.73923262]
 [ 0.4         2.05090205]
 [ 0.5         2.39744677]]
```

La primera columna de u tiene los valores de x
La segunda columna de u tiene los valores de y

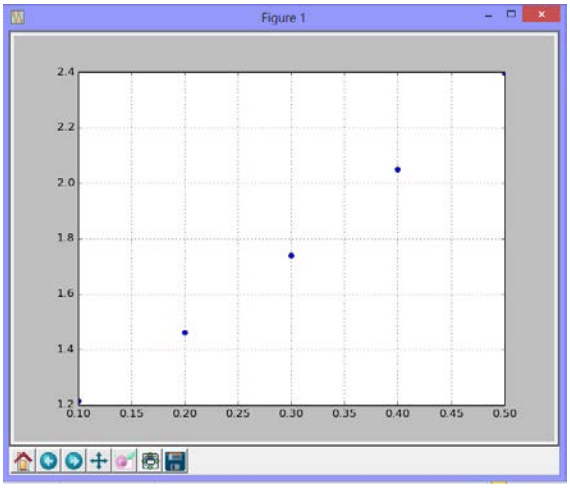
```
>>> np.set_printoptions(precision=4)
>>> print(u)
[[ 0.1     1.215 ]
 [ 0.2     1.461 ]
 [ 0.3     1.7392]
 [ 0.4     2.0509]
 [ 0.5     2.3974]]
```

Formateo de resultados (4 decimales)

```
>>> x=u[:,0]
>>> y=u[:,1]
>>> pl.plot(x,y,'o')
>>> pl.grid(True)
>>> pl.show()
```

Los valores de x.
Los valores de y
Gráficar los resultados con la librería Pylab

La primera columna de **u**, se escribe **`u[:,0]`** contiene los valores de **x**
La segunda columna de **u** se escribe **`u[:,1]`** contiene los valores de **y**
La marca **:** significa “todas las filas”



En los métodos que se revisarán posteriormente se desarrollan técnicas para usar la serie de Taylor sin necesidad de especificar las derivadas.

Solución numérica con el método **odeint** de la librería **Scipy** de **Python**

```
>>> from numpy import*
>>> from scipy.integrate import odeint
>>> def f(y,x):return y-x**2+x+1
>>> x=arange(0,0.6,0.1)
>>> y=odeint(f,1,x)
>>> print(x)
[ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5]
>>> print(y)
[[ 1.        ],
 [ 1.21517091],
 [ 1.46140275],
 [ 1.73985882],
 [ 2.05182469],
 [ 2.39872127]]
```

Note el orden de **x, y**
Dominio de y(x). Incluye x_0
Integrar. Se incluye y(x_0)

En las siguientes secciones se desarrollarán métodos numéricos con precisión similar o mejor que el método **Odeint** de Python. Adicionalmente, los métodos propuestos en este curso en su instrumentación computacional no requerirán especificar las derivadas y serán mas fáciles de utilizar que el método de Python.

Solución simbólica con el método **dsolve** de la librería simbólica **Sympy** de **Python**

Resolver la ecuación diferencial: $y' - y - x + x^2 - 1 = 0$

Con la condición inicial: $y(0) = 1$

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> dsolve(diff(y(x),x)-y(x)-x+x**2-1)
y(x) == (C1 + x*(x + 1)*exp(-x))*exp(x)          C1 es una constante
```

También puede mostrar en otro formato:

```
Eq(y(x), (C1 + x*(x + 1)*exp(-x))*exp(x))
```

```
>>> C1=Symbol('C1')                              Para determinar  C1
>>> y=(C1 + x*(x + 1)*exp(-x))*exp(x)            Solución simbólica
>>> y.subs(x,0)                                  Condición inicial  y(0)=1
C1
```

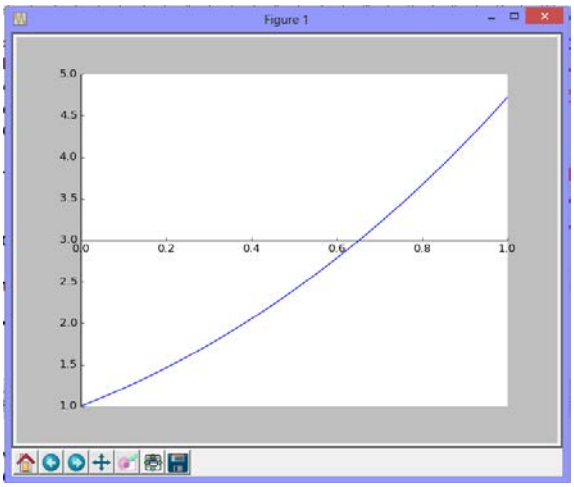
$y(0) = 1 \Rightarrow C1 = 1$

```
>>> y=y.subs(C1,1)
>>> y
(x*(x + 1)*exp(-x) + 1)*exp(x)                   Solución simbòlica explícita
>>> y=expand(y)                                  Simplificar
>>> y
x**2 + x + exp(x)
```

Solución analítica: $y(x) = x^2 + x + e^x$

Gráfico de la solución analítica con la función **plot** de la librería **Sympy**, en el intervalo [0,1]

```
>>> y=x**2+x+exp(x)
>>> plot(y,(x,0,1))
```



Si la ecuación diferencial es de tipo no-lineal, entonces el método simbólico computacional usualmento no puede llegar a una solución analítica.

Nota. El método **dsolve** de la librería **Sympy** de Python no entrega soluciones explícitas. Se requiere realizar las sustituciones para obtenerla. Otros programas computacionales con manejo simbólico entregan directamente la solución final explícita, en caso de que exista.

9.1.2 Obtención de derivadas de funciones implícitas

La instrumentación de la función **taylor3** requiere enviar como parámetros **f(x,y)** y **f'(x,y)**.

La derivada **f'(x,y)** debe obtenerse previamente en forma manual y debe ser enviada como parámetro.

En la siguiente sección se decribe una función para obtener derivadas de **y'(x) = f(x,y)** computacionalmente.

Para entender la notación de **Sympy** para derivar funciones implícitas se muestra el desarrollo de un ejemplo simple:

Ejemplo. Obtener la primera derivada de la función

$$y'(x) = f(x, y) = y - x^2 + x + 1, \text{ en donde } y=y(x)$$

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> f=y-x**2+x+1
>>> d=diff(f,x);print(d)                    Con este formato de f, diff considera a y
-2*x + 1                                      como constante y desaparece
>>> f=y(x)-x**2+x+1                         Esta notación define y como función de x
>>> d=diff(f,x);print(d)                    Ahora, diff realiza derivación implícita
-2*x + Derivative(y(x), x) + 1
>>> d=d.subs(Derivative(y(x),x),f);print(d)     Reemplaza y'(x) por f
-x**2 - x + y(x) + 2
>>> d=d.subs(y(x),y);print(d)                       Reemplaza y(x) por y
-x**2 - x + y + 2                           Derivada final en formato simple
```

Desarrollo de una función para obtener las derivadas sucesivas de una función implícita usando el manejo simbólico de la librería **Sympy**. La expresión resultante queda en formato simple y directamente evaluable, por lo tanto puede incorporarse en un método numérico para evaluar términos de la serie de Taylor evitando el envío de derivadas desde fuera del método numérico. Por simplicidad deben usarse los símbolos **'x, y'** como variables.

Esta función usa la función Derivative de Python para derivar implícitamente, y usa sustituciones para expresarla en forma simple. Es una contribución de interés para este tema.

Uso de la función derive:

derive(f,nd)

f: función **f(x,y)** en la cual **y=y(x)** siendo **x** la variable independiente
nd: orden de la derivada

```
import sympy as sp
x,y=sp.symbols('x,y')
def derive(f,nd):
    t=f
    for j in range(1,nd+1):
        d=sp.diff(f.subs(y,y(x)),x)
        f=d.subs(sp.Derivative(y(x),x),t).subs(y(x),y)
    return f
```

Ejemplo. Sea **y = y(x)**, obtenga la primera derivada en **x** de **f(x,y) = y^2sen(y)-x^2+x+1**

```
>>> from derive import derive
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> f=y**2*sin(y)-x**2+x+1
>>> d=derive(f,1)
>>> print(d)
-2*x + y**2*(-x**2 + x + y**2*sin(y) + 1)*cos(y) + 2*y*(-x**2 + x +
y**2*sin(y) + 1)*sin(y) + 1
```

Evaluación de la derivada

Ejemplo. Evaluar la derivada obtenida anteriormente. Use **x = 1.5, y = 2.3**

```
>>> r=d.subs(x,1.5).subs(y,2.3)          Sustitución de variables
>>> r
-2.39580345631014
>>> r.evalf(6)                           Función para mostrar
-2.39580                                 una cantidad fija de dígitos
```

9.1.3 Instrumentación de un método general para resolver una E.D.O con la serie de Taylor

Con la función **derive** se puede instrumentar una función para obtener puntos de la solución de una ecuación diferencial ordinaria de primer orden, lineal o no lineal, con una cantidad arbitraria de términos de la serie de Taylor con lo cual, en teoría, la precisión puede ser ilimitada:

Uso de la función: **taylorg(f, a, b, h, m, k)**

La función requiere especificar **f = f(x,y)** definida en forma simbólica, el punto inicial **($a=x_0$, $b=y_0$)** y los parámetros **h** (paso o distancia entre puntos), **m** (cantidad de puntos que se calcularán), **k** (orden de la derivada de **y'(x) q**ue se desea incluir en el desarrollo, **$k \geq 1$**).

Ecuación diferencial:

$$y'(x) = f(x,y), \quad y(x_0) = y_0,\ x_0 \leq x \leq x_n$$

Desarrollo de la serie de Taylor:

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2!} f^{(1)}(x_i, y_i) + \frac{h^3}{3!} f^{(2)}(x_i, y_i) + \ldots + \frac{h^{k+1}}{(k+1)!} f^{(k)}(x_i, y_i)$$

$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Error de truncamiento en cada paso: $E = O(h^{k+2})$

```
from derive import *
import numpy as np
import sympy as sp
x,y=sp.symbols('x,y')
def taylorg(f,a,b,h,m,k):
    u=np.zeros([m,2])
    D=[ ]
    for j in range(1,k+1):
        D=D+[derive(f,j)]          #Vector de derivadas simbólicas

    for i in range(m):
        g=f.subs(x,a).subs(y,b)
        t=b+h*g
        for j in range(1,k+1):
            z=D[j-1].subs(x,a).subs(y,b)
            t=float(t+h**(j+1)/sp.factorial(j+1)*z)    #Evaluar Taylor
        b=t
        a=a+h
        u[i,0]=a
        u[i,1]=b
    return u
```

Ejemplo. Calcule 5 puntos de solución de la ecuación $y' - y - x + x^2 - 1 = 0$, $y(0) = 1$. Use el desarrollo de la serie de Taylor hasta la tercera derivada de $f(x,y) = y'(x)$, con $h = 0.1$

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2!} f^{(1)}(x_i, y_i) + \frac{h^3}{3!} f^{(2)}(x_i, y_i) + \frac{h^4}{4!} f^{(3)}(x_i, y_i)$$

$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, 4, 5$$

El error de truncamiento en cada paso será $O(h^5)$

Solución

$$f(x, y) = y'(x) = y - x^2 + x + 1$$

Con la función taylorg:

```
>>> from sympy import *
>>> from taylorg import *
>>> x,y=symbols('x,y')
>>> f=y+x-x**2+1
>>> u=taylorg(f,0,1,0.1,5,3)
>>> print(u)
[[ 0.1         1.21517083]
 [ 0.2         1.46140257]
 [ 0.3         1.7398585 ]
 [ 0.4         2.05182424]
 [ 0.5         2.39872064]]
```

Estos resultados tienen un error de truncamiento en el orden 10^{-6}

La primera columna de **u**, se escribe `u[:,0]` contiene los valores de **x**
La segunda columna de **u** se escribe `u[:,1]` contiene los valores de **y**
La marca **:** significa “todas las filas”

En las siguientes secciones se describirán algunos métodos numéricos clásicos equivalentes a usar más términos de la serie de Taylor y que no requieren construir las derivadas de **f(x,y)** ni usar variables de tipo simbólico.

Estos métodos usan aproximaciones para las derivadas y tienen la ventaja que pueden extenderse a sistemas de ecuaciones diferenciales y a ecuaciones diferenciales con derivadas de mayor orden, incluyendo el importante caso de las ecuaciones diferenciales no-lineales.

9.1.4 Fórmula de Euler

Las siguientes fórmulas constituyen los métodos clásicos para resolver numéricamente ecuaciones diferenciales ordinarias. Son equivalentes a usar varios términos de la serie de Taylor pero sustituyen las derivadas por aproximaciones, de tal manera que no se requiere especificarlas o usar el método de derivación implícita anterior. Las fórmulas y los algoritmos resultantes son independientes de una EDO particular.

Sea una ecuación diferencial ordinaria explícita de primer orden con una condición en el inicio:

$$\mathbf{f(x, y) = y'(x), \quad y(x_0) = y_0}$$

La fórmula de Euler usa los dos primeros términos de la serie de Taylor:

$$\mathbf{y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y''(z)}$$

$$\mathbf{= y_i + hf(x_i, y_i) + \frac{h^2}{2!}y''(z), \quad x_i \le z \le x_{i+1}}$$

Definición: Fórmula de Euler

$$\mathbf{y_{i+1} = y_i + h\, f(x_i, y_i)}$$
$$\mathbf{x_{i+1} = x_i + h, \qquad i = 0, 1, 2, \ldots}$$
$$\mathbf{E = \frac{h^2}{2!}y''(z) = O(h^2), \quad x_i \le z \le x_{i+1}}$$ (Error de truncamiento en cada paso)

Algoritmo para calcular puntos de la solución de una EDO de primer orden con la fórmula de Euler

1) Defina **f(x,y)** y la condición incial **(x_0, y_0)**
2) Defina **h** y la cantidad de puntos a calcular **m**
3) Para **i = 1, 2, ..., m**
4) $\mathbf{y_{i+1} = y_i + h\, f(x_i, y_i)}$.
5) $\mathbf{x_{i+1} = x_i + h}$
6) fin

Ejemplo. Obtenga dos puntos de la solución de la siguiente ecuación diferencial con la fórmula de Euler. Use h = 0.1

$$\mathbf{y' - y - x + x^2 - 1 = 0, \quad y(0) = 1}$$

Ecuación diferencial

$$\mathbf{y' = f(x, y) = y + x - x^2 + 1, \quad x_0 = 0, y_0 = 1, \ h = 0.1}$$

Cálculo de los puntos

i=0: $y_1 = y_0 + h\, f(x_0, y_0) = 1 + 0.1\, f(0, 1) = 1 + 0.1(1 + 0 - 0^2 + 1) = 1.2000;$
$x_1 = x_0 + h = 0 + 0.1 = 0.1$

i=1: $y_2 = y_1 + h\, f(x_1, y_1) = 1.2 + 0.1\, f(0.1, 1.2) = 1.2 + 0.1\,(1.2 + 0.1 - 0.1^2 + 1] = 1.4290$
$x_2 = x_1 + h = 0.1 + 0.1 = 0.2$

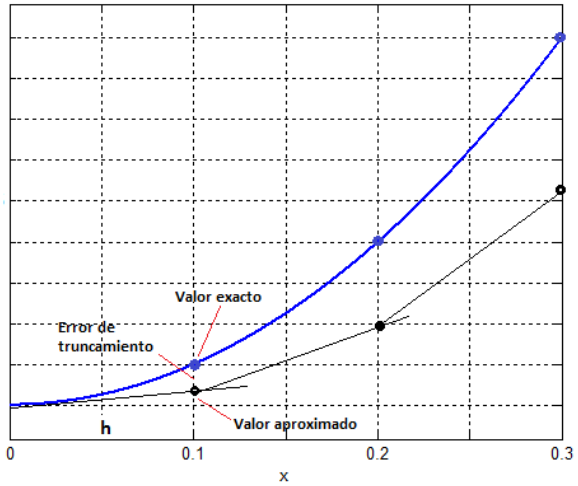
Para comprobar comparamos con la solución exacta**:** $y(x) = x + x^2 + e^x$

$y(0.1) = 1.2152$
$y(0.2) = 1.4614$

El error es significativo. Para reducirlo se pudiera reducir **h**. Esto haría que el error de truncamiento se reduzca pero si la cantidad de cálculos es muy grande, pudiera acumular error de redondeo. Una mejor estrategia es usar métodos más precisos que no requieran que **h** sea muy pequeño.

9.1.5 Error de truncamiento y error de redondeo

La fórmula de Euler utiliza la pendiente de la recta en cada punto para predecir y estimar la solución en el siguiente punto, a una distancia elegida **h**. La diferencia del punto calculado, con respecto al valor exacto es el error de truncamiento, el cual puede crecer al proseguir el cálculo.



En cada paso el error de truncamiento es:

$$E = \frac{h^2}{2} y''(z) = O(h^2),$$

Para reducir **E** se debe reducir **h** puesto que $h \to 0 \Rightarrow E \to 0$. Sin embargo, este hecho matemáticamente cierto, al ser aplicado tiene una consecuencia importante que es interesante analizar

Suponer que se desea calcular la solución **y(x)** en un intervalo fijo $x_0 \le x \le x_f$ mediante **m** puntos $x_i = x_0, x_1, x_2, \ldots, x_m$ espaciados regularmente en una distancia **h**:

$$h = \frac{x_f - x_0}{m}$$

Sea E_i el error de truncamiento en el paso **i**, entonces

$y_1 = y_0 + h\, f(x_0, y_0) + E_1$
$y_2 = y_1 + h\, f(x_1, y_1) + E_2 = y_0 + hf(x_0, y_0) + hf(x_1, y_1) + E_1 + E_2$
$y_3 = y_2 + h\, f(x_2, y_2) + E_3 = y_0 + hf(x_0, y_0) + hf(x_1, y_1) + hf(x_2, y_2) + E_1 + E_2 + E_3$
. . .
$y_m = y_0 + hf(x_0, y_0) + hf(x_1, y_1) + hf(x_2, y_2) + \ldots + hf(x_{m-1}, y_{m-1}) + E_1 + E_2 + E_3 + \ldots + E_m$

Siendo $E_i = \frac{h^2}{2} y''(z_i)$

El error de truncamiento acumulado es:

$$E = E_1 + E_2 + E_3 + \ldots + E_m = mh^2(\bar{D}) = \frac{x_f - x_0}{h} h^2(\bar{D}) = h[(x_f - x_0)\bar{D}]$$

En donde suponemos que existe un valor promedio $\bar{D}$ de los valores de $\frac{y''(z_i)}{2}$, independiente de **h**. Se muestra que el error de truncamiento acumulado es solamente de orden **O(h)**, por lo tanto **h** debe ser un valor mas pequeño que el previsto para asegurar que la solución calculada sea suficientemente precisa hasta el final del intervalo.

Por otra parte, cada vez que se evalúa $f(x_i, y_i)$ se puede introducir el error de redondeo R_i debido a los errores en la aritmética computacional y al dispositivo de almacenamiento. Entonces, el error de redondeo se pudiera acumular en cada paso y al final del intervalo se tendrá:

$$R = R_1 + R_2 + R_3 + \ldots + R_m = m(\bar{R}) = \frac{x_f - x_0}{h}(\bar{R}) = \frac{1}{h}[(x_f - x_0)\bar{R}]$$

$\bar{R}$ es algún valor promedio del error de redondeo en cada paso, independiente de **h**.

El error total acumulado al realizar los cálculos con la fórmula de Euler hasta el final del intervalo:

$$E_A = E + R = h[(x_f - x_0)\bar{D}] + \frac{1}{h}[(x_f - x_0)\bar{R}]$$

Si **m** es muy grande, **h** será muy pequeño y **E** será pequeño, pero al reducir **h**, el error de redondeo **R** puede crecer y llegar a ser mayor al error de truncamiento, por lo tanto el resultado perderá precisión en vez de aumentarla, lo cual se acepta equivocadamente como verdadero porque solamente se considera el error de truncamiento.

Como conclusión de lo anterior, es preferible usar fórmulas cuyo error de truncamiento **E** sea de mayor orden para que el valor de **h** no requiera ser muy pequeño si se buscan resultados con alta precisión. Esto retardará también el efecto del error de redondeo acumulado **R**.

Instrumentación computacional de la fórmula de Euler

Se define una función, **euler**, para obtener puntos de la solución de una EDO de primer orden con dos términos de la Serie de Taylor. La función requiere especificar **f(x,y)**, el punto inicial **(x_0, y_0)** y los parámetros **h** (paso o distancia entre puntos), y **m** (cantidad de puntos).

La función entrega una matriz **u** en la cual la primera columna contiene los valores de x, mientras que la segunda columna contiene los valores de y(x)

Ecuación diferencial: $y'(x) = f(x,y),\ y(x_0) = y_0,\ x_0 \le x \le x_n$
Fórmula de Euler: $y_{i+1} = y_i + hy'_i = y_i + hf(x_i,y_i)$
$x_{i+1} = x_i + h,\ i = 0, 1, 2, \ldots$

```
import numpy as np
def euler(f,x,y,h,m):
    u=np.zeros([m,2])
    for i in range(m):
        y=y+h*f(x,y)
        x=x+h
        u[i,0]=x
        u[i,1]=y
    return u
```

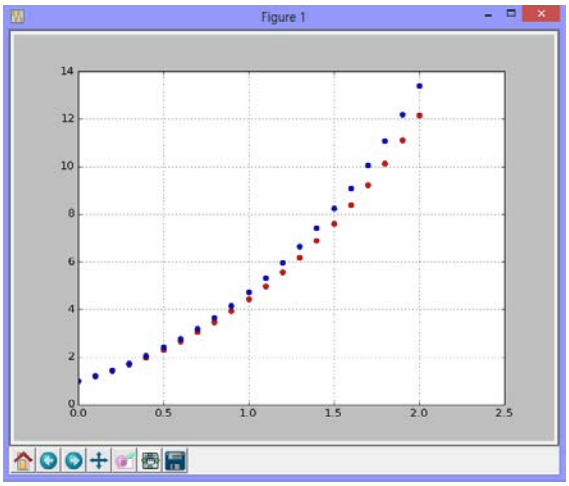
Ejemplo. Calcule 20 puntos de la solución de la ecuación $y' - y - x + x^2 - 1 = 0,\ y(0) = 1$ espaciados en una distancia h = 0.1, usando la función **euler**.

Grafique y compare con la solución analítica.

Solución

$f(x, y) = y - x^2 + x + 1$
$x_0 = 0,\ y_0 = 1,$
$h = 0.1,$
$m = 20$ (cantidad de puntos)

```
>>> import pylab as pl
>>> from euler import*
>>> def f(x,y):return y-x**2+x+1          Ecuación diferencial
>>> u=euler(f,0,1,0.1,20)
>>> pl.plot(u[:,0],u[:,1],'or')           Gráfico de la solución con Euler
>>> def y(x):return exp(x)+x**2+x         Solución analítica
>>> x=pl.arange(0,2.1,0.1)
>>> pl.plot(x,y(x),'ob')                  Gráfico de la solución analítica
>>> pl.grid(True)
>>> pl.show()
```



Puntos de la solución analítica y la solución numérica con la fórmula de Euler

9.1.6 Fórmula mejorada de Euler o fórmula de Heun

Sea la ecuación diferencial de primer orden con condición en el inicio:

$$y'(x) = f(x, y),\ y(x_0) = y_0$$

La fórmula de Heun o fórmula mejorada de Euler es aproximadamente equivalente a usar los tres primeros términos de la serie de Taylor con un artificio para sustituir la primera derivada de **f(x, y)**

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y''_i + \frac{h^3}{3!}y'''(z) = y_i + hf(x_i, y_i) + \frac{h^2}{2!}f'(x_i, y_i) + \frac{h^3}{3!}y'''(z),\ x_i \le z \le x_{i+1}$$

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2}f'(x_i, y_i) + O(h^3)$$

Para evaluar $f'(x_i, y_i)$ usamos una aproximación simple: $f'_i = \frac{f_{i+1} - f_i}{h} + O(h)$

$$y_{i+1} = y_i + hf_i + \frac{h^2}{2}[\frac{f_{i+1} - f_i}{h} + O(h)] + O(h^3) = y_i + hf_i + \frac{h}{2}f_{i+1} - \frac{h}{2}f_i + O(h^3)$$

$$y_{i+1} = y_i + \frac{h}{2}(f_i + f_{i+1}) + O(h^3) = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_{i+1})) + O(h^3)$$

Para evaluar $f_{i+1} = f(x_{i+1}, y_{i+1})$ se usa y_{i+1} calculado con la fórmula de Euler como aproximación inicial y se obtiene la fórmula de Heun o fórmula mejorada de Euler

Definición: Fórmula de Heun

$$\bar{y}_{i+1} = y_i + hf(x_i, y_i)$$

$$y_{i+1} = y_i + \frac{h}{2}\,(f(x_i, y_i) + f(x_{i+1}, \bar{y}_{i+1}))$$

$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

$$E = \frac{h^3}{3!}y'''(z) = O(h^3), \quad x_i \le z \le x_{i+1}$$ (Error de truncamiento en cada paso)

Se puede interpretar esta fórmula como una fórmula de corrección en la que se usa una primera estimación como valor inicial para una fórmula de mayor exactitud.

El error de truncamiento en cada paso es de tercer orden $O(h^3)$, y el error de truncamiento acumulado es de segundo orden $O(h^2)$, mejor que la fórmula de Euler.

Algoritmo para resolver una EDO de primer orden con la fórmula de Heun

1) **Defina f(x,y) y la condición incial (x_0, y_0)**
2) **Defina h y la cantidad de puntos a calcular m**
3) **Para i = 1, 2, ..., m**
4) $\bar{y}_{i+1} = y_i + hf(x_i, y_i)$
5) $y_{i+1} = y_i + \frac{h}{2}\,(f(x_i, y_i) + f(x_{i+1}, \bar{y}_{i+1}))$
6) $x_{i+1} = x_i + h$
7) **fin**

Ejemplo. Obtener dos puntos de la solución de la siguiente ecuación diferencial con la fórmula de Heun. Use h = 0.1

$$y' - y - x + x^2 - 1 = 0, \quad y(0) = 1$$

Solución

$$y' = f(x, y) = x - x^2 + y + 1, \quad x_0 = 0, y_0 = 1, \ h = 0.1$$

i=0: $\bar{y}_1 = y_0 + hf(x_0, y_0) = 1 + 0.1\, f(0, 1) = 1 + 0.1\,(0 - 0^2 + 1 + 1) = 1.2000$

$$y_1 = y_0 + \frac{h}{2}\,(f(x_0, y_0) + f(x_1, \bar{y}_1)) = 1 + \frac{0.1}{2}(f(0, 1) + f(0.1, 1.2) = 1.2145$$

$$x_1 = x_0 + h = 0 + 0.1 = 0.1$$

i=1: $\bar{y}_2 = y_1 + hf(x_1, y_1) = 1.2145 + 0.1\, f(0.1, 1.2145) = 1.44495$

$$y_2 = y_1 + \frac{h}{2}\left(f(x_1, y_1) + f(x_2, \bar{y}_2)\right)$$

$$= 1.2145 + \frac{0.1}{2}(f(0.1, 1.2145) + f(0.2, 1.44495) = 1.45997\ 25$$

$$x_2 = x_1 + h = 0.1 + 0.1 = 0.2$$

Para comprobar comparamos con la solución exacta**:** $y(x) = x + x^2 + e^x$

$y(0.1) = 1.2152$

$y(0.2) = 1.4614$

El error de truncamiento en cada paso está en el orden de los milésimos, coincidiendo aproximadamente con $E=O(h^3)$

Instrumentación computacional de la fórmula de Heun

Una función para obtener puntos de la solución de una EDO de primer orden con una condición en el inicio. La función requiere **f(x,y)**, el punto inicial (x_0, y_0) y los parámetros **h** (paso o distancia entre puntos), y **m** (cantidad de puntos).

La función entrega una matriz **u** en la cual la primera columna contiene los valores de x, mientras que la segunda columna contiene los valores de y(x)

Ecuación diferencial: $y'(x)=f(x,y),\ y(x_0)=y_0,\ x_0 \le x \le x_n$

Fórmula de Heun: $\bar{y}_{i+1} = y_i + hf(x_i, y_i)$

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f(x_{i+1}, \bar{y}_{i+1})\right)$$

$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

```
import numpy as np
def heun(f,x,y,h,m):
    u=np.zeros([m,2],dtype=float)
    for i in range(m):
        yn=y+h*f(x,y)
        y=y+h/2*(f(x,y)+f(x+h,yn))
        x=x+h
        u[i,0]=x
        u[i,1]=y
    return u
```

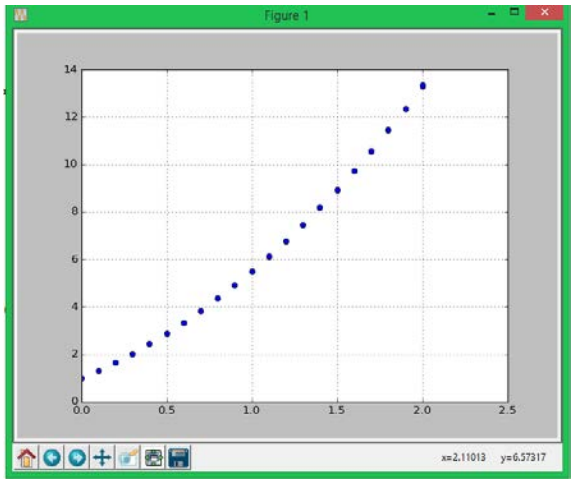
Ejemplo. Calcule 20 puntos de la solución de **$y' - y - \cos(x) + x^2 - 1 = 0$, con $y(0) = 1$** espaciados en una distancia h = 0.1, usando la función Heun

Grafique y compare con la solución analítica.

Solución

$f(x, y) = y - x^2 + \cos(x) + 1$
$x_0 = 0, y_0 = 1$
h = 0.1
m = 20 (cantidad de puntos)

```
>>> import pylab as pl
>>> import numpy as np
>>> from heun import*
>>> def f(x,y):return y-x**2+np.cos(x)+1          Ecuación diferencial
>>> u=heun(f,0,1,0.1,20)
>>> np.set_printoptions(precision=5)
>>> print(u)
[[  0.1       1.31425]                            Salida formateada con 5 dec.
 [  0.2       1.65843]
 [  0.3       2.03253]
 [  0.4       2.43659]
 [  0.5       2.87067]
 [  0.6       3.33488]
 [  0.7       3.82937]
  .......etc.......


