MÉTODOS NUMÉRICOS

Ejercicios prácticos con Matlab

Elena E. Álvarez Saiz
Universidad de Cantabria

RED educativa digital escartes

Métodos Numéricos

Ejercicios prácticos con Matlab

Elena E. Álvarez Saiz

Dpto. Matemática Aplicada y C. de la Computación

UC | Universidad de Cantabria

Fondo Editorial RED Descartes

RED educativa digital descartes .org proyecto descartes

Córdoba (España)
2025

Métodos Numéricos
Ejercicios prácticos con Matlab

Autores:
Elena E. Álvarez Saiz

Código JavaScript para el libro: Joel Espinosa Longi, IMATE, UNAM.
Recursos interactivos: DescartesJS, WebSim
Videos: Math-GPT
Fuentes: Lato y UbuntuMono
Imagen de portada: ilustración generada por Pollinations AI

Red Educativa Digital Descartes
Córdoba (España)
descartes@proyectodescartes.org
https://proyectodescartes.org

Proyecto iCartesiLibri
https://proyectodescartes.org/iCartesiLibri/

ISBN: 978-84-10368-24-8

Tabla de contenido

Prefacio

Los métodos numéricos son una herramienta esencial para quienes ejercen la ingeniería, ya que permiten resolver problemas complejos que a menudo no tienen una solución analítica simple. Estas técnicas posibilitan abordar situaciones reales mediante aproximaciones, convirtiéndose en un recurso indispensable en la práctica profesional.

Este libro está diseñado para quienes deseen adquirir una comprensión de los métodos numéricos para el cálculo de raíces de funciones no lineales, la aproximación de funciones mediante polinomios, la integración numérica y la resolución de ecuaciones diferenciales ordinarias.

Este libro ofrece:

- Explicaciones detalladas que conectan la teoría con la práctica, facilitando la comprensión de cómo y por qué funcionan los algoritmos.
- **230 ejemplos y ejercicios resueltos** paso a paso, que muestran cómo aplicar las técnicas numéricas en situaciones concretas.
- **38 herramientas interactivas** , gráficas y visuales que refuerzan las explicaciones y ayudan a entender mejor los conceptos abstractos.
- Una introducción progresiva a herramientas como Matlab/Octave, para que los lectores puedan implementar las soluciones y verificar sus propios resultados. Se incluye además una **herramienta para poder ejecutar código Octave online**, sin necesidad de instalar ningún programa que se complementa con un **asistente guiado por Inteligencia Artificial** para consultar sobre comandos y programación en Octave o Matlab. El acceso a esta herramienta se realiza desde el segundo icono que se muestra en el lateral derecho de cada página.

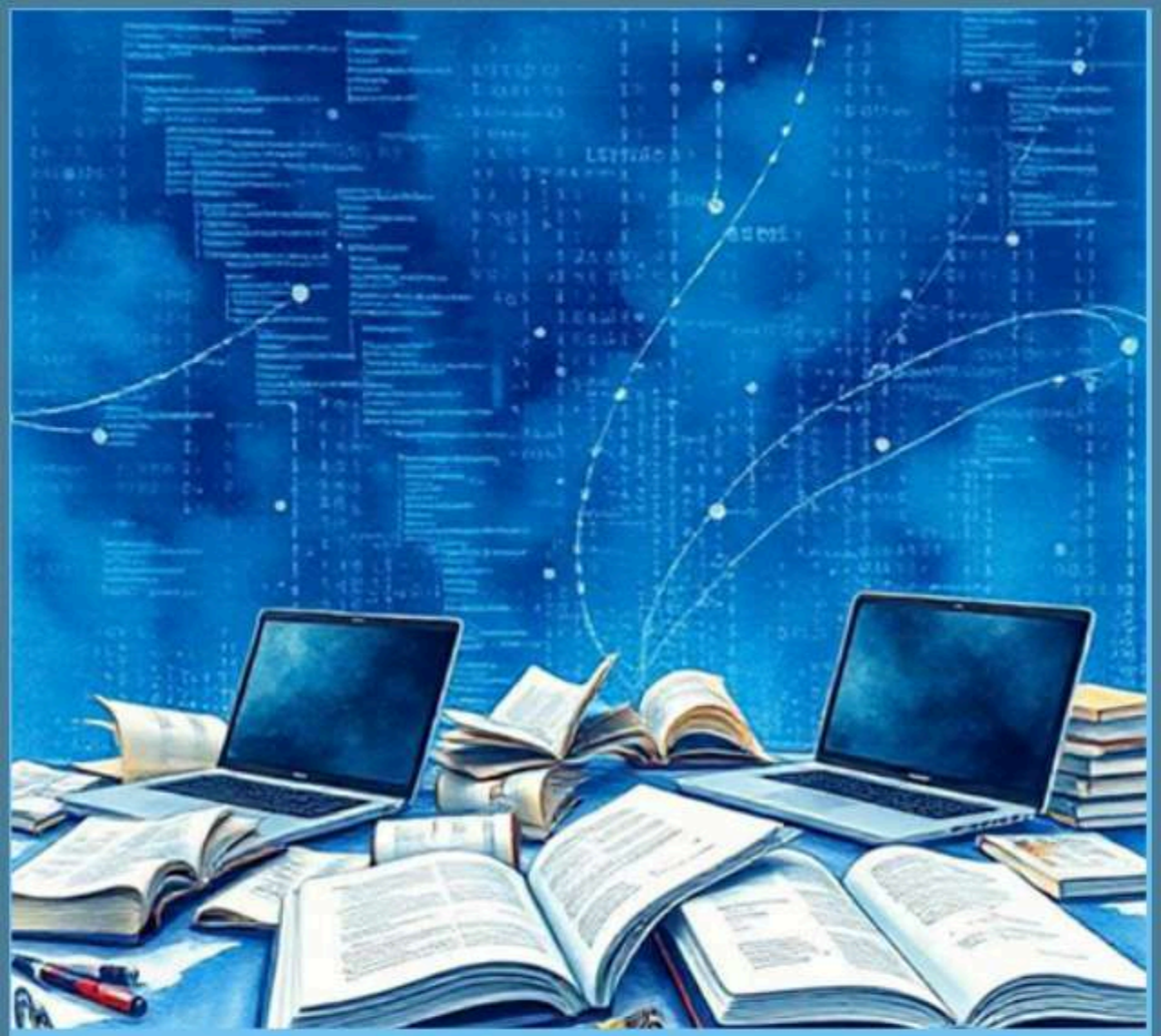
La orientación de este libro no solo pretende facilitar el aprendizaje, sino también despertar el interés por profundizar en el estudio de los métodos numéricos. La idea es acompañar al lector a través de una exposición clara y ordenada, donde cada contenido ayude a construir una base sólida que conecte los conceptos teóricos con su aplicación práctica.

Este libro toma como punto de partida los apuntes del profesor Eduardo Casas Rentería de la Universidad de Cantabria [6]. Su enfoque teórico ha servido de guía para estructurar los contenidos que aquí se presentan. El objetivo de este material es ofrecer una visión práctica, combinando explicaciones teóricas con ejemplos y ejercicios que ayuden a entender y dominar los conceptos básicos. Para apoyar los cálculos y facilitar el desarrollo de los ejemplos, se utiliza la programación en Matlab/Octave.

La estructura del libro se organiza en seis temas, que van desde los conceptos básicos hasta aplicaciones más avanzadas. El primer tema introduce los métodos numéricos y el uso de Matlab/Octave, sentando las bases para el aprendizaje y la implementación computacional. El segundo aborda la aritmética computacional básica, destacando los principios fundamentales para trabajar en un entorno digital. En el tercer tema se estudia la resolución aproximada de ecuaciones no lineales, explorando distintos métodos para encontrar raíces de funciones. El cuarto tema se centra en la aproximación de funciones mediante polinomios, una herramienta esencial en el análisis numérico. El quinto capítulo trata la integración numérica, presentando técnicas para calcular integrales de forma aproximada. Finalmente, se introduce la resolución de ecuaciones diferenciales ordinarias, ofreciendo métodos numéricos para su aproximación.

Soy consciente de que este libro puede contener algunas erratas o inconsistencias por lo que se anima a los lectores a compartir cualquier observación o corrección que consideren necesaria. Sus aportaciones serán de gran ayuda para mejorar futuras ediciones y asegurar que este recurso sea cada vez más útil.

Finalmente, quiero agradecer a la Red Educativa Digital Descartes y, de manera especial, a Juan Guillermo Rivera Berrío por compartir generosamente su conocimiento. Su experiencia y su compromiso con la educación han sido fundamentales para enriquecer este trabajo.



1 INTRODUCCIÓN A LOS MÉTODOS NUMERICOS Y A MATLAB

Capítulo

Introducción a los Métodos Numéricos y Matlab

1.1 Los métodos numéricos

El análisis numérico es una rama de las matemáticas que se ocupa de estudiar métodos y algoritmos para encontrar soluciones aproximadas a problemas que no pueden resolverse de manera exacta mediante técnicas analíticas tradicionales. Algunos de estos problemas incluyen la resolución de ecuaciones algebraicas y diferenciales, el cálculo de integrales o el tratamiento de sistemas de ecuaciones, tanto lineales como no lineales.

Al estudiar métodos numéricos, es importante considerar los siguientes aspectos:

- **Desarrollo de métodos.** El diseño y análisis de algoritmos que sean capaces de proporcionar soluciones aproximadas que sean lo suficientemente precisas para aplicarse a problemas reales.
- **Estimación y control del error.** Dado que las soluciones numéricas son aproximadas, resulta fundamental estudiar y medir los errores que se generan. Estos pueden deberse tanto a las limitaciones del propio método como a factores prácticos, como los errores de redondeo y truncamiento que aparecen al trabajar con la representación numérica en los computadores.
- **Convergencia de los métodos.** Para cada algoritmo, se debe analizar cómo se aproxima a la solución exacta a medida que aumenta el número de iteraciones o cálculos.

- **Estabilidad y rendimiento.** Un algoritmo numérico debe ser estable, lo que significa que pequeñas variaciones en los datos de entrada no deberían producir grandes diferencias en los resultados. Además, los métodos deben ser eficientes, es decir, ofrecer buenos tiempos de cálculo y utilizar los recursos computacionales de manera adecuada, algo especialmente importante cuando se enfrentan a problemas de gran tamaño o alta complejidad.

1.2 Ejemplo de introducción

Para ilustrar la aplicación de los métodos numéricos en problemas reales, se considera el caso de un sistema de baterías conectado a un panel solar que genera electricidad a lo largo del día. Se quiere modelar cómo la batería se carga con el tiempo, sabiendo que:

- La potencia solar $P_{solar}(t)$ varía a lo largo del día, dependiendo de la luz solar.
- La batería tiene una capacidad máxima de almacenamiento.
- La velocidad de carga disminuye a medida que la batería se acerca a su capacidad máxima (comportamiento típico de una batería).

La ecuación que podría modelizar cómo se almacena la energía en la batería podría ser la siguiente ecuación diferencial ordinaria (EDO):

$$\frac{dE}{dt} = P_{solar}(t) - kE(t)$$

donde:

- $E(t)$**:** Energía almacenada en la batería (kWh).
- $P_{solar}(t)$**:** Potencia solar disponible en función del tiempo (kW).

- k: Coeficiente de ineficiencia de la batería.
- t: Tiempo en horas.

Se recurre a técnicas numéricas para obtener una solución aproximada considerando los siguientes **datos y parámetros**:

- Capacidad máxima de la batería: 100 kWh.
- Coeficiente de ineficiencia (k): 0.05
- Tiempo de simulación: 24 horas.
- La potencia solar varía a lo largo del día considerando $t = 0$ el momento en el que sale el sol y tomando como valor máximo 50 kW a partir del cual disminuye siguiendo la función senoidal:

$$P_{solar}(t) = 50 \operatorname{sen}\left(\frac{\pi t}{12}\right)$$

La gráfica de la Figura 1.1 muestra la evolución de la energía almacenada en la batería en un día (24 horas) en función de la potencia solar disponible, utilizando como método de resolución el método de Euler que se analizará en detalle en el capítulo 6 de este libro. En la gráfica

- La línea azul muestra cómo la energía almacenada en la batería varía con el tiempo, dependiendo de la energía solar recibida.
- La línea discontinua naranja representa la potencia solar disponible a lo largo del día, que sigue una curva senoidal para simular el ciclo diurno.

Se observa cómo la batería se carga durante las horas de sol y comienza a descargarse lentamente debido a las ineficiencias cuando no hay suficiente energía solar disponible.

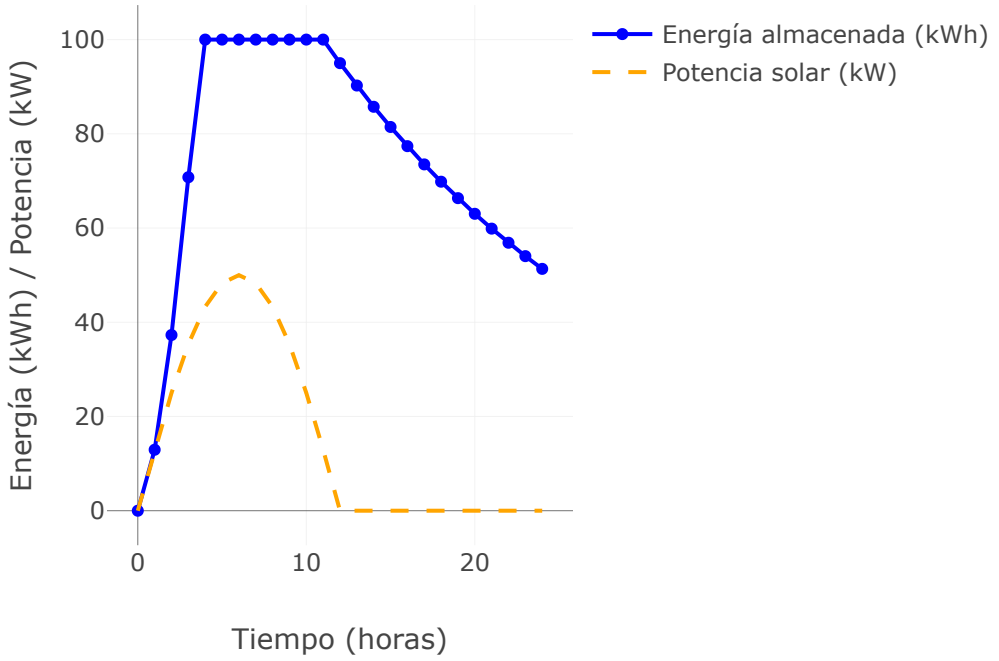


Figura 1.1. Simulación de almacenamiento de energía en la batería solar

¿En qué consiste el método numérico para obtener la solución $E(t)$?

El ejemplo descrito anteriormente responde a un problema de valor inicial (PVI) que busca encontrar la curva solución de una ecuación diferencial ordinaria de primer orden de la forma $\frac{dy}{dt} = f(t, y)$ con una condición inicial $y(t_0) = y_0$. La solución es la curva $y(t)$ que pasa por el punto (t_0, y_0) y tiene como pendiente de la curva en cualquier punto el valor establecido por la función f.

El **método de Euler**, que se verá en detalle en el capítulo 6, propone aproximar esta curva avanzando paso a paso a través de pequeños saltos en el eje x moviéndose suavemente por la recta tangente en cada instante ya que f permite calcular la pendiente a la curva en cada punto. En este proceso, se recalcula la dirección en cada paso.

A continuación se detalla el proceso a seguir en este método numérico, que se inicia desde el punto (t_0, y_0).

Pasos del Método de Euler

1. **Punto inicial:** Se parte del punto $y(t_0) = y_0$ dado por el PVI. Nuestra solución al problema es una curva que pasa por el punto (t_0, y_0).

2. **Aproximación por pasos:** A partir del punto inicial, se avanza con pasos pequeños Δt, calculando el valor aproximado de y en el siguiente punto usando la pendiente.

   Se obtendría así una lista de puntos (t_0, y_0), (t_1, y_1), ... (t_n, y_n) donde $t_n = t_{n-1} + \Delta t$

3. **Fórmula de actualización:** En cada paso, el valor de y se actualiza según la fórmula: $y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n)$ ya que $f(t_n, y_n)$ es la pendiente en el punto (t_n, y_n).

En este ejemplo introductorio que se está viendo, el problema de valor inicial es el siguiente,

$$\frac{dE}{dt} = P_{solar}(t) - kE(t) \qquad E(0) = 0$$

La aplicación del método de Euler para encontrar la solución de forma numérica consisté en aplicar la siguiente fórmula iterativa

$$E(n+1) = E(n) + \Delta t \cdot (P_{solar}(t_n) - k \cdot E(n))$$

donde Δt es el paso de tiempo (1 hora) y $E(n)$ es la energía almacenada en la batería en el instante n. Así, es posible calcular la energía almacenada en la batería a lo largo de 24 horas utilizando la potencia solar que varía con el tiempo. Se muestra seguidamente el cálculo de los primeros pasos.

Condición inicial:

- Energía inicial en la batería: $E(0) = 0\,\text{kWh}$

Paso 1: $t = 1\,\text{h}$

- $P_{\text{solar}}(0) = 50\sin(0) = 0\,\text{kW}$
- $E(1) = 0 + 1 \cdot (0 - 0.05 \cdot 0) = 0\,\text{kWh}$

Paso 2: $t = 2\,\text{h}$

- $P_{\text{solar}}(1) = 50\sin\left(\frac{\pi}{12} \cdot 1\right) \approx 12.94\,\text{kW}$
- $E(2) = 0 + 1 \cdot (12.94 - 0.05 \cdot 0) = 12.94\,\text{kWh}$

Paso 3: $t = 3\,\text{h}$

- $P_{\text{solar}}(2) = 50\sin\left(\frac{\pi}{12} \cdot 2\right) = 25\,\text{kW}$
- $E(3) = 12.94 + 1 \cdot (25 - 0.05 \cdot 12.94) \approx 37.29\,\text{kWh}$

Paso 4: $t = 4\,\mathrm{h}$

- $P_{\text{solar}}(3) = 50 \sin\left(\frac{\pi}{12} \cdot 3\right) \approx 36.60\,\mathrm{kW}$
- $E(4) = 37.29 + 1 \cdot (36.60 - 0.05 \cdot 37.29) \approx 72.24\,\mathrm{kWh}$

Paso 5: $t = 5\,\mathrm{h}$

- $P_{\text{solar}}(4) = 50 \sin\left(\frac{\pi}{12} \cdot 4\right) \approx 47.55\,\mathrm{kW}$
- $E(5) = 72.24 + 1 \cdot (47.55 - 0.05 \cdot 72.24) \approx 116.63\,\mathrm{kWh}$
  Pero, como la capacidad máxima de la batería es 100 kWh, se limita el valor a $E(5) = 100\,\mathrm{kWh}$.
  ...

El proceso continuaría realizando el cálculo para las 24 horas del día de forma similar.

1.3 Errores en cálculos numéricos

Los errores en métodos numéricos, aunque pequeños al inicio, pueden crecer y afectar gravemente los resultados. Comprenderlos es clave para evitar fallos. A continuación, se muestran ejemplos históricos con consecuencias significativas.

Misiles Patriot

El 25 de febrero de 1991, durante la Guerra del Golfo, una batería de misiles Patriot del ejército estadounidense ubicada en Dharan, Arabia Saudita, falló al intentar interceptar un misil Scud lanzado por las fuerzas iraquíes. El misil impactó en un cuartel del ejército estadounidense, causando la muerte de 28 soldados. Un informe de la entonces Oficina General de Contabilidad, titulado "Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia", identificó la causa del fallo: un error aritmético en los cálculos de tiempo del sistema, que provocó un desfase desde el inicio de la operación [1].

Figura 1.2. Misil Patriot [1]

Concretamente, el reloj interno del sistema medía el tiempo en décimas de segundo, y este valor se multiplicaba por 10 para obtener el tiempo en segundos. El cálculo se realizaba usando un registro de punto fijo de 24 bits. Como 1/10 tiene una expansión binaria infinita, se representó truncándolo a 24 bits después del punto decimal. Aunque el error de truncamiento era muy pequeño, al multiplicarse por el número grande que representaba el tiempo en décimas de segundo, acabó generando un error apreciable [1].

La explosión del cohete Ariane

El Programa Ariane comenzó en 1973 como una iniciativa europea para desarrollar un lanzador propio que asegurara su acceso independiente al espacio. El proyecto se llevó a cabo bajo la supervisión de la Agencia Espacial Europea (ESA, por sus siglas en inglés). Hasta mayo de 2003, el contratista principal fue el Centro Nacional de Estudios Espaciales (CNES) de Francia, momento en el que esta responsabilidad pasó al consorcio europeo EADS (European Aeronautic Defence and Space Company). [2].

La gestión comercial del lanzador Ariane estuvo a cargo de la sociedad Arianespace, creada en 1980, y en su desarrollo y construcción participaron alrededor de 40 compañías europeas. Todos los lanzamientos se realizaban desde el centro espacial de Kourou, en la Guayana Francesa, que disponía de varias plataformas de despegue y contaba con varios cientos de trabajadores de forma permanente.

El 4 de junio de 1996, el cohete Ariane 5 Flight 501, de la Agencia Espacial Europea (ESA), explotó apenas 40 segundos después de su despegue, a una altitud de 3,7 km, tras desviarse de la trayectoria prevista.

Figura 1.3. Cohete Ariane [2]

El vuelo del Ariane 5 Flight 501 era su primer lanzamiento tras una década de desarrollo, que había supuesto una inversión de más de 7000 millones de euros. Tanto el cohete como su carga estaban valorados en más de 500 millones de euros. La explosión fue provocada por un fallo en el sistema de guiado de la trayectoria, ocurrido 37 segundos después del despegue. La causa del error se localizó en el software encargado de controlar el sistema de referencia inercial (SRI).

En concreto, el fallo se produjo por una excepción al intentar convertir un número en punto flotante de 64 bits —relacionado con la velocidad horizontal del cohete respecto a la plataforma de lanzamiento— en un entero con signo de 16 bits. El problema fue que el valor que se intentaba almacenar superaba el máximo que puede representarse en 16 bits con signo, que es 32.767.

Para más información, se puede consultar [12].

El hundimiento de la plataforma petrolífera Sleipner A

El 23 de agosto de 1991, la plataforma petrolífera Sleipner A, propiedad de la empresa noruega Statoil, se hundió en el mar del Norte a 82 metros de profundidad. El accidente se debió a un error en el modelado numérico de la estructura mediante elementos finitos, que provocó una fuga de agua en uno de los 24 tanques de aire de 12 metros de diámetro, esenciales para la flotabilidad. La plataforma, que soportaba 57.000 toneladas de peso, más 200 personas y 40.000 toneladas de equipamiento, no pudo ser salvada debido a la incapacidad de las bombas de achique para evacuar el agua. El hundimiento supuso un coste estimado de 700 millones de euros [3].