>>> pl.grid(True)
>>> pl.show()
>>> pl.plot(u[:,0],u[:,1],'ob')                   Gráfico de la solución con Heun

                                                  Solución analítica
>>> def y(x):return 2*x - np.cos(x)/2 + np.exp(x)/2 + np.sin(x)/2 + x**2 + 1
>>> x=pl.arange(0,2.1,0.1)
>>> pl.plot(x,y(x),'ob')                          Gráfico de la solución analítica
>>> pl.grid(True)
>>> pl.show()
```



Puntos de la solución analítica y numérica con la fórmula de Heun

La diferencia en el gráfico no es significativa visualmente

NOTA. Observe el uso calificado de las funciones matemáticas con el prefijo **np.** o con el prefijo **pl.** Contienen definiciones similares.

```
np.cos()
pl.cos()
```

9.1.7 Fórmula de Runge-Kutta de segundo orden

Las fórmulas de Runge-Kutta utilizan artificios matemáticos para incorporar más términos de la serie de Taylor mediante aproximaciones para las derivadas.

Dada una ecuación diferencial de primer orden con una condición en el inicio:

$$y'(x) = f(x, y), \quad y(x_0) = y_0$$

La fórmula de Runge-Kutta de segundo orden consiste en determinar las constantes **a, b** de la siguiente expresión de tres términos:

$$y_{i+1} = y_i + aK_1 + bK_2$$

De tal manera que el desarrollo sea equivalente a la serie de Taylor con tres términos:

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2} f'(x_i, y_i) + O(h^3)$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Una forma directa de obtener la fórmula es compararla con la fórmula de Heun, la cual es equivalente a usar tres terminos de la Serie de Taylor:

$$y_{i+1} = y_i + hf(x_i, y_i)$$
$$y_{i+1} = y_i + \frac{h}{2} (f(x_i, y_i) + f(x_{i+1}, y_{i+1})) = y_i + \frac{1}{2} (hf(x_i, y_i) + hf(x_{i+1}, y_{i+1}))$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Si se definen:

$$K_1 = hf(x_i, y_i)$$
$$K_2 = hf(x_{i+1}, y_{i+1})) = hf(x_i + h, y_i + hf(x_i, y_i)) = hf(x_i + h, y_i + K_1)$$

Se utliza el promedio de K_1 y K_2 con lo que **a = 1/2, b = 1/2** en la fórmula propuesta:

Definición: Fórmula de Runge-Kutta de segundo orden

$$K_1 = hf(x_i, y_i)$$
$$K_2 = hf(x_i + h, y_i + K_1)$$
$$y_{i+1} = y_i + \frac{1}{2}(K_1 + K_2)$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$
$$E = \frac{h^3}{3!} y'''(z) = O(h^3), \quad x_i \le z \le x_{i+1}$$ (Error de truncamiento en cada paso)

Gráficamente, se puede interpretar que esta fórmula calcula cada nuevo punto usando un promedio de las pendientes en los puntos inicial y final en cada intervalo de longitud **h**.

El error de truncamiento en cada paso es de tercer orden $O(h^3)$, y el error de truncamiento acumulado es de segundo orden $O(h^2)$, mejor que la fórmula de Euler.

Algoritmo para resolver una EDO de primer orden con la fórmula de Runge_Kutta de segundo orden

1) **Defina f(x,y) y la condición incial (x_0, y_0)**
2) **Defina h y la cantidad de puntos a calcular m**
3) **Para i = 1, 2, ..., m**
4) $K_1 = hf(x_i, y_i)$
5) $K_2 = hf(x_i + h, y_i + K_1))$
6) $y_{i+1} = y_i + \frac{1}{2}(K_1 + K_2)$.
7) $x_{i+1} = x_i + h$
8) **fin**

Ejemplo. Obtener dos puntos de la solución de la siguiente ecuación diferencial con la fórmula de Runge-Kutta de segundo orden. Use h = 0.1

$$y' - y - x + x^2 - 1 = 0, \quad y(0) = 1$$

Solución

$$y' = f(x, y) = x - x^2 + y + 1, \quad x_0 = 0, y_0 = 1, \ h = 0.1$$

i=0: $K_1 = hf(x_0, y_0) = 0.1\, f(0, 1) = 0.1\,(0 - 0^2 + 1 + 1) = 0.2000;$

$K_2 = hf(x_0 + h, y_0 + K_1) = 0.1\, f(0.1, 1.2) = 0.1\,[0.1 - 0.1^2 + 1.2 + 1] = 0.2290$

$y_1 = y_0 + \frac{1}{2}(K_1 + K_2) = 1 + 0.5(0.2000 + 0.2290) = 1.2145$

$x_1 = x_0 + h = 0 + 0.1 = 0.1$

i=1: $K_1 = hf(x_1, y_1) = 0.1\, f(0.1, 1.2145) = 0.1\,(0.1 - 0.1^2 + 1.2145 + 1) = 0.2305;$

$K_2 = hf(x_1 + h, y_1 + K_1) = 0.1\, f(0.2, 1.4450) = 0.1\,[0.2 - 0.2^2 + 1.4450 + 1] = 0.2605$

$y_2 = y_1 + \frac{1}{2}(K_1 + K_2) = 1.2145 + 0.5(0.2305 + 0.2605) = 1.4600$

$x_2 = x_1 + h = 0.1 + 0.1 = 0.2$

Para comprobar comparamos con la solución exacta: $y(x) = x + x^2 + e^x$

$y(0.1) = 1.2152$

$y(0.2) = 1.4614$

El error de truncamiento en cada paso está en el orden de los milésimos, coincidiendo aproximadamente con **$E=O(h^3)$**

Instrumentación computacional de la fórmula de Runge-Kutta de segundo orden

Una función en Python para obtener puntos de la solución de una EDO de primer orden con una fórmula equivalente a usar tres términos de la Serie de Taylor pero sin construir la derivada **f'(x,y)**. La función requiere únicamente **f(x,y)**, el punto inicial **(x_0, y_0)** y los parámetros **h** (paso o distancia entre puntos), y **m** (cantidad de puntos).

La función entrega una matriz **u** en la cual la primera columna contiene los valores de x, mientras que la segunda columna contiene los valores de **y(x)**

Ecuación diferencial: **$y'(x)=f(x,y), y(x_0)=y_0, x_0 \leq x \leq x_n$**

Instrumentación en Python de la fórmula de Runge-Kutta de segundo orden

```
import numpy as np
def rk2(f,x,y,h,m):
    u=np.zeros([m,2],float)
    for i in range(m):
        k1=h*f(x,y)
        k2=h*f(x+h,y+k1)
        y=y+0.5*(k1+k2)
        x=x+h
        u[i,0]=x
        u[i,1]=y
    return u
```

Ejemplo. Calcule y grafique 20 puntos de la solución de la ecuación **$y' - y - x + x^2 - 1 = 0$, $y(0) = 1$** espaciados en una distancia h = 0.1, usando la función rk2

$f(x, y) = y - x^2 + x + 1$
$x_0 = 0$, $y_0 = 1$,
h = 0.1,
m = 20 (cantidad de puntos)

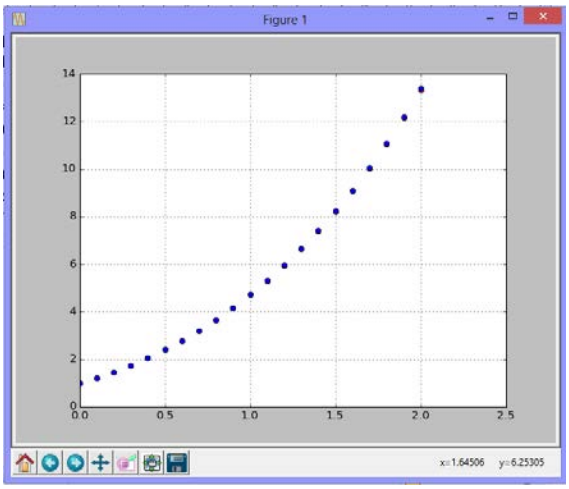
```
>>> import pylab as pl
>>> from rk2 import*
>>> import numpy as np
>>> def f(x,y):return y-x**2+x+1                    Ecuación diferencial
>>> u=rk2(f,0,1,0.1,20)
>>> np.set_printoptions(precision=5)
```

```
>>> print(u)
[[  0.1       1.2145 ]
 [  0.2       1.45997]
 [  0.3       1.73757]
 [  0.4       2.04856]
 [  0.5       2.39436]
 [  0.6       2.77652]
 [  0.7       3.19676]
 .......etc........
```

Tabla de resultados formateada

```
>>> pl.plot(u[:,0],u[:,1],'ob')
>>> pl.grid(True)
>>> pl.show()
```

Gráfico de puntos de la solución (x_i , y_i)



Puntos de la solución numérica con la fórmula rk2

9.1.8 Fórmula de Runge-Kutta de cuarto orden

La fórmula de Runge-Kutta de cuarto orden es la fórmula más utilizada para resolver una ecuación diferencial de primer orden con una condición en el inicio

Dada una ecuación diferencial de primer orden con una condición en el inicio:

$$y'(x) = f(x, y), \quad y(x_0) = y_0$$

La fórmula de Runge-Kutta de cuarto orden se obtiene determinando las constantes **a, b, c, d** de la siguiente expresión de cinco términos:

$$y_{i+1} = y_i + aK_1 + bK_2 + cK_3 + dK_4$$

De tal manera que el desarrollo sea equivalente a la serie de Taylor con 5 términos:

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2} f'(x_i, y_i) + \frac{h^3}{3!} f''(x_i, y_i) + \frac{h^4}{4!} f'''(x_i, y_i) + O(h^5)$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Mediante sustituciones de las derivadas por aproximaciones, se desarrolla una fórmula que no requiere especificar las derivadas de **f(x,y)**. La fórmula en el recuadro es la más popular:

Definición: Fórmula de Runge-Kutta de cuarto orden

$$K_1 = hf(x_i, y_i)$$
$$K_2 = hf(x_i + h/2, y_i + K_1/2)$$
$$K_3 = hf(x_i + h/2, y_i + K_2/2)$$
$$K_4 = hf(x_i + h, y_i + K_3)$$
$$y_{i+1} = y_i + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$
$$x_{i+1} = x_i + h, \qquad i = 0, 1, 2, \ldots$$
$$E = \frac{h^5}{5!} y^{(5)}(z) = O(h^5), \quad x_i \le z \le x_{i+1} \qquad \text{(Error de truncamiento en cada paso)}$$

Gráficamente, se puede interpretar en la fórmula que para cada punto calcula el promedio ponderado de las pendientes en los puntos inicial, medio y final para cada paso de longitud **h** asignando el doble de peso a las pendientes en el centro del intervalo

El error de truncamiento en cada paso es de quinto orden $O(h^5)$, y el error de truncamiento acumulado es de cuarto orden $O(h^4)$, suficientemente exacto para problemas prácticos comunes.

Algoritmo para resolver una EDO de primer orden con la fórmula de Runge-Kutta

1) **Defina f(x,y) y la condición inicial (x_0, y_0)**
2) **Defina h y la cantidad de puntos a calcular m**
3) **Para i = 1, 2, ..., m**
4) $K_1 = hf(x_i, y_i)$
5) $K_2 = hf(x_i + h/2, y_i + K_1/2)$
6) $K_3 = hf(x_i + h/2, y_i + K_2/2)$
7) $K_4 = hf(x_i + h, y_i + K_3)$
8) $y_{i+1} = y_i + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$.
9) $x_{i+1} = x_i + h$
10) **fin**

Ejemplo. Obtenga un punto de la solución de la siguiente ecuación diferencial con la fórmula de Runge-Kutta de cuarto orden. Use **h = 0.1**

$$y' - y - x + x^2 - 1 = 0, \quad y(0) = 1$$

Solución

$$y' = f(x, y) = x - x^2 + y + 1, \quad x_0 = 0, y_0 = 1, \quad h = 0.1$$

Cálculo de puntos

i=0:

$K_1 = hf(x_0, y_0) = 0.1\, f(0, 1) = 0.1\,(0 - 0^2 + 1 + 1) = 0.2000;$

$K_2 = hf(x_0 + h/2, y_0 + K_1/2) = 0.1\, f(0.05, 1.1) = 0.1\,(0.05 - 0.05^2 + 1.1 + 1) = 0.2148$

$K_3 = hf(x_0 + h/2, y_0 + K_2/2) = 0.1\, f(0.05, 1.1074) = 0.1\,(0.05 - 0.05^2 + 1.1074 + 1) = 0.2155$

$K_4 = hf(x_0 + h, y_0 + K_3) = 0.1\, f(0.1, 1.2155) = 0.1\,(0.1 - 0.1^2 + 1.2155 + 1) = 0.2305$

$$y_1 = y_0 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) = 1 + \frac{1}{6}[0.2 + 2(0.2148) + 2(0.2155) + 0.2305] = 1.2152$$

$x_1 = x_0 + h = 0 + 0.1 = 0.1$

Para comprobar comparamos con la solución exacta**:** $y(x) = x + x^2 + e^x$

$$y(0.1) = 1.2152$$

El error de truncamiento en cada paso está en el orden de los cienmilésimos, coincidiendo aproximadamente con $E=O(h^5)$. Los resultados tienen una precisión aceptable para la solución de problemas prácticos, por lo cual esta fórmula es muy utilizada

Instrumentación computacional de la fórmula de Runge-Kutta de cuarto orden

Una función en Python para obtener puntos de la solución de una EDO de primer orden con una fórmula equivalente a usar cinco términos de la Serie de Taylor. La función requiere especificar **f(x,y)**, el punto inicial **(x_0, y_0)** y los parámetros **h** (paso o distancia entre puntos), y **m** (cantidad de puntos).

La función entrega los vectores **u, v** conteniendo los puntos **x, y(x)**

Ecuación diferencial: **$y'(x) = f(x,y)$, $y(x_0) = y_0$, $x_0 \leq x \leq x_n$**

Fórmula de Runge_kutta:

$$K_1 = hf(x_i, y_i)$$
$$K_2 = hf(x_i + h/2, y_i + K_1/2)$$
$$K_3 = hf(x_i + h/2, y_i + K_2/2)$$
$$K_4 = hf(x_i + h, y_i + K_3)$$
$$y_{i+1} = y_i + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$

Instrumentación en Python

```
import numpy as np
def rk4(f,x,y,h,m):
    u=np.zeros([m,2],dtype=float)
    for i in range(m):
        k1=h*f(x,y)
        k2=h*f(x+h/2,y+k1/2)
        k3=h*f(x+h/2,y+k2/2)
        k4=h*f(x+h,y+k3)
        y=y+1/6*(k1+2*k2+2*k3+k4)
        x=x+h
        u[i,0]=x
        u[i,1]=y
    return u
```

Esta isntrumentación se generalizará posteriormente para resolver sistemas de ecuaciones dierenciales ordinarias de primer orden con condiciones en el inicio, usando un enfoque diferente.

Ejemplo. Calcule 20 puntos de la solución de la ecuación $y' - y - e^x + x^2 - 1 = 0$, $y(0) = 1$ espaciados en una distancia h = 0.1, usando la función **rk4**

Grafique y compare con la solución analítica.

Solución

$f(x, y) = y - x^2 + e^x + 1$
$x_0 = 0, y_0 = 1,$
$h = 0.1,$
$m = 20$ (cantidad de puntos)

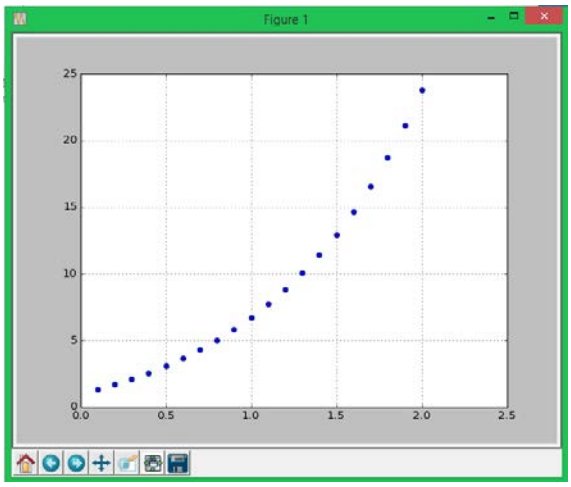
```
>>> import pylab as pl
>>> import numpy as np
>>> from rk4 import*
>>> def f(x,y):return y-x**2+np.exp(x)+1
>>> u=rk4(f,0,1,0.1,20)
>>> np.set_printoptions(precision=6)
>>> print(u)
[[  0.1        1.320517]
 [  0.2        1.68428 ]
 [  0.3        2.094956]
 [  0.4        2.556728]
 [  0.5        3.074358]
 [  0.6        3.653268]
 [  0.7        4.299623]
 [  0.8        5.020427]
 [  0.9        5.823636]
 [  1.         6.718274]
  .......etc.....

>>> pl.plot(u[:,0],u[:,1],'ob')
>>> pl.grid(True)
>>> pl.show()
```

Ecuación diferencial

Note el uso del identificador de librería para la función matemática

Tabla de resultados formateada

Gráfico de la solución con rk4



Para verificar precisión se usará como referencia un punto de la solución calculada. Un indicio del error de truncamiento es la diferencia entre la solución numérica y la solución analítica cuando x=1.

En la práctica no se dispone de la solución analítica, por lo que la estimación del error de truncamiento local se basa en el error de truncamiento de la serie de Taylor, asociado al método utilizado

Solución analítica:

```
y = exp(x)*(x + exp(-x) + 2*x*exp(-x) + x^2*exp(-x))
```

```
y(1) = 6.718281828459046
```

Solución numérica:

```
>>> u[9,1]
6.7182735390561872
```

Diferencia

```
0.000008289402859240624
```

NOTA. Actualmente se dispone de software computacional para encontrar la solución analítica para muchos problemas. En el caso de muchas ecuaciones diferenciales especialmente no lineales y de mayor orden, estas soluciones analíticas computacionales no se pueden obtener, por lo que los métodos numéricos siguen siendo una opción fácil de utilizar y analizar numéricamente mediante el parámetro h

9.2 Sistemas de ecuaciones diferenciales ordinarias de primer orden con condiciones en el inicio

Los métodos numéricos desarrollados para una ecuación diferencial ordinaria de primer orden pueden extenderse directamente a sistemas de ecuaciones diferenciales de primer orden.

Analizamos el caso de dos ecuaciones diferenciales ordinarias de primer orden simultáneas con condiciones en el inicio:

$$F(x, y, z, y') = 0, \quad y(x_0) = y_0$$
$$G(x, y, z, z') = 0, \quad z(x_0) = z_0$$

Se deben escribir en la notación adecuada para usar los métodos numéricos

$$y' = f(x, y, z), \quad y(x_0) = y_0$$
$$z' = g(x, y, z), \quad z(x_0) = z_0$$

9.2.1 Fórmula de Runge-Kutta de segundo orden extendida a sistemas de E. D. O. de primer orden

La fórmula de Runge-Kutta de segundo orden para un sistema de dos o más EDO's de primer orden con condiciones en el inicio, es una extensión directa de la fórmula para una EDO:

Definición: Fórmula de Runge-Kutta de segundo orden para un sistema de dos EDO de primer orden con condiciones en el inicio

$$K_{1,y} = hf(x_i, y_i, z_i)$$
$$K_{1,z} = hg(x_i, y_i, z_i)$$
$$K_{2,y} = hf(x_i + h, y_i + K_{1,y}, z_i + K_{1,z})$$
$$K_{2,z} = hg(x_i + h, y_i + K_{1,y}, z_i + K_{1,z})$$
$$y_{i+1} = y_i + \frac{1}{2}(K_{1,y} + K_{2,y})$$
$$z_{i+1} = z_i + \frac{1}{2}(K_{1,z} + K_{2,z})$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$
$$E = O(h^3), \quad x_i \leq z \leq x_{i+1}$$ (Error de truncamiento en cada paso)

Ejemplo. Obtenga dos puntos de la solución del siguiente sistema de ecuaciones diferenciales con la fórmula R-K de 2do. orden. Use **h = 0.1**

$$y' - x - y - z = 0,\ y(0) = 1$$
$$z' + x - y + z = 0,\ z(0) = 2$$

Solución

$$y' = f(x, y, z) = x + y + z,\quad x_0 = 0,\ y_0 = 1$$
$$z' = g(x, y, z) = -x + y - z,\quad x_0 = 0,\ z_0 = 2$$

Cálculo de dos puntos de la solución

i=0:

$$K_{1,y} = hf(x_0, y_0, z_0) = 0.1\, f(0, 1, 2) = 0.1\,(0 + 1 + 2) = 0.3$$
$$K_{1,z} = hg(x_0, y_0, z_0) = 0.1\, g(0, 1, 2) = 0.1\,(-0 + 1 - 2) = -0.1$$
$$K_{2,y} = hf(x_0+h, y_0+K_{1,y}, z_0+K_{1,z}) = 0.1\, f(0.1, 1.3, 1.9) = 0.1\,(0.1 + 1.3 + 1.9) = 0.33$$
$$K_{2,z} = hg(x_0+h, y_0+K_{1,y}, z_0+K_{1,z}) = 0.1g(0.1, 1.3, 1.9) = 0.1\,(-0.1 + 1.3 - 1.9) = -0.07$$
$$y_1 = y_0 + \frac{1}{2}(K_{1,y} + K_{2,y}) = 1 + 0.5(0.3 + 0.33) = 1.3150$$
$$z_1 = z_0 + \frac{1}{2}(K_{1,z} + K_{2,z}) = 2 + 0.5(-0.1 + (-0.07)) = 1.9150$$
$$x_1 = x_0 + h = 0 + 0.1 = 0.1$$

i=1:

$$K_{1,y} = hf(x_1, y_1, z_1) = 0.1\, f(0.1, 1.3150, 1.9150) = 0.333$$
$$K_{1,z} = hg(x_1, y_1, z_1) = 0.1\, g(0.1, 1.3150, 1.9150) = -0.07$$
$$K_{2,y} = hf(x_1+h, y_1+K_{1,y}, z_1+K_{1,z}) = 0.1\, f(0.2, 1.648, 1.845) = 0.3693$$
$$K_{2,z} = hg(x_1+h, y_1+K_{1,y}, z_1+K_{1,z}) = 0.1\, f(0.2, 1.648, 1.845) = -0.0397$$
$$y_2 = y_1 + \frac{1}{2}(K_{1,y} + K_{2,y}) = 1.6662$$
$$z_2 = z_1 + \frac{1}{2}(K_{1,z} + K_{2,z}) = 1.8602$$
$$x_2 = x_1 + h = 0.1 + 0.1 = 0.2$$

Para comprobar comparamos con la solución exacta

$$y(0.1) = 1.3160,\quad z(0.1) = 1.9150$$
$$y(0.2) = 1.6684,\quad z(0.2) = 1.8604$$

Los resultados calculados con la fórmula de Heun tienen al menos dos decimales exactos, y coinciden aproximadamente con $E=O(h^3)$

Instrumentación computacional de la fórmula de Runge-Kutta de segundo orden para resolver sistemas de E. D. O. de primer orden

La siguiente función usa recursos de manejo simbólico de la librería SymPy para instrumentar una generalización del método de Runge-Kutta de segundo orden para encontrar la solución numérica de una o mas ecuaciones diferenciales ordinarias de primer orden o ecuaciones diferenciales de mayor orden, con condiciones en el inicio.

La notación para describir las ecuaciones usa un formato distinto a las instrumentaciones anteriores

Definición de la función

```
rs = rk2n(F,V,U,h,m)
```

Parámetros de entrada

F: Vector con las definiciones de las ecuaciones diferenciales

V: Vector con las variables utilizadas. El primer componente debe ser la variable Independiente. Los nombres de variables pueden ser arbitrarios pero deben estar definidos como símbolos.

U: Vector con los valores iniciales.

h: Parámetro (distancia entre puntos de la variable independiente).

m: Cantidad de puntos de la solución requeridos.

Resultados

rs: Matriz en la cual la primera columna contiene las abscisa de los puntos de la solución y las otras columnas corresponden a los puntos de la solución para la otras variables en el orden en el que ingresaron.

```
#Runge-Kutta de segundo orden para n EDO - condiciones en el inicio
import sympy as sp
import numpy as np
def rk2n(F,V,U,h,m):
    nF=len(F)
    nV=len(V)
    K1=np.zeros([nF],dtype=sp.Symbol)
    K2=np.zeros([nF],dtype=sp.Symbol)
    rs=np.zeros([m,nV],dtype=float)
    T=list(np.copy(U))

    for p in range(m):
        for i in range(nF):
            K1[i]=F[i]
            K2[i]=F[i]
        for i in range(nF):
            for j in range(nV):
                K1[i]=K1[i].subs(V[j],float(T[j]))
            K1[i]=h*K1[i]
        for i in range(nF):
            K2[i]=K2[i].subs(V[0],float(T[0])+h)
            for j in range(1,nV):
                K2[i]=K2[i].subs(V[j],float(T[j])+K1[j-1])
            K2[i]=h*K2[i]

        T[0]=T[0]+h
        rs[p,0]=T[0]
        for i in range(nF):
            T[i+1]=T[i+1]+0.5*(K1[i]+K2[i])
            rs[p,i+1]=T[i+1]
    return rs
```

Ejemplo. Con la fórmula **rk2n** obtenga **20** puntos de la solución del siguiente sistema de ecuaciones diferenciales. Use **h = 0.1** Grafique y compare con la solución numérica de Python.

$$y' - x - y - z = 0, \quad y(0) = 1$$
$$z' + x - y + z = 0, \quad z(0) = 2$$

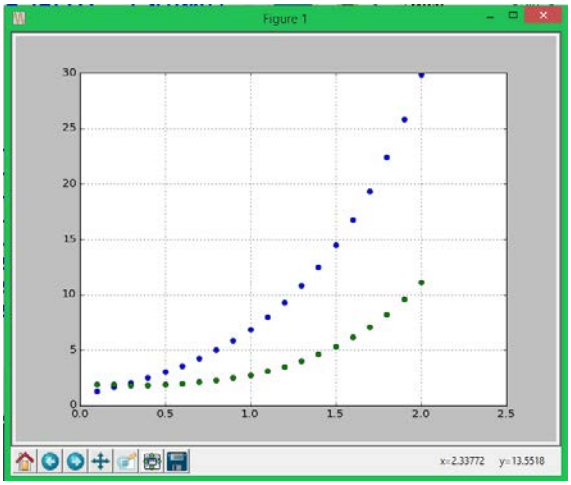
Solución

$$y' = f(x, y, z) = x + y + z, \quad x_0 = 0, y_0 = 1$$
$$z' = g(x, y, z) = -x + y - z, \quad x_0 = 0, z_0 = 2$$

```
>>> import sympy as sp
>>> import numpy as np
```

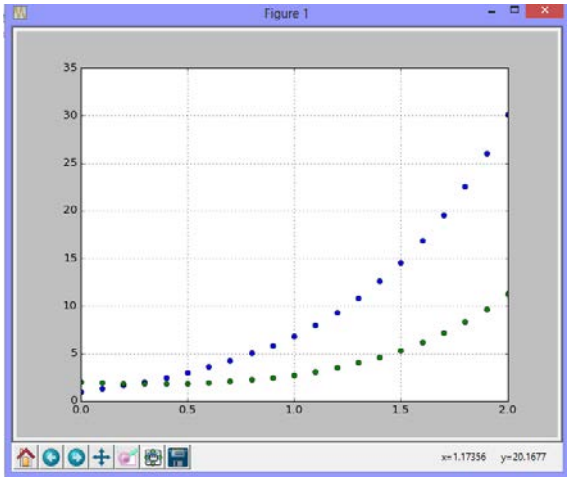
```
>>> import pylab as pl
>>> from rk2n import*
>>> x,y,z=sp.symbols('x,y,z')
>>> f=x+y+z
>>> g=-x+y-z
>>> rs=rk2n([f,g],[x,y,z],[0,1,2],0.1,20)
>>> np.set_printoptions(precision=6)
>>> print(rs)
[[  0.1        1.315      1.915    ]
 [  0.2        1.66615    1.86015 ]
 [  0.3        2.060442   1.836351]
 [  0.4        2.505725   1.845124]
 [  0.5        3.010867   1.888635]
 [  0.6        3.585926   1.969745]
 [  0.7        4.242353   2.092061]
 [  0.8        4.993218   2.26001 ]
 [  0.9        5.853473   2.478931]
 [  1.         6.840248   2.755175]
 [  1.1        7.973192   3.096234]
 [  1.2        9.274867   3.510892]
 [  1.3       10.771191   4.009398]
 [  1.4       12.491962   4.603672]
 [  1.5       14.471445   5.307537]
 [  1.6       16.749058   6.137004]
 [  1.7       19.370155   7.110579]
 [  1.8       22.38693    8.249642]
 [  1.9       25.859456   9.578868]
 [  2.        29.856883  11.126715]]