Para modelar los tanques de la plataforma se utilizó el programa NASTRAN, basado en el método de elementos finitos, aplicando una aproximación mediante un modelo elástico lineal. Sin embargo, esta aproximación resultó inadecuada y subestimó en un 47% los esfuerzos que debían soportar las paredes de los tanques. Como consecuencia, algunas paredes fueron diseñadas con un grosor insuficiente.

Un análisis posterior al accidente, realizado mediante el método de elementos finitos de forma más precisa, demostró que el diseño de la plataforma provocaría fugas en algunos de los tanques cuando estuviera sobre una lámina de agua de 62 metros de profundidad. La fuga real se produjo cuando la plataforma se encontraba sobre 65 metros de agua, lo que confirmó la causa del fallo.

Figura 1.4. Plataforma petrolífera [3]

1.4 Introducción a Matlab/Octave

Para desarrollar y resolver los ejercicios y ejemplos propuestos en este libro, se utilizará como herramienta principal de cálculo el programa Matlab. Su interfaz intuitiva y su amplia biblioteca de funciones lo convierten en una herramienta ideal para ilustrar conceptos matemáticos, realizar simulaciones y aplicar métodos computacionales en un entorno práctico.

Matlab es un acrónimo de "Matrix Laboratory", lo que refleja su origen como una herramienta diseñada para trabajar principalmente con matrices. Con el tiempo, el programa ha evolucionado para convertirse en un entorno de programación de alto nivel que permite realizar cálculos numéricos, análisis de datos, simulaciones, desarrollo de algoritmos y visualizar información.

Aunque Matlab es un software de pago, también se puede utilizar el software libre y gratuito Octave, que es compatible en muchos de sus comandos. Para facilitar la ejecución del código propuesto en los ejemplos y ejercicios, se pueden utilizar los siguientes enlaces sin necesidad de realizar ninguna instalación:

- octave online
- myCompiler

A continuación, se resumen las principales características de Matlab/Octave necesarias para seguir y comprender los ejemplos de los próximos capítulos. Para una explicación más detallada, se recomienda consultar la documentación oficial o la bibliografía recomendada [8].

Entorno de trabajo

La **ventana de Matlab** muestra un escritorio dividido en varias partes:

- Las órdenes se escriben en la ventana de comandos, **Command Window**.
- La ventana **Workspace** proporciona información sobre las variables utilizadas y permite explorar datos que cree o importe de archivos.
- **Current Folder**es la carpeta actual donde se ejecutan los archivos.

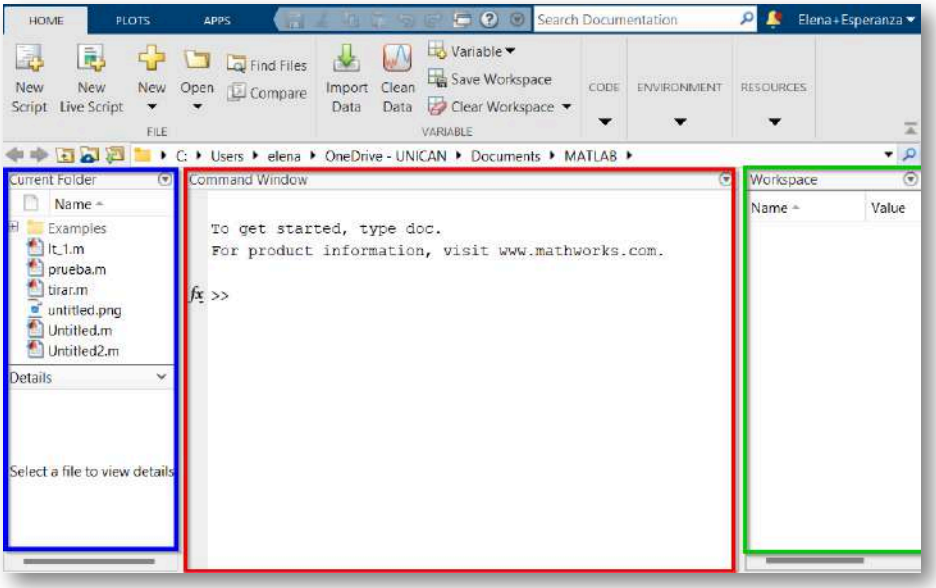


Figura 1.5. Ventana Matlab

Directorio de trabajo

Cuando se guarda el código en un archivo y se quiere utilizar más adelante, es necesario indicar al programa su ubicación. Matlab tiene preconfigurada la ruta donde almacena sus funciones nativas y las de los paquetes incluidos. Sin embargo, si se trabaja con archivos propios, se debe asegurar que el programa pueda acceder a ellos correctamente.

Matlab busca funciones y scripts en los directorios especificados por el comando `path`. El primero de ellos es siempre el especificado en el diálogo **Current Directory**.

En la figura aparece marcado en rojo el directorio actual. Tecleando el comando `path` en la ventana de comandos puede ver el listado de directorios activos.

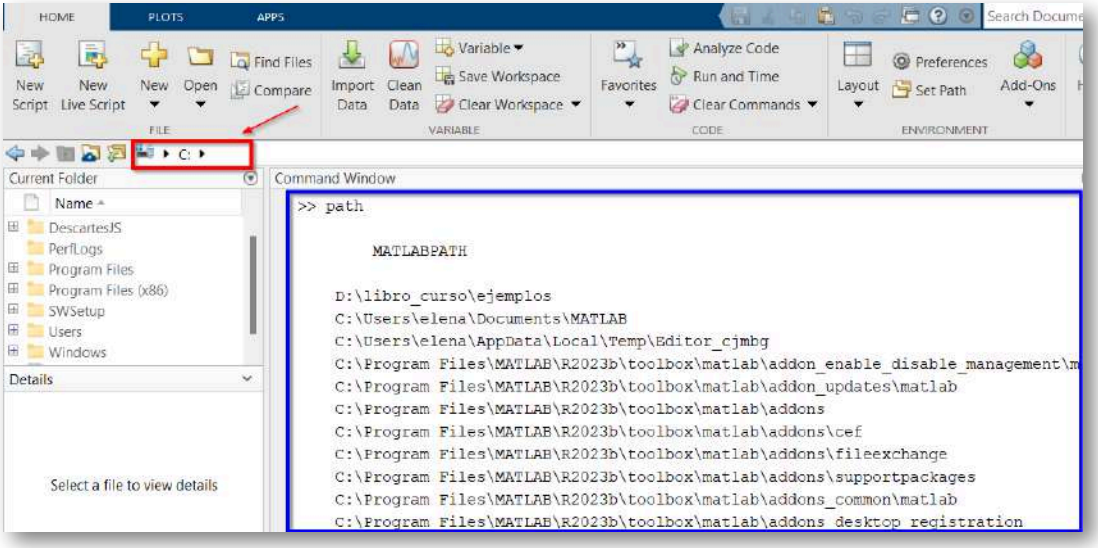


Figura 1.6. Directorio de trabajo

Ayuda de Matlab

Desde la ventana de comandos, se puede utilizar `help.` para obtener ayuda sobre la sintásis de un determinado comando.

```
help nombreDelComando
      Ejemplo:
  >>help plot
```

Además, desde el menú help, se puede acceder a la ayuda de MathWorks en el caso de Matlab.

Creación de variables

En Matlab/Octave, es sencillo crear variables, basta asignar un valor al nombre de la variable. Las variables se almacenan en la memoria y pueden contener números, matrices, cadenas de texto, entre otros. El contenido de una variable se puede recuperar y modificar a lo largo de una sesión de trabajo.

Se detallan a continuación algunas reglas a tener en cuenta a la hora de definir los **nombres de las variables**:

- El nombre de una variable puede tener letras, números y el guion de subrayar.
- El primer carácter tiene que ser una letra, `modulo2` es un nombre válido, pero no lo es `2modulo`.
- Las mayúsculas y las minúsculas tienen valor distinto. La variable `Modulo` es distinta de la variable `modulo`.
- En el nombre de una variable no puede haber espacios en blanco, `modulo1` es un nombre de variable válido, pero no lo es `modulo 1`.

- Existen nombres que deben evitarse porque tienen significado propio en Matlab: `ans, pi, Inf, i, j, ...` . Para obtener una lista de palabras reservadas teclea en la venta de comando `iskeyword()`.

Un ejemplo de declaración de variable:

```
x=5;    % x es una variable escalar
x=x+6;  % El valor de x es 11
```

Las variables creadas desde la línea de comandos pertenecen al espacio de trabajo (`workspace`).

Matlab ignora cualquier texto que vaya precedido por el símbolo %, por lo tanto, este símbolo sirve para incluir **comentarios**. Los ejemplos que siguen ilustran algunas características de las variables.

Ejemplo 1.1. ¿Qué resultado se obtendría con el siguiente código?

```
>> x=5;  2*x;  y=x^2;   x=y/x;
```

Solución

Ejemplo 1.2. Indica cuáles de las siguientes expresiones son incorrectas justificando las respuestas.

1. Numero-6+2
2. 8num=3*2
3. Num valores=3+2
4. A234_7899000=3
5. B-32=0

Solución

Visualizacion de números

El comando `format` en Matlab/Octave se utiliza para controlar la forma en que el programa muestra los números en la ventana de comandos. No cambia la precisión interna de los cálculos, solo afecta a la presentación de los resultados. Aquí tienes algunos de los modos más comunes de este comando al escribir el número π.

`format short`	4 dígitos después del punto decimal	3.1416
`format long`	15 dígitos después del punto decimal	3.141592653589793
`format shortEng`	Notacion decimal con 4 dígitos	3.1416e+00
`format longEng`	Notacion técnica larga	3.14159265358979e+000

Tabla 1.1. Formatos de números

Operaciones aritméticas

Matlab/Octave permite realizar operaciones aritméticas básicas de manera directa. Las operaciones estándar incluyen suma (+), resta (-), multiplicación (*), división (/) y exponenciación (^). Por ejemplo,

```
x = 4; y = x + 3; % Suma
z = x * 2; % Multiplicación
A= 2*(x + z); % El valor de A es 24
```

Precedencia de operadores

Las expresiones se evalúan de izquierda a derecha, la potencia tiene el orden de prioridad mayor, seguido del producto y la división (ambas

tienen la misma prioridad) y por último la suma y la resta (con igual prioridad entre ellas). Para alterar este orden se deben introducir adecuadamente paréntesis. Por ejemplo,

```
x= 2+3^4/2;      % x toma el valor 2+81/2
B =2+3^(4/2);    % B toma el valor 11=2+3^2
```

A continuación se presentan ejemplos para practicar operaciones aritméticas y comprender la jerarquía de operadores.

Ejemplo 1.3. Utiliza Matlab/Octave como calculadora para obtener el valor de

$$\frac{\left(\frac{\pi}{2}+4\right)2^{3+2^2}}{1+3^2}$$

Solución

Ejemplo 1.4. Calcula el valor de

$$\sqrt[3]{3\frac{27}{91}}+\frac{2}{4}\sqrt[4]{3\frac{27}{91}}-\frac{1}{2}\sqrt[5]{3\frac{27}{91}}$$

Solución

Ejemplo 1.5. Un proyecto cuesta $C_{\text{inst}} = 10\,000$ euros y genera un beneficio anual neto de $B_{\text{anual}} = 2\,000$ euros durante $N = 5$ años. Calcula el retorno de inversión (ROI) con una tasa de descuento de $r = 4\%$. Las fórmulas del beneficio total y retorno de la inversión son:

$$B_{\text{total}} = \sum_{t=1}^{N} \frac{B_{\text{anual}}}{(1+r)^t} \qquad ROI = \frac{B_{\text{total}} - C_{\text{inst}}}{C_{\text{inst}}} \cdot 100$$

Solución

Scripts o guiones

Un **script** es un archivo que contiene un conjunto de comandos de Matlab/Octave que se ejecutan en orden. El nombre del archivo, por ejemplo **nombre.m** , debe tener una secuencia de caracteres válidos sin espacios y extensión **.m**.

También se puede crear un guión desde el editor de Matlab/Octave eligiendo el menú **New Script.**

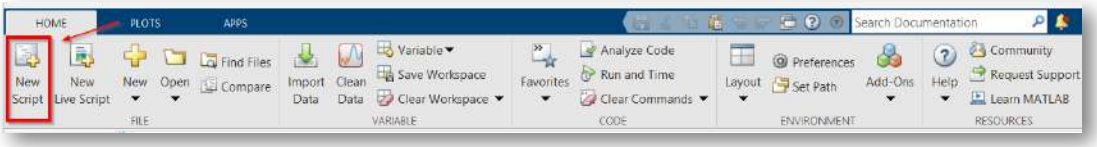


Figura 1.7. Menú New Script

Funciones anónimas

Algunas funciones sencillas pueden definirse con una única instrucción en la ventana de comandos mediante lo que se llaman funciones anónimas. Basta establecer su nombre, los argumentos de los que dependen separados por comas y los comandos que permiten definirla.

```
nombre_Funcion=@(argumentos) expresion_Funcion
```

Por ejemplo, para definir la función que calcula el área de un rectángulo a partir de su base y altura, habrá que escribir el código siguiente.

```
>>area_rectangulo=@(b,h) b*h
>>valor=area_rectangulo(4,3)
valor=
12
```

Ejemplo 1.6. Escribe una función anónima que permita pasar de grados Celsius a grados Fahrenheit.

Solución

Vectores

Los vectores en Matlab/Octave permiten almacenar secuencias de elementos que pueden disponerse como una fila o como una columna.

- **Vector fila**. Se define separando los elementos con espacios o comas (,).
- **Vector columna**. Se define separando los elementos con un punto y coma (;).
- **Vector equiespaciado en un intervalo**.

```
punto_inicial:paso:punto_final
```

Para generar un vector que comience en 2, avance de 2 en 2 y no supere 9, basta escribir el código siguiente.

```
>>v=4:2:9 &Devuelve: v=[4 6 8]
```

- **Comando linspace**. Genera un vector fila con números igualmente espaciados considerando un punto inicial, un punto final y el número de elementos.

```
linspace(valor_inicial,valor_final,num_puntos)
```

Por ejemplo, para generar un vector fila de 7 elementos equiespaciados siendo el primer valor 2 y el último 3 se deberá escribir:

```
>>v=linspace(2,3,7)
v=[2.0000 2.1667 2.3333 2.5000 2.6667 2.8333 3.0000]
```

A continuación se presenta un ejemplo que ilustra diversas formas de definir un vector.

Ejemplo 1.7. Genera los siguientes vectores eligiendo un valor cualquiera para n.

1. El vector cuyas componentes son $n,\ 2n, 3n, ..., 10n$.
2. El vector cuyas componentes son $n-1,\ n-3, n-5, ..., n-41$.
3. El vector cuyas componentes son $n^2+1,\ n^2+3, n^2+5, ..., n^2+2n-1$.
4. Un vector con $10+n$ componentes regularmente espaciadas entre 0 y π.

Solución

Matrices

Una matriz es una colección de valores en filas y columnas. Cada fila se separa con un punto y coma (;) y los elementos de una fila se separan con espacios o comas (,). Por ejemplo,

```
M = [1 2 3; 4 5 6; 7 8 9]; % Es una matriz 3x3
M = [1 2 3; 4 5 6]; % Matriz 2x3, 2 filas y 3 columnas
```

Se muestra cómo generar algunas matrices especiales.

Matriz de ceros

```
zeros(numFilas,numColumnas)
```

```
% Ejemplo: una matriz 4x3 de ceros
>>zeros(4,3)
```

Matriz de unos

```
ones(numFilas,numColumnas)
```

```
%Ejemplo: Matriz 4x3 con todos sus elementos unos
>>ones(4,3)
```

Matriz identidad

```
eye(orden)
```

```
% Ejemplo: Matriz identidad 3x3
>>eye(3)
```

Matriz diagonal

```
diag(vector)
```

```
%Ejemplo: Una matriz diagonal con el vector [2 3 4]
%en la diagonal
>>diag([2 3 4])
```

Matriz aleatoria

```
rand(m,n)      randi([imin imax],m,n)
```

```
%Ejemplo: Matriz 2x3  de valores aleatorios
>>rand(2,3)
%Ejemplo: Matriz 2x3 con valores aleatorios entre -5 y5
>>randi([-5 5],2,3)
```

Acceso a los elementos de una matriz

- Selección de un elemento específico indicando su fila y columna.

```
A = [1 2 3; 4 5 6; 7 8 9];
% Accede al elemento en la fila 2, columna 3
elemento = A(2, 3);
% Resultado: elemento = 6
```

- Selección de una fila.

```
A = [1 2 3; 4 5 6; 7 8 9]; fila2 = A(2, :);
% Accede a todos los elementos de la segunda fila
% Resultado: fila2 = [4 5 6]
```

- Selección de una olumna.

```
A = [1 2 3; 4 5 6; 7 8 9]; columna3 = A(:,3);
% Accede a todos los elementos de
%la tercera columna
% Resultado: columna3 = [3;6;9]
```

- Selección de la última fila o de la última columna de una matriz sin especificar su dimensión.

```
A = [1 2 3; 4 5 6; 7 8 9]; % Matriz 3x3
ultima_fila = A(end, :)
ultimas_columnas=A(:,2:end)
```

- Selección de submatrices especificando rango de filas y columnas.

```
A = [1 2 3; 4 5 6; 7 8 9];
% Submatriz con filas 1 y 2, y columnas 2 y 3
submatriz = A(1:2,2:3);% Resultado: [2 3;5 6]
```

- Selección de submatrices generadas a partir de condiciones.

```
A = [1 2 3; 4 5 6; 7 8 9];
elementos_mayores_a_5 = A(A>5);
% Resultado: elementos_mayores_a_5 = [6 7 8 9]
```

Operaciones con vectores y matrices

Matlab/Octave está diseñado para trabajar de manera óptima con matrices, permitiendo realizar operaciones como la trasposición, productos matriciales y otras manipulaciones comunes.

- Operaciones matriciales básicas.

```
A = [1 2; 3 4];B = [5 6; 7 8];
C = A+B;    % Suma de las matrices A y B
D = A*B;    % Producto de las matrices A y B
X = A\B;    % X es la matriz que cumple A*X=B
X = A/B;    % X es la matriz que cumple X*A=B
X = A^2;    % X es A*A
```

- Multiplicación, división, exponenciación elemento a elemento.

```
A = [1 2 3; 3 4 5];B = [6 7 8; 9 10 11];
C = A.*B;   % Multiplicación elemento a elemento
D = A./B;   % División elemento a elemento
D = A.^3;   % Potencia elemento a elemento
```

- Traspuesta de una matriz.

```
A = [1 2 3; 3 4 5]; % A es 2x3
A'  % Matriz traspuesta de A de orden 3x2
```

A continuación, se practicará la creación de vectores y matrices y el acceso a sus elementos en varios ejemplos.

Ejemplo 1.8. Dados los vectores fila $x = (4, 6, 2)$, $y = (3, -2, 4)$ y el número $k = 3$, realiza las siguientes operaciones componente a componente.

$$y - k\,; \quad ky\,; \quad \frac{x}{k} \quad ; 2x - y \quad x^2 + y^2\,; \quad \frac{3}{\sqrt[3]{x}} \quad ; \quad \frac{1+x}{y^4}$$

Solución

Ejemplo 1.9. Define una matriz A de orden 4 cualquiera y obtén los dos primeros elementos de la fila tercera.

Solución

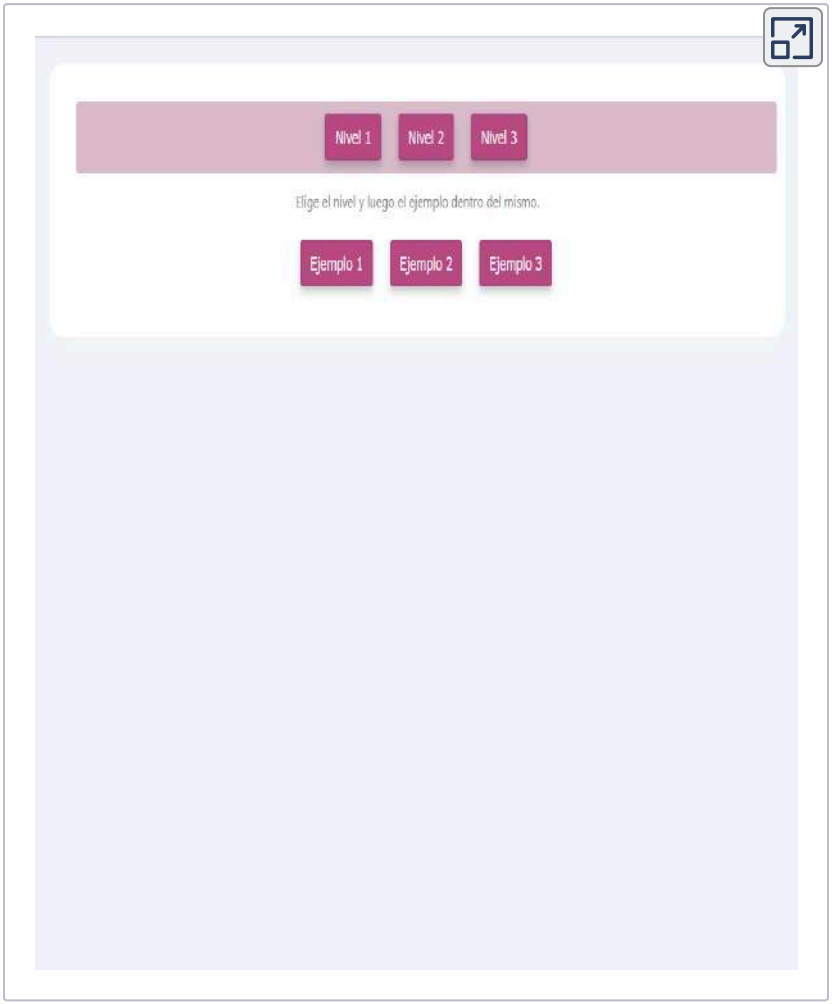
Ejemplo 1.10. Crea un vector que comience en 1 el valor máximo sea 22 y con salto 5. Crea otro vector con el mismo número de elementos que el anterior de forma que la suma de los dos vectores tenga todas sus componentes iguales.

Solución

Ejemplo 1.11. Genera una matriz de números aleatorias de 8x6 elementos. Selecciona la submatriz con las filas pares y las columnas impares.

Solución

Ejemplo 1.12. En el siguiente interactivo se pueden ver ejemplos prácticos de creación y manipulación de vectores y matrices en Matlab/Octave.



Interactivo 1.1. Ejercicios sobre matrices

Polinomios

En Matlab/Octave, los polinomios se representan como vectores de coeficientes ordenados de mayor a menor grado.

```
p = [3 -2 5 0 1];
% Representa 3x^4 - 2x^3 + 5x^2 + 0x + 1
```

Para evaluar un polinomio en un punto se utiliza el comando `polyval`. Para multiplicar, el comando `conv` y para dividir `deconv`.

```
x = 2; p1=[3 -2 5 0 1];p2=[2 3];
valor = polyval(p1, x); disp(valor) % Evalúa p(2)
pmul = conv(p1, p2);
[cociente, resto] = deconv(p1, p2);
```

Funciones nativas de Matlab/Octave

Los programas Matlab/Octave ofrecen una amplia variedad de funciones integradas para trabajar con números, vectores, matrices, gráficos y procesamiento de datos. A continuación se muestran algunas de las más comunes, organizadas en categorías.

- Funciones matemáticas básicas

```
v = [3 4 -1]; w=[3 4 5] % Vector fila
abs(v);      % Valor absoluto de cada elemento
sqrt(w);     % Raíz cuadrada de cada elemento
exp(v);      % Exponencial de cada elemento
log(w);      % Exponencial de cada elemento
round(w);    % Redondeo al entero más cercano
floor(w);    % Redondeo hacia abajo
round(w);    % Redondeo hacía arriba
mod(11,4)    % Residuo de la división 11 entre 4
```

- Funciones trigonométricas e hiperbólicas,

```
v = [3 4 -1];   % Vector fila
% Seno, coseno, tangente de cada elemento
sin(v), cos(v), tan(v)
% Arcoseno, arcocoseno, y arcotangente
asin(v), acos(v), atan(v)
% Seno, coseno, tangente hiperbólicas
asinh(v), acosh(v), atanh(v)
% Arcoseno, arcocoseno, y arcotangente
% hiperbólicas
asinh(v), acosh(v), atanh(v)
```

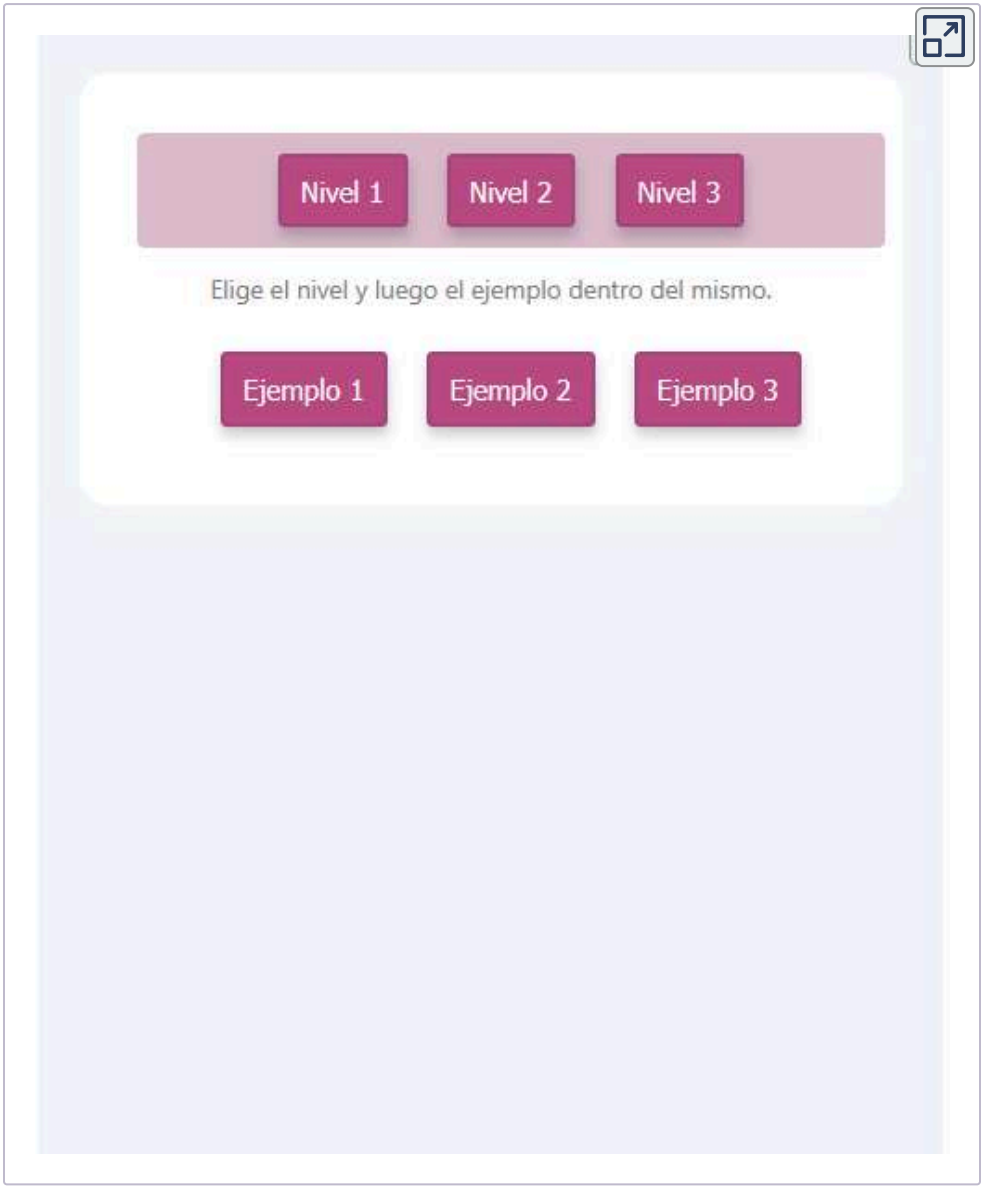
- Funciones para matrices y vectores.

```
A = [1 7 3;4 8 6];  % Matriz 2x3
size(A)     % Tamaño de A
length(A)   % Longitud de A, mayor dimensión
trace(A)    % Traza de A
sum(A)      % Suma de los elementos de A
prod(A)     % Producto de los elementos de A
mean(A)     % Promedio de los elementos de A
max(A)      % Máximo de los elementos de A
min(A)      % Mínimo de los elementos de A
sort(A)     % Ordena elementos de A
size(A)     % Devuelve un vector con el número
            % de columnas y filas de la matriz A
```

- Funciones para álgebra lineal.

```
A = [1 2; 3 4];     % Matriz 2x2
det(A)      % Determinante de A
inv(A)      % Inversa de A
eig(A)      % Autovalores de A
rank(A)     % Rango de A
```

Ejemplo 1.13. En el siguiente interactivo puedes ver distintos ejemplos de uso de algunas de las funciones anteriores.



Interactivo 1.2. Ejercicios sobre funciones en Matlab/Octave

Gráficas 2D

Una de las características más útiles de Matlab/Octave es su capacidad para crear gráficos de forma rápida y sencilla. Uno de los comandos a utilizar es `plot`.

```
plot(vectorx,vectory,opciones_gráfica)
```

```
% Ejemplo: Representar x^2 en [-10,10]
>> x = -10:0.1:10; % Vector de valores x
>> y = x.^2; % Cuadrado de cada valor de x
>> plot(x, y); % Crear la gráfica
```

En Matlab/Octave, puedes personalizar las gráficas usando opciones como color de línea (por ejemplo, 'r' para rojo), estilo de línea (como '--' para una línea discontinua), y símbolos para representar los puntos (por ejemplo, 'o' para círculos). También puedes combinar estas opciones, por ejemplo, 'b--o' para una línea azul discontinua con círculos.

Otros comandos gráficos:

- Abrir una ventana gráfica: `figure`.
- Añadir etiquetas en los ejes: `xlabel`, `ylabel`.
- Superponer gráficos dentro de una misma figura: `hold on`, `hold off`.
- Poner o quitar una rejilla al gráfico: `grid on`, `grid off`.
- Incluir un título de un gráfico: `title`.
- Generar una matriz de gráficos en una misma gráfica: `subplot(m,n,p)` divide la ventana de gráficos en una cuadrícula de mxn gráficas y selecciona la subgráfica p.

Ejemplo 1.14. Muestra en distintas gráficas los datos de temperatura de dos ciudades dadas en dos vectores.

```
% Datos de ejemplo de temperaturas a lo largo del día
horas = 0:1:23; % Horas de 0 a 23
temp_ciudad1 = [15, 14, 13, 13, 12, 12, 13, 15, 18, 22,
25, 27, 29, 30, 28, 26, 24, 22, 20, 18,
17, 16, 15, 14];
temp_ciudad2 = [10, 9, 9, 8, 8, 7, 8, 10, 13, 18, 20,
23, 25, 26, 24, 23, 21, 19, 17, 15, 13,
12, 11, 10];

% Gráfico de temperatura para la Ciudad 1
% Gráfico en azul
plot(horas, temp_ciudad1, 'b', 'LineWidth', 1.5);
hold on; % Mantiene la gráfica para añadir la segunda

% Gráfico de temperatura para la Ciudad 2
% Gráfico en rojo con línea discontinua
plot(horas, temp_ciudad2, 'r--', 'LineWidth', 1.5);

% Etiquetas y leyenda
xlabel('Hora del día');
ylabel('Temperatura (°C)');
title('Temperatura promedio durante el día en dos
ciudades');
legend('Ciudad 1', 'Ciudad 2');

% Mostrar la cuadrícula para facilitar la lectura
grid on;

% Desactivar hold para futuras gráficas
hold off;
```

Ejemplo 1.15. Una compañía eléctrica tiene la siguiente tarifa. Los primeros 100Kwh se pagarán a 2€ el Kwh, para los siguientes 200 Kwh costará 3 € y 6 de allí en adelante. Expresa el valor de la factura como una función de la cantidad de Kwh consumida al mes y representa la función.

Solución

Fundamentos sobre programación en Matlab/Octave

Matlab y Octave van más allá de simples calculadoras, incorporan un lenguaje propio de programación con el que se puede crear scripts y funciones, automatizar procesos, manejar bucles y condicionales entre otras posibilidades. A continuación, se presentan los fundamentos básicos del lenguaje de programación.

Funciones

Las **funciones** en Matlab/Octave permiten encapsular un conjunto de operaciones dentro de un bloque de código que puede ser reutilizado. Se definen en un archivo separado, también con la extensión **.m**. y con el mismo nombre que la función. En el siguiente ejemplo se muestra la función `cuadrado` que calcula, a partir de un valor de entrada, su cuadrado.

```
function resultado = cuadrado(x)
    resultado = x^2;
end
```

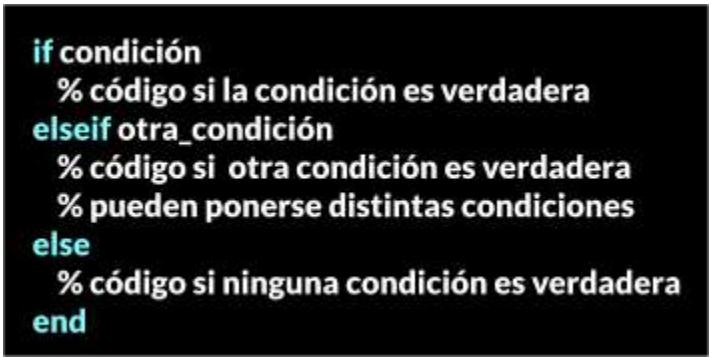
El fichero que tiene esta función deberá llamarse `cuadrado.m`. La función se invoca de la misma forma que las funciones predefinidas propias de Matlab/Octave desde la ventana de comandos o desde otro script.

```
a=cuadrado(2)+1
% Resultado: a=5;.
```

Para que Matlab pueda utilizar un script o función que hayas creado, el archivo debe guardarse en el directorio de trabajo. Este es el lugar donde Matlab busca por defecto los archivos cuando ejecutas comandos. Puedes ver o cambiar el directorio de trabajo en la barra superior del entorno de Matlab, o usar el comando `cd` desde la ventana de comandos.

Condicionales

Matlab/Octave admite la estructura de control condicional `if-else`, que se usa para ejecutar bloques de código en función de si se cumple o no una condición booleana.



En el ejemplo siguiente, si el valor de x es positivo se escribe el texto "x es positivo", en caso contrario, cuando es menor que cero, se escribe por pantalla "x es negativo" y, en otro caso, se escribe "x es cero".

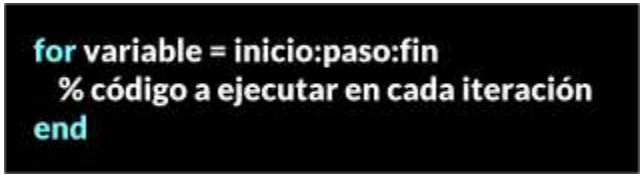
```
if x > 0
    disp('x es positivo');
elseif x <0
    disp('x es negativo');
else
    disp('x es cero');
end
```

Bucles

Matlab/Octave ofrecen dos tipos de estructuras de control para iterar: el bucle `for` y el bucle `while`, ambos permiten ejecutar un bloque de código repetidamente bajo distintas condiciones.

La instrucción `break` fuerza la salida del bucle, de modo que cualquier código restante dentro de esa iteración y las siguientes no se ejecutará. Por su parte, `continue` interrumpe únicamente la iteración actual.

- Ciclo `for`. Repite un conjunto de instrucciones cuando varía el índice en un conjunto de valores.



Con el siguiente código se imprime por pantalla el valor de i cuando va tomando los valores 1, 2, 3, ... ,10.

```
for i = 1:10
    disp(i);
end
```

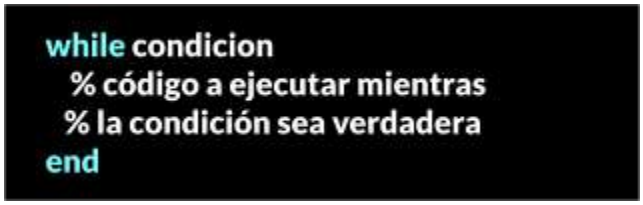
Se muestra cómo sumar los cuadrados de los primeros 20 números naturales.

```
suma = 0; % Inicialización de la suma
for i=1:20
    % Agrega el siguiente número a la suma
    suma = suma + i^2;
end
```

Este cálculo se puede hacer también sin utilizar bucles operando con vectores.

```
vec=1:20;
sum(vec.^2)
```

- Bucle `while`. Repite un conjunto de instrucciones mientras una condición es verdadera.



Como ejemplo, se verá cómo sumar los primeros números naturales hasta que la suma sea mayor o igual a 300.

```
suma = 0;   % Inicialización de la suma
n = 1;      % Número
while suma < 300.
    % Agrega el siguiente número a la suma
    suma = suma + n;
    % Incrementa el contador
    n = n + 1;
end
```

Se muestra ahora cómo se calcula la suma de los números pares desde 2 hasta 10 utilizando un ciclo `while`.

```
n = 2;      % Número inicial
suma = 0;   % Variable para almacenar la suma
% Bucle while
while n <= 10
    %Sumar el valor de n a la variable suma
    suma = suma +n;
    n = n + 2; % Incrementar n en 2
end
suma % Mostrar el resultado: 30
```

Este cálculo se puede hacer también incluyendo los números en un vector y, a continuación, aplicar la función `sum`, que devuelve la suma de sus elementos.

```
sum(2:2:10)
```

Manejo de archivos y datos

Se incluyen algunos comandos para la lectura y manejo de datos almacenados en ficheros.

- Abrir y leer dados de un archivo de texto: `fopen`, `fscanf`.

```
% Lee todos los números de archivo.txt
fileID = fopen('datos.txt', 'r');
data = fscanf(fileID, '%f');
fclose(fileID);
disp(data);
```

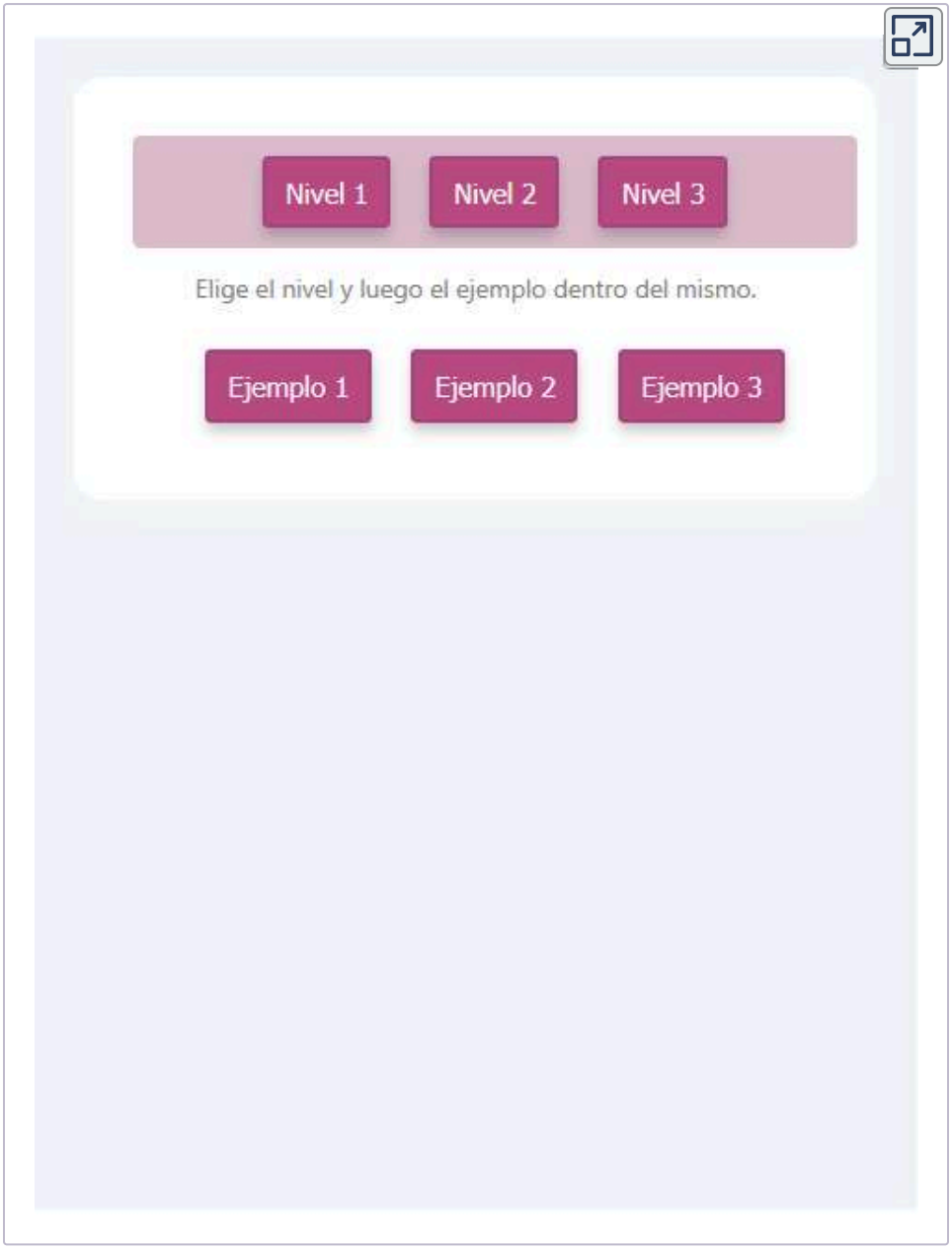
- Escritura de archivos: `fprintf`.

```
% Guarda el vector en un archivo de texto
data = [1.1, 2.2, 3.3];
fileID = fopen('output.txt', 'w');
fprintf(fileID, '%f\n', data);
fclose(fileID);
```

- Carga y guardado de variables: `save`, `load`.

```
% Guarda las variables a y b en archivo.mat
a = 5;
b = [1, 2, 3];
save('variables.mat', 'a', 'b');
load('variables.mat');
```

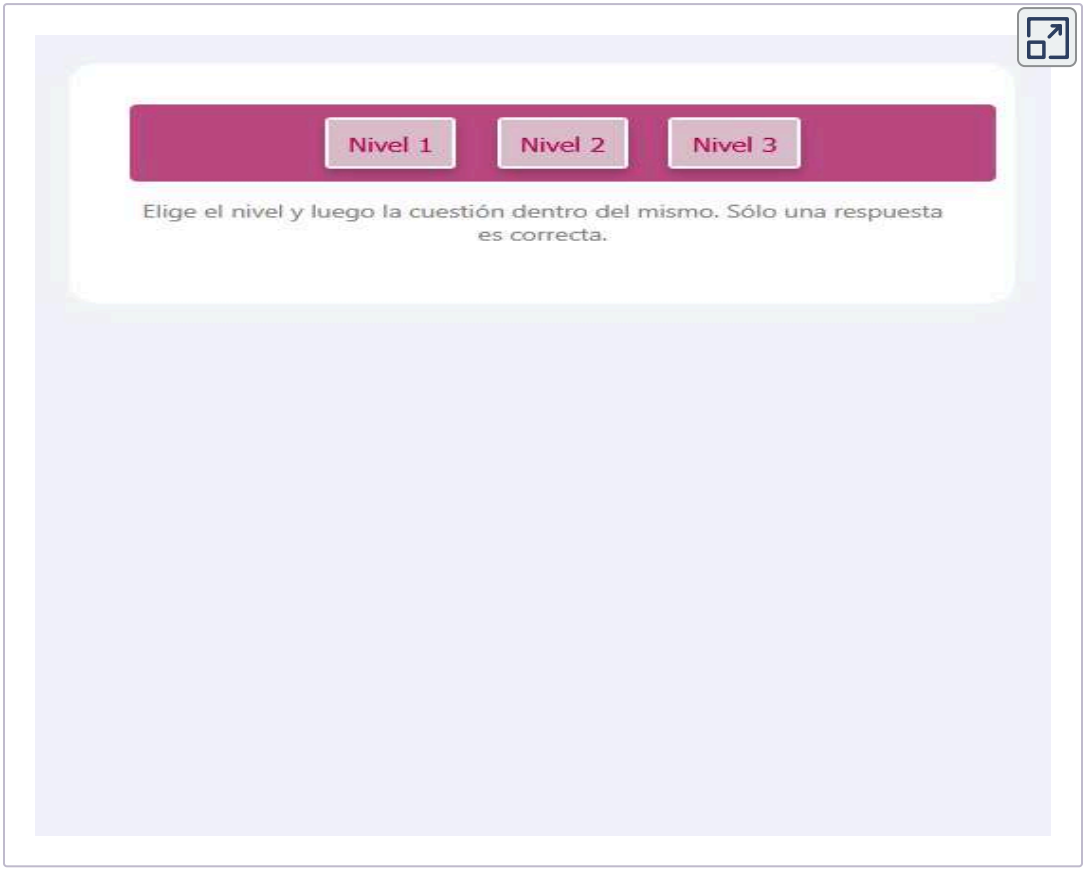
Ejemplo 1.16. En el siguiente interactivo puedes practicar con ciclos y bucles y algunas de las funciones vistas en este capítulo.