>>> pl.plot(rs[:,0],rs[:,1:3],'o')
>>> pl.grid(True)
>>> pl.show()
```

Gráfico de y(x), z(x)



Solución numérica con el método **Odeint** de la librería **SciPy** de **Python**

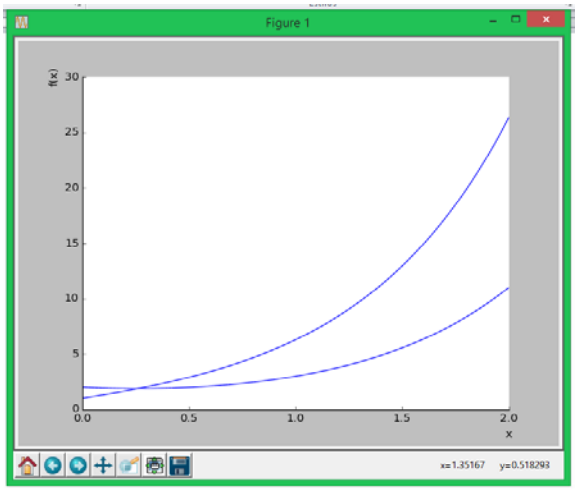
```
>>> from numpy import*
>>> import pylab as pl
>>> from scipy.integrate import odeint
>>> def fun(v0,x):                          Definición del sistema de EDO
        y=v0[0]
        z=v0[1]
        vec=[x+y+z,-x+y-z]
        return vec
>>> x=pl.arange(0,2.1,0.1)                  Variable independiente
>>> v0=[1,2]
>>> vsol=odeint(fun,v0,x)                   Vectores solución
>>> pl.plot(x,vsol,'o')                     Gráficos de los vectores
>>> pl.grid(True)
>>> pl.show()
```



NOTA. La notación utilizada en la instrumentación propuesta en este curso es más simple y general y permite experimentar con mayor claridad la obtención de la solución. Igualmente se puede usar para una o más ecuaciones diferenciales de primer orden, incluyendo ecuaciones no lineales, con condiciones en el inicio

Solución simbólica con el método **dsolve** de la librería **SymPy**

```
>>> from sympy import*
>>> x,y,z=symbols('x,y,z')
>>> dsolve([diff(y(x),x)-x-y(x)-z(x),diff(z(x),x)+x-y(x)+z(x)])
[y(x) == C1*exp(-sqrt(2)*x) + C2*exp(sqrt(2)*x), z(x) == C1*(-sqrt(2) -
1)*exp(-sqrt(2)*x) + C2*(-1 + sqrt(2))*exp(sqrt(2)*x)]
>>> C1,C2=symbols('C1,C2')
>>> y=C1*exp(-sqrt(2)*x) + C2*exp(sqrt(2)*x)
>>> z=C1*(-sqrt(2) - 1)*exp(-sqrt(2)*x) + C2*(-1 + sqrt(2))*exp(sqrt(2)*x)
>>> y.subs(x,0)                                    Condición inicial  y(0)=1
C1 + C2
>>> z.subs(x,0)                                    Condición inicial  z(0)=2
C1*(-sqrt(2) - 1) + C2*(-1 + sqrt(2))
>>> solve([C1+C2-1,C1*(-sqrt(2) - 1) + C2*(-1 + sqrt(2))-2])
{C2: 1/2 + 3*sqrt(2)/4, C1: -3*sqrt(2)/4 + 1/2}
>>> y=y.subs(C2,1/2 + 3*sqrt(2)/4).subs(C1,-3*sqrt(2)/4 + 1/2)
>>> z=z.subs(C2,1/2 + 3*sqrt(2)/4).subs(C1,-3*sqrt(2)/4 + 1/2)
>>> y=simplify(y)
>>> y
((2.0 + 3*sqrt(2))*exp(2*sqrt(2)*x) - 3*sqrt(2) + 2.0)*exp(-sqrt(2)*x)/4
>>> z=simplify(z)
>>> z
(-0.25*sqrt(2)*exp(2*sqrt(2)*x) + 1.0*exp(2*sqrt(2)*x) + 0.25*sqrt(2) +
1.0)*exp(-sqrt(2)*x)
>>> y.subs(x,0.1).evalf(6)
1.31102
>>> z.subs(x,0.1).evalf(6)
1.91970
>>> plot(y,z,(x,0,2))
```



Se observa alguna diferencia con la solución simbólica obtenida con otros programas computacionales

9.2.2 Fórmula de Runge-Kutta de cuarto orden para sistemas de EDO de primer orden y condiciones en el inicio

La fórmulas de Runge-Kutta de cuarto orden igualmente se pueden extender a sistemas de dos o más EDO's de primer orden con condiciones en el inicio.

Analizamos el caso de dos ecuaciones diferenciales ordinarias de primer orden con condiciones en el inicio, en las que **y'(x)** y **z'(x)** aparecen en forma explícita

$$F(x, y, z, y') = 0, \quad y(x_0) = y_0$$
$$G(x, y, z, z') = 0, \quad z(x_0) = z_0$$

Se pueden escribir en la notación para uso de los métodos numéricos

$$y' = f(x, y, z), \quad y(x_0) = y_0$$
$$z' = g(x, y, z), \quad z(x_0) = z_0$$

Definición: Fórmula de Runge-Kutta de cuarto orden para un sistema de dos EDO de primer orden con condiciones en el inicio

$$K_{1,y} = hf(x_i, y_i, z_i)$$
$$K_{1,z} = hg(x_i, y_i, z_i)$$
$$K_{2,y} = hf(x_i + h/2, y_i + K_{1,y}/2, z_i + K_{1,z}/2)$$
$$K_{2,z} = hg(x_i + h/2, y_i + K_{1,y}/2, z_i + K_{1,z}/2)$$
$$K_{3,y} = hf(x_i + h/2, y_i + K_{2,y}/2, z_i + K_{2,z}/2)$$
$$K_{3,z} = hg(x_i + h/2, y_i + K_{2,y}/2, z_i + K_{2,z}/2)$$
$$K_{4,y} = hf(x_i + h, y_i + K_{3,y}, z_i + K_{3,z})$$
$$K_{4,z} = hg(x_i + h, y_i + K_{3,y}, z_i + K_{3,z})$$
$$y_{i+1} = y_i + \frac{1}{6}(K_{1,y} + 2K_{2,y} + 2K_{3,y} + K_{4,y})$$
$$z_{i+1} = z_i + \frac{1}{6}(K_{1,z} + 2K_{2,z} + 2K_{3,z} + K_{4,z})$$
$$x_{i+1} = x_i + h, \quad i = 0, 1, 2, \ldots$$
$$E = O(h^5), \quad x_i \le z \le x_{i+1}$$ (Error de truncamiento en cada paso)

Instrumentación computacional de la fórmula de Runge-Kutta de cuarto orden para resolver sistemas de E. D. O. de primer orden

La siguiente función usa recursos de la librería simbólica **SymPy** de Python para instrumentar una generalización del método de Runge-Kutta de cuarto orden para encontrar la solución numérica de una o mas ecuaciones diferenciales ordinarias simultáneas explícitas de primer orden o ecuaciones diferenciales de mayor orden, con condiciones en el inicio.

La notación para describir las ecuaciones usa el formato de la función rk2n.

Definición de la función

```
rs = rk4n(F,V,U,h,m)
```

Parámetros de entrada

F: Vector con las definiciones de las ecuaciones diferenciales.

V: Vector con las variables utilizadas. El primer componente debe ser la variable Independiente. Los nombres de variables pueden ser arbitrarios pero deben estar definidos como símbolos.

U: Vector con los valores iniciales.

h: Parámetro (distancia entre puntos de la variable independiente).

m: Cantidad de puntos de la solución requeridos.

Resultados

rs: Matriz en la cual la primera columna contiene las abscisa de los puntos de la solución y las otras columnas corresponden a los puntos de la solución para la otras variables en el orden en el que ingresaron.

```
#Runge Kutta de cuarto orden para n EDO's
import sympy as sp
import numpy as np
def rk4n(F,V,U,h,m):
    nF=len(F)
    nV=len(V)
    K1=np.zeros([nF],dtype=sp.Symbol)
    K2=np.zeros([nF],dtype=sp.Symbol)
    K3=np.zeros([nF],dtype=sp.Symbol)
    K4=np.zeros([nF],dtype=sp.Symbol)
    rs=np.zeros([m,nV],dtype=float)
    T=list(np.copy(U))

    for p in range(m):
        for i in range(nF):
            K1[i]=F[i]
            K2[i]=F[i]
            K3[i]=F[i]
            K4[i]=F[i]
        for i in range(nF):
            for j in range(nV):
                K1[i]=K1[i].subs(V[j],float(T[j]))
            K1[i]=h*K1[i]
        for i in range(nF):
            K2[i]=K2[i].subs(V[0],float(T[0])+h/2)
            for j in range(1,nV):
                K2[i]=K2[i].subs(V[j],float(T[j])+K1[j-1]/2)
            K2[i]=h*K2[i]
        for i in range(nF):
            K3[i]=K3[i].subs(V[0],float(T[0])+h/2)
            for j in range(1,nV):
                K3[i]=K3[i].subs(V[j],float(T[j])+K2[j-1]/2)
            K3[i]=h*K3[i]
        for i in range(nF):
            K4[i]=K4[i].subs(V[0],float(T[0])+h)
            for j in range(1,nV):
                K4[i]=K4[i].subs(V[j],float(T[j])+K3[j-1])
            K4[i]=h*K4[i]
        T[0]=T[0]+h
        rs[p,0]=T[0]
        for i in range(nF):
            T[i+1]=T[i+1]+1/6*(K1[i]+2*K2[i]+2*K3[i]+K4[i])
            rs[p,i+1]=T[i+1]
    return rs
```

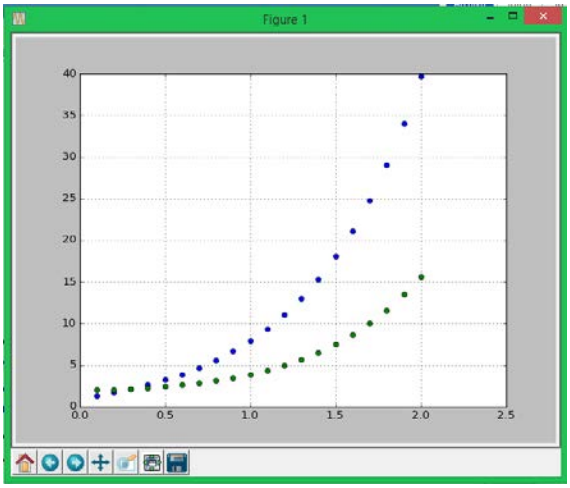
Ejemplo. Con la función **rk4n** obtenga **20** puntos de la solución del siguiente sistema de ecuaciones diferenciales. Use **h = 0.1** Grafique y compare con la solución numérica de Python.

$$y' - e^x - y - z + 1 = 0, \quad y(0) = 1$$
$$z' - y - \cos(x) + z = 0, \quad z(0) = 2$$

Solución

$$y' = f(x, y, z) = e^x + y + z - 1, \qquad x_0 = 0, y_0 = 1$$
$$z' = g(x, y, z) = y + \cos(x) - z, \qquad x_0 = 0, z_0 = 2$$

```
>>> import sympy as sp
>>> import numpy as np
>>> import pylab as pl
>>> from rk4n import*
>>> x,y,z=sp.symbols('x,y,z')
>>> f=sp.exp(x)+y+z-1                          Ecuaciones diferenciales
>>> g=y+sp.cos(x)-z                            Note el uso del identificador de librería
>>> rs=rk4n([f,g],[x,y,z],[0,1,2],0.1,20)
>>> np.set_printoptions(precision=6)
>>> print(rs)
[[  0.1        1.321371   2.015033]           Puntos de la solución formateados
 [  0.2        1.691314   2.060539]
 [  0.3        2.119421   2.137756]
 [  0.4        2.616595   2.248794]
 [  0.5        3.195301   2.396707]
 [  0.6        3.869843   2.585564]
 [  0.7        4.656686   2.820557]
 [  0.8        5.574832   3.108124]
   ......etc.......
>>> pl.plot(rs[:,0],rs[:,1:3],'o')                    Gráfico de y(x), z(x)
>>> pl.grid(True)
>>> pl.show()
```



Los métodos **rk2n** y **rk4n** pueden ser usados con una o más ecuaciones diferenciales ordinarias de primer orden con condiciones en el inicio, y también con ecuaciones diferenciales ordinarias de mayor orden, con condiciones en el inicio, que se puedan reducir a sistemas de ecuaciones diferenciales de primer orden. La notación puede ser diferente.

Librerías comunes

```
>>> import sympy as sp
>>> import numpy as np
>>> import pylab as pl
>>> from rk4n import*
```

Una EDO

```
>>> x,y=sp.symbols('x,y')
>>> f=y-x**2+x+1
>>> rs=rk4n([f],[x,y],[0,1],0.1,20)
>>> np.set_printoptions(precision=6)
>>> print(rs)
>>> pl.plot(rs[:,0],rs[:,1],'o')
>>> pl.grid(True)
>>> pl.show()
```

Dos EDO's

```
>>> x,y,z=sp.symbols('x,y,z')
>>> f=x+y+z
>>> g=-x+y-z
>>> rs=rk4n([f,g],[x,y,z],[0,1,2],0.1,20)
>>> np.set_printoptions(precision=6)
>>> print(rs)
>>> pl.plot(rs[:,0],rs[:,1:3],'o')
>>> pl.grid(True)
>>> pl.show()
```

Tres EDO's

```
>>> x,y,z,t=sp.symbols('x,y,z,t')
>>> f=x-y-z+t
>>> g=x-y+z-t-1
>>> d=x+y+z+t+2
>>> rs=rk4n([f,g,d],[x,y,z,t],[0,1,2,3],0.1,20)
>>> np.set_printoptions(precision=6)
>>> print(rs)
>>> pl.plot(rs[:,0],rs[:,1:4],'o')
>>> pl.grid(True)
>>> pl.show()
```

9.3 Ecuaciones diferenciales ordinarias de mayor orden y condiciones en el inicio

Mediante sustituciones estas ecuaciones se transforman en sistemas de ecuaciones diferenciales ordinarias de primer orden con condiciones en el inicio y se aplican los métodos numéricos como en la sección anterior.

Analizamos el caso de una ecuación diferencial ordinaria de segundo orden con condiciones en el inicio, en la que **y'(x)** y **y''(x)** aparecen en forma explícita

$$G(x, y, y', y'') = 0, \quad y(x_0) = y_0, \quad y'(x_0) = y'_0$$

Mediante la sustitución

$$z = y'$$

Se tiene

$$G(x, y, z, z') = 0$$

Se puede escribir como un sistema de dos ecuaciones diferenciales de primer orden siguiendo la notación anterior:

$$y' = f(x, y, z) = z$$
$$z' = g(x, y, z)$$ expresión que se obtiene despejando **z'** de **G**

Con las condiciones iniciales

$$y(x_0) = y_0$$
$$z(x_0) = y'_0 = z_0$$

Es un sistema de dos ecuaciones diferenciales de primer orden con condiciones en el inicio.

Ejemplo. Calcule un punto de la solución de la siguiente ecuación diferencial de segundo orden con condiciones en el inicio, con la fórmula de Runge-Kutta de cuarto orden, **h = 0.1**

$$y'' - y' - x + y + 1 = 0, \quad y(0) = 1, \quad y'(0) = 2$$

Solución

Mediante l**a sustitución z = y' se obtiene**

$$z' - z - x + y + 1 = 0$$

Constituyen un sistema de dos ecuaciones diferenciales de primer orden que se puede escribir

$$y' = f(x,y,z) = z, \quad y(0) = 1$$
$$z' = g(x,y,z) = x - y + z - 1, \quad z(0) = 2$$

Cálculo de los puntos de la solución

i=0:

$x_0 = 0, y_0 = 1, z_0 = 2$

$K_{1,y} = hf(x_0, y_0, z_0) = 0.1\ f(0, 1, 2) = 0.1\ (2) = 0.2$
$K_{1,z} = hg(x_0, y_0, z_0) = 0.1\ g(0, 1, 2) = 0.1\ (0 - 1 + 2 - 1) = 0$
$K_{2,y} = hf(x_0 + h/2, y_0 + K_{1,y}/2, z_0 + K_{1,z}/2) = 0.1f(0.05, 1.1, 2) = 0.1(\ 2) = 0.2$
$K_{2,z} = hgx_0 + h/2, y_0 + K_{1,y}/2, z_0 + K_{1,z}/2) = 0.1g(0.05, 1.1, 2) = 0.1(0.05-1.1+2-1) = -0.005$
$K_{3,y} = hf(x_0 + h/2, y_0 + K_{2,y}/2, z_0 + K_{2,z}/2) = 0.1\ f(\ 0.05, 1.1, 1.9975) = 0.1998$
$K_{3,z} = hg(x_0 + h/2, y_0 + K_{2,y}/2, z_0 + K_{2,z}/2) = 0.1\ g(\ 0.05, 1.1, 1.9975) = -0.0052$
$K_{4,y} = hf(x_0 + h, y_0 + K_{3,y}, z_0 + K_{3,z}) = 0.1\ f(0.1, 1.1998, 1.9948) = 0.1995$
$K_{4,z} = hgx_0 + h, y_0 + K_{3,y}, z_0 + K_{3,z}) = 0.1\ g0.1, 1.1998, 1.9948) = -0.0105$

$$y_1 = y_0 + \frac{1}{6}(K_{1,y} + 2K_{2,y} + 2K_{3,y} + K_{4,y}) = 1 + \frac{1}{6}[0.2+2(0.2)+2(0.1998)+0.1995] = 1.1998$$

$$z_1 = z_0 + \frac{1}{6}(K_{1,z} + 2K_{2,z} + 2K_{3,z} + K_{4,z}) = 2 + \frac{1}{6}[0+2(-0.005)+2(-0.0052)-0.0105]=1.9948$$

$x_1 = x_0 + h = 0 + 0.1 = 0.1$

Instrumentación computacional

Al transformar la ecuación diferencial de segundo orden a un sistema de ecuaciones diferenciales de primer orden se pueden usar las mismas funciones desarrolladas anteriormente. Para el ejemplo anterior se usará la función **rk4n**

$$y'' - y' - x + y + 1 = 0,\ y(0) = 1,\ y'(0) = 2$$

Solución

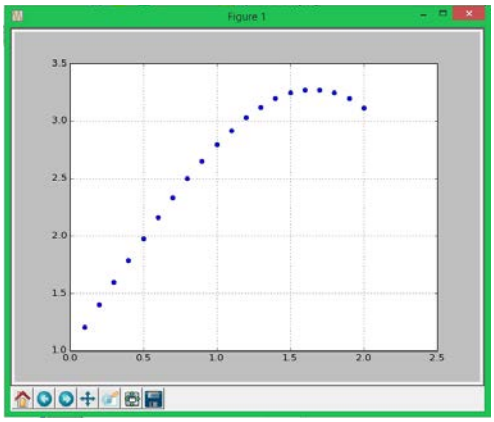
Mediante la sustitución **z = y'** se obtiene

$$z' - z - x + y + 1 = 0$$

Constituyen un sistema de dos ecuaciones diferenciales de primer orden que se puede escribir

$$y' = f(x,y,z) = z, \qquad y(0) = 1$$
$$z' = g(x,y,z) = x - y + z - 1, \qquad z(0) = 2$$

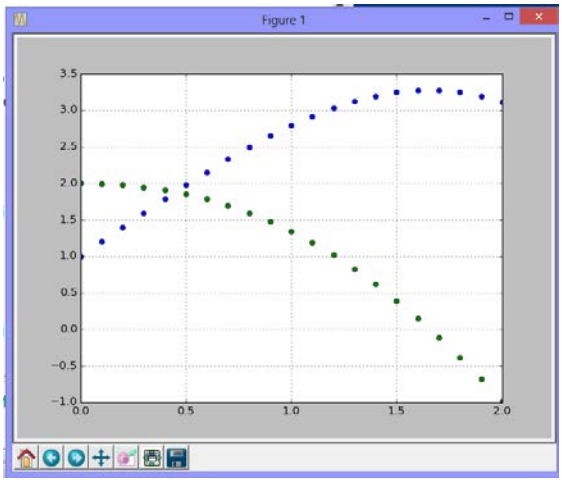
```
>>> import sympy as sp
>>> import numpy as np
>>> import pylab as pl
>>> from rk4n import*
>>> x,y,z=sp.symbols('x,y,z')
>>> f=z                                          Ecuaciones diferenciales
>>> g=x-y+z-1
>>> rs=rk4n([f,g],[x,y,z],[0,1,2],0.1,20)        Puntos de la solución
>>> np.set_printoptions(precision=10)
>>> print(rs)
[[ 0.1      1.1998291667  1.9948333333]
 [ 0.2      1.3986000708  1.9786692361]
 [ 0.3      1.5951634958  1.9505209571]
 [ 0.4      1.7882725577  1.9094239016]
 [ 0.5      1.9765855366  1.8544480978]
 [ 0.6      2.1586700014  1.7847116104]
 [ 0.7      2.3330083225  1.6993948609]
 [ 0.8      2.4980046591  1.5977557973]
  .....etc.....
>>> pl.plot(rs[:,0],rs[:,1],'ob')                Gráfico de y(x)
>>> pl.grid(True)
>>> pl.show()
```



Solución numérica de la ecuación diferencial de segundo orden del ejemplo anterior transformada a dos de primer orden, con el método **Odeint** de la librería **Scipy** de **Python**

y' = f(x,y,z) = z, **y(0) = 1**
z' = g(x,y,z) = x – y + z – 1, **z(0) = 2**

```
>>> from numpy import*
>>> import pylab as pl
>>> from scipy.integrate import odeint
>>> def fun(v0,x):                          Definición del sistema de EDO
        y=v0[0]
        z=v0[1]
        vec=[z, x-y+z-1]
        return vec
>>> x=pl.arange(0,2.1,0.1)                  Variable independiente  x = 0, 0.1,..2.0
>>> v0=[1,2]
>>> vsol=odeint(fun,v0,x)                   Vectores solución
>>> pl.plot(x,vsol,'o')                     Gráficos de los vectores
>>> pl.grid(True)
>>> pl.show()
```



```
>>> shape(vsol)                          El arreglo vsol contiene dos columnas
(21, 2)                                  de 21 elementos. La primera es y(x)
>>> print(vsol)
[[ 1.        ,  2.        ],            Fila 0, columnas 0 y 1
 [ 1.19982918,  1.99483344],            Fila 1, columnas 0 y 1
 [ 1.39860011,  1.97866944],
 [ 1.59516358,  1.95052131],
 . . . . . . . . . . .
 [ 3.24934419, -0.39055822],
 [ 3.19585332, -0.68155148],
 [ 3.11260243, -0.98547847]]             Fila 20, columnas 0 y 1
```

La solución coincide con la obtenida con el método rk4n pero la istrumentación desarrollada en rk4n es más simple de usar y los resultados más fáciles de interpretar. También se puede constatar que la solución calculada con el método rk4n es ligeramente más exacta:

Solución simbólica de la ecuación diferencial de segundo orden del ejemplo anterior con el método **dsolve** de la librería simbólica **SymPy** de **Python**

$$y'' - y' - x + y + 1 = 0,\ y(0) = 1,\ y'(0) = 2$$

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> dsolve(diff(diff(y(x),x))-diff(y(x),x)-x+y(x)+1)
y(x) == x + (C1*sin(sqrt(3)*x/2) + C2*cos(sqrt(3)*x/2))*sqrt(exp(x))

>>> C1,C2=symbols('C1,C2')
>>> y=x + (C1*sin(sqrt(3)*x/2) + C2*cos(sqrt(3)*x/2))*sqrt(exp(x))
>>> y.subs(x,0)                          Condición inicial y(0)=1
C2
>>> dy=diff(y,x)
>>> dy.subs(x,0)                         Condición inicial y'(0)=2
sqrt(3)*C1/2 + C2/2 + 1
```

```
y(0) = 1    ⇒    C2 = 1
y'(0) = 2   ⇒    sqrt(3)*C1/2 + C2/2 + 1 = 2   ⇒   C1 = sqrt(3)/3
```

De donde se puede obtener la solución simbólica exacta explícita

```
>>> y=y.subs(C2,1).subs(C1,sqrt(3)/3)
>>> y
x + (sqrt(3)*sin(sqrt(3)*x/2)/3 + cos(sqrt(3)*x/2))*sqrt(exp(x))
>>> y=simplify(y)
>>> y
x + 2*sqrt(3)*exp(x/2)*sin(sqrt(3)*x/2 + pi/3)/3
```

La solución simbólica escrita en la notación común:

$$y(x) = x + \frac{2}{3}\sqrt{3}\, e^{x/2} sen(\frac{\sqrt{3}}{2}x + \frac{\pi}{3})$$

Comparación de resultados numéricos y analíticos

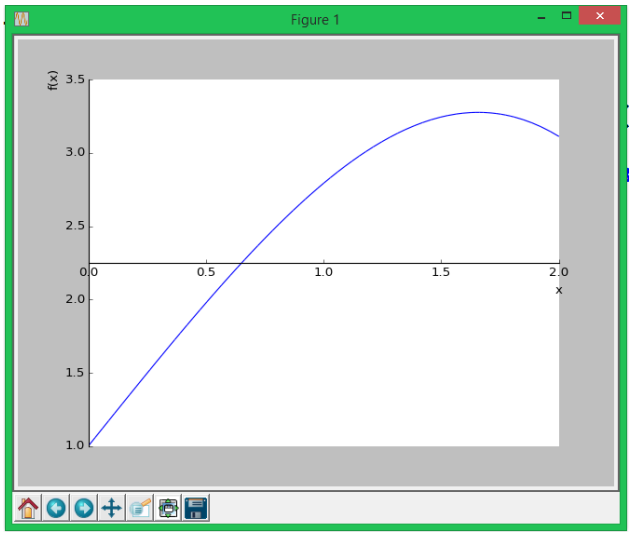
```
>>> y.subs(x,0.1).evalf(10)
1.199829168
```

Evaluar **y(0.1)**

Gráfico de la solución en el intervalo [0,2]

```
>>> plot(y,(x,0,2))
```

Función **plot** de la librería **SymPy**



Ejemplo. Obtenga la solución simbòlica con el método **dsolve** de SymPy para la siguiente ecuación diferencial: **$y'' + y' - y + 2x^2 - 3x - 4 = 0$, $0 \leq x \leq 2$, $y(0) = 1$, $y'(0) = 1$**

```
>>> from sympy import*
>>> x,y=symbols('x,y')
>>> dsolve(diff(diff(y(x),x))+diff(y(x),x)-y(x)+2*x**2-3*x-4)
y(x) == C1*exp(x*(-1 + sqrt(5))/2) + C2*exp(x*(-sqrt(5) - 1)/2) + 2*x**2 +
x + 1
>>> C1,C2=symbols('C1,C2')
>>> y=C1*exp(x*(-1 + sqrt(5))/2) + C2*exp(x*(-sqrt(5) - 1)/2) + 2*x**2 + x
+ 1
>>> y.subs(x,0)
C1 + C2 + 1
>>> dy=diff(y,x)
>>> dy.subs(x,0)
C1*(-1/2 + sqrt(5)/2) + C2*(-sqrt(5)/2 - 1/2) + 1
```

Condición inicial **y(0)=1**

Condición inicial **y'(0)=1**

Resolver el sistema **C1 + C2 + 1 = 1, C1*(-1/2 + sqrt(5)/2) + C2*(-sqrt(5)/2 - 1/2) + 1 = 1**

```
>>> solve([C1+C2, C1*(-1/2 + sqrt(5)/2) + C2*(-sqrt(5)/2 - 1/2)])
{C1: 0.0, C2: 0.0}
>>> y.subs(C1,0).subs(C2,0)
2*x**2 + x + 1                                    Solución simbólica
```

9.4 Ecuaciones diferenciales ordinarias no lineales

Los métodos numéricos pueden aplicarse igualmente para calcular la solución aproximada de ecuaciones diferenciales ordinarias no lineales, para las cuales no es posible o pudiese ser muy laborioso obtener la solución analítica. En los ejemplos anteriores también se probaron algunas ecuaciones diferenciales no lineales

Ejemplo. Obtenga numéricamente la solución de la ecuación

$$y'' + yy' - x + y - 3 = 0,\ y(0) = 1,\ y'(0) = 2,\ 0 \le x \le 2$$

Mediante la sustitución **z = y',** se obtiene: $z' + yz - x + y - 3 = 0$

Es un sistema de dos ecuaciones diferenciales de primer orden que se puede escribir

$$y' = f(x,y,z) = z,\ y(0) = 1$$
$$z' = g(x,y,z) = x - y - yz + 3,\ z(0) = 2$$

Solución con el método de Runge-Kutta de cuarto orden

```
>>> import sympy as sp
>>> import numpy as np
>>> from rk4n import*
>>> x,y,z=sp.symbols('x,y,z')
>>> f=z                                        Ecuaciones diferenciales
>>> g=x-y-y*z+3
>>> rs=rk4n([f,g],[x,y,z],[0,1,2],0.1,5)       Puntos de la solución
>>> np.set_printoptions(precision=6)
>>> print(rs)
[[ 0.1       1.19919   1.975993]               Solución formateada
 [ 0.2       1.393762  1.909031]
 [ 0.3       1.579876  1.808547]
 [ 0.4       1.75471   1.685216]
 [ 0.5       1.91651   1.54954 ]]
```

Nota. Los métodos simbólicos computacionales no pueden encontrar una solución para esta ecuación diferencial no lineal. Los métodos numéricos son una opción importante para explorar y analizar la solución.

Solución con el método numérico **Odeint** de **Python** para sistemas de EDO's

```
>>> from numpy import*
>>> from scipy.integrate import odeint

>>> def fun(v0,x):                          Definición del sistema de EDO
        y=v0[0]
        z=v0[1]
        vec=[z, x-y-y*z+3]
        return vec
>>> x=arange(0,0.6,0.1)                     Variable independiente
>>> v0=[1,2]
>>> vsol=odeint(fun,v0,x)                   Vectores solución

>>> print(x)
[ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5]
>>> print(vsol)
[[ 1.        ,  2.        ],
 [ 1.19919072,  1.97599116],
 [ 1.3937634 ,  1.90902847],
 [ 1.57987764,  1.80854407],
 [ 1.75471089,  1.68521327],
 [ 1.91651026,  1.54953843]]
```

9.5 Convergencia y estabilidad numérica

Los métodos numéricos que se utilizan para resolver una ecuación diferencial, como se muestra en los ejemplos anteriores, proporcionan una solución discreta aproximada.

Para algunas ecuaciones diferenciales ordinarias, únicamente se tiene esta solución discreta por lo que es de interés verificar de alguna manera su existencia y convergencia.

La convergencia numérica puede hacerse variando el parámetro **h** del método numérico seleccionado y cuantificando la tendencia de algunos puntos de control de la solución calculada

Un indicio de la existencia de la solución se puede observar en los resultados gráficos y numéricos verificando que no contenga puntos singulares.

Adicionalmente, es importante verificar si la solución obtenida es muy sensible a los errores en la formulación de la ecuación diferencial o en la condición inicial. Se puede detectar esta situación calculando numéricamente el problema original y el problema con alguna perturbación. Si la solución cambia significativamente puede interpretarse que el problema no está bien planteado en el sentido que la solución es muy sensible a los errores en los datos..

9.6 Ecuaciones diferenciales ordinarias con condiciones en los bordes

En esta sección revisaremos los métodos numéricos para resolver ecuaciones diferenciales ordinarias para las cuales se proporcionan condiciones iniciales en los bordes, siendo de interés conocer la solución en el interior de esta región, como en el siguiente ejemplo:

$$y'' - y' + y - 2e^x - 3 = 0,\ y(0) = 1,\ y(1) = 5,\ 0 \le x \le 1$$

9.6.1 Método de prueba y error (método del disparo)

Una opción para obtener la solución numérica consiste en realizar varios intentos suponiendo una condición adicional en el inicio para poder usar los métodos vistos anteriormente. Para el ejemplo anterior probamos:

$$y'' - y' + y - 2e^x - 3 = 0,\ y(0) = 1,\ y'(0) = 1,\ 0 \le x \le 1$$

Esta es ahora una ecuación diferencial de segundo orden con condiciones en el inicio, la cual se puede re-escribir como dos ecuaciones diferenciales de primer orden:

$$y' = f(x,y,z) = z,\quad y(0) = 1$$
$$z' = g(x,y,z) = 2e^x - y + z + 3,\quad z(0) = 1$$

Aquí se puede aplicar alguno de los métodos estudiados (Heun, Runge-Kutta). El cálculo debe continuar hasta llegar al otro extremo del intervalo de interés. Se debe comparar el resultado obtenido en el extremo derecho con el dato dado para ese borde: **y(1) = 5**.

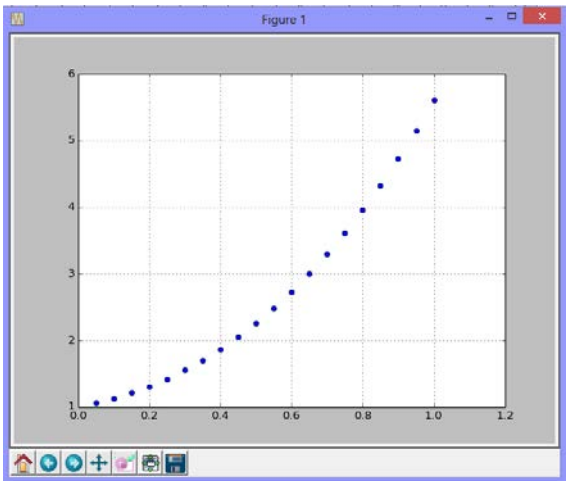
Esto permite suponer un mejor valor para la condición inicial supuesta y volver a calcular todo nuevamente. Este procedimiento se puede repetir varias veces.

Para el ejemplo se utilizó el método de Runge-Kutta de cuarto orden probando los valores iniciales **z(0) = y'(0) = 1, 0.5**. La figura muestra los resultados del primer intento

Usamos el método **rk4n**. Se calculan 20 puntos de la solución espaciados en **h = 0.05**

```
>>> import sympy as sp
>>> import numpy as np
>>> import pylab as pl
>>> from rk4n import*
>>> x,y,z=sp.symbols('x,y,z')
>>> f=z
>>> g=2*sp.exp(x)-y+z+3
>>> rs=rk4n([f,g],[x,y,z],[0,1,1],0.05,20)
>>> np.set_printoptions(precision=6)
>>> print(rs)
[[ 0.05      1.056376  1.257562]
 [ 0.1       1.126012  1.530496]
```

```
 [ 0.15      1.209688  1.819165]
      ... etc....
 [ 0.9       4.729771  8.258601]
 [ 0.95      5.15718   8.841171]
 [ 1.        5.614229  9.444205]]

>>> pl.plot(rs[:,0],rs[:,1],'ob')
>>> pl.grid(True)
>>> pl.show()
```



En este primer intento el valor que se obtiene para **y(x)** en el extremo **x = 1** es `5.614229`

Este valor es mayor a la condición dada en el extremo derecho: **y(1) = 5**, por lo tanto, en el siguiente intento reducimos el valor de **z(0) = y'(0)** a **0.5**:

```
>>> rs=rk4n([f,g],[x,y,z],[0,1,0.5],0.05,20)
```

En el extremo derecho se obtiene un valor menor al dato dado en la condición: `4.889117`

Para sistematizar la elección del valor inicial en el siguiente intento es preferible usar una interpolación para asignar el valor a **y'(0)**:

Sean y'_a, y'_b valores elegidos para **y'(0)** en dos intentos realizados
y_a, y_b valores obtenidos para **y** en el extremo derecho del intervalo, en los intentos
y_n valor suministrado como dato para el extremo derecho del intervalo
y'_0 nuevo valor corregido para **y'(0)** para realizar un nuevo intento

Usamos una recta para predecir el valor para y'_0:

$$y_b - y_n = \frac{y_b - y_a}{y'_b - y'_a}(y'_b - y'_0) \Rightarrow y'_0 = y'_b - \frac{y'_b - y'_a}{y_b - y_a}(y_b - y_n)$$

Para el ejemplo anterior, se tienen:

$\mathbf{y'_a = 1}$
$\mathbf{y'_b = 0.5}$
$\mathbf{y_a = 5.614229}$ (valor obtenido en el extremo derecho, con $\mathbf{y'_a = 1}$)
$\mathbf{y_b = 4.889117}$ (valor obtenido en el extremo derecho, con $\mathbf{y'_b = 0.5}$)
$\mathbf{y_n = 5}$ (dato)

Con lo que resulta

$$\mathbf{y'_0 = 0.5 - \frac{0.5 - 1}{4.889117 - 5.614229}(4.889117 - 5) = 0.5765}$$

Al realizar el tercer intento con este valor de **y'(0)** se comprueba que la solución calculada está muy cerca del valor dado en la condición. Se puede verificar que el punto final calculado $\mathbf{y(x_n)}$ coincide en cinco decimales con el dato suministrado **y(1) = 5** comparando con la solución analítica

```
>>> rs=rk4n([f,g],[x,y,z],[0,1,0.5765],0.05,20)
```

Se obtiene para x=1: y(1) = **5.000059**

Este método también se puede usar para resolver ecuaciones diferenciales ordinarias **no lineales** con condiciones en los bordes.

9.6.2 Método de diferencias finitas

Este es un enfoque más general para resolver ecuaciones diferenciales ordinarias con condiciones en los bordes. Consiste en sustituir las derivadas por aproximaciones de diferencias finitas. La ecuación resultante se denomina ecuación de diferencias y puede resolverse por métodos algebraicos.

El método de diferencias finitas discretiza el dominio continuo de **y(x)** a puntos x_i en el eje horizontal para los cuales se calcula la solución:

$$y(x),\ a \le x \le b \quad \Rightarrow \quad y(x_i),\ i = 0, 1, ..., n$$

Es importante usar en la sustitución aproximaciones del mismo orden para las derivadas, de tal manera que la ecuación de diferencias tenga consistencia en el error de truncamiento.

Aproximaciones de diferencias finitas de segundo orden $O(h^2)$ típicas:

$$y'_i = \frac{y_{i+1} - y_{i-1}}{2h} + O(h^2)$$

$$y''_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + O(h^2)$$

Ejemplo. Sustituya las derivadas por diferencias finitas en la EDO del ejemplo anterior

$$y'' - y' + y - 2e^x - 3 = 0,\ y(0) = 1, y(1) = 5$$

Solución

La sustitución convierte la ecuación original en una ecuación "discretizada":

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} - \frac{y_{i+1} - y_{i-1}}{2h} + y_i - 2e^{x_i} - 3 = 0,\ i = 1, 2, ..., n-1$$

Siendo **n** la cantidad de divisiones espaciadas en **h** en que se ha dividido el intervalo $0 \le x \le 1$

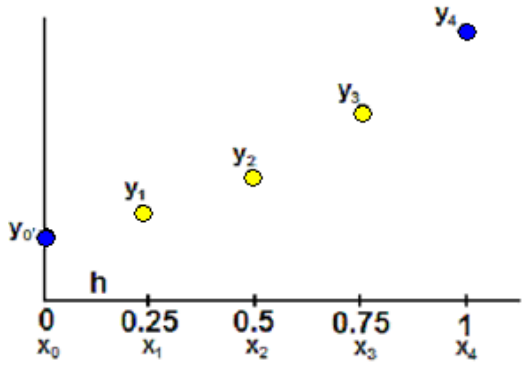
La ecuación resultante se denomina ecuación de diferencias con error de truncamiento $O(h^2)$.

Esta ecuación es consistente pues su límite, cuando $h \to 0$ es la ecuación diferencial original. Se espera que la solución calculada en los puntos, también tenga este comportamiento.

Para facilitar los cálculos es conveniente expresar la ecuación de diferencias en forma estándar agrupando términos:

$$(2+h)y_{i-1} + (2h^2-4)y_i + (2-h)y_{i+1} = 4h^2 e^{x_i} + 6h^2,\ i = 1, 2, ..., n-1$$

Para describir el uso de esta ecuación, supondremos que **h=0.25.** En la realidad debería ser más pequeño para que el error de truncamiento se reduzca. Con este valor de **h** el problema se puede visualizar de la siguiente manera



$y_0 = y(0) = 1$ (dato en el borde izquierdo)
$y_4 = y(1) = 5$ (dato en el borde derecho)
y_1, y_2, y_3 (son los puntos que se calcularán)

A continuación se aplica la ecuación de diferencias en los puntos especificados

$$(2+h)y_{i-1} + (2h^2-4)y_i + (2-h)y_{i+1} = 4h^2 e^{x_i} + 6h^2,\ i = 1, 2, 3$$

i=1: $(2+0.25)y_0 + (2(0.25^2)-4)y_1 + (2-0.25)y_2 = 4(0.25^2)\ e^{0.25} + 6(0.25^2)$

$$-3.875y_1 + 1.75y_2 = -1.5540$$

i=2: $(2+0.25)y_1 + (2(0.25^2)-4)y_2 + (2-0.25)y_3 = 4(0.25^2)\ e^{0.5} + 6(0.25^2)$

$$2.25y_1 - 3.875y_2 + 1.75y_3 = 0.7872$$

i=3: $(2+0.25)y_2 + (2(0.25^2)-4)y_3 + (2-0.25)y_4 = 4(0.25^2)\ e^{0.75} + 6(0.25^2)$

$$2.25y_2 - 3.875y_3 = -7.9628$$

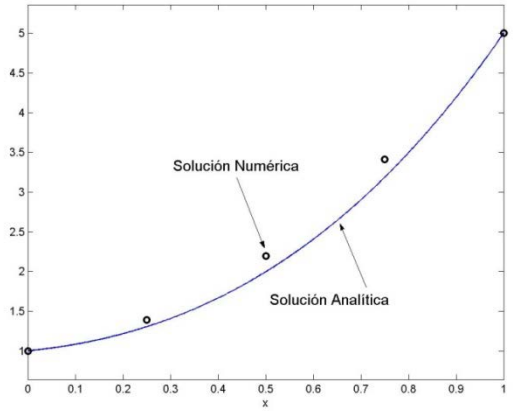
Estas tres ecuaciones conforman un sistema lineal

$$\begin{bmatrix} -3.875 & 1.75 & 0 \\ 2.25 & -3.875 & 1.75 \\ 0 & 2.25 & -3.875 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -1.5540 \\ 0.7872 \\ -7.9628 \end{bmatrix}$$

Cuya solución es:

$$y_1 = 1.3930,\ y_2 = 2.1964,\ y_3 = 3.4095$$

En el siguiente gráfico se visualizan los resultados calculados



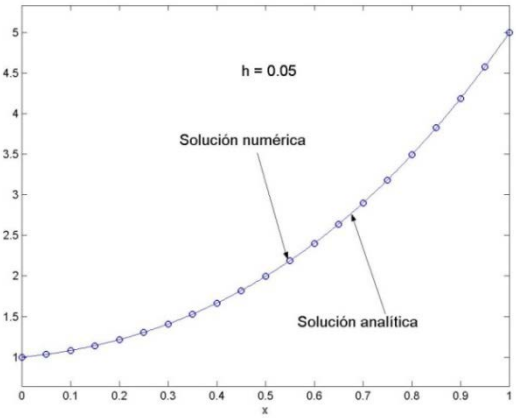
Comparación de la solución numérica y la solución analítica

La aproximación no es muy buena. Esto se debe a que la cantidad de puntos es muy pequeña. Si se desea una mejor aproximación se debe reducir **h.**

El siguiente gráfico muestra la solución calculada con **n=20.** A este valor de n le corresponde **h=0.05**.

Se obtiene un sistema de 19 ecuaciones lineales cuyas incógnitas son los puntos interiores.

Se observa que los resultados tienen una aproximación aceptable



Las ecuaciones que se obtienen con el método de diferencias conforman un sistema tridiagonal de ecuaciones lineales. Estos sistemas pueden resolverse con un algoritmo definido en un capítulo anterior el cual tiene eficiencia tipo **T(n) = O(n)**

Instrumentación computacional del método de diferencias finitas para una EDO

Dada la ecuación diferencial de segundo orden con condiciones en los bordes:

$$F(x, y(x), y'(x), y''(x)) = 0, (x_0, y_0), (x_n, y_n)$$

La siguiente instrumentación computacional del método de diferencias finitas corresponde a la solución de la ecuación de diferencias que resulta después del reemplazo de las derivadas y luego de ser escrita en forma estandarizada:

$$(P)y_{i-1} + (Q)y_i + (R)y_{i+1} = (S),\ i = 1, 2, 3, \ldots, n-1$$

En donde **P, Q, R, S** son expresiones que pueden contener puntos de **x** y **h**, siendo **x** la variable independiente. Para esta instrumentación, estas expresiones deben definirse como funciones en el lenguaje Python.

La función entrega en los vectores **u, v** los puntos calculados de la solución **x, y**.

Los puntos en los bordes: (x_0, y_0) y (x_n, y_n) son dados como datos, **n** es la cantidad de **sub intervalos** espaciados en una distancia **h**.

```
from tridiagonal import *
import numpy as np
def edodif(P,Q,R,S,x0,y0,xn,yn,n):
    h=(xn-x0)/n
    a=[];b=[];c=[];d=[]
    u=np.zeros([n-1,2],float)
    for i in range(0,n-1):
        x=x0+h*i
        a=a+[P(x,h)]                #diagonales del sistema tridiagonal
        b=b+[Q(x,h)]
        c=c+[R(x,h)]
        d=d+[S(x,h)]                #constantes del sistema tridiagonal
        u[i,0]=x
    d[0]=d[0]-a[0]*y0               #corrección para la primera ecuación
    d[n-2]=d[n-2]-c[n-2]*yn         #corrección para la última ecuación
    u[:,1]=tridiagonal(a,b,c,d)
    return u
```

Ejemplo. Use la función **edodif** para resolver la ecuación diferencial con condiciones en los bordes; **$y'' - y' + y - 2e^x - 3 = 0$, $y(0) = 1$, $y(1) = 5$,** con **n=20** sub intervalos

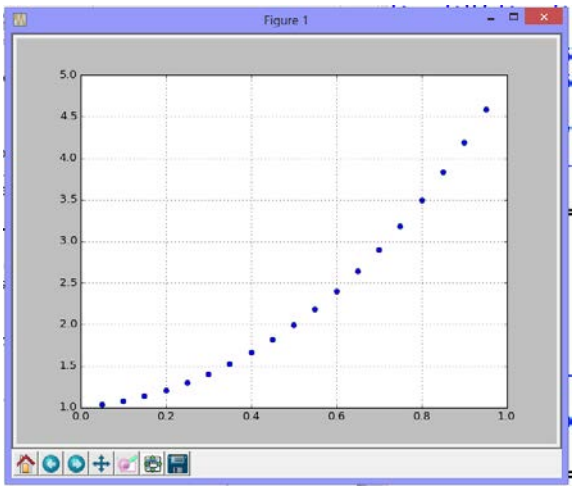
Solución

Forma estándar de la ecuación de diferencias para el ejemplo anterior, luego del reemplazo de las derivadas:

$$(2+h)y_{i-1} + (2h^2-4)y_i + (2-h)y_{i+1} = 4h^2 e^{x_i} + 6h^2,\ i=1, 2,..., n-1;\ y_0=y(0)=1;\ y_n=y(1)=5$$

Escribimos directamente en la ventana interactiva:

```
>>> import pylab as pl
>>> from edodif import*
>>> def P(x,h): return 2+h
>>> def Q(x,h): return 2*h**2-4
>>> def R(x,h): return 2-h
>>> def S(x,h): return 4*h**2*pl.exp(x)+6*h**2          note el uso del
>>> u=edodif(P,Q,R,S,0,1,1,5,20)                        identificador de librería
>>> print(u)
[[ 0.          1.03768279]      La primera columna de u contiene los valores de x
 [ 0.05        1.08745783]      La segunda columna de u contiene los valores de y
 [ 0.1         1.15008052]
 [ 0.15        1.22632555]
 [ 0.2         1.31698659]
   .....etc.....
>>> pl.plot(u[:,0],u[:,1],'ob')
>>> pl.grid(True)
>>> pl.show()
```



Los puntos de la solución están almacenados en la matriz u

9.6.3 Ecuaciones diferenciales ordinarias con condiciones en los bordes con derivadas

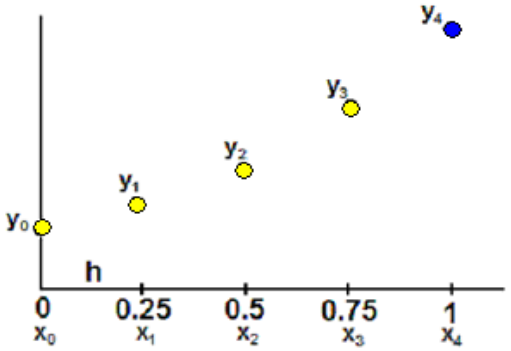
Consideremos una variación del problema anterior:

$$y'' - y' + y - 2e^x - 3 = 0,\ y'(0) = 0.5,\ y(1) = 5,\ 0 \le x \le 1$$

Luego de sustituir las derivadas y simplificar se tiene la ecuación de diferencias como antes:

$$(2+h)y_{i-1} + (2h^2-4)y_i + (2-h)y_{i+1} = 4h^2 e^{x_i} + 6h^2$$

Por simplicidad, consideremos que $h = 0.25$



El punto y_0 también es desconocido. Ahora aplicamos la ecuación de diferencias en cada uno de los puntos desconocidos, incluyendo el punto y_0

$$(2+h)y_{i-1} + (2h^2-4)y_i + (2-h)y_{i+1} = 4h^2 e^{x_i} + 6h^2,\ i = 0, 1, 2, 3$$

i=0: $(2+0.25)y_{-1} + (2(0.25^2)-4)y_0 + (2-0.25)y_1 = 4(0.25^2)\ e^0 + 6(0.25^2)$

$$2.25y_{-1} - 3.875y_0 + 1.75y_1 = 0.6250$$

i=1: $(2+0.25)y_0 + (2(0.25^2)-4)y_1 + (2-0.25)y_2 = 4(0.25^2)\ e^{0.25} + 6(0.25^2)$

$$2.25y_0 - 3.875y_1 + 1.75y_2 = 0.6960$$

i=2: $(2+0.25)y_1 + (2(0.25^2)-4)y_2 + (2-0.25)y_3 = 4(0.25^2)\ e^{0.5} + 6(0.25^2)$

$$2.25y_1 - 3.875y_2 + 1.75y_3 = 0.7872$$

i=3: $(2+0.25)y_2 + (2(0.25^2)-4)y_3 + (2-0.25)y_4 = 4(0.25^2)\ e^{0.75} + 6(0.25^2)$

$$2.25y_2 - 3.875y_3 = -7.8475$$

Se obtiene un sistema de cuatro ecuaciones con cinco incógnitas: $y_{-1},\ y_0,\ y_1,\ y_2,\ y_3$.

En las ecuaciones aparece un punto ficticio y_{-1}

Se puede usar una aproximación central de segundo orden para la primera derivada

$y'_0 = 0.5 = \dfrac{y_1 - y_{-1}}{2h}$ de donde se obtiene que $y_{-1} = y_1 - 2h(0.5) = y_1 - 0.25$

Esto permite eliminar el punto ficticio y_{-1} en la primera ecuación anterior:

i=0: $2.25(y_1 - 0.25) - 3.875y_0 + 1.75y_1 = 0.6250 \Rightarrow -3.875y_0 + 4y_1 = 1.1875$

Finalmente, el sistema se puede escribir:

$$\begin{bmatrix} -3.875 & 4 & 0 & 0 \\ 2.25 & -3.875 & 1.75 & 0 \\ 0 & 2.25 & -3.875 & 1.75 \\ 0 & 0 & 2.25 & -3.875 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1.1875 \\ 0.6960 \\ 0.7872 \\ -7.8475 \end{bmatrix}$$

Cuya solución es: $\mathbf{y_0 = 1.4738,\ y_1 = 1.7246,\ y_2 = 2.3216,\ y_3 = 3.3732}$

El sistema resultante tiene forma tridiagonal, y por lo tanto se puede usar el algoritmo específico muy eficiente para resolverlo.

Otra alternativa es la aplicación de la ecuación de diferencias solamente en los puntos interiores. Se obtendría un sistema de tres ecuaciones con las cuatro incógnitas: y_0, y_1, y_2, y_3. La cuarta ecuación se la obtendría con una fórmula de segundo orden para la derivada en el punto izquierdo definida con los mismos puntos desconocidos:

$$\mathbf{y'_0 = 0.5 = \frac{-3y_0 + 4y_1 - y_2}{2h}}$$

Pero este sistema no tendrá la forma tridiagonal, conveniente para resolver computacionalmente el sistema de ecuaciones con eficiencia.

Instrumentación computacional del método de diferencias finitas con derivadas en los bordes

La siguiente instrumentación del método de diferencias finitas permite resolver problemas de tipo similar al ejemplo anterior, con una derivada en el borde izquierdo. Instrumentaciones similares pueden desarrollarse si la derivada está en el borde derecho, o si ambos bordes contienen derivadas. La función entrega los **n-1** puntos calculados de la solución **x, y** en los vectores **u, v**

Dada la ecuación diferencial de segundo orden con condiciones en los bordes:

$$\mathbf{F(x, y(x), y'(x), y''(x)) = 0,}\ \text{dados}\ \mathbf{(x_0, y'_0), (x_n, y_n)}$$

Se conocen los datos en los bordes: $\mathbf{(x_0, y'_0)}$ y $\mathbf{(x_n, y_n)}$, **n** es la cantidad de sub intervalos espaciados a una distancia **h**.

La ecuación de diferencias que se obtiene luego de la sustitución de las derivadas debe escribirse en forma estandarizada

$$\mathbf{(P)y_{i-1} + (Q)y_i + (R)y_{i+1} = (S),\quad i = 1, 2, 3, \ldots, n-1,}\ \text{con los datos}\ \mathbf{(x_0, y'_0), (x_n, y_n)}$$

En donde **P, Q, R, S** son expresiones que pueden contener $\mathbf{x_i}$ y **h**

Dato de la derivada en el borde izquierdo:

$$\mathbf{y'_0 = \frac{y_1 - y_{-1}}{2h}} \quad \text{de donde se obtiene} \quad \mathbf{y_{-1} = y_1 - 2h(y'_0)}$$

Ecuación de diferencias aplicada en el borde izquierdo: **i = 0**

$$(\mathbf{P})(y_{-1}) + (\mathbf{Q})y_0 + (\mathbf{R})y_1 = (\mathbf{S})$$

Reemplazando el dato de la derivada

$$(\mathbf{P})(y_1 - 2hy'_0) + (\mathbf{Q})y_0 + (\mathbf{R})y_1 = (\mathbf{S})$$

Se obtiene la primera ecuación:

$$(\mathbf{Q})y_0 + (\mathbf{P} + \mathbf{R})y_1 = (\mathbf{S}) + (\mathbf{P})2hy'_0$$

```
from tridiagonal import *
import numpy as np
def edodifdi(P,Q,R,S,x0,dy0,xn,yn,n):
    h=(xn-x0)/n
    a=[];b=[];c=[];d=[]
    u=np.zeros([n,2],float)
    for i in range(0,n):
        x=x0+h*i
        a=a+[P(x,h)]
        b=b+[Q(x,h)]
        c=c+[R(x,h)]
        d=d+[S(x,h)]
        u[i,0]=x
    x=h
    c[0]=P(x,h)+R(x,h)
    d[0]=S(x,h)+P(x,h)*2*h*dy0
    d[n-1]=d[n-1]-c[n-1]*yn
    u[:,1]=tridiagonal(a,b,c,d)
    return u
```

Uso de la función EDODIFDI para resolver el ejemplo anterior, con **n=20** subintervalos

$$y'' - y' + y - 2e^{x} - 3 = 0,\ y'(0) = 0.5,\ y(1) = 5,\ 0 \le x \le 1$$

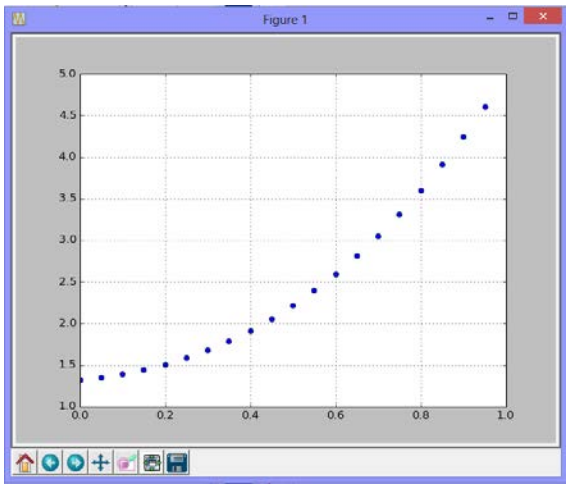
Forma estándar de la ecuación de diferencias

$$(2+h)y_{i-1} + (2h^2-4)y_i + (2-h)y_{i+1} = 4h^2\,\mathbf{e}^{x_i} + 6h^2$$

$$(\mathbf{P})y_{i-1} + (\mathbf{Q})y_i + (\mathbf{R})y_{i+1} = (\mathbf{S}),$$

Escribimos directamente en la ventana interactiva:

```
>>> import pylab as pl
>>> from edodifdi import*
>>> def P(x,h): return 2+h
>>> def Q(x,h): return 2*h**2-4
>>> def R(x,h): return 2-h
>>> def S(x,h): return 4*h**2*pl.exp(x)+6*h**2
>>> u=edodifdi(P,Q,R,S,0,0.5,1,5,20)
>>> print(u)
[[ 0.          1.31863092]
 [ 0.05        1.34898581]
 [ 0.1         1.39052186]
 [ 0.15        1.44398238]
 [ 0.2         1.51013237]
 [ 0.25        1.58975846]
 [ 0.3         1.68366867]
 [ 0.35        1.79269236]
  ..... etc .....
>>> pl.plot(u[:,0],u[:,1],'ob')
>>> pl.grid(True)
>>> pl.show()
```



9.6.4 Normalización del dominio de la E.D.O

Previo a la aplicación de los métodos numéricos es conveniente normalizar la ecuación diferencial llevándola al dominio **[0, 1]** mediante sustitución de variables. De esta manera son más significativas las expresiones para el error de truncamiento en términos de **h**.

Ecuación diferencial original

$$F(x, y(x), y'(x), y''(x)) = 0,\ y(a) = u,\ y(b) = v,\ a \le x \le b$$

Mediante las sustituciones:

$$x = (b - a)t + a:\quad t = 0 \Rightarrow x = a,\quad t = 1 \Rightarrow x = b,\ \Rightarrow \frac{dt}{dx} = \frac{1}{b - a}$$

$$y'(x) = \frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx} = \frac{1}{b - a}\frac{dy}{dt} = \frac{1}{b - a}y'(t)$$

$$y''(x) = \frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right) = \frac{d}{dt}\left(\frac{dy}{dx}\right)\frac{dt}{dx} = \frac{d}{dt}\left(\frac{1}{b - a}\frac{dy}{dt}\right)\frac{dt}{dx} = \frac{1}{(b - a)^2}\frac{d^2y}{dt^2} = \frac{1}{(b - a)^2}y''(t)$$

Se obtiene la ecuación diferencial normalizada

$$F(t, y(t), y'(t), y''(t)) = 0,\ y(0) = u, y(1) = v,\ 0 \le t \le 1$$

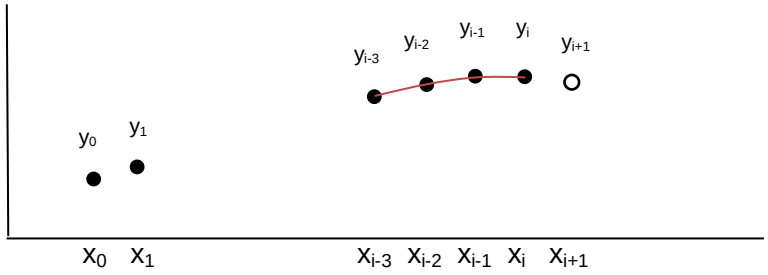
9.7 Ecuaciones diferenciales ordinarias con condiciones en el inicio: Fórmulas de pasos múltiples

Los métodos de un paso como Runge-Kutta calculan cada punto de la solución de una E.D.O. a una distancia **h** utllizando la información del punto anterior. Los métodos de pasos múltiples son fórmulas que utilizan varios puntos calculados y disponibles para calcular la solución en un nuevo punto.

9.7.1 Fórmulas de pasos múltiples de predicción

Estas fórmulas se usan para calcular la solución en un punto mediante un polinomio de interpolación colocado en varios puntos anteriores conocidos espaciados a una distancia **h**.