Interactivo 1.3. Ejercicios sobre ciclos y bucles

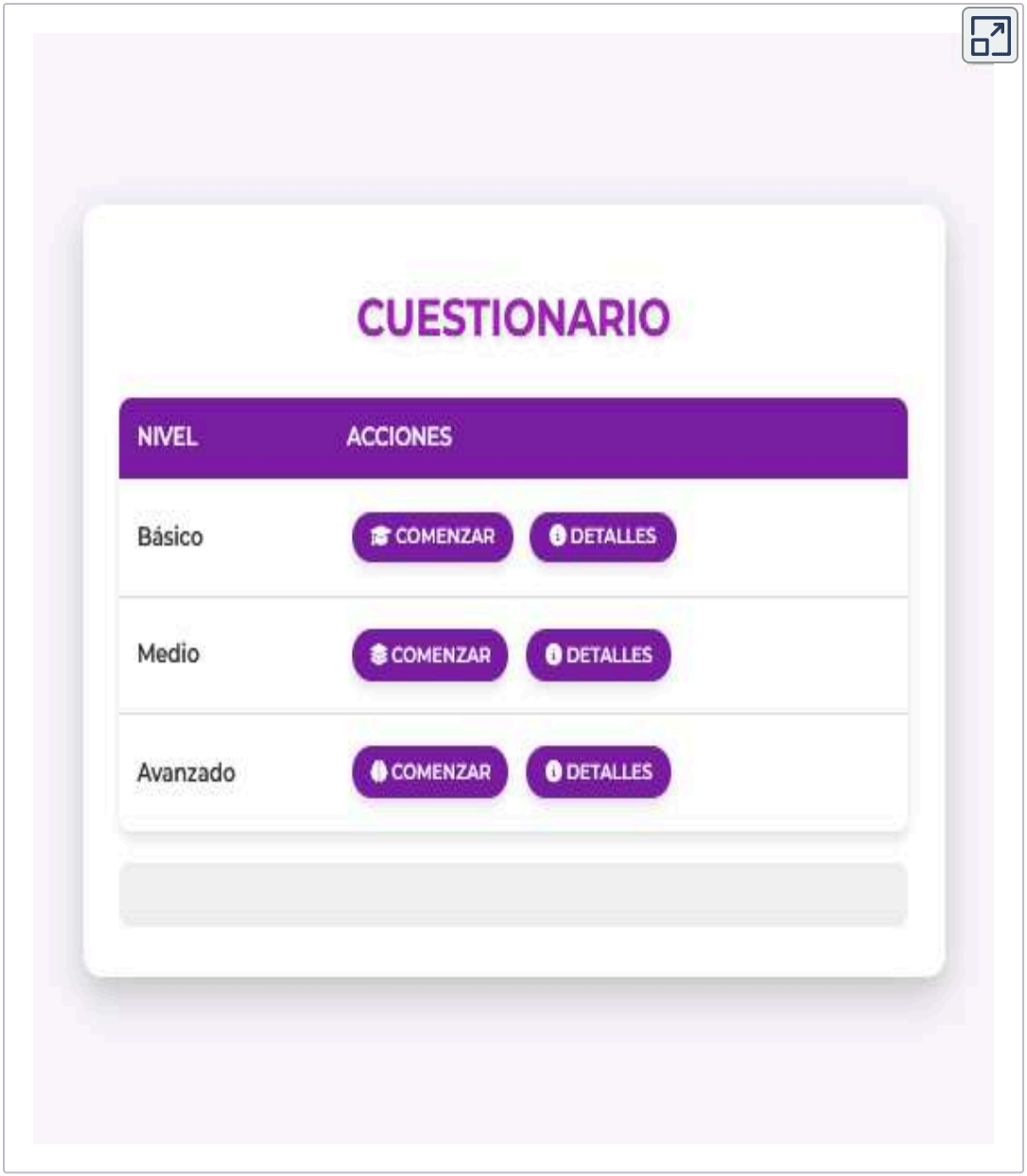
1.5 Autoevaluación

A continuación, se presenta un breve cuestionario de Matlab/Octave para consolidar los conceptos clave de este capítulo.



Interactivo 1.4. Test básico de Matlab/Octave

Con el siguiente interactivo puedes realizar una autoevaluación por niveles.



Interactivo 1.5. Autoevaluación del tema

1.6 Ejercicios

1 Define las tres variables siguientes: a=1.5, b=4, c=3.5. Calcula el valor de $d = \frac{a}{\frac{b}{c}+\frac{c^2}{a}}$.

Solución

2 Considera un gas que escapa de un tanque presurizado bajo un proceso adiabático. Su flujo másico se puede aproximar con la función:

$$f(P_{\text{ext}}, P_{\text{int}}, K) = P_{\text{int}}\sqrt{\frac{2K}{K-1}\left(1-\left(\frac{P_{\text{ext}}}{P_{\text{int}}}\right)^{\frac{K-1}{K}}\right)}$$

donde P_{int} es la presión interna en el tanque, P_{ext} es la presión externa fuera del tanque y K es el coeficiente adiabático del gas. Escribe esta expresión en Matlab/Octave y calcula su valor para los valores de $K = 1.4$, $P_{\text{ext}} = 1$ y $P_{\text{int}} = 2$. Utiliza una función anónima que dependa de estos tres parámetros.

Solución

3 Genera los siguientes vectores utilizando los operadores adecuados de Matlab/Octave:

- $v_1 = [1, 3, 5, ..., 25]$
- $v_2 = [0, 0.1, 0.2, ..., 1]$
- $v_3 = [\pi, 2\pi, 3\pi, ..., 10\pi]$
- $v_4 = [10, 9, ..., 1, 0]$

Solución

4 Genera un vector x de 30 componentes regularmente espaciadas entre 0 y π. Evalúa x en cada una de las funciones siguientes:

$$f(x) = \frac{\log(x+2)}{x} \quad f(x) = x^2 + x - e^x \quad f(x) = e^{x^2}\sin(x)$$

Solución

5 Realiza el ejercicio anterior utilizando funciones anónimas considerando que x puede ser un vector.

Solución

6 Representa gráficamente la función definida a trozos:

$$f(x) = \begin{cases} x^2 + 3x - 2, & \text{si } 1 \leq x < 3 \\ 7x^3 - 3, & \text{si } 3 \leq x \leq 4 \end{cases}$$

Solución

7 Diseña un script en Matlab/Octave que genere una matriz cuadrada A de tamaño $n \times n$, con n proporcionado por el usuario. Llena la matriz con valores aleatorios entre 1 y n, duplica los valores de la diagonal principal y reduce a la mitad los valores de la diagonal secundaria. Imprime la matriz original y la modificada.

Solución

8 Escribe un programa en Matlab/Octave que cree dos vectores X e Y de longitud 10 con números aleatorios entre 1 y 20. Compara elemento a elemento:

- Si $X[i] > Y[i]$, almacena la suma en un nuevo vector Z.
- Si $X[i] \leq Y[i]$, almacena el valor absoluto de su diferencia en Z.

Muestra el resultado final en Z.

Solución

9 Escribe un script que clasifique un conjunto de 24 mediciones de temperaturas en tres categorías: 'Frío' (< 10°C), 'Templado' (10-20°C], y 'Caluroso' (> 20°C). Muestra el número total de cada categoría.

Solución

10 Para $n = 10, 20, 40, 80, \ldots, 10240$, calcula la suma de los n primeros números naturales aplicando y sin aplicar la fórmula $S(n) = \frac{n(n+1)}{2}$. Muestra los resultados.

Solución

11 Escribe una M-función llamada `Multiplo` que, dados dos números k y h, devuelva:

- 1 si $k + h$ es múltiplo de 2.
- 2 si también es múltiplo de 3.
- 0 en caso contrario.

Solución

12 Construye un vector con los diez primeros términos de la sucesión:

$$x_{n+1} = \frac{1}{2 + x_n} \quad \text{con } x_1 = 1 \quad \text{para } n \geq 1$$

Solución

13 Escribe una M-función llamada `ContarPares` que reciba un vector como argumento y devuelva el número de componentes pares y un vector con dichas componentes.

Solución

14 Escribe una M-función que calcule la suma de las cifras de un número natural dado. Por ejemplo, `SumaCifras(325)` debería devolver 10.

Solución

15 Utilizando el algoritmo de Euclides, escribe una M-función que calcule el máximo común divisor de dos números. Ejemplo: `MCD(24, 15)` debería devolver 3. Nota: El MCD(a,b) con a>b es b si a es múltiplo de b. En otro caso, MCD(a,b)=MCD(b,r) siendo r el resto de dividir a entre b.

Solución

16 Escribe una M-función que, a partir de tres números dados, calcule la suma de los cuadrados de los dos números mayores. Por ejemplo, `SumaCuadrados(5, 3, 4)` debería devolver 41.

Solución

17 Escribe una M-función llamada `SuperaMedia` que reciba un vector y devuelva otro vector con los elementos que sean mayores o iguales a la media del vector de entrada.

Solución

RESUMEN

En este capítulo se ha subrayado la relevancia de los métodos numéricos para resolver problemas matemáticos complejos sin solución analítica exacta y se han analizado algunos ejemplos donde los errores que pueden surgir al realizar cálculos por ordenador pueden provocar soluciones erróneas. A continuación, se ha presentado MATLAB/Octave como el software de cálculo que se empleará a lo largo del libro para poner en práctica los conceptos teóricos facilitando tanto la implementación de algoritmos como la interpretación de resultados.

Algunos aspectos tratados en este capítulo son:

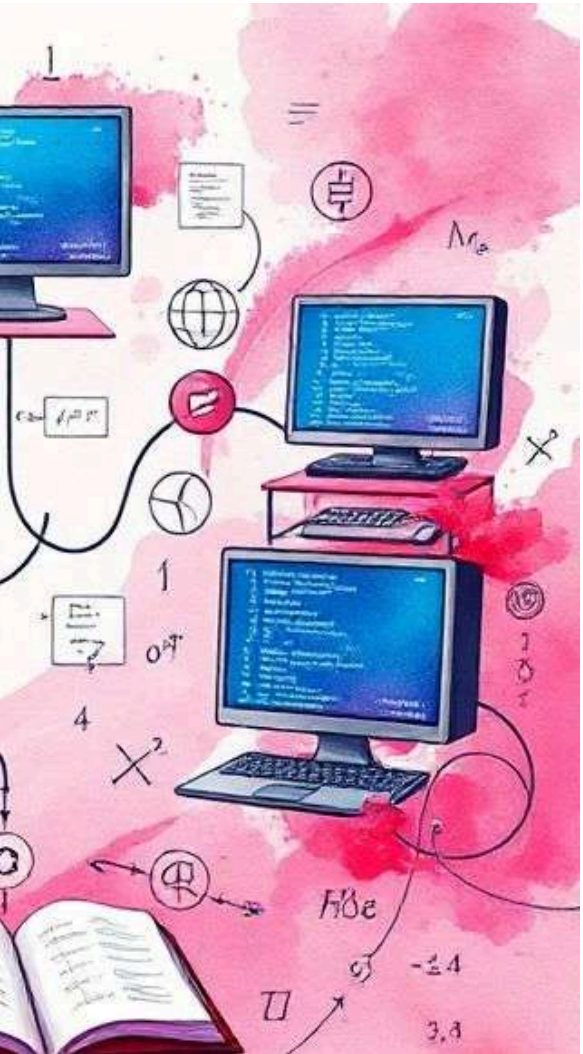
1. **Errores históricos relevantes.** Casos como el fallo de los misiles Patriot en 1991, donde un error en la representación del tiempo provocó un cálculo incorrecto, y la explosión del cohete Ariane 5 en 1996, causada por un fallo en la conversión de datos numéricos, son ejemplos clave que muestran cómo errores aparentemente pequeños pueden llevar a consecuencias catastróficas en sistemas críticos.

Para facilitar la resolución de problemas numéricos y la visualización de resultados, se ha introducido la herramienta computacional Matlab/Octave. Los contenidos que se han repasado son los siguientes:

2. **Entorno de trabajo, directorio de trabajo y ayuda de matlab.** El programa incluye una interfaz intuitiva para gestionar variables, explorar el directorio de trabajo y acceder a ayuda interactiva para funciones y comandos

3. **Creación de variables y operadores numéricos.** Las variables se crean automáticamente al asignarles un valor, y se pueden manipular mediante operadores aritméticos, de comparación y lógicos.

4. **Scripts y guiones.** Son conjuntos de comandos almacenados en archivos .m que permiten automatizar cálculos y organizar programas estructuradamente.

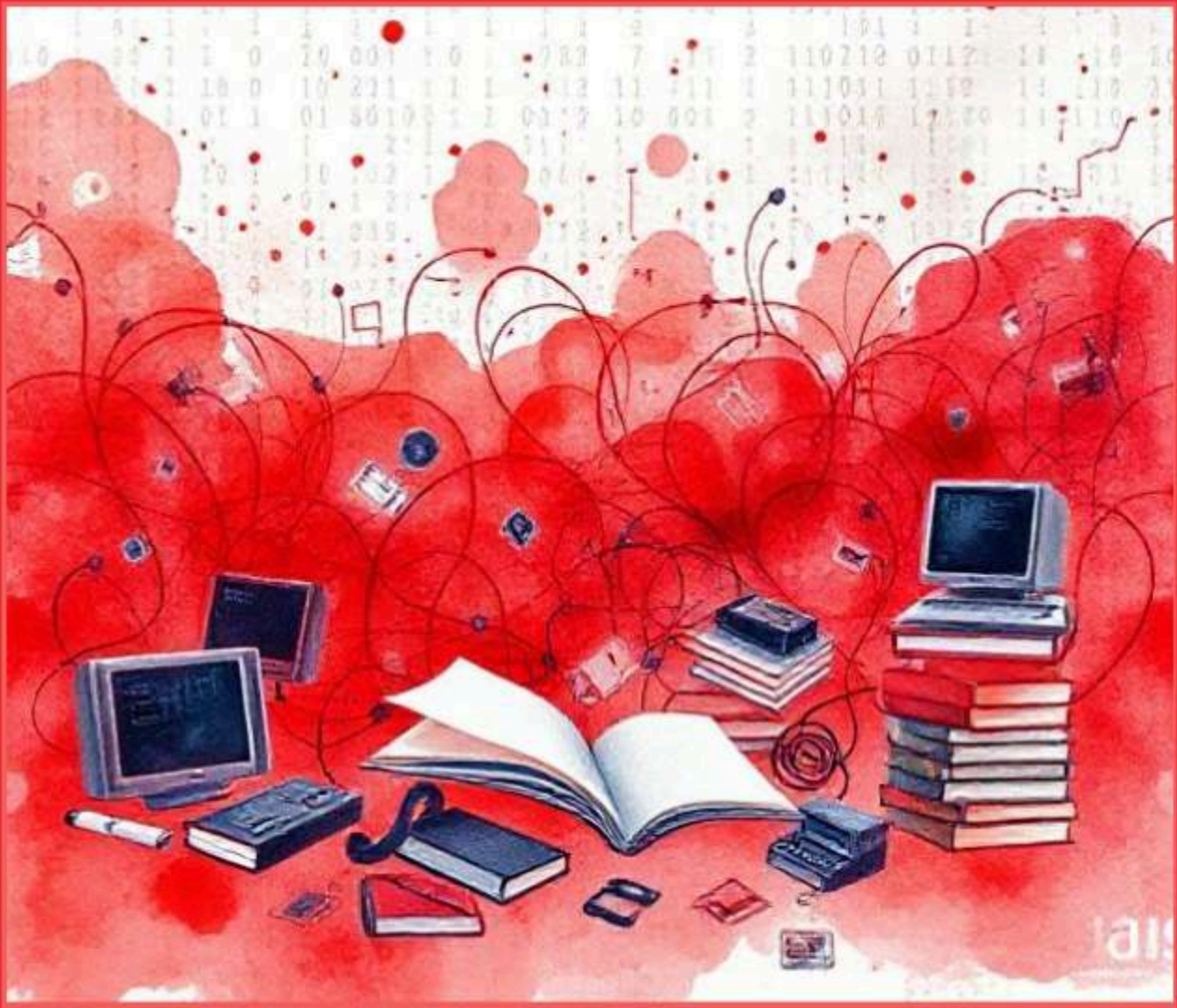


5. **Vectores y matrices.** El programa está diseñado para trabajar con vectores y matrices como estructuras de datos principales, permitiendo cálculos algebraicos y vectoriales de forma óptima.

6. **Funciones nativas.** Incluye un conjunto extenso de funciones nativas para realizar cálculos, facilitando las tareas numéricas comunes.

7. **Gráficas 2D.** Permite crear gráficas 2D para representar datos, incluyendo opciones para personalizar títulos, etiquetas y leyenda

8. **Introducción a la programacón.** La programación se basa en el uso de scripts, funciones definidas por el usuario y control de flujo (condicionales y bucles).



2 CUESTIONES BÁSICAS SOBRE ARITMÉTICA COMPUTACIONAL

Capítulo II

Cuestiones básicas sobre aritmética computacional

2.1 Introducción

Este tema tiene como objetivo mostrar la importancia de la aritmética computacional en el uso de métodos numéricos. Se analizará cómo los ordenadores representan los números y realizan operaciones, y se evidenciará que, debido a las limitaciones inherentes del sistema, como son el uso del sistema binario y la precisión finita, los resultados obtenidos no siempre son exactos y pueden presentar errores.

En particular, se verá que:

- Los ordenadores solo pueden **representar un subconjunto finito de los números reales**, ya que utilizan una cantidad limitada de bits para almacenar valores. Esto implica que muchos números, tanto raciones como irracionales, deben ser aproximados, lo que introduce errores en los cálculos a realizar con ellos.
- Las operaciones aritméticas pueden generar **errores acumulativos** debido a las limitaciones en la precisión.

A continuación, se mostrarán ejemplos numéricos que ayudan a entender mejor estos problemas y los errores que pueden aparecer al hacer cálculos en el ordenador. Para más información, se puede consultar [4], [10] y [6].

Ejemplo 2.1. Analiza el siguiente código Matlab y observa que al sumar $1\,000$ veces 0.1 no se obtiene 100.

```
suma=0;n=1000;
for k=1:n
    suma=suma+0.1;
end
format longEng;suma
```

El resultado debería ser 100 pero se obtiene un número próximo: $99.9999999999986e+000$

Ejemplo 2.2. Analiza el siguiente código Matlab y observa que los valores de la función $f(x) = (1-x)^6$ y los del polinomio resultado de expandir la potencia, $g(x)$, varían en las cercanías del punto 1. Representa la gráfica en $[0.996, 1.005]$ para observar estas diferencias.

```
f=@(x) (1-x).^6
%Polinomio expandida la potencia
%syms x ;expand((1-x).^6)
g=@(x) x.^6-6*x.^5+15* x.^4-20*x.^3+15*x.^2-6*x+1
x1=0.996:0.0001:1.005; format longEng;
%Comparamos resultados
[x1' f(x1)' g(x1)']
plot(x1,f(x1),x1,g(x1))
```

En la figura 2.1 se muestra la gráfica de la función $f(x)$ y la de la función $g(x)$ tras ejecutar el código anterior.

¿Por qué ocurre esto? La razón es que los ordenadores usan un formato (punto flotante binario) que no permite representar de forma precisa números como por ejemplo 0.1, 0.2 o 0.3. Como en general, los números están representados de manera inexacta, las operaciones aritméticas que se realicen también serán inexactas.

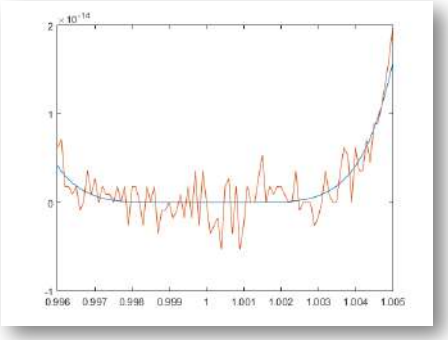


Figura 2.1. Gráficas de las funciones del ejemplo 2.2.

2.2 Sistemas de numeración

Para comprender y aplicar los métodos numéricos, es necesario conocer los sistemas numéricos, ya que constituyen la base para la representación y el tratamiento de los datos en los cálculos.

Un sistema de numeración consiste en un conjunto de símbolos que permiten representar cantidades numéricas, así como reglas para realizar operaciones con dichas cantidades. En este apartado, se verá el sistema decimal (base 10) y el binario (base 2), fundamentales en el ámbito del cálculo numérico. Mientras que el sistema decimal es el que se utiliza en la vida cotidiana, el binario es esencial en computación y en el procesamiento numérico, ya que los ordenadores representan y manipulan los datos a través de este sistema.

2.2.1 Sistema decimal

En el sistema decimal los números se representan utilizando 10 dígitos: 0,1,2,3,4,5,6,7,8,9. Además, este sistema de numeración es posicional, las cantidades se representan utilizando como base aritmética las potencias del número diez. Así, si $x \in \mathbb{R}^+$, se puede escribir

$$x = \sum_{-\infty}^{n} a_k \cdot 10^k \quad 0 \leq a_k \leq 9$$

Ejemplo 2.3. Escribe el número 709.3045 utilizando sumas de potencias de 10.

El número se puede escribir

$$709.3045 = 7 \cdot 10^2 + 9 \cdot 10^0 + 3 \cdot 10^{-1} + 4 \cdot 10^{-3} + 5 \cdot 10^{-4}$$

2.2.2 Sistema binario

Si se considera una base b, un número real x positivo se puede escribir de la forma

$$x = \sum_{-\infty}^{n} a_k \cdot b^k \quad 0 \leq a_k \leq b-1$$

o también

$$x = a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} a_{-3} \ldots$$

El Standard de IEEE (Institute of Electrical and Electronics Engineers) propone el uso de la base $b = 2$ en el almacenamiento y la aritmética sobre el computador. En este sistema de numeración los números se representan utilizando solamente las cifras cero y uno (0 y 1). Así, si x es positivo,

$$x = \sum_{-\infty}^{n} a_k \cdot 2^k \quad a_k \in \{0, 1\}$$

2.2.3 Conversión de decimal a binario

Para convertir un número decimal entero a binario, basta dividir el número y sus cocientes entre 2, acumulando los restos o residuos de la division. La lista de ceros y unos leídos de abajo a arriba, darán la expresión en binario. Se describe este proceso a continuación.

En el caso de que el número x sea entero, se comienza dividiendo x entre 2 obteniendo de cociente x_1 y resto r_1 con valor 0 o 1. Se podrá escribir entonces

$$x = 2 \cdot x_1 + r_1$$

El proceso se repite con x_1 y el resto de cocientes hasta que el cociente sea menor que 2

$$x = 2 \cdot (2 \cdot x_2 + r_2) + r_1 = 2^2 x_2 + 2r_2 + r_1 \ldots$$

Los números r_1, r_2... nos darán las cifras de la expresión binaria de x.

Ejemplo 2.4. Con ayuda del siguiente interactivo, introduce un número natural y observa cómo se expresa en binario.

Interactivo 2.1. Conversor decimal a binario

Para pasar a binario la parte fraccionaria $0.f_1f_2f_3\ldots$, se deben buscar $a_{-1}, a_{-2}, \ldots$ de forma que

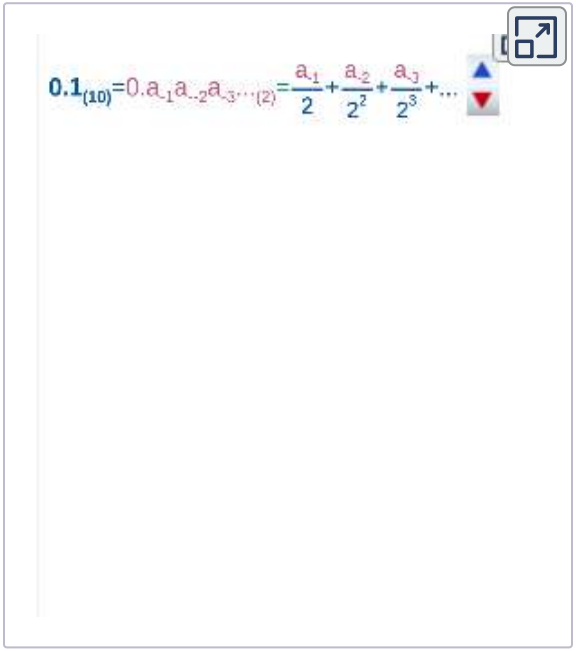
$$0.f_1f_2f_3\ldots = \frac{a_{-1}}{2} + \frac{a_{-2}}{2^2} + \frac{a_{-3}}{2^3} + \ldots$$

El proceso consiste en ir multiplicando por 2 y tener en cuenta el valor de la serie geométrica siguiente: $\sum_{k=1}^{\infty} \frac{1}{2^k} = 1$

Ejemplo 2.5. Convierte 0.1_{10} a binario.

Se tiene que,

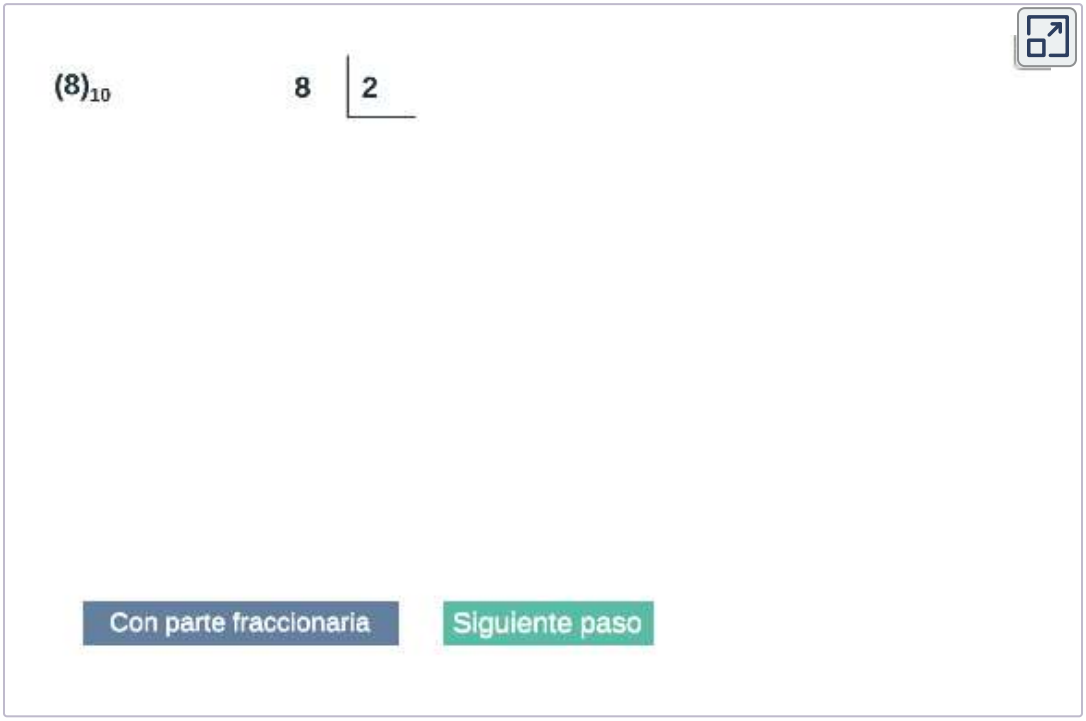
$$\begin{aligned}\frac{1}{10} &= \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^6} + \frac{0}{2^7} + \ldots \\ &= 0.0001\widehat{1001}_2\end{aligned}$$



Interactivo 2.2. Conversión a binario. Escena creada por Héctor A. Tabares Ospina y Juan Guillermo Rivera Berrío [11]

Teniendo en cuenta el ejemplo anterior, observa que el número 0.1 se escribe en base 2 con infinitas cifras distintas de cero.

Ejemplo 2.6. En la escena que se presenta a continuación, se muestra cómo hacer la conversión paso a paso a base 2 de ciertos números decimales. Se puede elegir entre números enteros y números con parte fraccionaria, observa el procedimiento para este último caso.

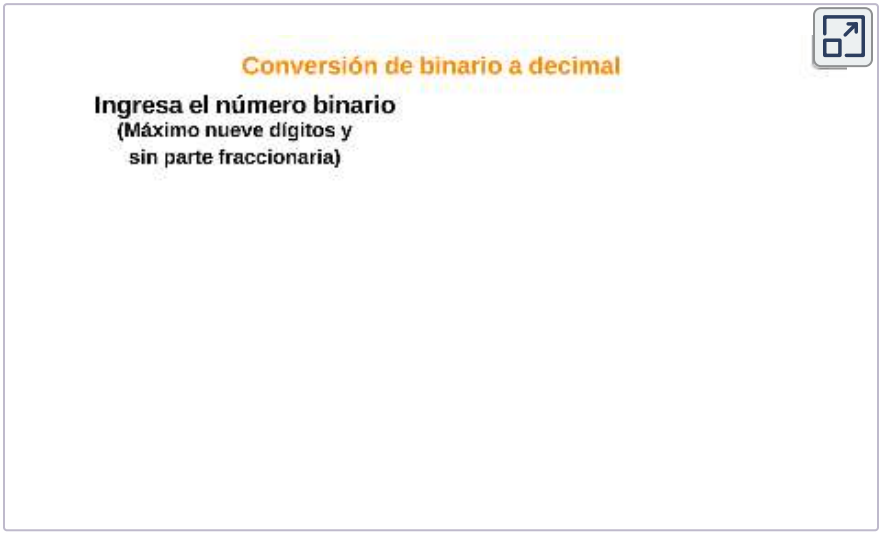


Interactivo 2.3. Escena creada por Héctor A. Tabares Ospina y Juan Guillermo Rivera Berrío [11]

2.2.4 Conversión de binario a decimal

Para convertir un número binario a decimal, basta multiplicar cada dígito binario por la potencia de 2 correspondiente a su posición y luego se suman todos los valores resultantes.

Ejemplo 2.7. Convierte a decimal un número binario sin parte fraccionaria con la siguiente herramienta interactiva.



Interactivo 2.4. Conversión a binario. Escena creada por Héctor A. Tabares Ospina y Juan Guillermo Rivera Berrío [11]

Se muestra en el siguiente ejemplo cómo se convierte a decimal un número binario con parte fraccionaria.

Ejemplo 2.8. Convierte el número $x = 10.111001_2$ a decimal

Se tiene que:

$$\begin{aligned} x &= 1\cdot 2^1 + 0\cdot 2^0 + 1\cdot 2^{-1} + 1\cdot 2^{-2} + \\ &\quad + 1\cdot 2^{-3} + 0\cdot 2^{-4} + 0\cdot 2^{-5} + 1\cdot 2^{-6} \\ &= 1 + 1/2 + 1/4 + 1/8 + 1/64 = 1.890625 \end{aligned}$$

2.3 Representación de los números reales en el ordenador

Un número real x en base 10 se puede escribir de forma que la primera cifra no nula aparecezca inmediatamente a la izquierda del punto decimal, es decir, de la forma

$$x = (-1)^s \times 10^e \times a.m$$

donde

- s es el signo (0 si $x \geq 0$, 1 si $x < 0$)
- $1 \leq a \leq 9$
- $m = m_1 m_2 m_3 \ldots$ con $0 \leq m_i \leq 9$ para todo i

Esta expresión se dice que está normalizada donde:

- s signo
- e exponente
- m mantisa

Ejemplo 2.9. Considera el número $x = 9028.304$ en base decimal, escribe su forma normalizada.

Se puede escribir de la forma

$$x = (-1)^0 \times 10^3 \times 9.028304$$

Como los ordenadores utilizan aritmética binaria, en base 2, cualquier número x se puede escribir normalizado de la forma

$$x = (-1)^s \times 2^e \times 1.m$$

Así, el número x se puede almacenar guardando el signo s, el exponente e y la mantisa m. Como en base 2 el primer dígito que precede a la mantisa es siempre 1, no es necesario almacenarlo. De esta forma se optimiza la representación mediante el llamado bit oculto.

Los números reales se representan en el ordenador según la norma IEEE 754, establecida en 1985 por el Institute of Electrical and Electronics Engineers (IEEE). Esta norma establece dos formatos a partir de la expresión normalizada en coma flotante: simple y doble precision .

> En un computador, la mantisa y el exponente se almacenan en palabras de memoria con un número limitado de bits. La cantidad de bits asignados a cada uno define el rango y la precisión de los números representados.

2.3.1 Simple precisión

Para representar un número real en precisión simple en el ordenador se utiliza:

- 1 bit para el signo.
- 8 bits para el exponente.
- 23 bits para la mantisa.

Teniendo en cuenta que se usan 8 bits para el exponente, sus valores posibles variarían desde 0 hasta 255:

$$\underbrace{00000000}_{8\ bits} \quad \text{hasta} \quad \underbrace{11111111}_{8\ bits} \equiv 1 + 2 + ... + 2^7 = 2^8 - 1 = 255$$

Con objeto de poder considerar las potencias de 2 negativas se hace un desplazamiento del exponente para que represente valores desde -127 a 128. Así, en precisión simple, un exponente real de 0 se almacena

como 127, un exponente de -1 como 126, y un exponente de +1 como 128. En general, la relación entre el exponente real y el almacenado es la siguiente:

$$e_{almacenado} = e_{real} + 127$$

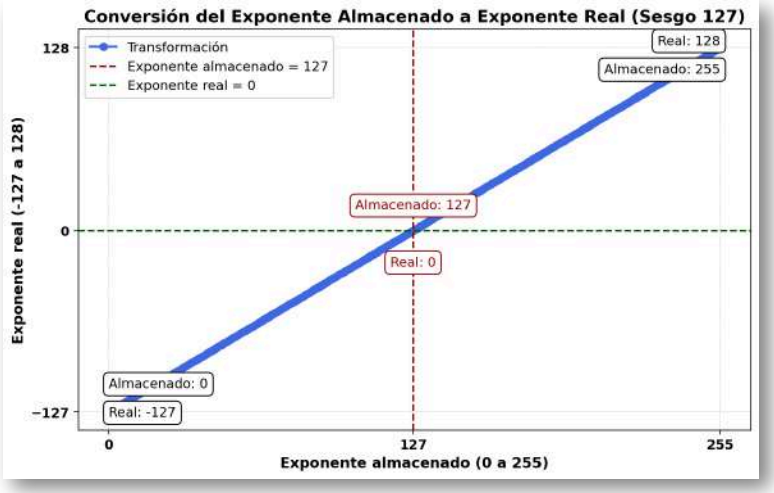


Figura 2.2. Exponente real y almacenado

Al valor 127 se le llama sesgo. El sesgo es un número fijo que depende del formato, en simple precisión es 127 pero en doble precisión es 1023. Con esta representacion, los números que se pueden representar en precisión simple son:

$$x = (-1)^s \times 2^{e-127} \times \ 1.m \qquad 0 < e < 255$$

Los valores extremos del exponente, esto es 0 y 255, se reservan para números especiales (cero e infinito).

Ejemplo 2.10. Para el número 3.125_{10} escribe cómo sería su codificación en precisión simple.

Se tiene que

$$3.125_{10} = 11.001_2 = 1.1001 \times 2^1$$

Por lo tanto, $m = 1001$ que ajustada a 23 bits es $1001\underbrace{0\ldots0}_{19\ bits}$.

Para el exponente, considerando el sesgo, se codificará $e = 127 + 1 = 128 = 2^7$. Este número en binario es 10000000.

En definitiva, la representacion en precisión simple del número decimal 3.125 es la siguiente:

$$0\ \ 10000000\ \ 1001\underbrace{0\ldots0}_{19\ bits}$$

Ejemplo 2.11. Practica en la conversión de números decimales a precisión simple según el estándar IEEE 754 con ayuda de la siguiente herramienta interactiva.

Interactivo 2.5. Conversor a IEEE 32

2.3.2 Doble precisión

En Matlab la precisión que se usará para los cálculos será doble precisión. En este caso se utiliza:

- 1 bit para el signo.
- 11 bits para el exponente.
- 52 bits para la mantisa.

Teniendo en cuenta que el exponente se almacena en 11 bits, los valores que se pueden considerar para el exponente son desde 0 hasta 2047:

$$\underbrace{0....0}_{11\ bits} \quad \text{hasta} \quad \underbrace{1....1}_{11\ bits} \equiv 1 + 2 + ... + 2^{10} = 2^{11} - 1 = 2047$$

Para poder considerar las potencias de 2 negativas se toma un desplazamiento o sesgo del exponente de 1023. Así, es posible representar los siguientes números:

$$x = (-1)^s \times 2^{e-1023} \times \ 1.m \qquad 0 < e < 2047$$

Se consideran las siguientes representaciones especiales:

- Cero: $e = 0, m = 0.$
- Infinito: $e = 2047, m = 0.$
- NAN: $e = 2047, m \neq 0.$

Ejemplo 2.12. Codifica el número 3.125_{10} en doble precisión.

Se tiene

$$3.125_{10} = 11.001_2 = 1.1001 \times 2^1$$

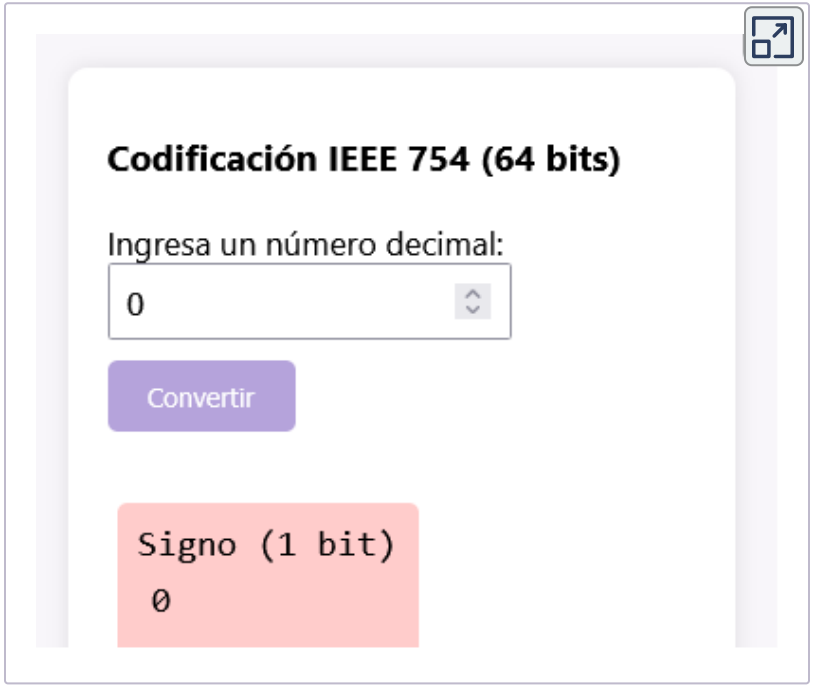
donde la mantisa es $m = 1001$ que ajustada a 52 bits es $1001\underbrace{0\ldots0}_{48\ bits}$.

Respecto al exponente, el valor a almacenar es $e = 1023 + 1 = 1024 = 2^{10}$, que en binario es 10000000000.

El número se almacenará de la forma:

$$\underbrace{0}_{signo}\ \underbrace{10000000000}_{exponente}\qquad \underbrace{10010\ldots0}_{mantisa,\ 48\ últimos\ bits\ 0}$$

Ejemplo 2.13. Practica la conversión de números decimales a doble precisión según el estándar IEEE 754 con la siguiente herramienta interactiva.



Interactivo 2.6. Conversión a IEEE 64

2.3.2.1 Rango y precisión

Dado que la representación de números reales bajo estos formatos es aproximada, hay dos conceptos importantes en la aritmética en punto flotante.

- Rango: Se refiere al conjunto de valores donde existen números representables. Los valores máximo y mínimo normalizados estrictamente positivos representables en doble precisión son:

  Mayor número normalizado: `realmax` en Matlab

  $$2^{1023} \times 1.\underbrace{1....1}_{52\ bits} \equiv 2^{1023}\left(1+\sum_{j=1}^{52}\frac{1}{2^j}\right) \approx 1.79 \times 10^{308}$$

  Menor número normalizado: `realmin` en Matlab

  $$2^{-1022} \times 1.00...\underbrace{0}_{bit\ 52} \equiv 2^{-1022} \approx 2.22 \times 10^{-308}$$

  Si se considera que el bit escondido es cero, se pueden obtener números menores que el mínimo número normalizado mediante la siguiente representación:

  $$x = (-1)^s \times 2^{-1022} \times \ 0.f$$

  Estos números se dicen que están desnormalizados.

- Precisión: Da la diferencia entre dos números representables consecutivos, depende del número de bits de la mantisa. Debido a los 52 bits de la mantisa, la precisión en términos de dígitos decimales es aproximadamente: $log_{10}(2^{52}) \approx 15.95$ dígitos.

Ejemplo 2.14. Obtén el menor número positivo desnormalizado.

El mínimo número desnormalizado positivo ocurre cuando la mantisa tiene un 1 en el bit menos significativo $m = 2^{-52}$. El exponente real será

$$2^{-1022} \cdot 2^{-52} = 2^{-1074}$$

Todos los números menores que este valor se consideran 0.

```
>>2^(-1074)
ans=
    4.9407e-324
>>2^(-1075}
ans=
    0
```

Los números representados en la máquina no están igualmente distribuidos en la recta real, hay muchos más cerca del cero.

El número más pequeño representable después del 1 utilizando representación binaria en punto flotante con doble precisión es:

$$1 = (+1) \times 2^0 \times 1.\underbrace{00\ldots0}_{52}$$

$$1 + \varepsilon = (+1) \times 2^0 \times 1.\underbrace{00\ldots01}_{52}$$

El valor de ϵ se llama épsilon de la máquina y en Matlab se identifica con eps: $\varepsilon_m = 2^{-52}$.

El número más pequeño ε de forma que $1 - \varepsilon \neq 1$ es 2^{-53}. Por tanto, las distancias entre el número que antecede a 1 y el que le precede son respectivamente $\varepsilon_m/2$ y ε_m.

Para cualquier otro número real, la distancia entre x y el número máquina más próximo en Matlab se utiliza `eps(x)`.

2.4 Errores de redondeo y aritmética computacional

2.4.1 Error absoluto y relativo

Para cuantificar el error, se introducen los conceptos de error absoluto y error relativo. Si $\tilde{p}$ es una aproximación de p, se define:

- Error absoluto: $|\tilde{p} - p|$
- Error relativo: $\frac{|\tilde{p}-p|}{|p|}$

El error absoluto indica la desviación de una medición respecto al valor real, expresada en las mismas unidades. El error relativo, en cambio, representa esa desviación como una fracción del valor real; si se multiplica por 100, se obtiene el error porcentual, lo que permite comparar errores en magnitudes de distinta escala.

En general, en los métodos numéricos no se conoce el valor verdadero, por lo tanto tampoco se conocerá el error real. Lo frecuente es tener una estimación del error a partir de una cota positiva del error máximo.

Ejemplo 2.15. Calcula el error absoluto y relativo siendo $p = 3$ y $\tilde{p} = 3.1$. Repite el cálculo para $p = 3\,000$ y $\tilde{p} = 3\,100$.

En el caso $p = 3$, $\tilde{p} = 3.1$ se tiene

$$E_a = |3 - 3.1| = 0.1 \qquad E_r = \frac{E_a}{p} = 3\%$$

Para $p = 3\,000$, $\tilde{p} = 3\,100$, se tiene

$$E_a = |3\,000 - 3\,100| = 100 \ , \quad E_r = \frac{E_a}{p} = 3\%$$

En este ejemplo se muestra que si bien el error absoluto es distinto, el error relativo es el mismo. El error relativo de 3% indica que la medición tiene un error de aproximadamente el 3% respecto al valor real.

Ejemplo 2.16. En 1862 el físico Foucault, utilizando un espejo giratorio, calculó la velocidad de la luz en $298\,000$ km/s. Aceptando como exacta la velocidad de $299\,776$ km/s, estima el error absoluto y el error relativo cometido por Foucault.

Por otra parte, la determinación de la constante universal h realizada por Planck en 1913 dio el valor de 6.41×10^{-27} erg seg. Adoptando el valor de 6.626×10^{-27} erg seg, estima el error absoluto y relativo cometido por Planck.

¿Cuál de las dos medidas es más precisa?

En la tabla siguiente se recogen los errores absoluto y relativo para cada estimación.

Medición	Valor estimado	Valor real	Error absoluto	Error relativo
Velocidad de la luz (Foucault)	298 000	299 776	1 776	5.920661e-03
Constante de Planck	6.41e-27	6.626e-27	2.13e-28	3.216246e-02

Para determinar cuál es la medición más precisa, se analiza el error relativo:

- Foucault tiene un error relativo menor (~0.00592), lo que indica mayor precisión.
- Planck tiene un error relativo mayor (~0.03216), lo que sugiere menor precisión.

2.4.2 Errores de redondeo

En el sistema decimal, existen distintas formas de redondear un número a t decimales. En el proceso denominado truncamiento (o redondeo hacia el cero) se eliminan todos los números ubicados a la derecha del t-ésimo decimal. La magnitud del error puede ser tan grande como 10^{-t}. En el redondeo al más cercano (tambien llamado redondeo óptimo) debe elegirse el número con t decimales que sea más cercano a x.

Ejemplo 2.17. En una máquina hipotética que utilice un sistema decimal con dos dígitos decimales, escribe el número 3.246 usando truncamiento o redondeo.

El número 3.246 se representaría como 3.24 si se usa truncamiento o bien 3.25 si se usa redondeo. El truncamiento elimina todas las cifras que no pueden ser representadas por la máquina, en este caso a partir del tercer decimal. El redondeo, toma la primera cifra no representable, en este ejemplo, sería el 6 y le suma una unidad al último dígito si es mayor o igual a 5 o lo trunca si es menor que 5.

Se considera el número

$$x = (-1)^s \times 2^{e_x} \times 1.f \qquad -1023 < e_x < 1024$$

y se denota por $fl(x)$ a su representación en punto flotante, es decir, al número que mejor representa a x dentro de las limitaciones del sistema de representación numérica, se tendrá

$$|x - fl\,(x)| \leq 2^{e_x} \cdot 2^{-53} \Rightarrow \frac{|x - fl\,(x)|}{|x|} \leq 2^{-53}$$

Por lo tanto, el épsilon de la máquina es una cota superior del error de redondeo en el sistema punto flotante.

Recuerda que $fl(x)$

- Es un número representable en el sistema.
- Es el número más cercano a x dentro del rango representable.
- Si x está exactamente entre dos números representables, el valor de $fl(x)$ dependerá de la política de redondeo (por ejemplo, redondeo al más cercano, truncamiento, etc.).

2.4.3 Propagación de errores

Si $\otimes$ es una operación aritmética como una suma, resta, multiplicación o división, el cálculo sigue el siguiente proceso:

1. Almacena los números x e y en coma flotante.
2. Después realiza la operación con los valores representados.
3. Finalmente da el resultado en punto flotante.