En el siguiente gráfico se supondrán conocidos los puntos $y_{i-k} \ldots y_{i-3}, y_{i-2}, y_{i-1}, y_i$ mientras que se desea calcular el punto y_{i+1}



Obtención de la fórmula de pasos múltiples:

Dada la E. D. O.

$$y'(x) = dy/dx = f(x,y), \quad y(x_0) = y_0$$

Reescribir como

$$dy = f(x,y)dx = y'(x)dx$$

Integrando desde un punto conocido $y(x_{i-k})$ hasta un nuevo punto $y(x_{i+1})$

$$\int_{x_{i-k}}^{x_{i+1}} dy = \int_{x_{i-k}}^{x_{i+1}} y'(x)dx \quad \Rightarrow \quad y_{i+1} = y_{i-k} + \int_{x_{i-k}}^{x_{i+1}} y'(x)dx$$

Para obtener una fórmula aproximada se usa el polinomio de diferencias finitas regresivas incluyendo al punto **i** y los puntos anteriores:

$$y'(x) = p_n(x) = p_n(S) = f_i + \binom{S}{1}\Delta f_{i-1} + \binom{S+1}{2}\Delta^2 f_{i-2} + \binom{S+2}{3}\Delta^3 f_{i-3} + \ldots + \binom{S+n-1}{n}\Delta^n f_{i-n}$$

$$E = \binom{S+n}{n+1}h^{n+1}y^{(n+1)}(z), \qquad S = \frac{x - x_i}{h}$$

Sustituyendo el polinomio y cambiando los límites y el diferencial

$$y_{i+1} = y_{i-k} + \int_{x_{i-k}}^{x_{i+1}} p_n(x)dx = y_{i-k} + h\int_{-k}^{1} p_n(s)ds$$

Se obtiene la siguiente expresión con la que se pueden generar fórmulas de pasos múltiples denominadas fórmulas de predicción:

$$y_{i+1} = y_{i-k} + h\int_{-k}^{1} p_n(s)ds$$

La fórmula tiene dos parámetros para elegir: **n, k**

Ejemplos:

1) $n = 0,\ k = 0:\quad y_{i+1} = y_i + h\int_0^1 f_i ds = y_i + hf_i = y_i + hf(x_i, y_i)$

Es la conocida fórmula de Euler

2) $n = 2,\ k = 0:\quad y_{i+1} = y_i + h\int_0^1 [f_i + s\Delta f_{i-1} + \frac{s(s-1)}{2}\Delta^2 f_{i-2}]ds$

$$= y_i + h[f_i + \frac{\Delta f_{i-1}}{2} + \frac{5}{12}\Delta^2 f_{i-2}]$$

Sustituyendo las diferencias finitas:

$$\Delta f_{i-1} = f_i - f_{i-1}, \qquad \Delta^2 f_{i-2} = \Delta f_{i-1} - \Delta f_{i-2} = f_i - f_{i-1} - (f_{i-1} - f_{i-2}) = f_i - 2f_{i-1} + f_{i-2}$$

Se obtiene finalmente:

$$y_{i+1} = y_i + \frac{h}{12}[23f_i - 16f_{i-1} + 5f_{i-2}], \qquad E = O(h^2)$$

Observaciones importantes

1) Las fórmulas de predicción requieren tener puntos disponibles antes de su aplicación. Estos puntos deben ser calculados con mucha precisión. El método adecuado para esto es el método de Runge-Kutta

2) La integración extiende el polinomio de interpolación hasta el punto **i+1** que no pertenece al dominio del polinomio. Por lo tanto se ha realizado una extrapolación, que en general no no produce resultados confiables.

3) La ventaja de estas fórmulas es que requieren menos cálculos en cada paso pues usan puntos ya calculados. Esto fue importante en la época en la que no se disponía de dispositivos computacionales automatizados. Una aplicaciòn de interés es usar el método para proyectar el valor de una función y su derivada fuera del intervalo que contiene a los puntos conocidos.

9.7.2 Fórmulas de pasos múltiples de corrección

Estas fórmulas son el complemento a las fórmulas de predicción. En las fórmulas de pasos múltiples de corrección se coloca el polinomio de interpolación incluyendo el punto **i+1** calculado como una primera estimación con la fórmula de predicción. El nuevo resultado corrige el resultado anterior y produce una mejor aproximación.

Dada la ecuación diferencial ordinaria de primer orden:

$$\mathbf{y'(x) = dy/dx = f(x,y),\quad y(x_0) = y_0}$$

Reescribirla como

$$\mathbf{dy = f(x,y)\,dx = y'(x)\,dx}$$

Integrando desde un punto arbitrario $\mathbf{y(x_{i-k})}$ hasta el punto desconocido $\mathbf{y(x_{i+1})}$

$$\int_{x_{i-k}}^{x_{i+1}} dy = \int_{x_{i-k}}^{x_{i+1}} y'(x)dx \quad \Rightarrow \quad y_{i+1} = y_{i-k} + \int_{x_{i-k}}^{x_{i+1}} y'(x)dx$$

Para obtener una fórmula aproximada de corrección se usa el polinomio de diferencias finitas regresivas incluyendo al punto **i+1** y puntos anteriores:

$$p_n(x) = p_n(S) = f_{i+1} + \binom{S}{1}\Delta f_i + \binom{S+1}{2}\Delta^2 f_{i-1} + \binom{S+2}{3}\Delta^3 f_{i-2} + \ldots + \binom{S+n-1}{n}\Delta^n f_{i-n+1}$$

$$E = \binom{S+n}{n+1} h^{n+1} y^{(n+1)}(z), \qquad s = \frac{x - x_{i+1}}{h}$$

Sustituyendo **y'(x)** por el polinomio y cambiando los límites y el diferencial

$$y_{i+1} = y_{i-k} + h\int_{-k-1}^{0} p_n(s)ds$$

Esta expresión se usa para generar fórmulas de pasos múltiples de corrección:

La fórmula tiene dos parámetros para elegir: **n, k**

Ejemplo:

$$n = 1,\ k = 0: \quad y_{i+1} = y_i + \frac{h}{2}[f_i + f_{i+1}]$$

Es la conocida fórmula mejorada de Euler o fórmula de Heun

9.7.3 Métodos de Predicción – Corrección

La combinación de una fórmula de pasos múltiples de predicción con una fórmula de pasos múltiples de corrección constituye un método de Predicción – Corrección. Uno de estos métodos que ha sido estudiado se denomina fórmula de Adams – Moulton. Se la obtiene con los parámetros **n=3, k=0** en las fórmulas establecidas anteriormente. El desarrollo detallado permite también conocer el error de truncamiento correspondiente.

Fórmula de Predicción de Adams-Moulton:

$$y_{i+1} = y_i + \frac{h}{24}[55f_i - 59f_{i-1} + 37f_{i-2} + 9f_{i-3}], \qquad E_P = \frac{251}{720}h^5 y^v(z)$$

Fórmula de Corrección de Adams-Moulton:

$$y_{i+1} = y_i + \frac{h}{24}[9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}], \qquad E_C = -\frac{19}{720}h^5 y^v(z)$$

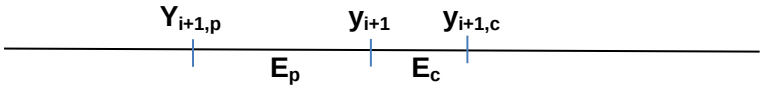
Esta combinación de fórmulas permite establecer un esquema de exactitud para el error de truncamiento con el planteamiento siguiente:

Sean

y_{i+1}	valor exacto, desconocido
$y_{i+1,p}$	valor calculado con la fórmula de predicción de Adams Moulton
$y_{i+1,c}$	valor calculado con la fórmula de corrección de Adams Moulton

Suponer que los valores de las derivadas en el error son aproximadamente iguales.

Considerando la siguiente disposición de los valores calculados y el valor exacto:



$$\frac{Ec}{Ec + Ep} = \frac{Ec}{y_{i+1,c} - y_{i+1,p}} \cong \frac{19/720}{251/720 + 19/729} \cong \frac{1}{14}$$

De donde se puede establecer el siguiente criterio para el error de truncamiento del método:

$$| y_{i+1,c} - y_{i+1,p} | \ \cong \ 14\ Ec$$

Para aplicar este criterio, suponer que se desea que el error en el resultado final de y_{i+1} caculado con la fórmula de corrección no exceda a $E_c < 10^{-m}$, entonces deberá cumplirse:

$$| y_{i+1,c} - y_{i+1,p} | < 14x10^{-m}$$

Si no se cumple, debe entenderse que el valor de **h** tendría que reducirse para reducir el error.

9.8 Ejercicios con ecuaciones diferenciales ordinarias

1. Obtenga dos puntos de la solución de la siguiente ecuación diferencial utilizando tres términos de la Serie de Taylor. Use **h = 0.1**

$$y' - 2x + 2y^2 + 3 = 0,\ y(0)=1$$

2. Dada la siguiente ecuación diferencial ordinaria de primer orden

$$y' - 2y + 2x^2 - x + 3 = 0,\ y(0)=1.2$$

a) Obtenga dos puntos de la solución con la fórmula de Euler. Use **h = 0.1**
b) Obtenga dos puntos de la solución con la fórmula de Heun. Use **h = 0.1**
c) Obtenga dos puntos de la solución con la fórmula de Runge-Kutta de cuarto orden. Use **h = 0.1**
d) Compare con la solución exacta: $y(x) = x/2 + x^2 - 11/20\ e^{2x} + 7/4$

3. Al resolver una ecuación diferencial con un método numérico, el error de truncamiento tiende a acumularse y crecer. Use el método de Euler con **h=0.1** para calcular 10 puntos de la solución de la ecuación: $y' - 2x + 5y - 1 = 0,\ y(0) = 2$

Compare con la solución analítica $y(x) = 2x/5 + 47/25\ e^{-5x} + 3/25$. Observe que el error de truncamiento tiende a reducirse. Explique este comportamiento.

4. La solución exacta de la ecuación diferencial: $y' - 2xy = 1,\ y(0)=y_0$ es

$$y(x) = e^{x^2}\left(\frac{\sqrt{\pi}}{2}\,erf(x) + y_0\right),\ \text{donde}\ erf(x) = \frac{1}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

Encuentre **y(0.4)** de la ecuación diferencial: $y' - 2xy = 1,\ y(0)=1$

a) Con la fórmula de Runge-Kutta, **h=0.2**
b) Evaluando la solución exacta con la cuadratura de Gauss con dos puntos

5. Dada la siguiente ecuación diferencial ordinaria de segundo orden

$$y'' - y' - sen(x) + y + 1 = 0,\ y(0)=1.5,\ y'(0) = 2.5$$

a) Obtenga dos puntos de la solución con la fórmula de Heun. (**h = 0.1**)
b) Obtenga dos puntos de la solución con la fórmula de Runge-Kutta de cuarto orden. (**h = 0.1**)
c) Compare con la solución computacional obtenida con el método **odeint** de Python

6. Dada la siguiente ecuación diferencial

$$y'' + y' - y + 2x^2 - 3x - 4 = 0,\ 0 \le x \le 2,\ y(0) = 1,\ y'(0) = 1$$

a) Obtenga dos puntos de la solución con la fórmula de Runge-Kutta de cuarto orden. (**h = 0.1**).
b) Compare con la solución simbólica obtenida con **dsolve** de la librería SymPy

7. Dada la siguiente ecuación diferencial

$$2y''(x) - 3y'(x) + 2x = 5,\ y(1) = 2,\ y(3) = 4$$

a) Normalice la ecuación diferencial en el intervalo **[0, 1]**
b) Use el método de diferencias finitas y obtenga la solución, **h = 0.2** en el intervalo normalizado.

8. Dada la siguiente ecuación diferencial

$$y'' + y' - y + 2x^2 - 3x - 4 = 0,\ 2 \le x \le 5,\ y(2) = 11,\ y(5) = 56$$

a) Normalice la ecuación al intervalo $0 \le t \le 1$
b) Sustituya las derivadas por aproximaciones de diferencias finitas de segundo orden y simplifique a la forma : $P\, y_{i-1} + Q\, y_i + R\, y_{i+1} = S$
c) Resuelva el sistema resultante y obtenga la solución. Use **h=0.2**

9. La siguiente es una ecuación diferencial no lineal conocida con el nombre de Ecuación de Van der Pol y describe un sistema vibratorio masa-resorte-amortiguador:

$$\text{x(t):}\quad mx'' + \mu(x^2 - 1)x' + kx = 0,\quad x(0)=0.75,\ x'(0)=0$$

En donde **x** es la amplitud de vibración, **t** es tiempo, **m** es masa**, k** es la constante elástica del resorte y μ es el coeficiente de amortiguamiento.

Con la fórmula de Heun calcule **x(t), t=0.1, 0.2, 0.3. ..., 1.0,** use $\mu = 4$, **k=1, m=1**

10. La distribución de temperatura u(x) en estado estable de una barra de longitud L con una fuente de calor Q(x), temperatura constante en el extremo derecho y aislada en el extremo izquierdo, está dada por

$$u'' + Q(x) = 0,\quad u'(0) = 0,\ u(L) = T$$

Resuelva para **T=100, $Q(x)=x^2$, L=1.** Use el método de diferencias finitas, **h=0.2**

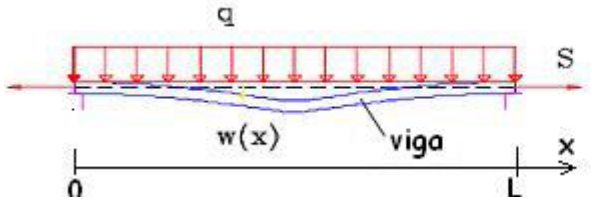
11. Para resolver la siguiente ecuación diferencial ordinaria no lineal

$$y'' = x(y')^2,\ 1 \le x \le 2,\quad y(1) = 1.1071,\ y'(1) = -0.4$$

Use la fórmula mejorada de Euler (fórmula de Heun) con **h = 0.25, $1 \le x \le 2$**

Tabule los resultados obtenidos junto con los valores de la solución exacta:
y(x) = arc cot(x/2). Comente el resultado de esta comparación.

12. Un problema importante en ingeniería es la deflexión de una viga de sección transversal uniforme sujeta a una carga distribuida y colocada con sus extremos apoyados de forma tal que no pueden deflectarse, como se muestra en la figura:



La ecuación que describe esta situación física es: $\frac{d^2w}{dx^2} = \frac{S}{EI}w + \frac{qx}{2EI}(x - L)$

Siendo w(x): deflexión a una distancia x desde el extremo izquierdo.
q: intensidad de carga por unidad de longitud. E: módulo de elasticidad.
S: tensión en los puntos extremos. I: momento principal de inercia. L: longitud de la viga.

La condición de no deflexión en los extremos se expresa con: w(0) = w(L) = 0.

Suponga una viga de acero con las siguientes características:
L = 10 m, q = 1460 N/m, E = 20.7 x 10^7 kPa, S = 4.4 kN, I = 5 m^4.

Calcule con un método numérico la deflexión de la viga cada 2 m.

13. Una masa **m = 1/8** se suspende en un resorte de constante **k = 2**. Inicialmente se la estira hacia abajo **1.0** y se aplica una fuerza externa **f(t) = 8sen(t).** Entonces el desplazamiento vertical **Y(t)** se describe con la siguiente ecuación, en donde **t** es tiempo:

$$Y'' + \frac{k}{m}Y = \frac{f(t)}{m}$$

Con el método de diferencias finitas, **h=0.1,** encuentre **Y(t)** entre: **Y(0) = -1** y **Y(1) = 3**

14. Un sistema amortiguado y forzado resorte-masa tiene la ecuación diferencial ordinaria para sus movimientos:

$$m\frac{d^2x}{dt^2} + a\left|\frac{dx}{dt}\right|\frac{dx}{dt} + kx = \mathrm{F_0 sen(wt)}$$

Donde x = desplazamiento a partir de la posición de equilibrio, t = tiempo, m= 2 kg masa, a = 5 N/(m/s)^2 y k= 6 N/m. El término de amortiguamiento es no lineal y representa el amortiguamiento del aire. La función de fuerza F_0sen(wt) tiene valores $F_{0\,=}$ 2.5 N y w = 0.5 rad/s. Las condiciones iniciales son:

Velocidad inicial, $\frac{dx}{dt} = 0\ m/s$
Desplazamiento inicial, x = 1 m

a) Resuelva esta ecuación con un método de Runge-Kutta de 2do orden durante el periodo $0 \leq t \leq 15\ s$, usando h=0.1 s. (sólo plantee la formulación)

b) Realice los cálculos para tres pasos, incluyendo la estimación del error.

15. Al integrar una ecuación diferencial de primer orden **y'(x) = f(x,y)** entre x_{i-1} y x_{i+1} se obtiene la expresión: $y_{i+1} = y_{i-1} + \int_{x_{i-1}}^{x_{i+1}} f(x,y)\,dx = y_{i-1} + \int_{x_{i-1}}^{x_{i+1}} y'(x)\,dx$,

a) Use la aproximación: $y'(x) \cong p_2(s) = f_i + s\Delta f_{i-1} + \frac{(s+1)s}{2}\Delta^2 f_{i-2}$

en donde $s = \frac{x - x_i}{h}$.

Obtenga una fórmula aproximada con la cual pueda obtener puntos de la solución.

b) Con la fórmula calcule **y(0.08)** de la solución de la ecuación **(1 + 2x) y' + 0.9 y = 0**, dados la condición inicial **(0, 1)** y dos puntos de la solución: **(0.02,0.9825), (0.04, 0.9660).** Use **h=0.02**

16. Sea **P(t)** el número de individuos de una población en el tiempo **t**, medido en años. Con ciertas suposiciones, la tasa de crecimiento demográfico está dada por la ecuación diferencial no lineal:

$$\frac{dP(t)}{dt} = bt - k[P(t)]^2$$

Suponga que **P(0)=50976, b=2.9x10^{-2}, k=1.4x10^{-7}**. Calcule la población después de dos años. Construya una fórmula para calcular puntos de **P** desarrollando tres términos de la Serie de Taylor. Use **h=0.5** y estime numéricamente el error de truncamiento

17. Dada la siguiente ecuación diferencial:

y(x): y'' + y' - y + 3x^2 - 5x - 6 = 0, 4≤x≤9, y(4) = 53, y(9) = 253

Use la fórmula de Runge-Kutta de cuarto orden como soporte para aplicar el método del disparo con **h=0.2**

Realice dos intentos eligiendo valores para **y'(4).** Para el tercer intento use un valor para **y'(4)** obtenido mediante interpolación lineal con los resultados de los dos primeros intentos. Muestre el cuadro de resultados y el gráfico del tercer intento.

18. Dada la siguiente ecuación diferencial:

y(x): y'' + y' - y + 3x^2 - 5x - 6 = 0, 4≤x≤9, y(4) = 53, y(9) = 253

a) Normalice la ecuación mediante las sustituciones indicadas en las notas del curso.

b) Sustituya las derivadas y lleve la ecuación diferencial a la forma

$$\mathbf{(P)y_{i-1} + (Q)y_i + (R)y_{i+1} = (S)}$$

c) Resuelva la ecuación diferencial normalizada, con dominio $\mathbf{0 \leq t \leq 1}$
Usando el método de diferencias finitas con **h=0.1**

d) Muestre una tabla con los resultados calculados.

e) Muestre en un gráfico los resultados obtenidos de la solución calculada.

10 MÉTODO DE DIFERENCIAS FINITAS PARA RESOLVER ECUACIONES DIFERENCIALES PARCIALES

En este capítulo se estudiará el método de diferencias finitas aplicado a la resolución de ecuaciones diferenciales parciales de segundo orden para funciones de dos variables.

Sea **u** una función que depende de las variables independientes **x, y**. La siguiente ecuación es la forma general de una ecuación diferencial parcial de segundo orden:

$$\mathbf{u(x,y):}\quad F\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \frac{\partial^2 u}{\partial x \partial y}\right) = 0$$

Una forma de clasificar estas ecuaciones es por su tipo: parabólicas, elípticas e hiperbólicas,

En forma similar a las ecuaciones diferenciales ordinarias, el método numérico que usaremos consiste en sustituir las derivadas por aproximaciones de diferencias finitas. El objetivo es obtener una ecuación denominada ecuación de diferencias que pueda resolverse por métodos algebraicos. Esta sustitución discretiza el dominio con espaciamiento que debe elegirse.

10.1 Aproximaciones de diferencias finitas

Las siguientes son algunas aproximaciones de diferencias finitas de uso común para aproximar las derivadas de **u** en un punto (x_i, y_j), siendo Δx, Δy espaciamiento constante en las direcciones **x** e **y** respectivamente. El término a la derecha en cada fórmula representa el error de truncamiento.

$$\frac{\partial u_{i,j}}{\partial x} \cong \frac{u_{i+1,j} - u_{i,j}}{\Delta x}, \qquad E = -\frac{\Delta x}{2}\frac{\partial^2 u(z)}{\partial x^2} = O(\Delta x), \quad x_i \le z \le x_{i+1} \tag{8.1}$$

$$\frac{\partial u_{i,j}}{\partial y} \cong \frac{u_{i,j+1} - u_{i,j}}{\Delta y}, \qquad E = -\frac{\Delta y}{2}\frac{\partial^2 u(z)}{\partial y^2} = O(\Delta y), \quad y_i \le z \le y_{i+1} \tag{8.2}$$

$$\frac{\partial u_{i,j}}{\partial y} \cong \frac{u_{i,j} - u_{i,j-1}}{\Delta y}, \qquad E = -\frac{\Delta y}{2}\frac{\partial^2 u(z)}{\partial y^2} = O(\Delta y), \quad y_{i-1} \le z \le y_i \tag{8.3}$$

$$\frac{\partial u_{i,j}}{\partial x} \cong \frac{u_{i+1,j} - u_{i-1,j}}{2(\Delta x)}, \qquad E = -\frac{(\Delta x)^2}{6}\frac{\partial^3 u(z)}{\partial^3 x} = O(\Delta x)^2, \quad x_{i-1} \le z \le x_{i+1} \tag{8.4}$$

$$\frac{\partial^2 u_{i,j}}{\partial x^2} \cong \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{2(\Delta x)}, \qquad E = -\frac{(\Delta x)^2}{12}\frac{\partial^4 u(z)}{\partial x^4} = O(\Delta x)^2, \quad x_{i-1} \le z \le x_{i+1} \tag{8.5}$$

$$\frac{\partial^2 u_{i,j}}{\partial y^2} \cong \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta y)^2}, \qquad E = -\frac{(\Delta y)^2}{12}\frac{\partial^4 u(z)}{\partial y^4} = O(\Delta y)^2, \quad y_{i-1} \le z \le y_{i+1} \tag{8.6}$$

10.2 Ecuaciones diferenciales parciales de tipo parabólico

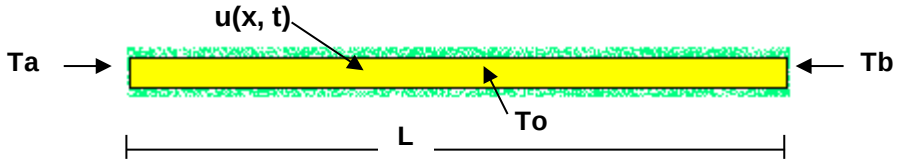
Estas ecuaciones se caracterizan porque en su dominio una de las variables no está delimitada. Para aplicar el método de diferencias finitas usaremos un ejemplo básico para posteriormente interpretar los resultados obtenidos.

Ejemplo. Resolver la ecuación de difusión en una dimensión con los datos suministrados

$$\mathbf{u(x,t):}\quad \frac{\partial^2 u}{\partial x^2} = k\frac{\partial u}{\partial t}$$

Suponer que **u** es una función que depende delas variables **x**, **t**, en donde **u** representa valores de temperatura, **x** representa posición, mientras que **t** es tiempo,

Esta ecuación se puede asociar al flujo de calor en una barra muy delgada aislada transversalmente y sometida en los extremos a alguna fuente de calor. La constante **k** depende del material de la barra. La solución representa la distribución de temperaturas en cada punto **x** de la barra y en cada instante **t**



Ta, Tb son valores de temperatura de las fuentes de calor aplicadas en los extremos. En este primer ejemplo suponer que son valores constantes. **To** es la temperatura inicial y **L** es la longitud de la barra.

Estas condiciones se pueden expresar de manera simbólica en un sistema de coordenadas

u(0, t) = Ta, t ≥ 0
u(L, t) = Tb, t ≥ 0
u(x, 0) = To, 0 < x < L

Para aplicar el método de diferencias finitas, debe discretizarse el dominio de **u** mediante una malla con puntos en un plano en el cual el eje horizontal representa la posición x_i mientras que el eje vertical representa el tiempo t_j.

$$\mathbf{u = u(x,t),\ 0 \le x \le L,\ t \ge 0} \quad \Rightarrow \quad \mathbf{u(x_i, t_j) = u_{i,j}\ ;\ i = 0, 1, ..., n;\ j = 0, 1, 2, ...}$$

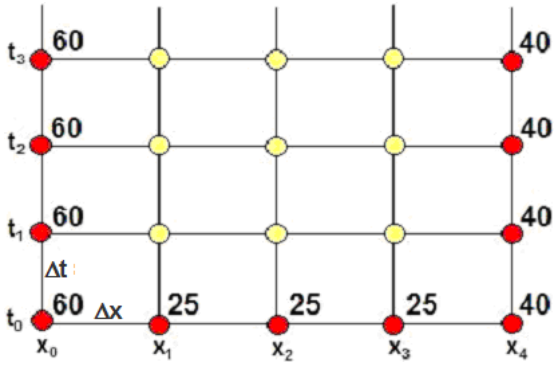
El método de diferencias finitas permitirá aproximar **u** en los puntos $\mathbf{(x_i, t_j)}$

Para el ejemplo supondremos los siguientes datos, en las unidades que correspondan

Ta = 60
Tb = 40
To = 25
k = 4
L = 1

Decidimos además, para simplificar la aplicación del método que $\Delta x = 0.25,\ \Delta t = 0.1$

Con esta información se define el dominio de $u_{i,j}$. En la malla se representan los datos en los bordes y los puntos interiores cuyo valor debe calcularse:



10.2.1 Un esquema de diferencias finitas explícito

Para el primer intento elegimos las fórmulas (**8.5**) y (**8.2**) para sustituir las derivadas de la ecuación diferencial:

$$\frac{\partial^2 u}{\partial x^2} = k\frac{\partial u}{\partial t} \quad \Rightarrow \quad \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2 = k\frac{u_{i,j+1} - u_{i,j}}{\Delta t} + O(\Delta t)$$

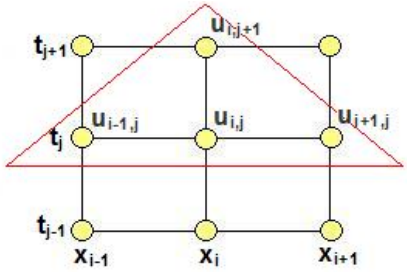
Se obtiene la ecuación de diferencias

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} = k\frac{u_{i,j+1} - u_{i,j}}{\Delta t},$$ cuyo error de truncamiento es $E = O(\Delta x)^2 + O(\Delta t)$.

Si $\Delta x, \Delta t \to 0 \ \Rightarrow\ E \to 0$ y la ecuación de diferencias tiende a la ecuación diferencial parcial

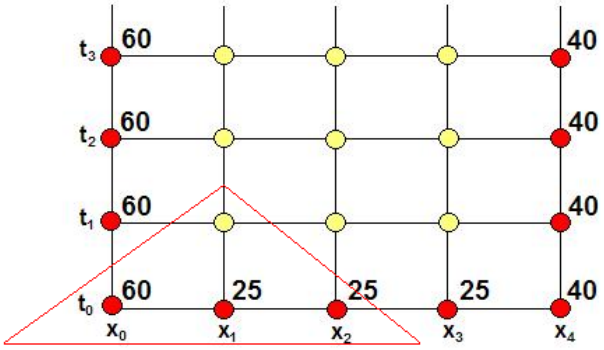
Para que esta sustitución sea consistente, todos los términos de la ecuación deben tener un error de truncamiento de orden similar. Esto significa que Δt debe ser menor que Δx, aproximadamente en un orden de magnitud.

Es conveniente analizar cuales puntos están incluidos en la ecuación de diferencias. Para esto consideramos un segmento de la malla y marcamos los puntos de la ecuación.



Los puntos que están incluidos en la ecuación de diferencia conforman un triángulo. Este triángulo puede colocarse en cualquier lugar de la malla asignando a **i, j** los valores apropiados.

Por ejemplo, si **i=1, j=0**, la ecuación de diferencias se ubica en el extremo inferior izquierdo de la malla. Los puntos en color rojo son los datos conocidos. Los puntos en amarillo son los puntos que deben calcularse.



Se puede observar que solo hay un punto desconocido en la ecuación. Por lo tanto, esta ecuación de diferencias proporciona un **método explícito** de cálculo**.** Esto significa que cada punto de la solución puede obtenerse en forma individual y directa cada vez que se aplica la ecuación.