La representación de la operación es entonces

$$fl\left(x \otimes y\right) = fl\left(fl\left(x\right) \otimes fl\left(y\right)\right)$$

En consecuencia, el orden en el que se realizan las operaciones puede modificar el resultado final. Con el siguiente ejemplo se muestra que puede no cumplirse la propiedad asociativa.

Ejemplo 2.18. Considera $x = 10^{20}, y = -10^{20}, z = 1$ y calcula con Matlab/Octave $x + (y + z)$ y $(x + y) + z$.

Ejecuta el siguiente código y observa que los valores obtenidos son 0 y 1, respectivamente.

```
x=10^20;y=-10^20;z=1;
x+(y+z)
(x+y)+z
```

Ejemplo 2.19. Calcula con Matlab/Octave la siguiente operación $x = 1 - 3 \cdot (4/3 - 1)$. Observa que el valor no es cero.

```
x=1- 3*(4/3 - 1)
x =
    2.2204e-16
```

Ejemplo 2.20. Se considera una máquina hipotética con base decimal y 2 dígitos de mantisa, es decir, dos cifras decimales en la parte fraccionaria de la representación normalizada. Calcula la diferencia de 1025 y de 912.3.

El valor exacto es 112.7. El calculo sería de la siguiente manera

$$fl(1025) = 1.02 \cdot 10^3 \quad fl(912.3) = 9.12 \cdot 10^2$$

$$fl(1025 - 912.3) = fl(112.7) = 1.12 \cdot 10^2$$

$$fl(fl(x) - fl(y)) = fl(108) = 1.08 \cdot 10^2$$

Además, cuando se realiza una cadena de operaciones, los errores de redondeo pueden acumularse de modo que resten validez al resultado. Para controlar la propagación de errores de redondeo, es necesario buscar algoritmos con el número mínimo de operaciones convenientemente dispuestas para evitar estos problemas y que a la vez el tiempo de ejecucion sea aceptable.

Para un análisis detallado sobre los errores en cálculos numéricos, como son los errores de redondeo, truncamiento y su propagación, se

2.4.4 Error por cancelación

Este error se produce cuando se calcula la diferencia de dos números bastante próximos y el redondeo del ordenador iguala esa diferencia a cero.

Ejemplo 2.21. Dados $x = 1 + 2^{-54}$ e $y = 1$, calcula con Matlab $x - y$. Realiza la misma operación con $x = 2^{60} + 2^6$ e $y = 2^{60}$.

Nuestro ordenador establecerá en ambos casos que $x - y = 0$ cuando en realidad en el primer caso $x - y \approx 5,5 \cdot 10^{-17}$ y en el segundo 2^6.

```
% Primer ejemplo
x=1+2^-54;y=1;x-y
% Segundo ejemplo
x=2^60+2^6;y=2^60;x-y
```

Ejemplo 2.22. Considera la función

$$f(x) = (e^x - 1)^2 + \left(\frac{1}{\sqrt{1+x^2}} - 1\right)^2$$

Se quiere calcular $f'(1) \approx \frac{f(1+h)-f(1)}{h}$ para valores de h pequeños. Genera un vector de valores de h desde 10^{-1} hasta 10^{-17} y observa qué valor se obtendría para $f'(1)$ utilizando la aproximación anterior. Realiza también el cálculo derivando la función y sustituyendo en el punto 1. Compara ambos resultados.

Se escribe el código que evalía la aproximación de $f'(1)$

```
f=@(x) (exp(x)-1).^2+(1./sqrt(1+x.^2)-1).^2;
k=-1:-1:-17;h=10.^(k);
v=(f(1+h)-f(1))./h;
format longEng
disp([h' v'])
```

Observa que a partir del valor $k = -16$ se obtiene 0 porque se produce cancelación. Realizando la derivada de la función y sustituyendo en el punto 1, se obtiene el valor 9.54865532212975587.

Ejemplo 2.23. Se sabe que $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$. Calcula una aproximación de e^{-40} con los 100 primeros términos de la serie. Repite después el cálculo de e^{-x} teniendo en cuenta que $e^{-x} = \frac{1}{e^x}$. ¿Qué se observa?

Observa que para calcular e^{-40} el valor de x es menor que cero, los términos de la serie exponencial irán cambiando de signo y, cuando $|x|$ es grande, el valor de los sumandos será cada vez más pequeño. Esto implica que en algún momento se producirá un error de cancelación cuando se sume "el siguiente" término.

Si se considera el cálculo tomando el inverso de e^{40}, se evita este problema de cancelación ya que todos los términos son positivos. El cálculo con Matlab/Octave se muestra a continuación.

```
x=40; %Inverso del número a calcular e^-40=1/e^40
suma=1;p=1;
for k=1:99
    p=p*(x/k);
    suma=suma+p;
end
1/suma
x=-40; %Sin considerar el inverso
suma=1;p=1;
for k=1:99
    p=p*(x/k);
    suma=suma+p;
end
suma
%Valor que da Matlab
exp(-40)    % 4.248354255291589e-018
```

En los siguientes ejemplos, se muestra como una reformulación del cálculo evita el problema de cancilación en algunas situaciones.

Ejemplo 2.24. Considera la función

$$f(x) = \frac{1}{1+2x} - \frac{1-x}{1+x}$$

Observa que para valores x próximos a 0 se puede producir cancelacion evaluando esta función en puntos próximos a 0 directamente. Observa cómo evitar esta cancelación considerando la expresión equivalente resultado de restar las dos fracciones

$$f(x) = \frac{2x^2}{(1+2x)(1+x)}$$

Se escribe el código Matlab/Octave para realizar los cálculos.

```
%Se considera la primera expresión como resta de
fracciones
f=@(x) 1./(1+2*x)-(1-x)./(1+x);
%Se realiza la suma para evitar la cancelación
g=@(x) 2*x.^2./((1+2*x).*(1+x));
format longEng
%Se considera un vector de valores próximos a cero
k=-1:-1:-17;
h=10.^k;
%Se evalúan ambas expresiones
f1=f(h);
g1=g(h);
%Se imprimen los valores de h y los valores
%obtenidos para comparar
disp([h' f1' g1'])
```

Ejemplo 2.25. Reformula la expresión $f(x) = \frac{1-\cos x}{x}$ para evitar la cancelacion en puntos próximos a cero obteniendo una expresión equivalente multiplicando y dividiendo por $1+\cos x$.

La expresión equivalente es $f(x) = \frac{1-cos^2(x)}{x(1+cos/x)} = \frac{sen^2(x)}{x(1+cos/x)}$.

```
f=@(x) (1-cos(x))./x;
%Se multiplica y divide por (1+cos(x))
g=@(x) sin(x).^2./(x.*(1+cos(x)));
k=-1:-1:-17;h=10.^k;f1=f(h);g1=g(h);
format longEng
disp([h' f1' g1'])
```

Ejemplo 2.26. Reformula la expresión

$$f(x) = \sqrt{x + \frac{1}{x}} - \sqrt{x - \frac{1}{x}}$$

para valores grandes de forma que se evite la cancelación.

Multiplicando y dividiendo por el conjungado del numerador

$$f(x) = \frac{\sqrt{x^2+1} - \sqrt{x^2-x}}{x} = \frac{2}{x\left(\sqrt{x^2+1} + \sqrt{x^2-x}\right)}$$

```
f=@(x) sqrt(x+1./x)-sqrt(x-1./x);
%Se multiplica y divide por el conjugado
g=@(x) 2./(x.*(sqrt(x.^2+1)+sqrt(x.^2-x)));
k=1:25;h=10.^k;f1=f(h);g1=g(h);
format longEng
disp([h' f1' g1'])
```

2.4.5 Error por desbordamiento

Con frecuencia, una operación aritmética con dos números válidos da como resultado un número tan grande o tan pequeño que el ordenador no puede manejarlo; como consecuencia, se produce un overflow o un underflow, respectivamente.

Ejemplo 2.27. Multiplica por 2 el valor máximo representable en punto flotante, `realmax` para ilustrar el concepto de desbordamiento. Divide después por 2 el valor mínimo positivo normalizado, `realmin` y observa que se representa por cero.

Se realizan los cálculos con Matlab/Octave.

```
% Valor máximo representable
maxVal = realmax;
% Se intenta exceder el valor máximo
num = maxVal * 2;
disp(['Desbordamiento resulta en: ', num2str(num)]);
```

```
% Valor mínimo positivo normalizado representable
minVal = realmin;
% Se intenta obtener un valor menor al mínimo
num = minVal * 1e-100;
disp(['Subdesbordamiento resulta en: ',
num2str(num)]);
```

2.5 Inestabilidad numérica

En el siguiente ejemplo se verá que si el algoritmo no es adecuado, los errores de redondeo pueden conducir a resultados incorrectos. Este fenómeno, conocido como inestabilidad numérica, puede evitarse seleccionando algoritmos más apropiados.

Ejemplo 2.28. Para cada $n \geq 0$ se considera la integral

$$I_n = \int_0^1 x^n sen\pi x \, dx$$

Utilizando integración por partes, se sabe que

$$I_o = \frac{2}{\pi} \quad , \quad I_1 = \frac{1}{\pi}$$
$$I_{n+2} = \frac{1}{\pi}\left[1 - \frac{(n+2)(n+1)}{\pi} I_n\right] \qquad n \geq 0$$

Calcula utilizando esta recurrencia el valor de I_{30} e I_{31} a partir de $I - 0$ e I_1. ¿Son los resultados aproximaciones satisfactorias de la integral? Propón una alternativa eficiente despejando I_n en función de I_{n+2} para calcular en forma descente I_{30} e I_{31} teniendo en cuenta que I_n para n grande es próxima a cero.

Realizando el cálculo directo I_{30} e I_{31} de forma iterativa con Matlab/Octave se obtienen valores negativos. El valor de I_{30} que se otiene es -0.357484694711417 y para el valor de I_{31} se obtiene -2.312387998625434

```
a=2/pi; v(1)=1/pi;v(2)=1/pi-2*a/pi^2;
format longEng
for k=3:31;
    v(k)=1/pi-k*(k-1)*v(k-2)/pi^2;
end
disp([[1:31]' v'])
```

Esto es imposible porque el integrando es positivo y por tanto las integrales deben ser positivas.

Se verá cómo obtener los valores de estas integrales de otra forma. Despejando I_n en función de I_{n+2} se tiene:

$$\frac{(n+2)(n+1)}{\pi} I_n = 1 - \pi I_{n+2}$$
$$I_n = \frac{\pi}{(n+2)(n+1)}(1 - \pi I_{n+2})$$
$$I_n = \frac{\pi^2}{(n+2)(n+1)}\left(\frac{1}{\pi} - I_{n+2}\right)$$

Si se toman límites cuando n tiene a infinito, se puede ver que la sucesión I_n tiende a cero. Basta tener en cuenta que

$$\left| \int_0^1 x^n sen\,(\pi x)\,dx \right| \leq \int_0^1 x^n dx = \frac{x^{n+1}}{n+1}\Big|_{x=0}^{x=1} = \frac{1}{n+1}$$

Por ello, para utilizar esta expresión hacia atrás, se debe partir de dos valores I_{n+2} e I_{n+1} próximos a cero para un valor de n alto. En el siguiente código se ha considerado I_{56} e I_{55} como 0.

```
a=56;
w(56)=0;w(55)=0;
for k=a-2:-1:1;
    w(k)=pi^2/((k+2)*(k+1))*(1/pi-w(k+2));
end
disp([[1:56]' w'])
```

El valor de I_{30}, utilizando este cálculo, es 0.003139287076703, y el valor de I_{31} es 0.002950500704051.

Ejemplo 2.29. Se considera la integral:

$$y_n = \int_0^1 \frac{x^n}{x+5} dx$$

Teniendo en cuenta los cálculos siguientes, se puede encontrar una ley de recurrencia para el valor de y_n:

$$y_n = \int_0^1 \frac{x^{n-1}x}{x+5} dx = \int_0^1 \frac{x^{n-1}\,(x+5-5)}{x+5} dx =$$

$$= \int_0^1 x^{n-1} dx - 5\int_0^1 \frac{x^{n-1}}{x+5} dx = \frac{1}{n} - 5y_{n-1}$$

Se pide calcular de forma recurrente el valor de y_{20} con la expresión obtenida: $y_n = \frac{1}{n} - 5y_{n-1}$. Considera después la fórmula de recurrencia en sentido descendente para obtener el valor de y_{20} resultado de despejar y_{n-1} en función de y_n:

$$y_{n-1} = \frac{1}{5n} - \frac{y_n}{5}$$

Justifica los resultados obtenidos.

En el cálculo recurrente de $y_n = \frac{1}{n} - 5y_{n-1}$ se puede observar que se obtiene un valor negativo, lo cual es absurdo porque el integrando es positivo. Este error se produce porque se restan dos números del mismo signo de magnitud similar.

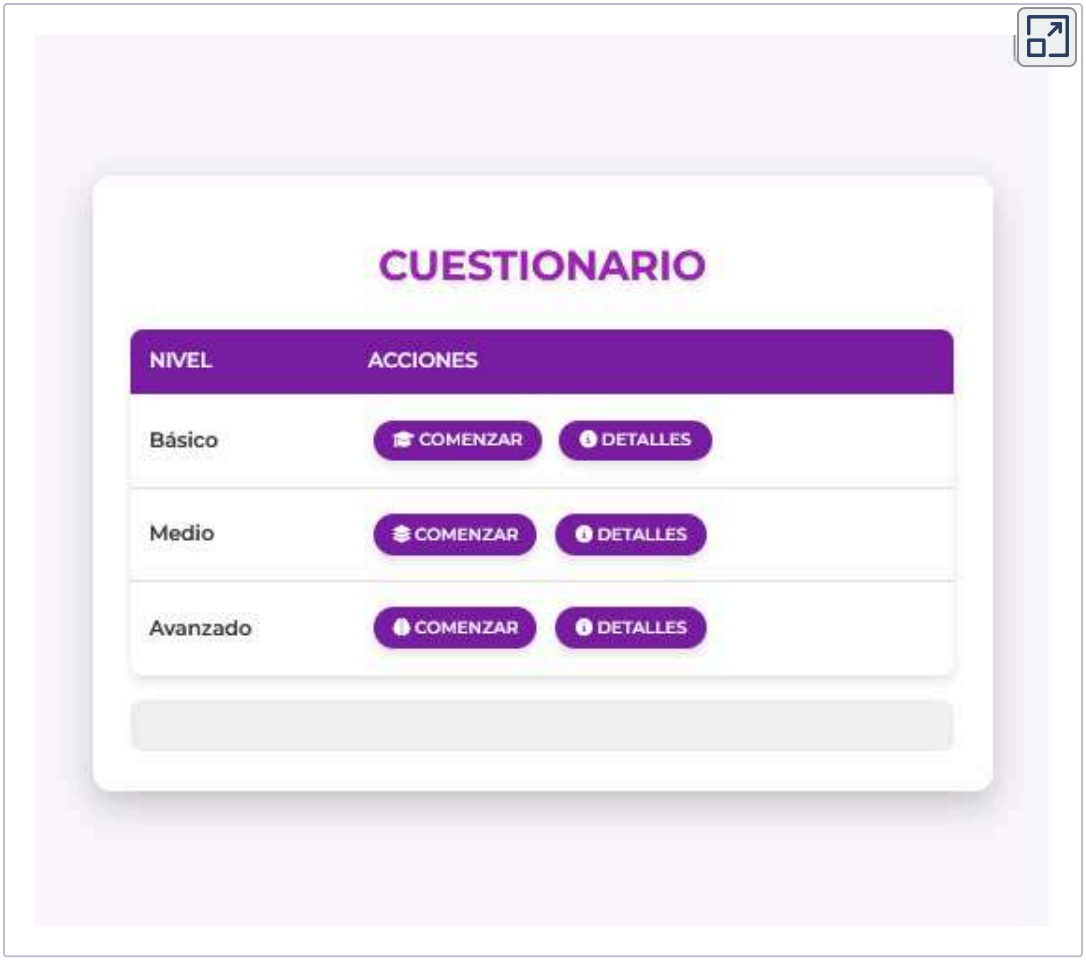
El código para obtener y_{20} con esta ley de recurrencia es:

```
clear all
y(1)=1-5*(log(6)-log(5));
format longEng
for k=2:20;
    y(k)=1/k-5*y(k-1);
end
disp([[1:20]' y'])
```

Tomando límites en $y_n = \frac{1}{n} - 5y_{n-1}$ cuando n tiende a ∞, se ve que la sucesión y_n converge a 0. Por lo tanto, considerando por ejemplo $y_{60} = 0$, se calcula de nuevo el valor de y_{20} utilizando la ley de recurrencia descendente $y_{n-1} = \frac{1}{5n} - \frac{y_n}{5}$.

```
y(60)=0;
format longEng
for k=60:-1:2;
    y(k-1)=1/(5*k)-y(k)/5;
end
disp([(1:60)' y'])
%y(20) es 7.997523028232164e-03
```

2.6 Autoevaluación



Interactivo 2.7. Actividad de autoevaluación

2.7 Ejercicios

1. Calcula los números cuya representación en punto flotante de precisión doble es
   - $0\ 10000000100$
     $0111110000000000000000000000000000000000000000000000000000$
   - $0\ 10000000010$
     $1011001000100000000000000000000000000000000000000000000000$

Solución

2. Escribe la representación normalizada en el estandar IEEE de doble precisión del número decimal -17.25.

Solución

3. ¿Qué estimación de las siguientes es más precisa

$$\frac{9}{11} = 0.818 \quad \sqrt{18} = 2.243$$

Solución

4. Calcula el error absoluto, el error relativo y el número de cifras significativas de la aproximación de π mediante
   - $p* = 22/7$ (Antiguo Egipto, siglo XXVI a.C.).
   - $p* = 223/71$ (Arquímedes, Antigua Grecia, siglo III a.C.).
   - $p* = 3.14159$ (Liu Hui, China, año 265).
   - $p* = 355/113$ (Zu Chongzhi, China, año 480).

Solución

5. Determina el mayor intervalo en el cual debe estar $p*$ para que aproxime a $\sqrt{2}$ con un error menor que 10^{-2}.

Solución

6 Resuelve el siguiente sistema con Matlab

$$\begin{cases} 17x_1 + 5x_2 = 22 \\ 1.7x_1 + 0.5x_2 = 2.2 \end{cases}$$

Es sencillo ver que una solución obvia es $x_1 = x_2 = 1$ aunque el sistema tiene infinitas soluciones ya que la segunda ecuacion es la primera dividida por 10. ¿Qué sucede al resolverlo con Matlab/Octave?

Solución

7 Se quiere calcular p_n como $(1/3)^n$ para $n > 0$ considerando $p_0 = 1$ definiendo $p_n = (1/3)p_{n-1}$ para $n > 1$. Otra forma de generar la sucesión es definiendo $p_0 = 1$ y $p_1 = 1/3$ y calculando para $n > 2$

$$p_n = \frac{10}{3}p_{n-1} - p_{n-2}$$

¿Alguno de los dos métodos es inestable?

Solución

8 Calcula la siguinete suma $\sum_{n=1}^{10000000} \frac{1}{n}$ en orden usual y luego en orden opuesto, es decir, empezando la suma desde el último sumando hacia atrás. Explicar las diferencias obtenidas e indique cuál es el resultado más preciso.

Solución

9 Indica cómo evaluar con Matlab la expresión

$$\frac{1}{x+h} - \frac{1}{x}$$

para los valores de h siguientes 10^{-2}, 10^{-4}, ..., 10^{-22} tomando como valor x un número cualquiera. Escribe una reformulación de esta expresión de forma que no se produzca cancelación. Utiliza el formato longEng para mostrar los resultados en ambos casos.

Solución

10 Justifica qué ocurre cuando se evalúa la expresión

$$\sqrt{x^2+1} - 1$$

para valores próximos a cero. ¿Cómo se podría hacer más estable? Evalúa estas expresiones para 20 puntos equidistantes entre 0 y 0.2.

Solución

11 Calcula una fórmula alternativa para la expresión

$$-\log\left(x-\sqrt{x^2-1}\right)$$

que no sea susceptible de pérdida de cifras significativas cuando x es grande y positiva. Analiza la diferencia de resultado con una y otra expresión con valores de x que sean potencias de 10 de exponente natural entre 6 y 16.

Solución

12 Considera la sucesion recurrente siguiente convergente a π

$$x_1 = 2 \quad x_{n+1} = 2^{n-\frac{1}{2}}\sqrt{1-\sqrt{1-4^{1-n}x_n^2}}$$

Explica el comportamiento que se obtiene al calcular con Matlab/Octave el valor de x_{550}.

Solución

13 Considera

$$y_n = \int_0^1 x^n e^x dx \quad (n \geq 0)$$

Utiliza integración por partes para obtener la siguiente relación de recurrencia

$$y_{n+1} = e - (n+1)\, y_n$$

Calcula y_{20} utilizando esta recurrencia. ¿Qué se observa? ¿Qué ocurre si se utiliza la ley de recurrencia hacia atrás siguiente: $y_n = \frac{e-y_{n+1}}{n+1}$?

Solución

RESUMEN

En este capítulo se han visto los fundamentos sobre los errores en cálculo numérico, destacando cómo los sistemas computacionales representan y manipulan números, y cómo esto puede originar errores que afectan a los resultados.

Además, se ha analizado los errores de redondeo y truncamiento, destacando la importancia de desarrollar algoritmos numéricos que minimicen estos problemas en aplicaciones prácticas.

Los aspectos clave analizados en este tema son:

1. **Introducción a los errores en cálculo numérico.** Los errores en los sistemas computacionales son inevitables debido a la naturaleza finita con la que se representan números reales y se realizan operaciones matemáticas en el ordenador. Esta limitación se debe al uso de representaciones en punto flotante, que introducen imprecisiones al redondear o truncar los valores. Entender estas limitaciones es esencial para abordar problemas numéricos con precisión.

2. **Clasificación de los errores**. Los errores numéricos se clasifican principalmente en dos tipos: errores de redondeo y errores de truncamiento. Los errores de redondeo ocurren al ajustar un número real a la precisión limitada del ordenador, mientras que los errores de truncamiento surgen al aproximar funciones o procesos matemáticos, como por ejemplo cuando se utilizan aproximaciones por series para representar funciones. Además, los errores de los algoritmos aparecen también cuando éstos no convergen convenientemente hacia la solución deseada.

3. **Minimización de errores**. Reducir el impacto de los errores numéricos requiere el desarrollo de algoritmos y estrategias específicas, como el control de precisión, el análisis de estabilidad de los métodos y la elección de representaciones numéricas adecuadas. Estas herramientas permiten minimizar la propagación de errores a lo largo de los cálculos y garantizar la validez de los resultados, especialmente en aplicaciones como la ingeniería, la física y las finanzas.

3 RESOLUCIÓN APROXIMADA ECUACIONES NO LINEALES

Capítulo III

Resolución aproximada de ecuaciones no lineales

3.1 Introducción

> El propósito de los métodos numéricos para resolver ecuaciones no lineales es encontrar raíces aproximadas, ya que la mayoría de las ecuaciones no tienen soluciones exactas o estas no son computacionalmente viables. Mediante métodos iterativos se va consiguiendo aproximar la solución dentro de un margen de error considerado [7].

Entre las aplicaciones más importantes de los métodos numéricos está la resolución de ecuaciones no lineales, una tarea esencial en disciplinas como física, ingeniería, economía y muchas otras áreas de la ciencia. Estas ecuaciones, donde la variable aparece de forma no lineal son clave para modelar fenómenos reales, desde el diseño de estructuras hasta la dinámica de sistemas físicos. Sin embargo, muchas veces estas ecuaciones no tienen soluciones exactas en términos algebraicos, por lo que se buscan soluciones aproximadas.

Se considera por ejemplo la siguiente ecuación obtenida a partir de la segunda ley de Newton para calcular la velocidad de un paracaidista,

$$v = \frac{g \cdot m}{c}\left(1 - e^{-\frac{c}{m}t}\right)$$

donde la velocidad v depende de la variable independiente tiempo, t, g es la constante de gravitación, c el coeficiente de resistencia y m es la masa del paracaidista.

Masa (kg): 70
Constante de Resistencia (Ns/m): 25
Tiempo máximo (s): 60

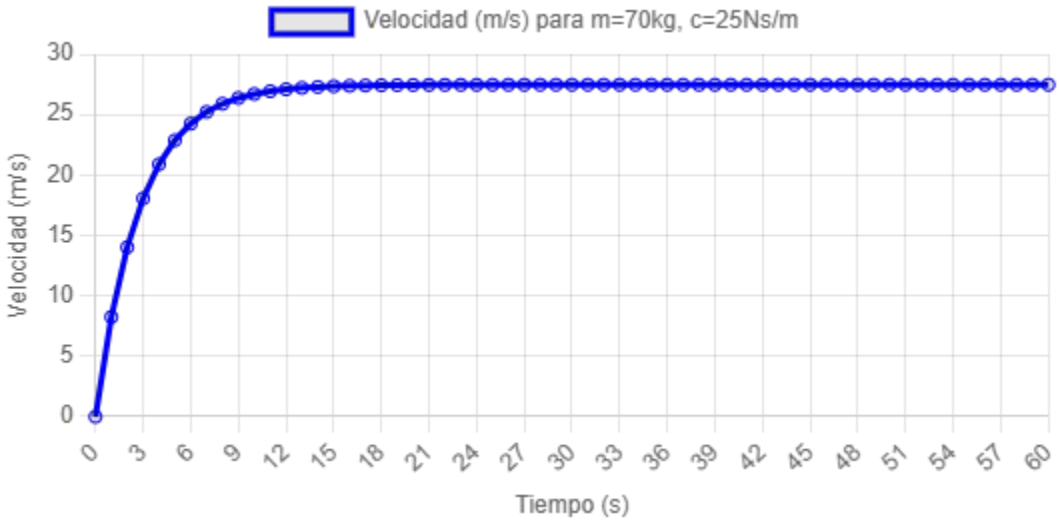


Figura 3.1. Simulador de la velocidad de un paracaidista

Si se quisiera obtener el coeficiente de resistencia del paracaidista con una masa dada, ¿cómo se obtendría este valor para alcanzar una velocidad determinada en un periodo establecido? La respuesta se obtendría calculando c que hace cero a la siguiente función

$$f(c) = \frac{g \cdot m}{c}\left(1 - e^{-\frac{c}{m}t}\right) - v$$

esto es, habría que calcular el valor de c tal que $f(c) = 0$.

Cuando ocurre como en este caso, y no es posible encontrar analíticamente el cero de la función o pudiendo encontrarlo su cálculo es complicado, se utilizan algoritmos que determinan, en un número finito de pasos, un valor aproximado de la raíz buscada. Estos métodos de aproximación generalmente son iterativos y conducen a una solución a través de una sucesión $x_0, x_1, ..., x_k, ...$ que converge al valor α que cumple $f(\alpha) = 0$.

A lo largo de la historia, la resolución de ecuaciones no lineales ha evolucionado desde métodos manuales hasta complejos algoritmos implementados en ordenadores modernos. En la antigüedad, los babilonios y griegos ya aplicaban principios numéricos para resolver problemas matemáticos. Así, el método de bisección, que tiene sus orígenes en la antigüedad, utiliza un enfoque simple: dividir un intervalo y localizar la raíz mediante evaluaciones intermedias.

Otros métodos, como el de Newton-Raphson, desarrollado en el siglo XVII por Isaac Newton [1] y Joseph Raphson [2], aprovechan herramientas del cálculo diferencial utilizando la pendiente de las tangentes de la función.

En el siglo XX, con el desarrollo de la computación, matemáticos como Alan Turing[3] y John von Neumann[4] sentaron las bases para implementar estos métodos en máquinas, haciendo posible resolver problemas que antes eran inabordables.

---

[1] Isaac Newton (1642-1727) fue un físico, matemático y astrónomo inglés, considerado uno de los científicos más influyentes de la historia. Desarrolló las leyes del movimiento y la gravitación universal, así como importantes contribuciones al cálculo, óptica y mecánica. Su obra más destacada, Philosophiæ Naturalis Principia Mathematica, sentó las bases de la física clásica.

[2] Joseph Raphson (1648-1715) fue un matemático inglés conocido por su trabajo en análisis numérico. Publicó el método de Newton-Raphson en su obra Analysis Aequationum Universalis (1690), una técnica iterativa para encontrar raíces de ecuaciones no lineales. Su versión mejorada del método de Newton fue más algebraica y general, contribuyendo al desarrollo de la matemática aplicada.

[3] Alan Turing (1912–1954) fue un matemático británico, considerado uno de los padres de la computación moderna. Desarrolló el concepto de la máquina de Turing, base teórica de los algoritmos, y desempeñó un papel crucial en la Segunda Guerra Mundial al descifrar los códigos nazis de la máquina Enigma. Su trabajo sentó las bases de la inteligencia artificial y la informática teórica..

[4] John von Neumann (1903-1957) fue un matemático, físico y pionero de la computación húngaro-estadounidense. Contribuyó al desarrollo de la teoría de juegos, la mecánica cuántica y los métodos numéricos en computación científica. Fue clave en la creación de las primeras computadoras y en la formulación de la arquitectura de Von Neumann, base de los sistemas computacionales modernos.

En el siguiente video se da una visión rápida de los métodos más importantes que se tratan en este capítulo.

Video 3.1. Métodos de resolución de ecuaciones no lineales

Se comenzará con el método de bisección, una técnica basada en el teorema de Bolzano, que garantiza la existencia de raíces en un intervalo y ofrece una convergencia segura aunque lenta. Luego se estudiará el método del punto fijo, que transforma la ecuación en una forma iterativa y permite analizar la convergencia local y global de la solución. Posteriormente, se explorará el método de Newton-Raphson, que emplea la derivada de la función para obtener una convergencia rápida, aunque con ciertas limitaciones respecto a su punto de inicio. Finalmente, se analizará el método de la secante, que evita el cálculo de derivadas.

A la vez que se van mostrando los distintos métodos, se irán presentado ejemplos prácticos para ilustrar el comportamiento de cada uno de ellos, comparando sus ventajas y limitaciones en diferentes situaciones. Para más información, se puede consultar [10] y [6].

3.2 Raíz o cero de una función

A cualquier solución de la ecuación $f(x) = 0$ se le llama cero o raíz de la funcion. Cuando f es una función continua en un intervalo abierto I, α un punto de I y $m \geq 1$ un número entero, se dice que α es un cero de f de multiplicidad m si

$$f(x) = (x - \alpha)^m q(x) \qquad x \neq \alpha$$

Se puede demostrar que si f es $C^m(I)$, es decir, es derivable hasta el orden m con derivadas continuas, y α es un punto de I, la función tiene un cero de multiplicidad m en α si

$$f(\alpha) = f'(\alpha) = ... = f^{(m-1}(\alpha) = 0, \qquad f^{(m}(\alpha) \neq 0$$

3.2.1 Localización y separación de raíces

En un primer paso para obtener las raíces de una función, se busca un intervalo donde se encuentre uno o varios ceros de la función. Posteriormente, se buscan intervalos que contengan una sola raíz. El siguiente teorema ayuda en esta localización y separación de raíces.

TEOREMA DE BOLZANO: Si f es una función continua en $[a, b]$ de manera que $f(a)f(b) < 0$, entonces existe al menos un cero de la función en el intervalo (a, b).

El hecho de que la función f tenga un cero, no significa que sea único. Si se añaden condiciones adicionales, como por ejemplo la monotonía estricta de f en un intervalo I, sí se podría garantizar que la solución de $f(x) = 0$ es única en dicho intervalo.

Se recuerda que en el caso de que la función sea derivable, si $f'(x) > 0$ en todo punto de I la función es estrictamente creciente en I. Análogamente, si $f'(x) < 0$ en I entonces es estrictamente decreciente.

Si la función es estrictamente creciente o estrictamente decreciente en un intervalo donde hay cambio de signo en los extremos, la función tiene un único cero en dicho intervalo.

3.3 Algoritmos iterativos

Para la aproximación al cero de la función f se utilizan métodos iterativos. En el caso de los de un paso, estos métodos consisten en:

1. Partir de un valor inicial x_0 que se supone suficientemente próximo a la solución buscada.
2. Iterar, es decir, obtener el término x_{k+1} a partir de x_k mediante una fórmula de la forma $x_{k+1} = G(x_k)$ con $k \geq 0$.

Este proceso depende tanto de la función G como del punto inicial x_0 y los algoritmos a utilizar deberán dar respuesta a cuestiones sobre su convergencia como son:

1. la sucesión x_k que se obtiene mediante el método iterativo, ¿es convergente?
2. en caso de ser convergente, ¿es el límite de la sucesión la solución de $f(x) = 0$?
3. ¿cuántas iteraciones hay que realizar para conseguir que el error de aproximación sea menor que una cantidad prefijada?
4. ¿cómo evoluciona el error al realizar las iteraciones?

3.3.1 Convergencia de un algoritmo

Se dice que un algoritmo converge globalmente, si la sucesión generada es convergente hacia una solución de la ecuación cualquiera que sea el punto inicial x_0 que se tome.

Cuando la solución converge hacia la solución únicamente si el punto x_0 pertenece a un cierto entorno, la convergencia se dice que es local.

No siempre es mejor considerar un algoritmo que converja globalmente, a veces, un algoritmo localmente convergente, puede tener una "velocidad de convergencia" mayor si bien este tipo de algoritmos plantean la dificultad de cómo elegir el intervalo adecuado donde considerar x_0.

El método de aproximaciones sucesivas o punto fijo, que consiste en partir de un punto x_0 y calcular los demás términos de la sucesión x_n de la forma $x_{n+1} = g(x_n)$ para obtener un punto fijo de g, tiene una convergencia más lenta que el método de Newton que converge localmente.

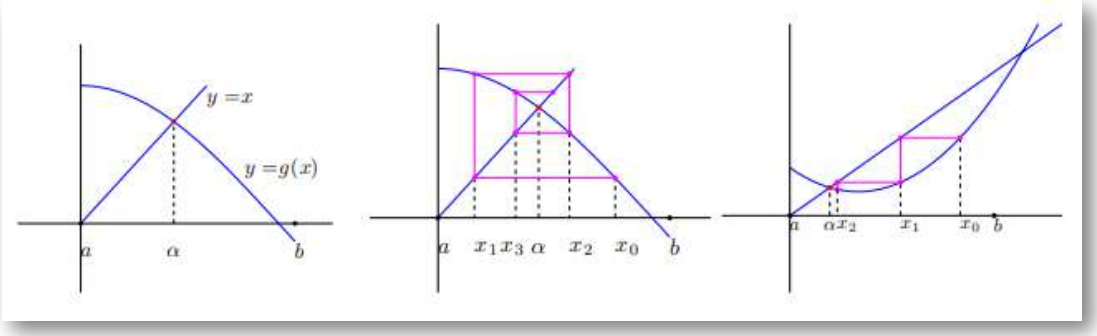


Figura 3.2. Representación método del punto fijo

3.3.2 Velocidad de convergencia

Se distinguen distintas velocidades de convergencia que se pasarán a definir.

Dada una sucesión de números reales, $\{x_k\}_{k=1}^{\infty}$, convergente hacia un punto $\overline{x}$, se dice que su orden o velocidad de convergencia es p, con $p \geq 1$ si existe una constante $C_p \geq 0$ tal que:

$$|x_{k+1} - \overline{x}| \leq C_p |x_k - \overline{x}\,|^p$$

para todo $k \geq k_0$ siendo k_0 entero.

En particular,

- si p=1 y $C_p < 1$, la convergencia es lineal. Si $C_p \geq 1$, la convergencia es sublineal.
- si p=2, la convergencia es cuadrática.
- si p=3, la convergencia es cúbica.

Ejemplo 3.1. Se considera la sucesión $x_k = \frac{1}{6} - \frac{1}{3}\left(-\frac{1}{2}\right)^k$ que converge a $1/6$. Se ve que la convergencia es lineal.

Se puede comprobar que

$$\left|x_{k+1} - \frac{1}{6}\right| \leq \frac{1}{2}\left|x_k - \frac{1}{6}\right|$$

ya que

$$\left|x_{k+1} - \frac{1}{6}\right| = \frac{1}{3 \cdot 2^{k+1}} \qquad \left|x_k - \frac{1}{6}\right| = \frac{1}{3 \cdot 2^k}$$

Ejemplo 3.2. Comprueba que las sucesiones $a_k = 1/k, b_k = 1/k^3, c_k = 1/2^k$ convergen linealmente a 0 y que la sucesion $d_k = 3^{-2^k}$ converge cuadráticamente a 0.

En efecto,

$$a_k = \frac{1}{k} \quad \left|\frac{1}{k+1} - 0\right| < \left|\frac{1}{k} - 0\right| \quad p = 1$$

$$b_k = \frac{1}{k^3} \quad \left|\frac{1}{(k+1)^3} - 0\right| < \left|\frac{1}{k^3} - 0\right| \quad p = 1$$

$$c_k = \frac{1}{2^k} \quad \left|\frac{1}{2^{k+1}} - 0\right| < \left|\frac{1}{2^k} - 0\right| \quad p = 1$$

Para $d_k = 3^{-2^k}$

$$\left(3^{2^k}\right)^2 = 3^{2^{k+1}} \Rightarrow \left|\frac{1}{3^{2^{k+1}}} - 0\right| < \left|\frac{1}{3^{2^k}} - 0\right|^2 \quad p = 2$$

Ejemplo 3.3. Supón que $C_p = 0.5$ y que $|x_k - \overline{x}| < 0.01$. Considerando $p = 1$ y $p = 2$, analiza cómo se aproxima al valor límite, que se denotará por $\overline{x}$, considerando el término $k+1$ y $k+2$ de la sucesión $\{x_k\}$.

Se tiene que

- Si $p = 1$,

$$|x_{k+1} - \overline{x}| < C_p\,|x_k - \overline{x}| < 5 \cdot 10^{-1} \cdot 10^{-2}$$

$$|x_{k+2} - \overline{x}| < C_p\,|x_{k+1} - \overline{x}| < 5 \cdot 10^{-1} \cdot (5 \cdot 10^{-3}) = 25 \cdot 10^{-4}$$

- Si $p = 2$,

$$|x_{k+1} - \overline{x}| < C_p\,|x_k - \overline{x}|^2 < 5 \cdot 10^{-1} \cdot (10^{-2})^2 = 5 \cdot 10^{-5}$$

$$|x_{k+2} - \overline{x}| < C_p\,|x_{k+1} - \overline{x}|^2 < 5 \cdot 10^{-1} \cdot (5 \cdot 10^{-5})^2 = 125 \cdot 10^{-11}$$

Se dice que la sucesión $\{x_k\}_{k=1}^{\infty}$ converge superlinealmente a $\bar{x}$, si existe una sucesión $\{\varepsilon_k\}_{k=1}^{\infty}$ convergente a 0 tal que

$$|x_{k+1} - \bar{x}| \leq \varepsilon_k |x_k - \bar{x}| \quad \forall k \geq 0$$

Se puede observar que si la sucesión tiene una velocidad de convergencia p mayor que 1 entonces converge superlinealmente. Bastaría tomar

$$\varepsilon_k = C_p |x_k - \bar{x}|^{p-1} \quad \forall k \geq 0$$

El siguiente resultado permite determinar un test de parada de las iteraciones cuando la convergencia es superlineal.

Si $\{x_k\}_{k=1}^{\infty}$ converge superlinealmente a $\bar{x}$ entonces

$$\lim_{k \to \infty} \frac{|x_{k+1} - x_k|}{|x_k - \bar{x}|} = 1$$

En este caso, una vez establecida la precisión que se quiere alcanzar, $\varepsilon > 0$, puede considerarse como test de parada:

$$|x_{k+1} - x_k| < \varepsilon$$

se compara así dos iteraciones consecutivas y se mira la coincidencia de dígitos que se considere (fijado por el valor de ϵ).

Ejemplo 3.4. Cuando la convergencia es lineal puede ocurrir que la diferencia entre los valores obtenidos en dos iteraciones, $|x_{k+1} - x_k|$, sea pequeña y, sin embargo, x_{k+1} esté alejado de la solución. Considerar la sucesión $x_{k+1} = \frac{x_k}{1.1} + 3$ con $x_0 = 100$ que converge a 33. Analiza la diferencia entre dos términos consecutivos para k desde 1 hasta 18.

El límite de la sucesión se obtiene resolviendo la ecuación

$$\bar{x} = \frac{\bar{x}}{1.1} + 3 \Rightarrow x\left(1 - \frac{1}{1.1}\right) = 3 \Rightarrow \bar{x} = 33$$

Se analiza en la siguiente tabla cómo los valores de la sucesión están próximos y, sin embargo, están lejos del valor límite.

k	x_k	$\lvert x_{k+1} - x_k \rvert$
0	100.0000	-
1	93.0909	6.9091
..	..	..
7	69.5798	2.1570
8	68.9844	0.5954
9	68.0767	0.9077
10	67.7970	0.2797
11	67.6337	0.1633
12	67.5398	0.0939
13	67.4907	0.0491
14	67.4633	0.0274
15	67.4476	0.0157
16	67.4387	0.0089
17	67.4330	0.0057
18	67.4295	0.0035

3.4 Método de la bisección

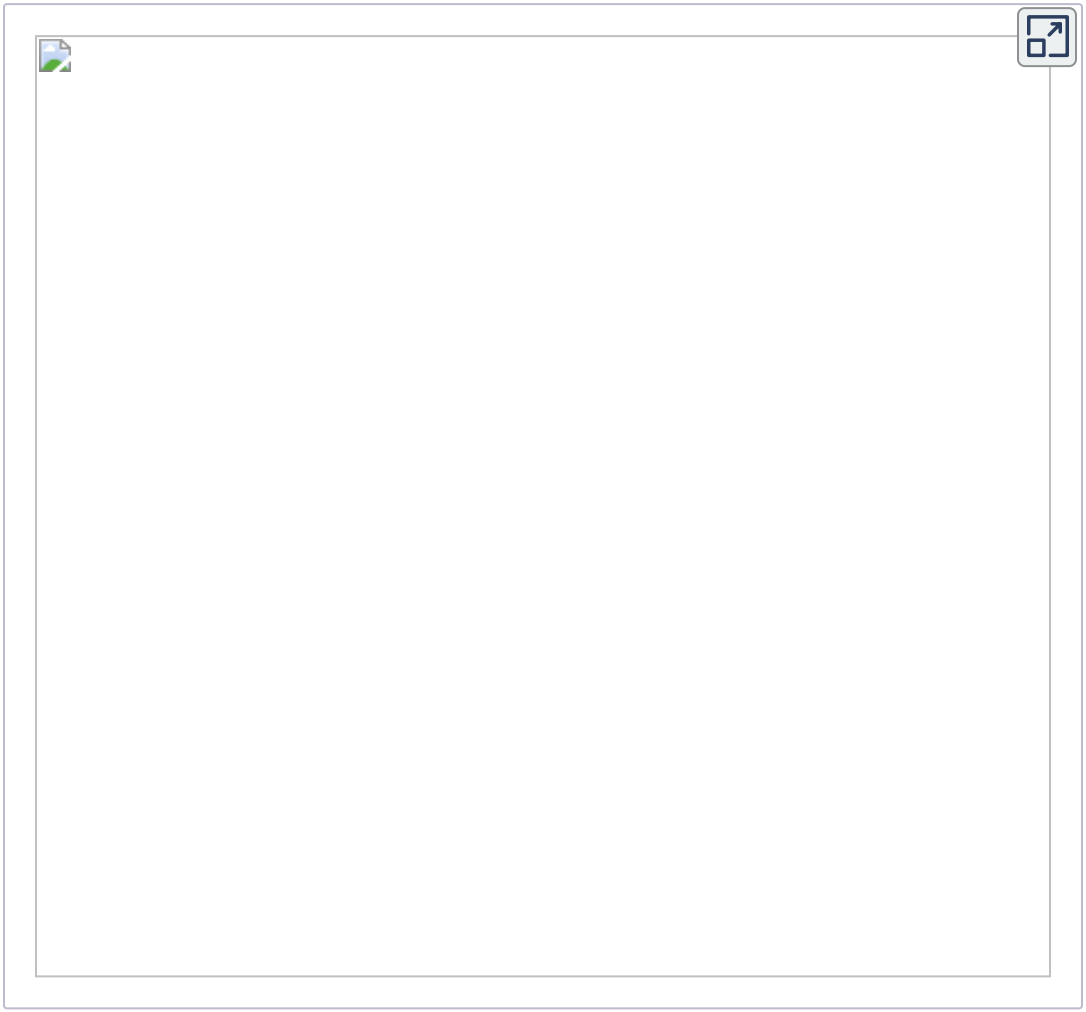
Aplicando el teorema de Bolzano, una forma sencilla de aproximar el cero de una función continua es dividir el intervalo en dos y elegir el subintervalo en el que se sigue produciendo el cambio de signo. Repitiendo este proceso se puede llegar a conseguir una aproximación de la solución de $f(x) = 0$ tan precisa como se desee.

Video 3.2. Método de la bisección

Los pasos del algoritmo de Bisección son:

1. Elige $[a, b]$ donde la función f cambia de signo.
2. Calcula el punto medio de $[a, b]$, esto es, $c = (a + b)/2$.
3. Si $f(a)$ y $f(c)$ tienen signos opuestos, la raíz está en el intervalo $[a, c]$; de lo contrario, está en $[c, b]$.

4. Redefine el intervalo $[a, b]$ como $[a, c]$ si $f(a)$ y $f(c)$ tienen signos opuestos, o como $[c, b]$ si $f(c)$ y $f(b)$ tienen signos opuestos.
5. Repite los pasos 2-4 hasta que $[a, b]$ sea lo suficientemente pequeño o $f(c)$ sea próximo a cero con la precisión deseada.