Despejamos el punto desconocido $u_{i,j+1}$

$$u_{i,j+1} = \frac{\Delta t}{k(\Delta x)^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + u_{i,j}$$

Definiendo $\lambda = \dfrac{\Delta t}{k(\Delta x)^2} \quad \Rightarrow \quad u_{i,j+1} = \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + u_{i,j}$

La ecuación de diferencias se puede escribir

$$u_{i,j+1} = \lambda u_{i-1,j} + (1-2\lambda)u_{i,j} + \lambda u_{i+1,j}, \quad i = 1, 2, 3; \quad j = 0, 1, 2, \ldots$$

La forma final de la ecuación de diferencias se obtiene sustituyendo los datos del ejemplo:

$$\lambda = \frac{\Delta t}{k(\Delta x)^2} = \frac{0.1}{4(0.25)^2} = 0.4$$

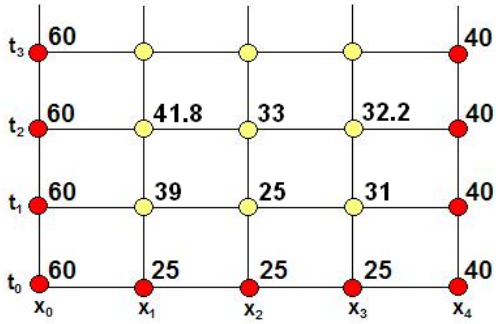
$$u_{i,j+1} = 0.4u_{i-1,j} + 0.2u_{i,j} + 0.4u_{i+1,j}, \quad i = 1, 2, 3; \quad j = 0, 1, 2, \ldots$$

La solución se debe calcular sucesivamente para cada nivel t_j hasta donde sea de interés analizar la solución de la ecuación diferencial. Calculemos dos niveles de la solución:

j = 0, i = 1: $u_{1,1} = 0.4u_{0,0} + 0.2u_{1,0} + 0.4u_{2,0} = 0.4(60) + 0.2(25) + 0.4(25) = 39$
i = 2: $u_{2,1} = 0.4u_{1,0} + 0.2u_{2,0} + 0.4u_{3,0} = 0.4(25) + 0.2(25) + 0.4(25) = 25$
i = 3: $u_{3,1} = 0.4u_{2,0} + 0.2u_{3,0} + 0.4u_{4,0} = 0.4(25) + 0.2(25) + 0.4(40) = 31$
j = 1, i = 1: $u_{1,2} = 0.4u_{0,1} + 0.2u_{1,1} + 0.4u_{2,1} = 0.4(60) + 0.2(39) + 0.4(25) = 41.8$
i = 2: $u_{2,2} = 0.4u_{1,1} + 0.2u_{2,1} + 0.4u_{3,1} = 0.4(39) + 0.2(25) + 0.4(31) = 33$

i = 3: $u_{3,2} = 0.4u_{2,1} + 0.2u_{3,1} + 0.4u_{4,1} = 0.4(25) + 0.2(31) + 0.4(40) = 32.2$

En el siguiente gráfico se anotan los resultados para registrar el progreso del cálculo



Para que la precisión de la solución sea aceptable, los incrementos **Δx** y **Δt** deberían ser mucho más pequeños, pero esto haría que la cantidad de cálculos involucrados sea muy grande para hacerlo manualmente.

10.2.2 Análisis de estabilidad del método de diferencias finitas

La estabilidad del método de diferencias finitas es el estudio del crecimiento del error de truncamiento en el proceso cálculo referido a la variable en la cual avanza la solución. Si el método es estable, entonces el error de truncamiento permanece acotado y se dice que el método de diferencias finitas es estable.

Criterio de von Neumann

Supondremos que **Δx** se ha fijado y se desea determinar **Δt** para que el método de diferencias finitas sea estable, es decir que el error de truncamiento no crezca mientras el cálculo avanza en la dirección de la variable no acotada **t**.

Se define el coeficiente de crecimiento del error de truncamiento en dos iteraciones consecutivas en la dirección t:

$$M = \left| \frac{E_{i,j+1}}{E_{i,j}} \right|$$

Entonces, si **M ≤ 1**, el error no crecerá en **t** y el método de diferencias finitas es estable.

El objetivo es establecer alguna relación entre **Δx** y **Δt**. El método de Neumann asigna una forma exponencial compleja al error de truncamiento $E_{i,j}$ en cada punto (x_i, t_j). De esta manera el crecimiento der error de truncamiento queda expresado mediante **t**

$$E_{i,j} = e^{\sqrt{-1}\,\beta x_i + \alpha t_j}, \quad E_{i,j+1} = e^{\sqrt{-1}\,\beta x_i + \alpha(t_j + \Delta t)} \Rightarrow \frac{E_{i,j+1}}{E_{i,j}} = \frac{e^{\sqrt{-1}\,\beta x_i + \alpha(t_j + \Delta t)}}{e^{\sqrt{-1}\,\beta x_i + \alpha t_j}} = e^{\alpha \Delta t}$$

$M \le 1 \Rightarrow e^{\alpha \Delta t} \le 1$. Es la condición de estabilidad para este método

Análisis de la estabilidad del método explícito para la EDP del ejemplo propuesto

Ecuación de diferencias:

$$u_{i,j+1} = \lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + u_{i,j}, \quad \lambda = \frac{\Delta t}{k(\Delta x)^2}$$

La ecuación de errores para el análisis de estabilidad se obtiene restando la ecuación de diferencias, de la ecuación de diferencias en la que a cada término se le agrega el error de truncamiento respectivo:

$$u_{i,j+1} + E_{i,j+1} = \lambda(u_{i-1,j} + E_{i-1,j} - 2(u_{i,j} + E_{i,j}) + u_{i+1,j} + E_{i+1,j}) + u_{i,j} + E_{i,j}$$
$$-u_{i,j+1} = -\lambda(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) - u_{i,j}$$

Sumando las ecuaciones y sustituyendo los errores de truncamiento en la forma exponencial compleja se obtiene:

$$e^{\sqrt{-1}\,\beta x_i + \alpha(t_j+\Delta t)} = \lambda(e^{\sqrt{-1}\,\beta(x_i+\Delta x)+\alpha t_j} - 2e^{\sqrt{-1}\,\beta x_i+\alpha t_j} + e^{\sqrt{-1}\,\beta(x_i-\Delta x)+\alpha t_j}) + e^{\sqrt{-1}\,\beta x_i+\alpha t_j}$$

Dividiendo por $e^{\sqrt{-1}\,\beta x_i+\alpha t_j}$

$$e^{\alpha\Delta t} = \lambda(e^{\sqrt{-1}\,\beta\Delta x} - 2 + e^{\sqrt{-1}\,\beta_i(-\Delta x)}) + 1$$

Sustituyendo las fórmulas conocidas:

$$e^{\pm\sqrt{-1}\,\beta\Delta x} = \cos(\beta\Delta x) \pm \sqrt{-1}\,\text{sen}(\beta\Delta x)$$

Se obtiene luego de simplificar

$$e^{\alpha\Delta t} = \lambda(2\cos(\beta\Delta x) - 2) + 1$$

La estabilidad está condicionada a: $|e^{\alpha\Delta t}| \le 1$

$$|\lambda(2\cos(\beta\Delta x) - 2) + 1| \le 1$$

Sustituyendo los valores extremos de $\cos(\beta\Delta x) = 1, \quad \cos(\beta\Delta x) = -1$

$\cos(\beta\Delta x) = 1$: $|\lambda(2(1) - 2) + 1| \le 1 \Rightarrow 1 \le 1$

$\cos(\beta\Delta x) = -1$: $|\lambda(2(-1) - 2) + 1| \le 1 \Rightarrow |\lambda(-4) + 1| \le 1 \Rightarrow -2 \le -4\lambda \le 0 \Rightarrow 2 \ge 4\lambda \ge 0$

$$\lambda \ge 0 \ \wedge \ \lambda \le \frac{1}{2}$$

Se obtiene la condición para que este método sea estable:

$$\lambda \le \frac{1}{2} \quad \Rightarrow \quad \frac{\Delta t}{k(\Delta x)^2} \le \frac{1}{2}$$

Si se cumple esta condición, el error de truncamiento no crecerá. Si se fija **k** y **Δx,** debería elegirse **Δt** para que se mantenga la condición anterior.

Se puede verificar que si no se cumple esta condición, el método se hace inestable rápidamente y los resultados obtenidos son incoherentes. Esto puede interpretarse como que la ecuación de diferencias ya no es consistente con la ecuación original.

Desde el punto de vista del error de truncamiento del método de diferencias finitas, el parámetro **Δx** debería ser suficientemente pequeño para que los resultados tengan precisión aceptable.

Según la expresión del error de truncamiento: $\mathbf{E = O(\Delta x)^2 + O(\Delta t),}$ para que las aproximaciones sean consistentes numéricamente, **Δt** debería ser aproximadamente un orden de magnitud menor que **Δx**, además de cumplir con la condición de estabilidad.

Ejemplo. En la EDP anterior se desea que el error de truncamiento en el método de diferencias finitas esté en el orden de **0.0001**. Suponer **k = 4**

Consistencia numérica

$$\mathbf{E = O(\Delta x)^2 + O(\Delta t) \quad \Rightarrow \quad \Delta x = 0.01 \quad \Rightarrow \quad \Delta t = 0.0001}$$

Condición de estabilidad

$$\lambda = \frac{\Delta t}{k(\Delta x)^2} \leq \frac{1}{2} \quad \Rightarrow \quad \Delta t \leq \frac{1}{2}k(\Delta x)^2 = \frac{1}{2}4(0.01)^2 = 0.0002$$

Por lo tanto, si se elije **Δt** = **0.0001** se cumplen las condiciones de estabilidad y consistencia numérica referidas al error de truncamiento.

Desde el punto de vista del error de redondeo, **Δx** y **Δt** no deberían ser demasiado pequeños para evitar la acumulación del error en las operaciones aritméticas. Debido a que aumenta la cantidad de niveles de solución en la dirección **t**. Los cálculos aritméticos deberán realizarse con mayor precisión para que el error de redondeo no influya en los resultados.

10.2.3 Instrumentación computacional del método explícito de diferencias finitas para una E.D.P. de tipo parabólico

El siguiente programa es una instrumentación en Python para resolver el ejemplo anterior y es una referencia para aplicarlo a problemas similares. Los resultados obtenidos se muestran gráficamente.

Para la generalización es conveniente expresar la ecuación de diferencias en forma estándar

$$u_{i,\,j+1} = (P)\,u_{i-1,\,j} + (Q)u_{i,\,j} + (R)u_{i+1,j},\ \ i = 1, 2, 3, \ldots, n-1;\ \ j = 1, 2, 3, \ldots$$

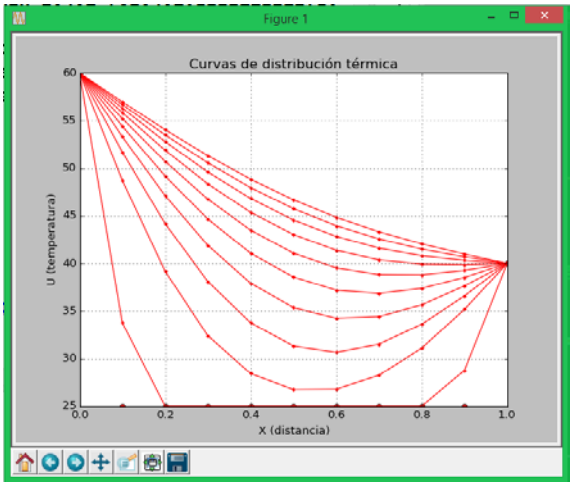
En donde **P, Q, R** dependen de los datos de la ecuación que se desea resolver.

Para el ejemplo propuesto se tiene: $u_{i,j+1} = \lambda u_{i-1,j} + (1-2\lambda)u_{i,j} + \lambda u_{i+1,j}$

```
# Método explícito de diferencias finitas para una EDP parabólica
# U(i,j+1)=(P)U(i-1,j) + (Q)U(i,j) + (R)U(i+1,j)

def edpdif(P,Q,R,U,m):
    u=[U[0]]
    for i in range(1,m):
        u=u+[P*U[i-1]+Q*U[i]+R*U[i+1]]      # Cálculo de puntos
    u=u+[U[m]]
    return u

import pylab as pl
m=10                            # Número de puntos en x
n=100                           # Número de niveles en t
Ta=60; Tb=40                    # Condiciones en los bordes
To=25                           # Condición en el inicio
dx=0.1; dt=0.01                 # incrementos
L=1                             # longitud
k=4                             # dato especificado
U=[Ta]                          # Asignación inicial
for i in range(1,m):
   U=U+[To]
U=U+[Tb]
lamb=dt/(k*dx**2)               #Parámetro lambda
P=lamb
Q=1-2*lamb
R=lamb
```

```
pl.title('Curvas de distribución térmica');
pl.xlabel('X (distancia)');
pl.ylabel('U (temperatura)')
x=[0]
for i in range(1,m+1):
    x=x+[i*dx]                # Coordenadas para el gráfico
pl.plot(x,U,'or')             # Distribución inicial
for j in range(n):
    U=edpdif(P,Q,R,U,m)
    if j%10==0:
        pl.plot(x,U,'-r');    # curvas cada 10 niveles de t
        pl.plot(x,U,'.r')     # puntos
pl.grid(True)
pl.show()
```

Resultado gráfico



Curvas de distribución térmica para el ejemplo.
La distribución tiende a una distribución lineal.

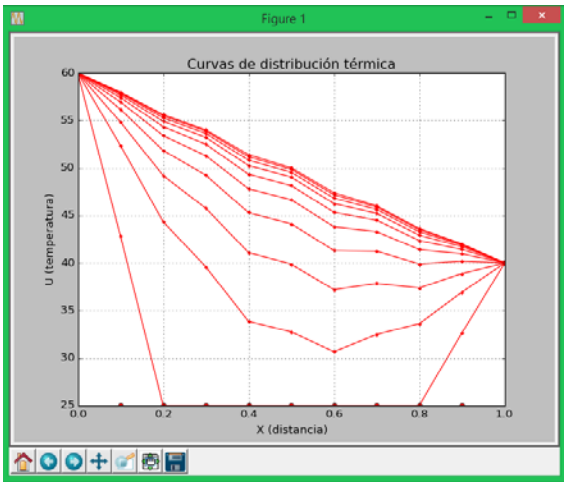
Los resultados numéricos están almacenados en los vectores **x, U**

Para esta prueba se utilizó $\Delta x = 0.1$, $\Delta t = 0.01$ por consistencia numérica

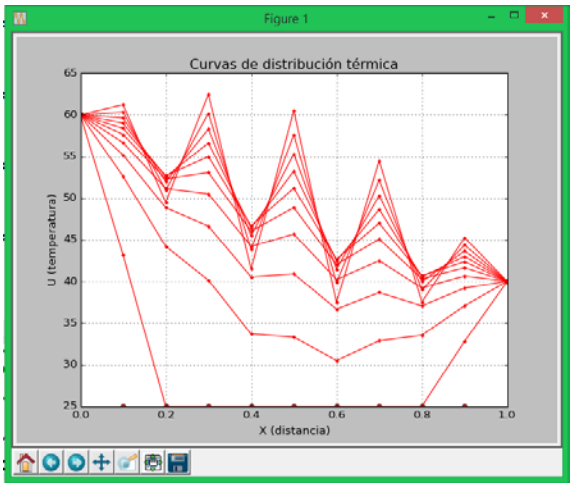
Por la condición de estabilidad: $\lambda = \frac{\Delta t}{k(\Delta x)^2} = \frac{0.01}{4(0.1)^2} = 0.25 < \frac{1}{2}$ se cumple la condición.

Con la instrumentación se puede verificar que al incrementar Δt a un valor tal que $\lambda > 0.5$, ya no se cumple la condición y el método se hace inestable. En este caso, los resultados que se obtienen se distorsionan. Se puede verificar con $\lambda = 0.51,\ 0.52,\ \ldots$

Prueba con $\lambda = 0.51$. La solución comienza a distorsionarse



Prueba con $\lambda = 0.52$. La solución se ha distorsionado y es incoherente.



10.2.4 Un esquema de diferencias finitas implícito

En un segundo intento elegimos las aproximaciones (8.5) y (8.3) para sustituir las derivadas de la ecuación diferencial

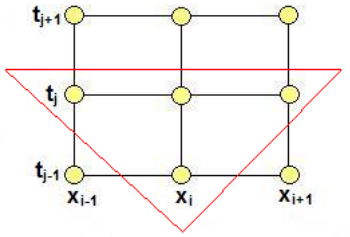
$$\frac{\partial^2 u}{\partial x^2} = k\frac{\partial u}{\partial t} \quad \Rightarrow \quad \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2 = k\frac{u_{i,j} - u_{i,j-1}}{\Delta t} + O(\Delta t)$$

Se obtiene la ecuación de diferencias

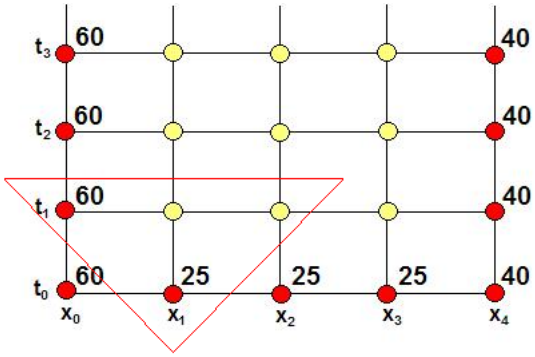
$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} = k\frac{u_{i,j} - u_{i,j-1}}{\Delta t}$$

Su error de truncamiento es igualmente $\mathbf{E = O(\Delta x)^2 + O(\Delta t)}$, debiendo cumplirse que $\mathbf{\Delta t < \Delta x}$

Marcamos los puntos incluidos en esta ecuación de diferencias.



Los puntos marcados conforman un triángulo de puntos invertido. Este triángulo correspondiente a los puntos incluidos en la ecuación de diferencias y puede colocarse en cualquier lugar de la malla asignando a **i, j** los valores apropiados. Por ejemplo, si **i=1, j=1**, la ecuación de diferencias se aplicada en el extremo inferior izquierdo de la malla:



La ecuación de diferencias ahora contiene dos puntos desconocidos. Si la ecuación se aplica sucesivamente a los puntos **`i=2, j=1`** e **`i=3, j=1`**, se obtendrá un sistema de tres ecuaciones lineales y su solución proporcionará el valor de los tres puntos desconocidos.

Esta ecuación de diferencias genera un **método implícito** para obtener la solución.

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} = k\frac{u_{i,j} - u_{i,j-1}}{\Delta t}, \quad E = O(\Delta x)^2 + O(\Delta t), \quad \Delta t < \Delta x$$

$$\frac{\Delta t}{k(\Delta x)^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) = u_{i,j} - u_{i,j-1}$$

Definiendo $\lambda = \frac{\Delta t}{k(\Delta x)^2}$, la ecuación se puede escribir

$$\lambda u_{i-1,j} + (-1-2\lambda)u_{i,j} + \lambda u_{i+1,j} = -u_{i,j-1}, \quad i = 1, 2, 3; \quad j = 1, 2, 3, \ldots$$

Finalmente con los datos $\lambda = \frac{\Delta t}{k(\Delta x)^2} = \frac{0.1}{4(0.25)^2} = 0.4$, se obtiene la forma final

$$0.4u_{i-1,j} - 1.8u_{i,j} + 0.4u_{i+1,j} = -u_{i,j-1}, \quad i = 1, 2, 3; \quad j = 1, 2, 3, \ldots$$

La cual genera un sistema de ecuaciones lineales para obtener la solución en cada nivel t_j

Calcular un nivel de la solución con este método:

j = 1, i = 1:	$0.4u_{0,1} - 1.8u_{1,1} + 0.4u_{2,1} = -u_{1,0}$	
	$0.4(60) - 1.8u_{1,1} + 0.4u_{2,1} = -25$	$\Rightarrow \quad -1.8\,u_{1,1} + 0.4u_{2,1} = -49$
i = 2:	$0.4u_{1,1} - 1.8u_{2,1} + 0.4u_{3,1} = -u_{2,0}$	
	$0.4u_{1,1} - 1.8u_{2,1} + 0.4u_{3,1} = -25$	$\Rightarrow \quad 0.4u_{1,1} - 1.8\,u_{2,1} + 0.4u_{3,1} = -25$
i = 3:	$0.4u_{2,1} - 1.8u_{3,1} + 0.4u_{4,1} = -u_{3,0}$	
	$0.4u_{2,1} - 1.8u_{3,1} + 0.4(40) = -25$	$\Rightarrow \quad 0.4u_{2,1} - 1.8\,u_{3,1} = -41$

Se tiene el sistema lineal

$$\begin{bmatrix} -1.8 & 0.4 & 0 \\ 0.4 & -1.8 & 0.4 \\ 0 & 0.4 & -1.8 \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \end{bmatrix} = \begin{bmatrix} -49 \\ -25 \\ -41 \end{bmatrix}$$

Cuya solución es

$$\begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \end{bmatrix} = \begin{bmatrix} 33.0287 \\ 26.1290 \\ 28.5842 \end{bmatrix}$$

Un análisis de estabilidad demuestra que el método implícito no está condicionado a la magnitud de $\lambda = \frac{\Delta t}{k(\Delta x)^2}$ como en el método explícito. Sin embargo, Δx y Δt con $\Delta t < \Delta x$ en aproximadamente un orden de magnitud deberían ser suficientemente pequeños para que la ecuación de diferencias sea consistente con la ecuación original.

10.2.5 Instrumentación computacional del método implícito para una E.D.P. de tipo parabólico

La siguiente instrumentación del método de diferencias finitas implícito permite resolver problemas de tipo similar al ejemplo anterior. La ecuación de diferencias debe escribirse en forma estandarizada

$$(P)u_{i-1,j} + (Q)u_{i,j} + (R)u_{i+1,j} = -u_{i,j-1}, \quad i = 1, 2, 3, \ldots, m-1; \; j = 1, 2, 3, \ldots$$

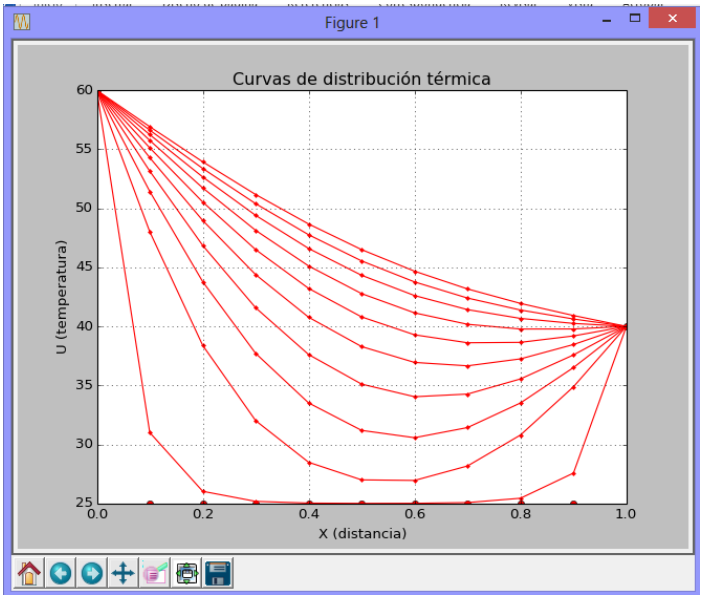
En donde **P**, **Q**, **R** dependen de los datos de la ecuación que se desea resolver.
Para el ejemplo propuesto se tiene: $\lambda u_{i-1,j} + (-1-2\lambda)u_{i,j} + \lambda u_{i+1,j} = -u_{i,j-1}$

Se define una función **EDPDIFPI** que genera y resuelve el sistema de ecuaciones lineales en cada nivel. El sistema de ecuaciones lineales resultante tiene forma tridiagonal, y por lo tanto se puede usar un algoritmo específico muy eficiente, la función **tridiagonal** cuya instrumentación fue realizada en capítulos anteriores.

```
# Método implícito de diferencias finitas para una EDP parabólica
# (P)U(i-1,j) + (Q)U(i,j) + (R)U(i+1,j) = -U(i,j-1)

from tridiagonal import*
def edpdifpi(P, Q, R, U, m):
# Método de Diferencias Finitas Implícito
    a=[];b=[];c=[];d=[]
    for i in range(m-2):
        a=a+[P]
        b=b+[Q]
        c=c+[R]
        d=d+[-U[i+1]]
    d[0]=d[0]-a[0]*U[0]
    d[m-3]=d[m-3]-c[m-3]*U[m-1]
    u=tridiagonal(a,b,c,d)
    U=[U[0]]+u+[U[m-1]]
    return U

import pylab as pl
m=11                          # Número ecuaciones: m-1
n=100                         # Número de niveles en t
Ta=60; Tb=40                  # Condiciones en los bordes
To=25                         # Condición en el inicio
dx=0.1; dt=0.01               # incrementos
L=1                           # longitud
k=4                           # dato especificado
U=[Ta]                        # Asignación inicial
```

```
for i in range(1,m-1):
   U=U+[To]
U=U+[Tb]
lamb=dt/(k*dx**2)
P=lamb
Q=-1-2*lamb
R=lamb
pl.title('Curvas de distribución térmica');
pl.xlabel('X (distancia)');
pl.ylabel('U (temperatura)')
x=[]
for i in range(m):
    x=x+[i*dx]                # Coordenadas para el gráfico
pl.plot(x,U,'or')             # Distribución inicial
for j in range(n):
    U=edpdifpi(P,Q,R,U,m)
    if j%10==0:
        pl.plot(x,U,'-r');    # curvas cada 5 niveles de t
        pl.plot(x,U,'.r')
pl.grid(True)
pl.show()
```

Resultado gráfico



Curvas de distribución térmica para el ejemplo. La distribución tiende a ser uniforme.

Los resultados numéricos están almacenados en los vectores **x, U**

10.2.6 Práctica computacional

Probar los métodos explícito e implícito instrumentados computacionalmente para resolver el ejemplo anterior cambiando Δx y Δt para analizar la convergencia de los esquemas de diferencias finitas.

Realizar pruebas para verificar la condición de estabilidad condicionada para el método explícito e incondicional para el método implícito. Probar con $\lambda = 0.4, 0.5, 0.6$

10.2.7 Condiciones variables en los bordes

Analizamos la ecuación anterior cambiando las condiciones inicial y en los bordes.

Suponer que inicialmente la barra se encuentra a una temperatura que depende de la posición mientras que en el borde derecho se aplica una fuente de calor que depende del tiempo y en el extremo izquierdo hay una pérdida de calor, según las siguientes especificaciones:

u(x,t): $\frac{\partial^2 u}{\partial x^2} = k\frac{\partial u}{\partial t}$, $0 \le x \le 1,\ t \ge 0$

$\frac{\partial u(0,t)}{\partial x} = -5,$ $t > 0$

$u(1, t) = 20 + 10\ sen(t),\ t \ge 0$

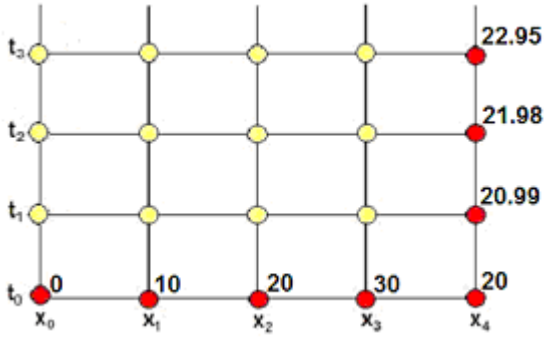
$u(x, 0) = 40\ x,$ $0 \le x \le 1$

$\Delta x = 0.25,\ \Delta t = 0.1,\ k = 4$

Para aplicar el método de diferencias finitas, debe discretizarse el dominio de **u**

$u(x_i, t_j) = u_{i,j}$; $i = 0, 1, ..., n;\ j = 0, 1, 2, ...$

La red con los datos incluidos en los bordes inferior y derecho:



Los puntos en color amarillo, en el borde izquierdo y en el centro, son desconocidos

Usamos el esquema de diferencias finitas implícito visto anteriormente:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} = k\frac{u_{i,j} - u_{i,j-1}}{\Delta t}, \quad E = O(\Delta x)^2 + O(\Delta t), \ \Delta t < \Delta x$$

$$\frac{\Delta t}{k(\Delta x)^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) = u_{i,j} - u_{i,j-1}$$

Definiendo $\lambda = \frac{\Delta t}{k(\Delta x)^2}$, la ecuación se puede escribir

$$\lambda u_{i-1,j} + (-1-2\lambda)u_{i,j} + \lambda u_{i+1,j} = -u_{i,j-1}, \ i = 1, 2, 3; \ j = 1, 2, 3, \ldots$$

Finalmente se tiene la ecuación de diferencias con los datos $\lambda = \frac{\Delta t}{k(\Delta x)^2} = \frac{0.1}{4(0.25)^2} = 0.4$,

Ahora la ecuación es aplicada en cada punto desconocido, incluyendo el borde izquierdo:

$$0.4u_{i-1,j} - 1.8u_{i,j} + 0.4u_{i+1,j} = -u_{i,j-1}, \ i = 0, 1, 2, 3; \ j = 1, 2, 3, \ldots$$

La cual genera un sistema de ecuaciones lineales para obtener la solución en cada nivel t_j

A continuación calculamos un nivel de la solución con este método:

j = 1,

i = 0: $0.4u_{-1,1} - 1.8u_{0,1} + 0.4u_{1,1} = -u_{0,0} \Rightarrow 0.4u_{-1,1} - 1.8u_{0,1} + 0.4u_{1,1} = 0$ **(1)**

i = 1: $0.4u_{0,1} - 1.8u_{1,1} + 0.4u_{2,1} = -u_{1,0} \Rightarrow 0.4u_{0,1} - 1.8u_{1,1} + 0.4u_{2,1} = -10$ **(2)**

i = 2: $0.4u_{1,1} - 1.8u_{2,1} + 0.4u_{3,1} = -u_{2,0} \Rightarrow 0.4u_{1,1} - 1.8u_{2,1} + 0.4u_{3,1} = -20$ **(3)**

i = 3: $0.4u_{2,1} - 1.8u_{3,1} + 0.4u_{4,1} = -u_{3,0}$

$0.4u_{2,1} - 1.8u_{3,1} + 0.4(20.99) = -30 \Rightarrow 0.4u_{2,1} - 1.8\,u_{3,1} = -38.4$ **(4)**

Se tiene un sistema de cuatro ecuaciones y cinco puntos desconocidos: $u_{-1,1}$, $u_{0,1}$, $u_{1,1}$, $u_{2,1}$, $u_{3,1}$ incluyendo el punto ficticio $u_{-1,1}$

El dato adicional conocido: $\frac{\partial u(0,t)}{\partial x} = -5$ es aproximado ahora mediante una fórmula de diferencias finitas central:

$$\frac{\partial u_{0,j}}{\partial x} = \frac{u_{1,j} - u_{-1,j}}{2(\Delta x)} = -5, \quad j = 1, 2, 3, \ldots, \ E = O(\Delta x)^2$$

De donde se obtiene $u_{-1,j} = u_{1,j} + 5(2\Delta x)$ para sustituir en la ecuación **(1)** anterior:

j = 1, i = 0: $0.4u_{-1,1} - 1.8u_{0,1} + 0.4u_{1,1} = 0$

$0.4[u_{1,j} + 5(2\Delta x)] - 1.8u_{0,1} + 0.4u_{1,1} = 0 \Rightarrow -1.8u_{0,1} + 0.8u_{1,1} = -1$

En notación matricial:

$$\begin{bmatrix} -1.8 & 0.8 & 0 & 0 \\ 0.4 & -1.8 & 0.4 & 0 \\ 0 & 0.4 & -1.8 & 0.4 \\ 0 & 0 & 0.4 & -1.8 \end{bmatrix} \begin{bmatrix} u_{0,1} \\ u_{1,1} \\ u_{2,1} \\ u_{3,1} \end{bmatrix} = \begin{bmatrix} -1 \\ -10 \\ -20 \\ -38.4 \end{bmatrix}$$

Cuya solución es: $u_{0,1} = 5.4667,\ u_{1,1} = 11.0500,\ u_{2,1} = 19.2584,\ u_{3,1} = 25.6130$

El sistema de ecuaciones lineales resultante tiene forma tridiagonal, y por lo tanto se puede usar un algoritmo específico muy eficiente para resolverlo.

10.2.8 Instrumentación computacional para una E.D.P. con derivadas en los bordes

La siguiente instrumentación del método de diferencias finitas implícito permite resolver problemas con una derivada en el borde izquierdo. Instrumentaciones similares se pueden desarrollar si la derivada está a la derecha o en ambos bordes.

Dada la ecuación diferencial parcial parabólica con condiciones en los bordes:

$$u=u(x,t),\quad F(t, x, u, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}) = 0,\quad u(x,t_0)=T_0,\quad \frac{\partial u(0,t)}{\partial x} = \delta_0,\quad u(1,t)=T_b$$

Luego de sustituir las derivadas para el método implícito se escribe la ecuación de diferencias estandarizada:

$$(P)u_{i-1,j} + (Q)u_{i,j} + (R)u_{i+1,j} = -u_{i,j-1}, \qquad i = 1, 2, 3, \ldots, m-1$$

Dato de la derivada en el borde izquierdo (i=0)

$$\frac{\partial u_{0,j}}{\partial x} = \frac{u_{1,j} - u_{-1,j}}{2(\Delta x)} = \delta_0 \quad \Rightarrow \quad u_{-1,j} = u_{1,j} - 2(\Delta x)\delta_0$$

Ecuación de diferencias aplicada en el borde izquierdo: $i = 0$

$$(P)u_{-1,j} + (Q)u_{0,j} + (R)u_{1,j} = -u_{i,j-1}$$

Reemplazo de la derivada:

$$(P)(u_{1,j} - 2(\Delta x)\delta_0) + (Q)u_{0,j} + (R)u_{1,j} = -u_{i,j-1}$$

$$(Q)u_{0,j} + (P+R)u_{1,j} = -u_{i,j-1} + (P)\,2(\Delta x)\delta_0$$

Ejemplo. Resolver computacionalmente la siguiente ecuación diferencial parcial con una derivada en el borde izquierdo

$$u(x,t): \quad \frac{\partial^2 u}{\partial x^2} = k\frac{\partial u}{\partial t}, \quad 0 \le x \le 1, \quad t \ge 0$$

$$\frac{\partial u(0,t)}{\partial x} = -5, \qquad t \ge 0$$

$$u(1,t) = 60, \quad t \ge 0$$

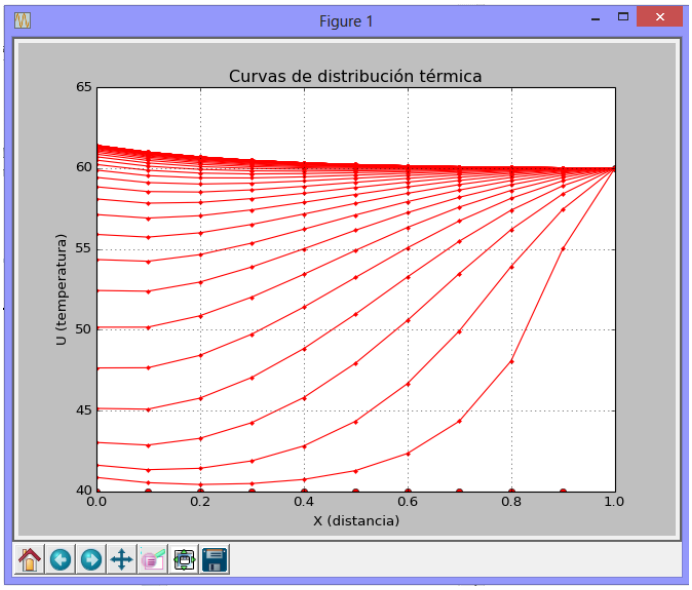
$$u(x,0) = 40, \quad 0 < x < 1$$

$$\Delta x = 0.1, \ \Delta t = 0.1, \ k = 4$$

```
# Solución de una EDP con una derivada en un borde
from tridiagonal import*
def edpdifpid(P,Q,R,U,der0,dx,m):
# Método de Diferencias Finitas Implícito
    a=[];b=[];c=[];d=[]
    for i in range(m-1):
        a=a+[P]
        b=b+[Q]
        c=c+[R]
        d=d+[-U[i+1]]
    c[0]=P+R;
    d[0]=d[0]+2*dx*P*der0
    d[m-2]=d[m-2]-c[m-2]*U[m-1]
    u=tridiagonal(a,b,c,d)
    U=u+[U[m-1]]
    return U

import pylab as pl
m=11                          # Número ecuaciones: m-1
n=50                          # Número de niveles en t
der0=-5                       # Derivada en el borde izquierdo
Tb=60                         # Condiciones en los bordes
To=40                         # Condición en el inicio
dx=0.1                        # incrementos
dt=0.1
L=1                           # longitud
k=4                           # dato especificado
U=[]                          # Asignación inicial
for i in range(m-1):
   U=U+[To]
U=U+[Tb]
lamb=dt/(k*dx**2)
P=lamb
Q=-1-2*lamb
R=lamb
```

```
pl.title('Curvas de distribución térmica');
pl.xlabel('X (distancia)');
pl.ylabel('U (temperatura)')
x=[]
for i in range(m):
    x=x+[i*dx]                          # Coordenadas para el gráfico
pl.plot(x,U,'or')                             # Distribución inicial
for j in range(n):
    U=edpdifpid(P,Q,R,U,der0,dx,m)
    pl.plot(x,U,'-r');                  # curvas cada 5 niveles de t
    pl.plot(x,U,'.r')
pl.grid(True)
pl.show()
```

Resultado gráfico



Curvas de distribución térmica

10.2.9 Método de diferencias finitas para EDP no lineales

Si la ecuación tiene términos no lineales, se puede adaptar un método de diferencias finitas explícito como una primera aproximación. Si se usa un método implícito, se obtendrá un sistema de ecuaciones no lineales el cual es un problema numérico complejo de formular y resolver.

Ejemplo. Formule un esquema de diferencias finitas explícito para resolver la siguiente ecuación diferencial parcial no lineal del campo de la acústica:

$$\mathbf{u(x,t):}\quad \frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - k\frac{\partial^2 u}{\partial x^2}$$

Solución

Si **x** es una variable delimitada y **t** no es acotada, la solución progresará en **t**

Se usarán las siguientes aproximaciones:

$$\frac{\partial u_{i,j}}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} + O(\Delta t)$$

$$\frac{\partial u_{i,j}}{\partial x} = \frac{u_{i+1,j} - u_{i-1,j}}{2(\Delta x)} + O(\Delta x)^2$$

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2$$

Sustituyendo en la EDP

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = -u_{i,j}\frac{u_{i+1,j} - u_{i-1,j}}{2(\Delta x)} - k\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2}$$

Con error de truncamiento: $E = O(\Delta t) + O(\Delta x)^2$, esto requiere que $\Delta t < \Delta x$

Se obtiene un esquema explícito de diferencias finitas que permitirá calcular cada punto en la malla del dominio discretizado de la ecuación diferencial. Previamente deben definirse las condiciones en los bordes así como los parámetros **k, Δt, Δx**. También debería analizarse la estabilidad del método, considerando especialmente que es un método explícito.

$$u_{i,j+1} = \Delta t\left(-u_{i,j}\frac{u_{i+1,j} - u_{i-1,j}}{2(\Delta x)} - c\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2}\right) + u_{i,j}$$

10.3 Ecuaciones diferenciales parciales de tipo elíptico

Este tipo de ecuaciones se caracteriza porque su dominio es una región cerrada. Para aplicar el método de diferencias finitas usaremos un ejemplo particular para posteriormente interpretar los resultados obtenidos.

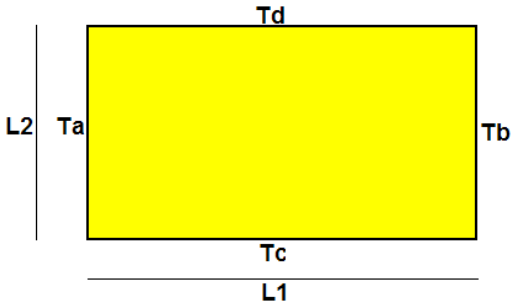
Ejemplo. Resolver la ecuación de difusión en dos dimensiones:

$$\mathbf{u(x,y):} \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Suponer que **u** es una función que depende de **x, y,** en donde **u** representa valores de temperatura, **x, y** representan posición.

Solución

Esta ecuación se puede asociar al flujo de calor en una placa muy delgada aislada térmicamente en sus caras superior e inferior y sometida en los bordes a alguna condición. La solución representa la distribución final de temperaturas en la placa en cada punto **(x, y)**



Ta, Tb, Tc, Td son valores de temperatura, suponer constantes, de alguna fuente de calor aplicada en cada borde de la placa. **L1, L2** son las dimensiones de la placa. ¿Porque no es necesario definir alguna constante que dependa de material de la placa?

Estas condiciones se pueden expresar simbólicamente en un sistema de coordenadas **X-Y:**

u(0, y) = Ta, **0 < y < L2**
u(L1, y) = Tb, **0 < y < L2**
u(x, 0) = Tc, **0 < x < L1**
u(x, L2) = Td, **0 < x < L1**

El valor en las esquinas es un promedio de los valores adyacente.

Para aplicar el método de diferencias finitas, debe discretizarse el dominio de **u** mediante una malla con puntos $\mathbf{u(x_i, y_j)}$, en la cual $\mathbf{x_i}$, $\mathbf{y_j}$ representan coordenadas

$$\mathbf{u=u(x,y),\ \ 0 \le x \le L1,\ \ 0 \le y \le L2 \Rightarrow\ \ u(x_i, y_j) = u_{i,j}\ ;\ i = 0, 1, ..., n;\ \ j = 0, 1, 2, ..., m}$$

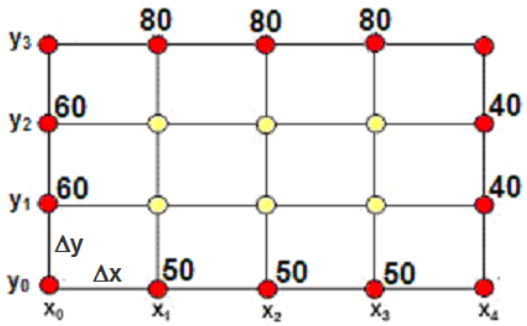
El método de diferencias finitas permitirá encontrar **u** en estos puntos.

Para el ejemplo supondremos los siguientes datos, en las unidades que correspondan

Ta = 60, **Tb = 40**
Tc = 50, **Td = 80**
L1 = 2, **L2 = 1.5**

Supondremos además que $\mathbf{\Delta x = 0.5,\ \Delta y = 0.5}$

Con esta información describimos el dominio de **u** mediante una malla con puntos en dos dimensiones en la cual el eje horizontal representa la posición x_i mientras que el eje vertical representa y_j. En esta malla se representan los datos en los bordes y los puntos interiores que deben ser calculados



10.3.1 Un esquema de diferencias finitas implícito

Elegimos las siguientes aproximaciones de diferencias finitas para sustituir las derivadas de la ecuación diferencial

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2$$

$$\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O(\Delta y)^2$$

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2 + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O(\Delta y)^2 = 0$$

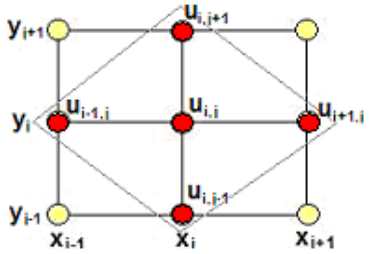
Se obtiene la ecuación de diferencias

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0$$

Con error de truncamiento $\mathbf{E = O(\Delta x)^2 + O(\Delta y)^2}$

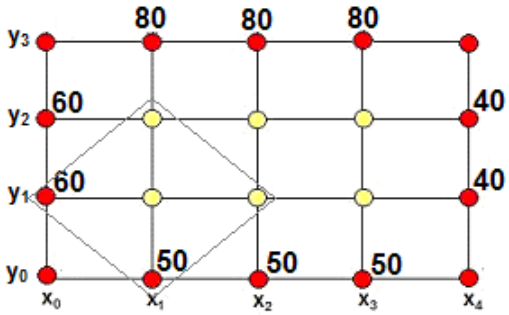
Para que esta sustitución sea consistente, **Δx** debe ser muy cercano a **Δy** en magnitud.

Es conveniente analizar los puntos incluidos en la ecuación de diferencias. Para esto consideramos un segmento de la malla y marcamos los puntos de la ecuación.



Los puntos marcados conforman un rombo. Este rombo describe los puntos incluidos en la ecuación de diferencias y puede colocarse en cualquier lugar de la malla asignando a **i, j** los valores apropiados.

Por ejemplo, si **i=1**, **j=1**, la ecuación de diferencias se aplica al extremo inferior izquierdo de la malla.



Se puede observar que la ecuación de diferencias contiene tres puntos desconocidos

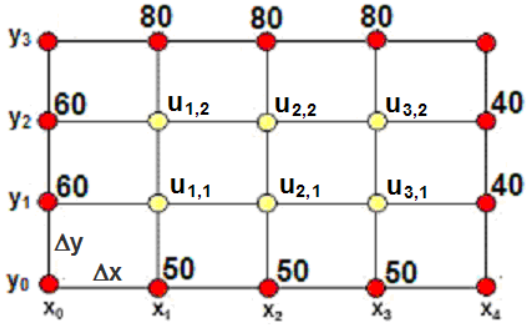
Si se aplica a todos los puntos interiores: **i = 1, 2, 3** con **j = 1, 2** se obtendrá un sistema de seis ecuaciones lineales con los seis puntos desconocidos cuyos valores se pueden determinar resolviendo el sistema. Por lo tanto, la ecuación de diferencias proporciona un **método implícito** para obtener la solución. Una simplificación adicional se obtiene haciendo $\Delta x = \Delta y$

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0$$

Forma final de la ecuación de diferencias:

$$u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} = 0 \,, \quad i = 1, 2, 3; \; j = 1, 2$$

Esta ecuación se aplica en cada punto desconocido de la malla



j = 1, i = 1: $60 + 50 + u_{2,1} + u_{1,2} - 4u_{1,1} = 0 \Rightarrow -4u_{1,1} + u_{2,1} + u_{1,2} = -110$

i = 2: $50 + u_{1,1} + u_{3,1} + u_{2,2} - 4u_{2,1} = 0 \Rightarrow u_{1,1} - 4u_{2,1} + u_{3,1} + u_{2,2} = -50$

i = 3: $50 + 40 + u_{3,2} + u_{2,1} - 4u_{3,1} = 0 \Rightarrow u_{2,1} - 4u_{3,1} + u_{3,2} = -90$

j = 2, i = 1: $60 + 80 + u_{1,1} + u_{2,2} - 4u_{1,2} = 0 \Rightarrow u_{1,1} - 4u_{1,2} + u_{2,2} = -140$

i = 2: $80 + u_{1,2} + u_{2,1} + u_{3,2} - 4u_{2,2} = 0 \Rightarrow u_{2,1} + u_{1,2} - 4u_{2,2} + u_{3,2} = -80$

i = 3: $80 + 40 + u_{2,2} + u_{3,1} - 4u_{3,2} = 0 \Rightarrow u_{3,1} + u_{2,2} - 4u_{3,2} = -120$

Se tiene obtiene un sistema de ecuaciones lineales

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,2} \\ u_{3,2} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \end{bmatrix} = \begin{bmatrix} -110 \\ -50 \\ -90 \\ -140 \\ -80 \\ -120 \end{bmatrix}$$

Cuya solución es

$$\begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \end{bmatrix} = \begin{bmatrix} 57.9917 \\ 56.1491 \\ 51.3251 \\ 65.8178 \\ 65.2795 \\ 59.1511 \end{bmatrix}$$

Se ha usado un método directo para resolver el sistema de este ejemplo particular

Planteamiento de un método iterativo de diferencias finitas para resolver la ecuación elíptica del ejemplo anterior.

Se observa en el sistema de ecuaciones resultante que en cada ecuación intervienen hasta cinco puntos de la red y su forma es diagonal dominante por lo que también se podrían usar métodos iterativos dado que la convergencia es segura.

Fórmula iterativa. Se despejan los componentes de la diagonal y se rotulan las iteraciones:

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)}\right), \quad k = 0, 1, 2, \ldots \qquad \text{(iteraciones)}$$

$$i = 1, 2, 3, \ldots \qquad j = 1, 2, 3, \ldots$$

Al observar esta fórmula, se concluye que en la fórmula iterativa el valor en cada punto es un promedio aritmético simple de los cuatro puntos a su alrededor. De manera análoga, esto corresponde a la interpretación física del problema.

Partiendo de una distribución inicial, el cálculo en cada punto debe repetirse hasta que la diferencia entre dos iteraciones consecutivas sea menor que una toleranvia especificada.

Para acelerar la convergencia, el valor inicial asignado a los puntos interiores puede ser el valor promedio de los valores en los bordes.

En la instrumentación computacional que se describe en esta sección, se prefiere desarrollar un método iterativo para obtener la solución puesto que no es simple instrumentar computacionalmente un generador de ecuaciones para los métodos directos.

10.3.2 Instrumentación computacional para una E.D.P. de tipo elíptico

La siguiente instrumentación en Python está diseñada para resolver una ecuación diferencial parcial elíptica con condiciones constantes en los bordes. Se supondrá además que **Δx = Δy**

Ecuación de diferencias

$$u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} = 0 \text{ , } i = 1, 2, 3, \ldots \quad ; \; j = 1, 2, 3, \ldots$$

Fórmula iterativa

$$u_{i,j}^{(k+1)} = \frac{1}{4}(u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)}) \text{ , } \quad k = 0, 1, 2, \ldots \qquad \text{(iteraciones)}$$

$$i = 1, 2, 3, \ldots \qquad j = 1, 2, 3, \ldots$$

En la siguiente instrumentación se usa el método de Gauss-Seidel para calcular la solución partiendo de valores iniciales iguales al promedio de los valores de los bordes.

```
# Programa para resolver una EDP Elíptica
# con condiciones constantes en los bordes
from numpy import*
Ta=60;Tb=60;Tc=50;Td=70     # Bordes izquierdo, derecho, abajo, arriba
n=10                        # Puntos interiores en dirección hor. (X)
m=10                        # Puntos interiores en dirección vert.(Y)
miter=100                   # Máximo de iteraciones
e=0.001                     # Error de truncamiento relativo 0.1%
u=zeros([n+2,m+2],float)
for i in range(n+2):
    u[i,0]=Tc
    u[i,m+1]=Td
for j in range(m+2):
    u[0,j]=Ta
    u[n+1,j]=Tb
p=0.25*(Ta+Tb+Tc+Td)            # valor inicial interior promedio
for i in range(1,n-1):
    for j in range(1,m-1):
        u[i,j]=p
```

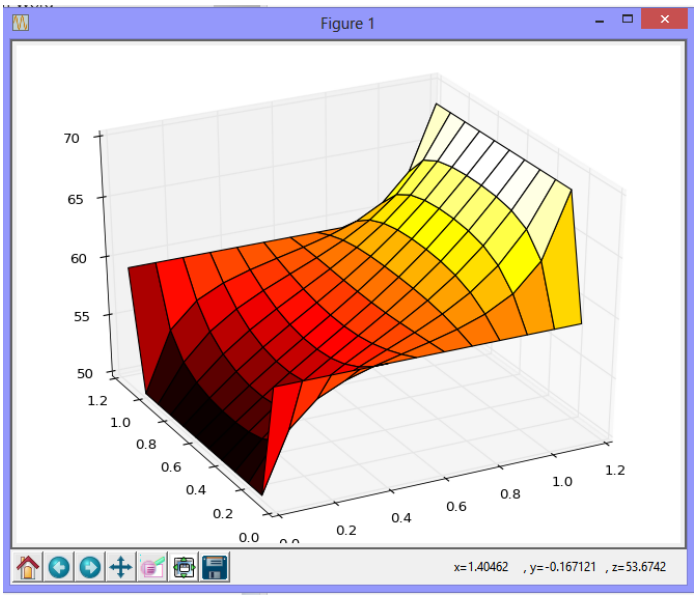
```
k=0                                  # conteo de iteraciones
converge=False                             # señal de convergencia
while k<miter and not converge:
      k=k+1
      t=u.copy()
      for i in range(1,n+1):
          for j in range(1,m+1):
              u[i,j]=0.25*(u[i-1,j]+u[i+1,j]+u[i,j+1]+u[i,j-1])
      if linalg.norm((u-t),inf)/linalg.norm(u,inf)<e:
          converge=True

if converge:
    for i in range(n+2):                  # Malla con la solución final
        print([float('%5.2f' % (u[i,j])) for j in range(m+2)])

    print('Conteo de iteraciones: ',k)    # Conteo de iteraciones

    import pylab as pl
    from mpl_toolkits.mplot3d import Axes3D      # Gráfico 3D
    fig=pl.figure()
    ax=Axes3D(fig)
    x=pl.arange(0,1.2,0.1)
    y=pl.arange(0,1.2,0.1)
    X,Y=pl.meshgrid(x,y)
    ax.plot_surface(X,Y,u,rstride=1,cstride=1,cmap='hot')
    pl.show()

else:
    print('No converge')
```

Cuadro de resultados de **u** (temperaturas en los nodos de la malla)

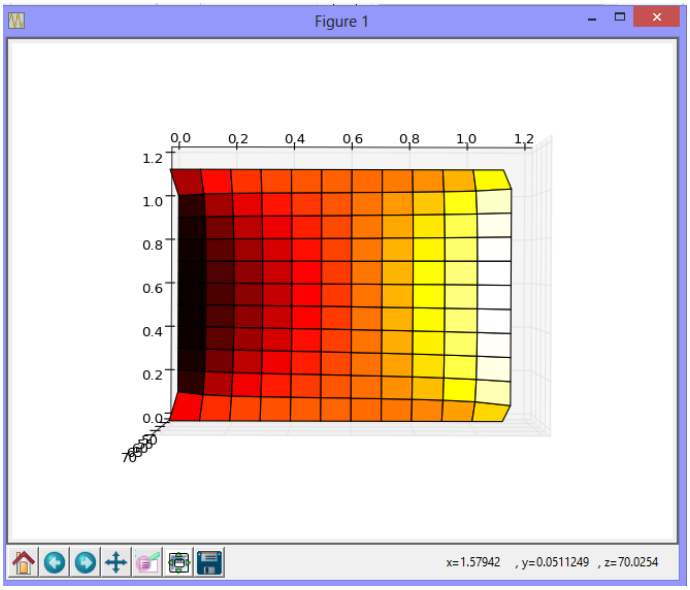
```
[60.0, 60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0]
[50.0, 55.06, 57.11, 58.15, 58.82, 59.36, 59.88, 60.48, 61.26, 62.47, 64.73, 70.0]
[50.0, 53.15, 55.29, 56.75, 57.86, 58.81, 59.77, 60.85, 62.17, 63.93, 66.44, 70.0]
[50.0, 52.31, 54.26, 55.83, 57.17, 58.42, 59.68, 61.08, 62.71, 64.7,  67.14, 70.0]
[50.0, 51.93, 53.71, 55.3,  56.76, 58.17, 59.63, 61.21, 63.01, 65.08, 67.44, 70.0]
[50.0, 51.78, 53.48, 55.08, 56.59, 58.07, 59.62, 61.29, 63.16, 65.25, 67.56, 70.0]
[50.0, 51.79, 53.51, 55.11, 56.63, 58.12, 59.66, 61.33, 63.19, 65.27, 67.57, 70.0]
[50.0, 51.97, 53.78, 55.4,  56.87, 58.29, 59.74, 61.32, 63.1,  65.14, 67.47, 70.0]
[50.0, 52.37, 54.35, 55.97, 57.33, 58.58, 59.85, 61.22, 62.83, 64.78, 67.18, 70.0]
[50.0, 53.2,  55.39, 56.89, 58.01, 58.98, 59.94, 60.99, 62.29, 64.01, 66.49, 70.0]
[50.0, 55.09, 57.18, 58.24, 58.92, 59.47, 59.99, 60.58, 61.34, 62.53, 64.75, 70.0]
[60.0, 60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0,  60.0]
```

Conteo de iteraciones: 41

Representación gráfica en tres dimensiones de la solución calculada **u**



Con el cursor se puede rotar el gráfico en la pantalla y observarlo desde diferentes perspectivas. La vista superior en el siguiente gráfico muestra la malla con los colores. Los colores mas claros representan mayor temperatura



Ejemplo. Proponer un **método de diferencias finitas iterativo** para resolver la ecuación de Poisson en tres dimensiones:

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}=f(x,y,z)$$

Plantear el método con condiciones en los bordes igual a cero y **f(x,y,z)=-10** en un cubo unitario. Utilzar $\Delta x=\Delta y=\Delta z=1/3$

Sustituyendo las aproximaciones de diferencias finitas extendidas a tres dimensiones:

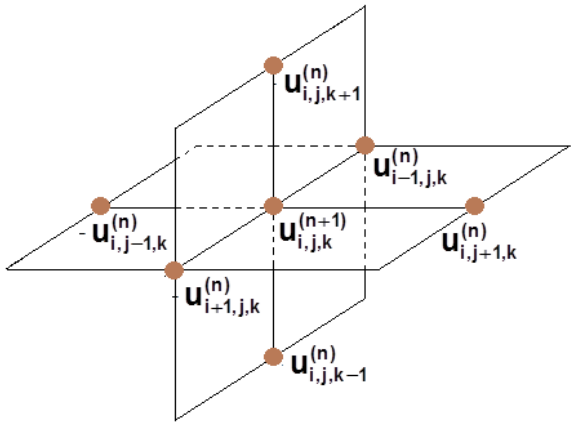
$$\frac{u_{i-1,j,k}-2u_{i,j,k}+u_{i+1,j,k}}{(\Delta x)^2}+\frac{u_{i,j-1,k}-2u_{i,j,k}+u_{i,j+1,k}}{(\Delta y)^2}+\frac{u_{i,j,k-1}-2u_{i,j,k}+u_{i,j,k+1}}{(\Delta z)^2}=-10$$

$$i=1,2,\quad j=1,2,\quad k=1,2$$

A partir de esta representación se puede proponer un método iterativo (Gauss-Seidel) como en el caso bidimensional anterior en esta sección:

$$u_{i,j,k}^{(n+1)}=\frac{1}{6}(u_{i-1,j,k}^{(n)}+u_{i+1,j,k}^{(n)}+u_{i,j-1,k}^{(n)}+u_{i,j+1,k}^{(n)}+u_{i,j,k-1}^{(n)}+u_{i,j,k+1}^{(n)})+\frac{10}{9},\quad n=0,1,2,\ldots \text{ (iteraciones)}$$

Malla tridimensional



Estado inicial (n=0): $\forall i \forall j \forall k\ (u_{i,j,k}^{(0)}=0)$

Iteraciones

n=0:

$$u_{1,1,1}^{(1)}=\frac{1}{6}(u_{0,1,1}^{(0)}+u_{2,1,1}^{(0)}+u_{1,0,1}^{(0)}+u_{1,2,1}^{(0)}+u_{1,1,0}^{(0)}+u_{1,1,2}^{(0)})+\frac{10}{9}$$

$$=\frac{1}{6}(0+0+0+0+0+0)+\frac{10}{9}=1.1111$$

$$u_{2,1,1}^{(1)} = \frac{1}{6}(u_{1,1,1}^{(1)} + u_{3,1,1}^{(0)} + u_{2,0,1}^{(0)} + u_{2,2,1}^{(0)} + u_{2,1,0}^{(0)} + u_{2,1,2}^{(0)}) + \frac{10}{9}$$
$$= \frac{1}{6}(1.1111 + 0 + 0 + 0 + 0 + 0) + \frac{10}{9} = 1.2963$$

Etc.

Ejemplo. Proponer un método de diferencias finitas directo para resolver la ecuación diferencial parcial de tipo elíptico:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \cos\,(x+y) + \cos\,(x-y) = 0\,,\ 0 \le x \le \pi/2\,,\ 0 \le y \le \pi/4$$

Sujeta a las condiciones de frontera:

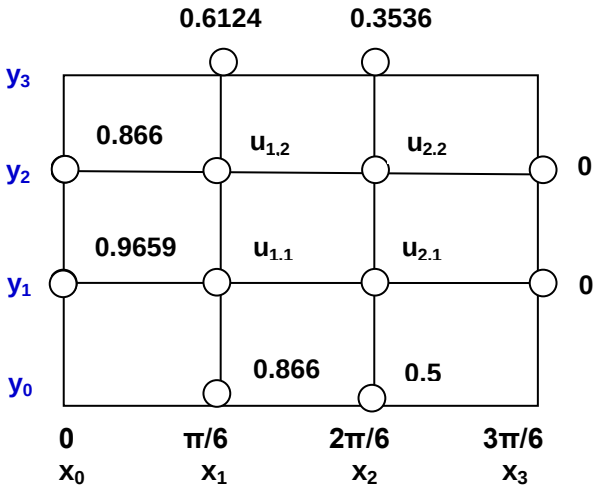
$$u(0,y) = \cos(y)\,,\ u(\pi/2, y) = 0\,,\ 0 \le y \le \pi/4$$

$$u(x,0) = \cos(x)\,,\ u(x,\pi/4) = \frac{\cos(x)}{\sqrt{2}}\,,\ 0 \le x \le \pi/2$$

Utilizar como tamaño de paso $\Delta x = \pi/6$ y $\Delta y = \pi/12$

Solución

Los datos se colocan en la malla



Sustituir las derivadas:

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2$$
$$\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O(\Delta y)^2$$

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + \cos(x_i + y_j) + \cos(x_i - y_j) = 0,$$

$$E = O(\Delta x)^2 + O(\Delta y)^2$$

$$(\Delta y)^2(u_{i+1.j} - 2u_{i,j} + u_{i-1,j}) + (\Delta x)^2(u_{i.j+1} - 2u_{i,j} + u_{i,j-1}) = -(\Delta x)^2(\Delta y)^2(\cos(x_i + y_j) + \cos(x_i - y_j))$$

$$(\Delta y)^2(u_{i+1.j} + u_{i-1,j}) + (\Delta x)^2(u_{i.j+1} + u_{i,j-1}) - 2((\Delta y)^2 + (\Delta x)^2)u_{i,j} = -(\Delta x)^2(\Delta y)^2(\cos(x_i + y_j) + \cos(x_i - y_j))$$

Aplicando esta ecuación de diferencias para **i=1,2; j=1,2** y sustituyendo **Δx, Δy** y las condiciones en los bordes, se obtendrá un sistema de cuatro ecuaciones lineales de cuya resolución se determinarán los cuatro puntos interiores.

10.4 Ecuaciones diferenciales parciales de tipo hiperbólico

En este tipo de ecuaciones el dominio está abierto en uno de los bordes Para aplicar el método de diferencias finitas usaremos un ejemplo particular para posteriormente interpretar los resultados obtenidos.

Ejemplo. Ecuación de la onda en una dimensión: **u(x, t):** $\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$

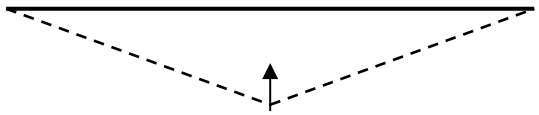
Suponer que **u** es una función que depende de **x, t** en donde **u** representa el desplazamiento vertical de la cuerda en función de la posición horizontal **x,** y el tiempo **t**, mientras que **c** es una constante. **L** es la longitud de la cuerda.

Suponer que los extremos de la cuerda están sujetos y que la longitud es **L = 1**

$$u = u(x, t),\ 0 < x < 1,\quad t \geq 0$$
$$u(0, t) = 0,\quad t \geq 0$$
$$u(1, t) = 0,\quad t \geq 0$$

Inicialmente la cuerda es estirada desplazando su punto central una distancia de **0.25**

$$u(x, 0) = \begin{cases} -0.5x, & 0 < x \leq 0.5 \\ 0.5(x-1), & 0.5 < x < 1 \end{cases}$$



Al soltar la cuerda sin velocidad, desde la posición indicada en el instante inicial:

$$\frac{\partial u(x,0)}{\partial t} = 0\ ,\quad 0 < x < 1$$

10.4.1 Un esquema de diferencias finitas explícito para resolver la EDP hiperbólica

Para resolver el ejemplo propuesto se usará un método de diferencias finitas explícito

El dominio se discretiza mediante una malla con puntos en dos dimensiones en la cual el eje horizontal representa la posición x_i mientras que el eje vertical representa el tiempo t_j.

$$u=u(x,t),\ 0 < x < L,\ t \geq 0 \quad \Rightarrow \quad u(x_i, t_j) = u_{i,j}\ ;\ i = 0, 1, ..., n;\ j = 0, 1, 2, ...$$

Elegimos las siguientes aproximaciones de diferencias finitas para las derivadas:

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta x)^2$$

$$\frac{\partial^2 u_{i,j}}{\partial t^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta t)^2} + O(\Delta t)^2$$

$$\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta t)^2} + O(\Delta x)^2 = c^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O(\Delta t)^2$$

Se obtiene la ecuación de diferencias:

$$\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta t)^2} = c^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2}$$

Cuyo error de truncamiento es $\mathbf{E = O(\Delta x)^2 + O(\Delta t)^2}$. Por consistencia numérica, Δx y Δt deberían ser aproximadamente del mismo orden.

Esta ecuación se puede expresar en la forma:

$$u_{i,j+1} - 2u_{i,j} + u_{i,j-1} = \frac{c^2(\Delta t)^2}{(\Delta x)^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$$

Mediante el análisis de estabilidad se demuestra que si $\frac{c^2(\Delta t)^2}{(\Delta x)^2} \leq 1$, la solución calculada con este método explícito de diferencias finitas es estable.

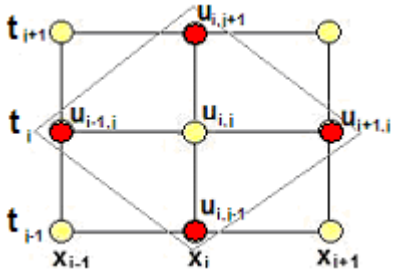
Para el ejemplo, suponer que $c = 2$, $\Delta x = 0.2$, entonces de la condición anterior se tiene:

$$\frac{2^2(\Delta t)^2}{(0.2)^2} = 1 \Rightarrow \Delta t = 0.1$$

Esta sustitución permite además simplificar:

$$u_{i,j+1} + u_{i,j-1} = u_{i+1,j} + u_{i-1,j}$$

La ecuación incluye cuatro puntos dispuestos en un rombo.



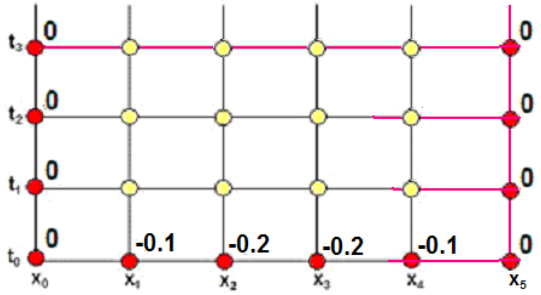
La solución progresa en la dirección **t** por lo que despejamos el punto en el vértice superior.

$$u_{i,j+1} = u_{i+1,j} + u_{i-1,j} - u_{i,j-1}, \quad i = 1, 2, 3, 4; \quad j = 1, 2, 3, \ldots \qquad (1)$$

Se puede notar observando el gráfico que para obtener cada nuevo punto de la solución se requieren conocer dos niveles previos de la solución

Describimos el dominio de **u** mediante una malla con puntos en dos dimensiones en la cual el eje horizontal representa la posición $\mathbf{x_i}$ mientras que el eje vertical representa tiempo $\mathbf{t_j}$.

En esta malla se representan los datos en los bordes y los puntos interiores que van a ser calculados



Siguiendo una estrategia usada anteriormente, expresamos el dato adicional $\frac{\partial u(x,0)}{\partial t} = 0$ mediante una fórmula de diferencias central:

$$\frac{\partial u_{i,0}}{\partial t} = \frac{u_{i,1} - u_{i,-1}}{2(\Delta t)} = 0 \quad \Rightarrow \quad u_{i,-1} = u_{i,1} \qquad (2)$$

Esta ecuación incluye el punto $u_{i,-1}$ entonces debe evaluarse la ecuación **(1)** en **t = 0**:

$$j = 0: \quad u_{i,1} = u_{i+1,0} + u_{i-1,0} - u_{i,-1}, \quad i = 1, 2, 3, 4$$

Sustituyendo en esta ecuación el resultado **(2)** se elimina el punto ficticio $u_{i,-1}$

$$u_{i,1} = u_{i+1,0} + u_{i-1,0} - u_{i,1}$$

$$u_{i,1} = \frac{1}{2}(u_{i+1,0} + u_{i-1,0}), \; i = 1, 2, 3, 4 \qquad (3)$$

La ecuación **(3)** debe aplicarse únicamente cuando t **= 0,** para encontrar $u_{i,j}$ en el nivel j=1

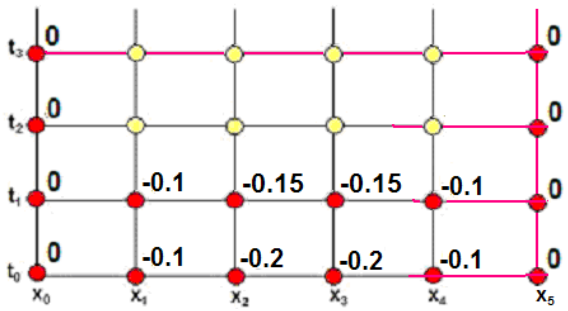
j = 0, i = 1: $u_{1,1} = \frac{1}{2}(u_{2,0} + u_{0,0}) = \frac{1}{2}(-0.2 + 0) = -0.1$

i = 2: $u_{2,1} = \frac{1}{2}(u_{3,0} + u_{1,0}) = \frac{1}{2}(-0.2 + (-0.1)) = -0.15$

i = 3: $u_{3,1} = \frac{1}{2}(u_{4,0} + u_{2,0}) = \frac{1}{2}(-0.1 + (-0.2)) = -0.15$

i = 4: $u_{4,1} = \frac{1}{2}(u_{5,0} + u_{3,0}) = \frac{1}{2}(0 + (-0.2)) = -0.1$

Los valores calculados son colocados en la malla:



Ahora se tienen dos niveles con puntos conocidos. A partir de aquí se debe usar únicamente la ecuación **(1)** como un esquema explícito para calcular directamente cada punto en los siguientes niveles **j,** cada nivel a una distancia $\Delta t = 0.1$

$$u_{i,j+1} = u_{i+1,j} + u_{i-1,j} - u_{i,j-1}, \quad i = 1, 2, 3, 4; \qquad j = 1, 2, 3, \ldots$$

j = 1, **i = 1:** $u_{1,2} = u_{2,1} + u_{0,1} - u_{1,0} = -0.15 + 0 - (-0.1) = -0.05$

i = 2: $u_{2,2} = u_{3,1} + u_{1,1} - u_{2,0} = -0.15 + (-0.1) - (-0.2) = -0.05$

i = 3: $u_{3,2} = u_{4,1} + u_{2,1} - u_{3,0} = -0.1 + (-0.15) - (-0.2) = -0.05$

i = 4: $u_{4,2} = u_{5,1} + u_{3,1} - u_{4,0} = 0 + (-0.15) - (-0.1) = -0.05$

j = 2, **i = 1:** $u_{1,3} = u_{2,2} + u_{0,2} - u_{1,1} = -0.05 + 0 - (-0.1) = 0.05$

i = 2: $u_{2,3} = u_{3,2} + u_{1,2} - u_{2,1} = -0.05 + (-0.05) - (-0.15) = 0.05$

i = 3: $u_{3,3} = u_{4,2} + u_{2,2} - u_{3,1} = -0.05 + (-0.05) - (-0.15) = 0.05$

i = 4: $u_{4,3} = u_{5,2} + u_{3,2} - u_{4,1} = 0 + (-0.05) - (-0.1) = 0.05$

j = 3, **i = 1:** $u_{1,4} = u_{2,3} + u_{0,3} - u_{1,2} = 0.05 + 0 - (-0.05) = 0.1$

i = 2: $u_{2,4} = u_{3,3} + u_{1,3} - u_{2,2} = 0.05 + 0.05 - (-0.05) = 0.15$

i = 3: $u_{3,4} = u_{4,3} + u_{2,3} - u_{3,2} = 0.05 + 0.05 - (-0.05) = 0.15$

i = 4: $u_{4,4} = u_{5,3} + u_{3,3} - u_{4,2} = 0 + 0.05 - (-0.05) = 0.1$

etc.

10.4.2 Instrumentación computacional para una E.D.P. de tipo hiperbólico

La siguiente instrumentación del método de diferencias finitas permite resolver problemas de tipo similar al ejemplo anterior: extremos fijos, estiramiento central, velocidad inicial nula y condición $\frac{c^2(\Delta t)^2}{(\Delta x)^2} = 1$ cuya ecuación de diferencias es:

$u_{i,1} = \frac{1}{2}(u_{i+1,0} + u_{i-1,0})$, **i = 1, 2, 3, . . . , m-1** (Para el primer nivel **j = 1**)

$u_{i,j+1} = u_{i+1,j} + u_{i-1,j} - u_{i,j-1}$ **i = 1, 2, 3, . . . , m-1** (Para los siguientes niveles **j = 2, 3, 4, ...**)

El esquema de cálculo utilizado es explícito. Cada punto es calculado directamente.

Se muestra la solución calculada numérica y gráficamente luego de calcular **n** niveles en la variable t.

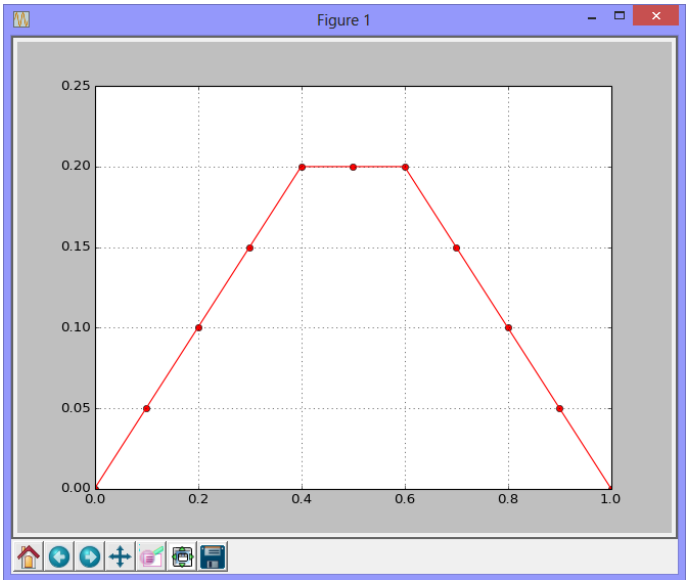
Los datos se especifican en el programa y se describen mediante anotaciones su significado para realizar otras pruebas.

```
# Método de Diferencias Finitas explícito: EDP Hiperbólica

from numpy import*
import pylab as pl
m=11                                  # Número de puntos en x
n=10                                  # Número de niveles en t
c=2                                   # dato especificado
L=1                                   # longitud
dx=L/(m-1)                            # incremento
dt=sqrt(dx**2/c**2)                    # para cumplir la condición
U0=zeros([m])                         # Extremos fijos

x=0
for i in range(1,m-1):          # Nivel inicial
    x=x+dx
    if x<L/2:
        U0[i]=-0.5*x           # Expresión para el desplazamiento
    else:
        U0[i]= 0.5*(x-1)

U1=[U0[0]]                            # Primer nivel
for i in range (1,m-1):
   U1=U1 + [0.5*(U0[i-1]+U0[i+1])]
U1=U1+[U0[m-1]]

for j in range(1,n+1):                     # Siguientes niveles
    Uj=[U1[0]]
    for i in range(1,m-1):
         Uj=Uj + [U1[i+1]+U1[i-1]-U0[i]]
    Uj=Uj + [U1[m-1]]
    U0=U1                             # Actualizar niveles anteriores
    U1=Uj

    # Mostrar la solución en cada nivel
    print('%4d'%j,[float('%5.2f' % (Uj[j])) for j in range(m)])

# Mostrar el gráfico de la solución en el último nivel
x=[]
for i in range(m):
    x=x+[i*dx]                        # Coordenadas para el gráfico

pl.grid(True)
pl.plot(x,Uj,'or')                    # Graficar puntos y cuerda
pl.plot(x,Uj,'-r')
pl.show()
```

Resultados numéricos en cada nivel (posición de los puntos de la cuerda)

```
 1 [0.0, -0.05, -0.1,  -0.15, -0.15, -0.15, -0.15, -0.15, -0.1,  -0.05, 0.0]
 2 [0.0, -0.05, -0.1,  -0.1,  -0.1,  -0.1,  -0.1,  -0.1,  -0.1,  -0.05, 0.0]
 3 [0.0, -0.05, -0.05, -0.05, -0.05, -0.05, -0.05, -0.05, -0.05, -0.05, 0.0]
 4 [0.0,  0.0,  -0.0,   0.0,  -0.0,   0.0,   0.0,   0.0,   0.0,   0.0,  0.0]
 5 [0.0,  0.05,  0.05,  0.05,  0.05,  0.05,  0.05,  0.05,  0.05,  0.05, 0.0]
 6 [0.0,  0.05,  0.1,   0.1,   0.1,   0.1,   0.1,   0.1,   0.1,   0.05, 0.0]
 7 [0.0,  0.05,  0.1,   0.15,  0.15,  0.15,  0.15,  0.15,  0.1,   0.05, 0.0]
 8 [0.0,  0.05,  0.1,   0.15,  0.2,   0.2,   0.2,   0.15,  0.1,   0.05, 0.0]
 9 [0.0,  0.05,  0.1,   0.15,  0.2,   0.25,  0.2,   0.15,  0.1,   0.05, 0.0]
10 [0.0,  0.05,  0.1,   0.15,  0.2,   0.2,   0.2,   0.15,  0.1,   0.05, 0.0]
```

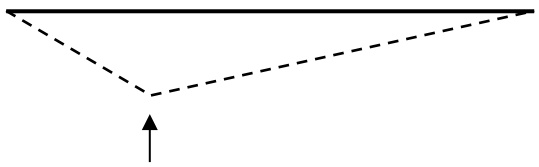
Resultado gráfico en el nivel **n = 10** (posición de los puntos de la cuerda)



En la siguiente prueba el desplazamiento inicial es asimétrico. En el punto **x=0.25**, se tensa **0.25** hacia abajo y se suelta la cuerda. Las otras condiciones se mantienen igual.

Ecuación que describa la cuerda en el instante inicial:

$$u(x,0) = \begin{cases} -x, & 0 < x \le 0.25 \\ \frac{1}{3}(x-1), & 0.25 < x < 1 \end{cases}$$



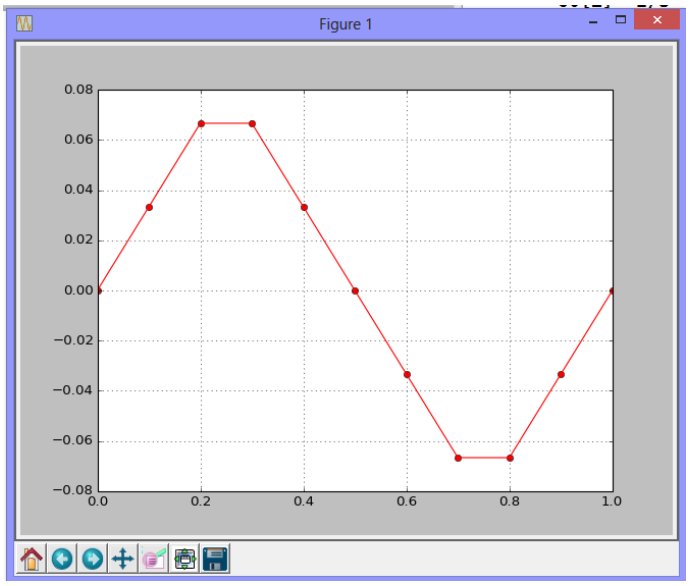
En el programa anterior debe hacerse la siguiente sustitución en las líneas para el desplazamiento inicial de la cuerda. Adicionalmente cambiar **n = 14**

```
    for i in range(1,m-1):
        x=x+dx
        if x<L/4:
            U0[i]=-x
        else:
            U0[i]= 1/3*(x-1)
```

Resultados numéricos en cada nivel (posición de los puntos de la cuerda)

```
 1 [0.0, -0.07, -0.1,  -0.13, -0.17, -0.17, -0.13, -0.1,  -0.07, -0.03, 0.0]
 2 [0.0, -0.0,  -0.03, -0.07, -0.1,  -0.13, -0.13, -0.1,  -0.07, -0.03, 0.0]
 3 [0.0,  0.03,  0.03,  0.0,  -0.03, -0.07, -0.1,  -0.1,  -0.07, -0.03, 0.0]
 4 [0.0,  0.03,  0.07,  0.07,  0.03, -0.0,  -0.03, -0.07, -0.07, -0.03, 0.0]
 5 [0.0,  0.03,  0.07,  0.1,   0.1,   0.07,  0.03,  0.0,  -0.03, -0.03, 0.0]
 6 [0.0,  0.03,  0.07,  0.1,   0.13,  0.13,  0.1,   0.07,  0.03, -0.0,  0.0]
 7 [0.0,  0.03,  0.07,  0.1,   0.13,  0.17,  0.17,  0.13,  0.1,   0.07, 0.0]
 8 [0.0,  0.03,  0.07,  0.1,   0.13,  0.17,  0.2,   0.2,   0.17,  0.1,  0.0]
 9 [0.0,  0.03,  0.07,  0.1,   0.13,  0.17,  0.2,   0.23,  0.2,   0.1,  0.0]
10 [0.0,  0.03,  0.07,  0.1,   0.13,  0.17,  0.2,   0.2,   0.17,  0.1,  0.0]
11 [0.0,  0.03,  0.07,  0.1,   0.13,  0.17,  0.17,  0.13,  0.1,   0.07, 0.0]
12 [0.0,  0.03,  0.07,  0.1,   0.13,  0.13,  0.1,   0.07,  0.03,  0.0,  0.0]
13 [0.0,  0.03,  0.07,  0.1,   0.1,   0.07,  0.03,  0.0,  -0.03, -0.03, 0.0]
14 [0.0,  0.03,  0.07,  0.07,  0.03,  0.0,  -0.03, -0.07, -0.07, -0.03, 0.0]
```

Resultado gráfico en el nivel **n = 14** (posición de los puntos de la cuerda)

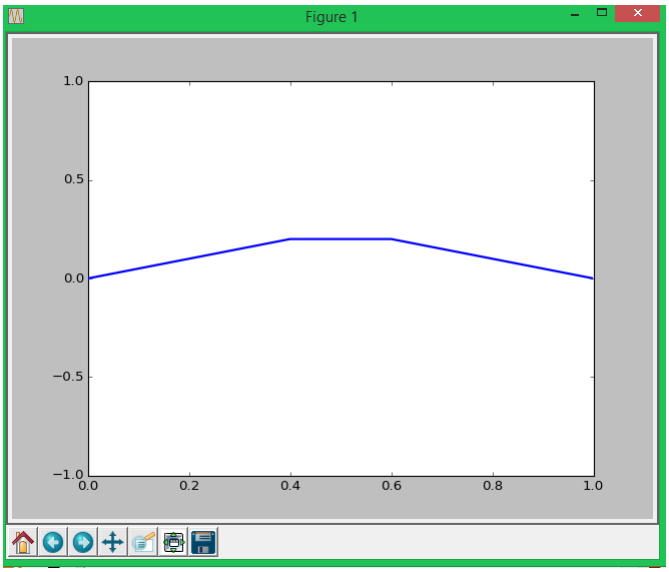


10.4.3 Instrumentación computacional con animación para una E.D.P. de tipo hiperbólico

En esta instrumentación del ejemplo anterior se utiliza la librería MatPlotLib para visualizar el efecto de movimiento de la cuerda

```
# Método de Diferencias Finitas explícito: EDP Hiperbólica
# Programa para animación de la figura

global m,c,L,dx,dt
m=51                                # Número de puntos en x
c=2                                 # dato especificado
L=1                                 # longitud
dx=L/(m-1)                          # incremento
dt=(dx**2/c**2)**0.5                # para cumplir la condición

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

def posicion_inicial():
    global U0,U1
    U0=np.zeros([m])                        # Extremos fijos
    x=0
    for i in range(1,m-1):                  # Nivel inicial
        x=x+dx
        if x<L/2:
            U0[i]=-0.5*x              # Expresión para el desplazamiento
        else:
            U0[i]= 0.5*(x-1)
    U1=[U0[0]]                             # Primer nivel
    for i in range (1,m-1):
       U1=U1 + [0.5*(U0[i-1]+U0[i+1])]
    U1=U1+[U0[m-1]]

# Definir la figura, ejes, y elemento que se va a animar
fig = plt.figure()
ax = plt.axes(xlim=(0,1), ylim=(-1,1))
linea, = ax.plot([], [], lw=2)

# Iniciar la función de animación
def inicio():
    linea.set_data([], [])
    return linea,
```

```
# Función con los datos de animación. Es llamada secuencialmente
def animar(j):
    global U0,U1
    Uj=[U1[0]]
    for i in range(1,m-1):
        Uj=Uj + [U1[i+1]+U1[i-1]-U0[i]]  # Posición actual de la cuerda
    Uj=Uj + [U1[m-1]]
    U0=U1                                # Actualizar niveles anteriores
    U1=Uj
    x=np.arange(0,1+dx,dx)
    y=Uj                                 # Coordenadas para el gráfico
    linea.set_data(x, y)
    return linea,

posicion_inicial()

# Animación. blit=True para redibujar solo las partes que han cambiado

anim = animation.FuncAnimation(fig, animar, init_func=inicio,
frames=100, interval=20, blit=True)
plt.show()
```



Instante de la cuerda en movimiento

10.5 Ejercicios con ecuaciones diferenciales parciales

1. Dada la siguiente ecuación diferencial parcial de tipo parabólico

$$\frac{\partial^2 u}{\partial x^2} = c\frac{\partial u}{\partial t}, \quad u = u(x,t), \quad 0 < x < 2, \quad t > 0,$$ con las condiciones

1) u(x,0)=25 sen(x), 0<x<2
2) u(0,t)=10 t, t≥0
3) $\frac{\partial u(2,t)}{\partial x} = 5, t \geq 0$

a) Calcule manualmente dos niveles de la solución con el método explícito de diferencias finitas. Utilice **Δx=0.5, c=1.6**

Para que el método sea estable use la condición $\frac{\Delta t}{c(\Delta x)^2} = \frac{1}{2}$

Use una aproximación de diferencias finitas de segundo orden para la condición 3)

b) Calcule dos niveles de la solución con el método implícito de diferencias finitas. Use las mismas especificaciones dadas en **1.**

2. Con el criterio de Neumann, analice la estabilidad del **método implícito** de diferencias finitas para resolver la EDP de tipo parabólico. Demuestre que es incondicionalmente estable.

$$\frac{\partial^2 u}{\partial x^2} = k\frac{\partial u}{\partial t} \quad \Rightarrow \quad \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} = k\frac{u_{i,j} - u_{i,j-1}}{\Delta t}$$

3. Resuelva la siguiente ecuación diferencial parcial de tipo elíptico con el método de diferencias finitas.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

u(0, y) = 10, 0≤y≤3
u(2, y) = 20 sen (πy), 0≤y≤3
$\frac{\partial u(x,0)}{\partial y} = 20$, **0≤x≤2**
u(x, 3) = 25x, 0≤x≤2

Determine la solución en los puntos interiores. Use **$\Delta x = \Delta y = 0.5$**

4. La siguiente ecuación diferencial parcial de tipo hiperbólico describe la posición **u** de cierta cuerda en cada punto **x**, en cada instante **t**

$$\frac{\partial^2 u}{\partial t^2} = (1+2x)\frac{\partial^2 u}{\partial x^2}, \quad 0<x<1,\ t>0$$

Use las siguientes condiciones iniciales y de borde:

$$u(x,0) = 0;\ 0 \le x \le 1$$

$$\frac{\partial u(x,0)}{\partial t} = x(1-x)$$

Use el método de diferencias finitas con $\Delta x = \Delta t = 0.25$, y encuentre la solución cuando $t = 1$

5. Resolver el siguiente problema de valor en la frontera:

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = x + y\ ,\ 0 < x < 1\ ,\ 0 < y < 1 \\ u(0,y) = 0\ ,\ u(1,y) = \dfrac{y^2}{6}\ ,\ 0 \le y \le 1 \\ u(x,0) = u(x,1) = \dfrac{1}{6}x^3\ ,\ 0 \le x \le 1 \end{cases}$$

a) Sustituya las derivadas por aproximaciones de diferencias finitas de segundo orden y simplifique

b) Construya la malla de nodos. Use $\Delta x=\Delta y=1/3$

c) Obtenga el sistema de ecuaciones aplicando las condiciones de frontera

d) Resuelva el sistema y especifique el valor de **u** en cada nodo interior de la malla

6. La ecuación de advección-difusión es un modelo que se usa para estudiar el problema de transporte de contaminantes. Calcule dos niveles de la siguiente ecuación de advección - difusión longitudinal usando un método de diferencias finitas implícito.

Use $\Delta x = 0.25,\ \Delta t = 0.1$

$$\frac{\partial u}{\partial t} + 2\frac{\partial u}{\partial x} = 5\frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le 1, \quad t \ge 0$$

$$u(0,x) = 10x, \quad 0 \le x \le 1$$

$$u(t,1) = 5, \quad t > 0$$

$$\frac{\partial u}{\partial x}(0,t) = 0, \quad t > 0$$

7. Se tiene la ecuación de calor en estado estable

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=\frac{x}{y}+\frac{y}{x}, \qquad R=(1,2)x(1,2)$$

Condiciones de frontera

$$u(x,1)=x\ln x;\ \ 1\le x\le 2$$
$$u(x,2)=x\ln(4x^2);\ \ 1\le x\le 2$$
$$u(1,y)=y\ln y;\ \ 1\le y\le 2$$
$$u(2,y)=2y\ln(2y);\ \ 1\le y\le 2$$

a) Plante el sistema asociado considerando $\Delta x = 0.1,\ \Delta y = 0.1$. (Solo planteo)

b) Encuentre las soluciones estimadas considerando $\Delta x = 0.25,\ \Delta y = 0.25$.

BIBLIOGRAFÍA

[1] Burden R., Faires J. ***Análisis Numérico***, Thomson Learning, 2002, Mexico

[2] Gerald, C., ***Applied Numerical Análysis***, Addison-Wesley Publishing Company

[3] Conte S., Boor C., ***Análisis Numérico Elemental***, McGraw-Hill

[4] Carnahan B., Luther H. Wilkes J., ***Applied Numerical Methods***, John Wiley & Sons, Inc

[5] Nakamura S., ***Análisis Numérico y Visualización Gráfica con MATLAB***, Prentice-Hall

[6] Rodríguez, L., ***Python Programación***, 2014

[7] Rodríguez, L., ***Análisis Numérico Básico con el soporte de MATLAB***, 2012