Interactivo 3.1. Método de la bisección

Observa que se cumple

$$|x_k - \overline{x}| < \frac{1}{2}(b_k - a_k) = \frac{1}{2^2}(b_{k-1} - a_{k-1}) = \ldots$$

$$= \frac{1}{2^{k+1}}(b-a) \underset{k\to\infty}{\to} 0$$

Por lo tanto, este resultado asegura la convergencia del método de la Bisección.

Ejemplo 3.5. Considerando las funciones y los intervalos de la tabla,

$f(x)$	a	b
$cos(x) - x$	0	2
$sen(x)$	-0.5	$\pi/2$
$x^2 - sen(x)$	0.5	2

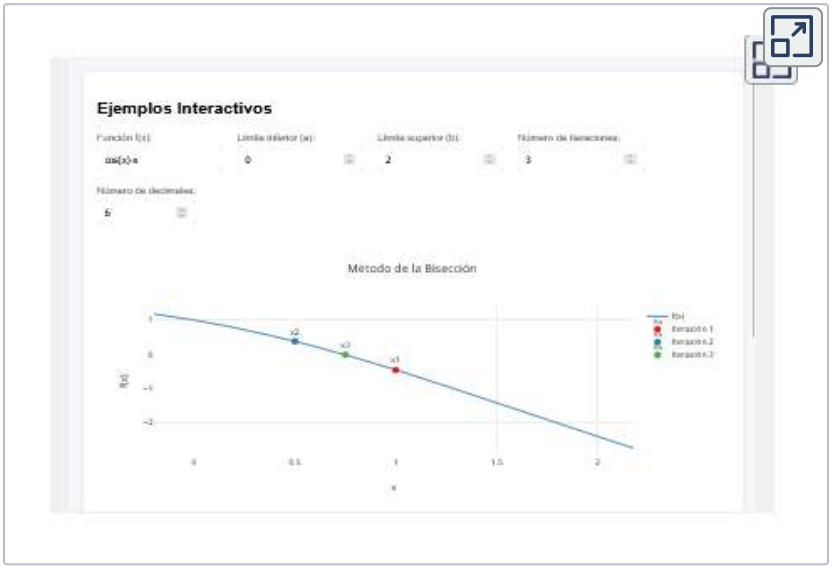
calcula las tres primeras iteraciones del método de la bisección. Comprueba previamente que la función en cada intervalo tiene únicamente un cero.

En primer lugar, se ve que cada función tiene un único cero en el intervalo indicado.

- La función $f(x) = \cos(x) - x$ es continua y cumple además que $f(0) \cdot f(2) < 0$. Esto significa que al menos tiene un cero en $[0, 2]$. Como $f'(x) = -sen(x) - 1 < 0$ la función es estrictamente decreciente en ese intervalo por lo que f tiene un único cero en $[0, 2]$.
- La función $f(x) = sen(x)$ es continua y cumple que $f(-0.5) \cdot f(\pi/2) < 0$. Esto significa que al menos tiene un cero en $[-0.5, \pi/2]$. Como además en ese intervalo $f'(x) = \cos(x) > 0$, la función es estrictamente creciente en $[-0.5, \pi/2]$ por lo que f tiene un único cero en este intervalo.

- La función $f(x) = x^2 - sen(x)$ es continua y cumple además que $f(0.5) \cdot f(2) < 0$. Esto significa que al menos tiene un cero en $[0.5, 2]$. Como $f'(x) = 2x + \cos(x) > 1 + 0 > 0$, la función es estrictamente creciente en ese intervalo y f tiene un único cero en $[0.5, 2]$.

Para calcular los valores de las tres primeras iteraciones del método de la bisección, se puede utilizar la herramienta interactiva siguiente introduciendo la función y los valores de a y b para cada caso. En el interactivo se debe escribir seno de x como sin(x).

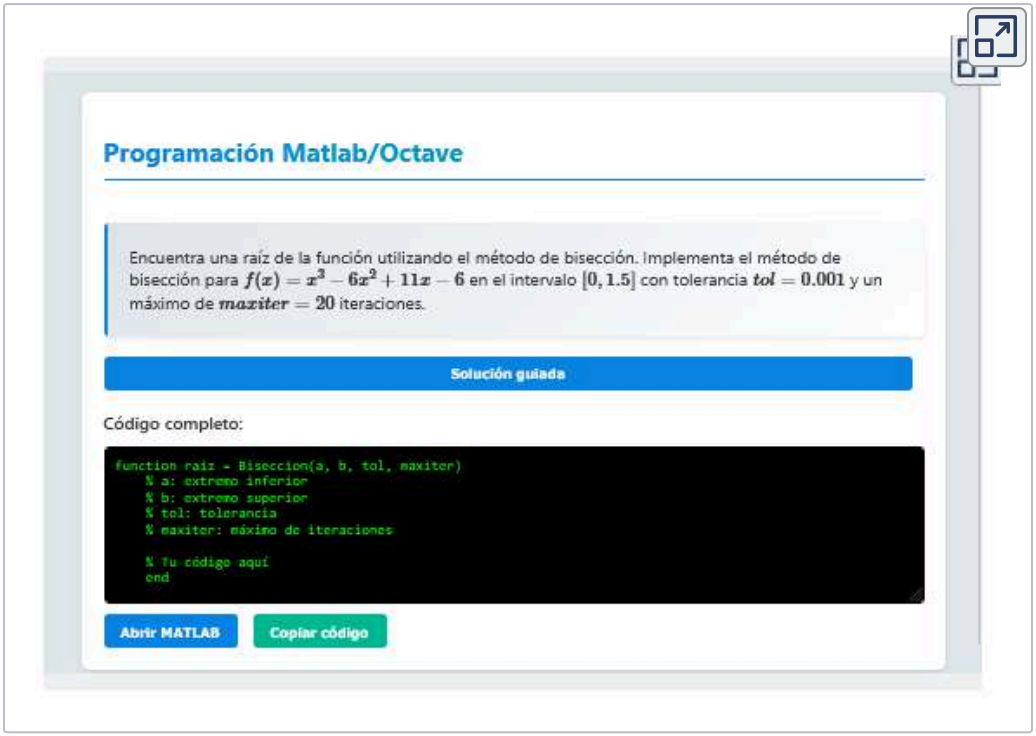


Interactivo 3.2. Método de la bisección

Ejemplo 3.6. Define una función Matlab/Octave `Biseccion(a,b,tol,maxiter)` que tenga como parámetros los extremos del intervalo `a` y `b`, la tolerancia `tol`y el máximo de iteraciones `maxiter` para calcular los ceros de la función $f(x) = x^3 - 6x^2 + 11x - 6$ en el intervalo $[0, 1.5]$ tomando como tolerancia el valor 0.001 y el valor de 30 como máximo de iteraciones. Nota: En este polinomio las raíces se pueden calcular de forma exacta, son 1, 2 y 3 por lo que este ejemplo únicamente tiene interés como ejemplo teórico.

Calcula después el cero que está dentro del intervalo $[1.5, 2.5]$ y el que se encuentra en el intervalo $[2.5, 3.5]$.

La herramienta interactiva siguiente construye el código de la función `Biseccion2(a,b,tol,maxiter)`. Para ello, se debe pulsar en el botón solucion guiada e ir avanzando paso a paso para obtener el código.



Interactivo 3.3. Programación método de la bisección

Definida esta función en un fichero .m, bastaría calcular los ceros en cada intervalo escribiendo en la ventana de comandos:

```
Biseccion(0,1.5,10^-3,30)  %Solución: 1.0000
Biseccion(1.5,2.5,10^-3,30)  %Solución: 2.0000
Biseccion(2.5,3.5,10^-3,30)  %Solución: 3.0000
```

Ejemplo 3.7. Modifica el código del ejemplo anterior para considerar como parámetro también la función cuyo cero se quiere calcular: `Biseccion(fun,a,b,tol,maxiter)`.
Calcula con esta función el cero de $f(x) = x^3 - 6x^2 + 11x - 6$ en el intervalo $[0, 1.5]$, con una tolerancia 10^{-6} y un máximo de 25 iteraciones.

Suponiendo que en el intervalo $[a, b]$ hay cambio de signo de f, la función pedida podría ser la siguiente.

```
function raiz = Biseccion2(f, a, b, tol, maxIter)
    raiz='';
    for i = 1:maxIter
        c = (a + b) / 2;
        fc = f(c); fa = f(a);
        if fa * fc < 0
            b = c;
        else
            a = c;
        end
        if abs(f(c)) < tol
            raiz = c;
            break;
        end
    end
end
```

Para calcular el cero de $f(x) = x^3 - 6x^2 + 11x - 6$ en el intervalo $[0, 1.5]$, con una tolerancia 10^{-6} y un máximo de 25 iteraciones, se deberá escribir el código siguiente.

```
f=@(x) x.^3-6*x.^2+11*x-6;
%Se comprueba que hay cambio de signo
f(0)*f(1.5)
Biseccion2(f,0,1.5,10^-6,25)
%Solución: 1.0000
```

3.5 Método del punto fijo

Dada la ecuación $f(x) = 0$, los pasos a seguir para aplicar el método del punto fijo son:

1. Despejar de la ecuación anterior x, obteniendo $x = g(x)$. El problema de encontar el cero de f se reduce a encontrar el punto fijo de g, esto es, encontrar $\overline{x}$ cumpliendo $\overline{x} = g(\overline{x})$.

2. Partiendo de un valor inicial x_0, se debe iterar de la forma
$$x_{k+1} = g(x_k), \quad k = 0, 1, ...$$
con el objeto de que x_k converja a $\overline{x}$.

Video 3.3. Método del punto fijo

En los resultados teóricos que se enuncian a continuación, se dan condiciones suficientes para que haya un punto fijo de una función g en un intervalo y para que la convergencia sea a un único punto fijo.

Sea g continua en $[a,b]$ que aplica $[a,b]$ en $[a,b]$, entonces g tiene un punto fijo en $[a,b]$.

Sea g continua en $[a,b]$ con g' cumpliendo

$$|g'(x)| \leq L < 1 \quad para\ \ todo\ x \in (a,b)$$

Si x_0 es un punto cualquiera en (a,b) entonces la solución

$$x_{k+1} = g(x_k) \quad k \geq 0$$

converge al único punto fijo en $[a,b]$.

Además, se verifica

$$|x_{k+1} - \overline{x}| \leq \frac{L^k |x_1 - x_o|}{1-L} \quad k = 1, 2, ...$$

Si al utilizar el método del punto fijo x_{k+1} y x_k coinciden dentro de la exactitud especificada ϵ, no significa que $\overline{x} \approx x_{k+1}$ con la misma exactitud. En general, esta afirmación no es correcta. Si $g'(x)$ está próxima a la unidad, entonces la cantidad $|x_k - \overline{x}|$ puede ser grande, aún cuando $|x_{k+1} - x_k|$ sea extremadamente pequeña.

Ejemplo 3.8. Considera la función $g(x) = x - x^2/10$ que tiene como punto fijo el 0. Comprueba que la derivada en las proximidades del 0 es próxima a 1 y analiza los valores que se obtienen con el método del punto fijo.

Efectivamente, la función tiene un punto fijo ya que se cumple $g(0) = 0$.

Además, la derivada de la función es $g'(x) = 1 - x/5$ en el punto 0 vale 1. Se representa en primer lugar la función y la recta $y = x$ y, después, se calculan distintas iteraciones comenzando en el punto 0.5 para analizar cómo es la convergencia.

```
g=@(x) x-x.^2/10; h=@(x) x;
% Gráfica de la función
fplot(g, [-2, 2]); hold on;
fplot(h, [-2, 2], 'r--'); % Línea y = x
xlabel('x');ylabel('g(x)');
title('Método del Punto Fijo para g(x) = x -
x^2/10');
grid on;
legend('g(x)', 'y=x');
```

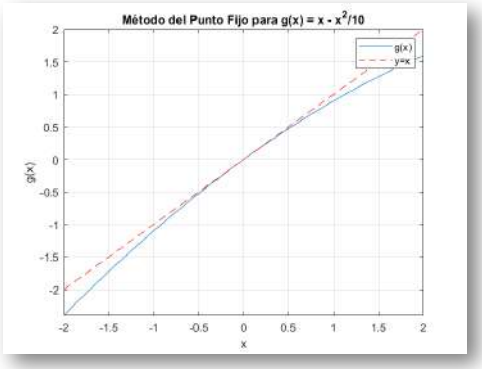
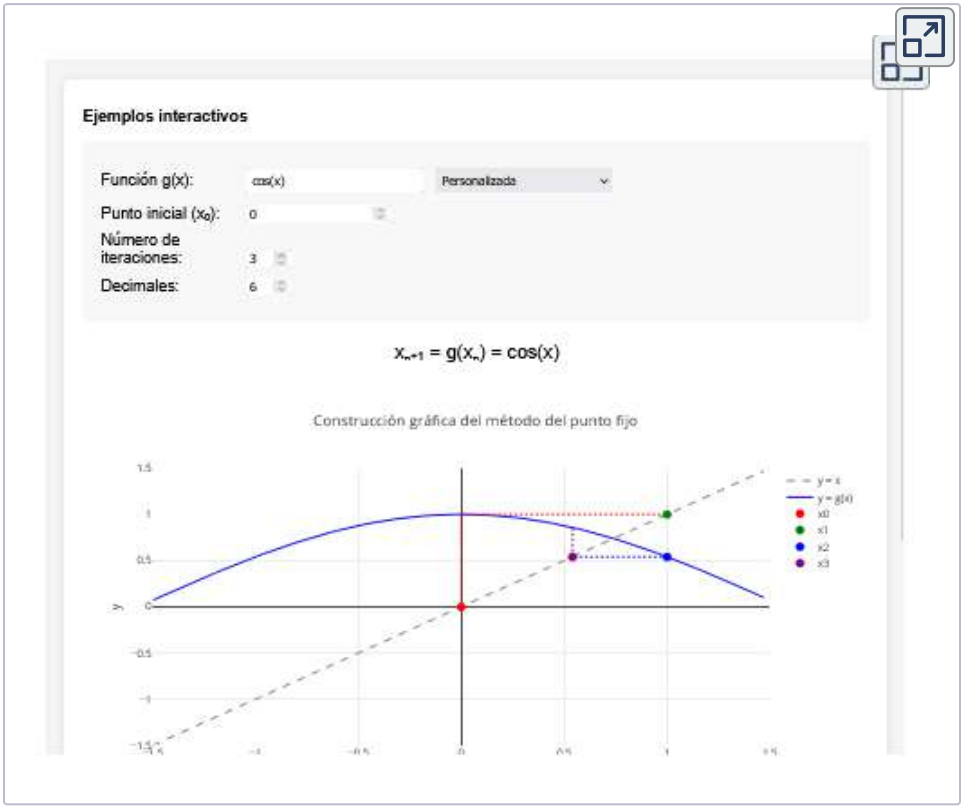


Figura 3.3. Gráfica del ejemplo

Iterando, se ve que la convergencia es muy lenta, después de 100 iteraciones, el valor es 0.0828.

```
x(1)=0.5;maxIter=100;
for k=2:maxIter
    x(k)=g(x(k-1));
end
x(maxIter)
plot(1:maxIter,x,'o');
```

Ejemplo 3.9. Con el siguiente interactivo se pueden obtener las primeras iteraciones del método del punto fijo introduciendo distintas funciones y puntos iniciales. En el ejemplo se muestran las tres primeras iteraciones del método para calcular la raíz de $\cos(x) - x = 0$, tomando $g(x) = cos(x)$. Además, se representa gráficamente cómo obtener el valor de una iteración a partir de la anterior.



Interactivo 3.4. Ejemplos del método del punto fijo

Ejemplo 3.10. Escribe el código Matlab para calcular la raíz de $f(x) = cos(x) - x$ considerando el proceso iterativo siguiente:

$$x_o = 0.5 \ , \quad x_{k+1} = \cos\left(x_k\right) \quad k, 0, 1, 2, ...$$

tomando una tolerancia $\epsilon = 10^{-16}$.

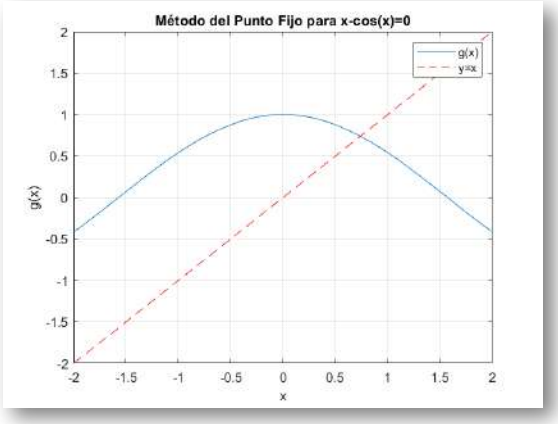


Figura 3.4. Gráficas del ejemplo

El código podría ser el siguiente:

```
g=@(x) cos(x); xk=0.5;
format long
error=1;iter=0;
while error >10^-16
    xk1=g(xk);
    error=abs(xk-xk1);
    xk=xk1;
    iter=iter+1;
end
disp('Iteraciones')
iter
disp('Raiz')
xk
```

Ejecutándolo, se observa que a partir de la iteración 89 se estabiliza la sucesión obteniendo como cero el valor 0.739085133215161.

Ejemplo 3.11. Calcula los ceros de $f(x) = x - e^{-x^2}$ en $[0, 1]$ utilizando el método del punto fijo tomando las funciones $g(x) = e^{-x^2}$ y $h(x) = (-log(x))^{1/2}$. Empieza en el punto 0.5 realizando 100 iteraciones y analiza los resultados que se obtienen con cada función.

Con ayuda de la herramienta vista anteriormente, se pueden obtener las primeras iteraciones en el caso de la función g. Visualmente se puede intuir que la sucesión converge.

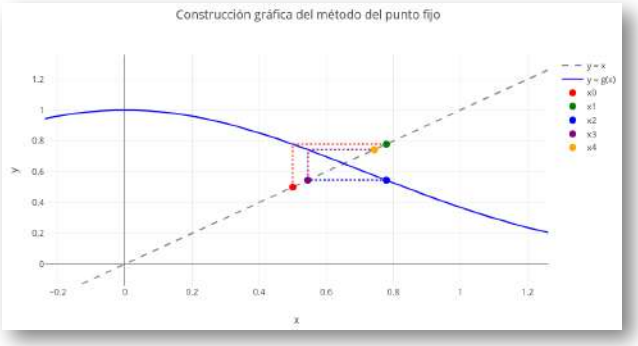


Figura 3.5. Iteraciones del punto fijo para la función e^{-x^2}

En el código siguiente se realizan hasta un máximo de iteraciones indicado en `maxIter` para conseguir la precisión daad.

```
% Se representa la función g y la recta y=x
g=@(x) exp(-x.^2);x=0:0.01:1;
plot(x,g1(x),x,x)
% Número máximo de iteraciones y tolerancia
xk=0.5;tol=10^-10;maxIter=100;format long
for  k=1:maxIter
    xk1=g(xk);
    if abs(xk1-xk)<tol
        xk=xk1; break
    end
    xk=xk1;
end
[xk g(xk) k]
```

Se puede ver que la función cumple la condición suficiente para asegurar la convergencia. Para ello basta calcular el máximo de g' en $[0,1]$ que se alcanza en el punto 0.5, cumpliéndose

$$|g'(x)| = \left|-2x\, e^{-x^2}\right| < 1 \qquad x \in [0,1]$$

En el segundo caso, $h(x) = (-log(x))^{1/2}$, la representación de las primeras iteraciones hace intuir que no habrá convergencia.

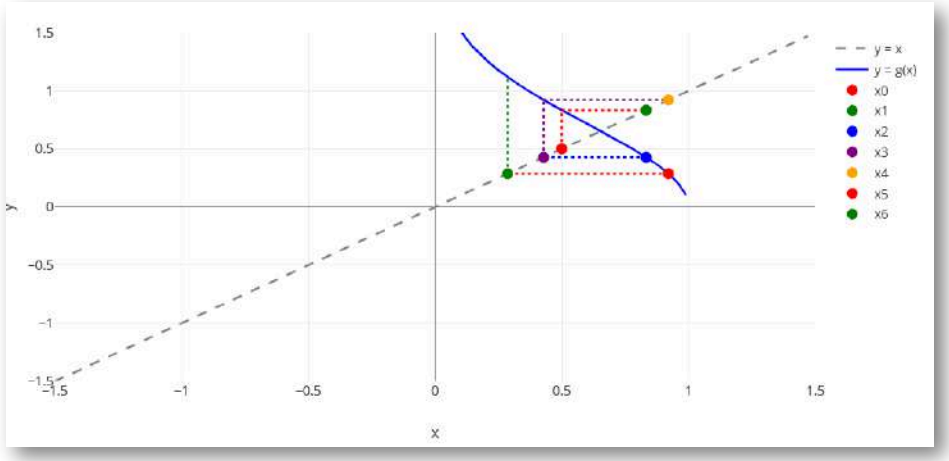


Figura 3.6. Iteraciones del punto fijo para la función $h(x) = (-log(x))^{1/2}$,

Con el siguiente código se pueden obtener las 5 primeras iteraciones. El valor absoluto de la derivada en 0.5 no es menor que 1 por lo que no se tendría garantizada la convergencia comenzando a iterar en 0.5.

```
h=@(x) (-log(x)).^0.5;
%Derivada
h1=@(x) -0.5*((-log(x)).^-0.5)./x;
h1(0.5)  %-1.201122408786450
%Iteraciones
xk=0.5;
for k=1:5
    xk=h(xk)
end
```

En este ejemplo, se ha visto que el método del punto fijo aplicado a $g(x)$ permite obtener el cero de la función f, que es 0.652918640419205, y que con la función h, el método del punto fijo no sería convergente comenzando a iterar desde 0.5.

Ejemplo 3.12. Acota las raíces de $p(x) = x^3 + 3x^2 - 1 = 0$ y encontrar las tres raíces reales del polinomio mediante el método del punto fijo.

Se acotan en primer lugar los ceros de la función y se analiza qué funciones se pueden considerar para aplicar el punto fijo en cada intervalo.

Se tiene que $p'(x) = 3x^2 + 6x = 3x(x+2)$ tiene dos ceros en el punto 0 y en -2. En el punto $(0, -1)$ hay un mínimo y en el punto $(-2, 3)$ un máximo. Luego p tiene tres raíces reales. Aislando estas tres raíces se tiene que están situadas en los intervalos [-3,-2], [-1,0], [0,1].

- En el intervalo [-3,-2], dividiendo por x^2 al ser x distinto de 0, habría que calcular las raíces de $x + 3 - 1/x^2 = 0$. Se puede considerar $g(x) = 1/x^2 - 3$ cumpliéndose en dicho intervalo
$$|g'(x)| = \left|-2x^{-3}\right| \leq \frac{1}{4} < 1$$

- En el intervalo [-1,0], la raíz a calcular será de la ecuación $x^2(x+3) = 1$. Se puede considerar $g(x) = -\sqrt{(x+3)^{-1}}$ que aplica el intervalo $[-1, 0]$ dentro del intervalo $[-1, 0]$. En este caso,
$$|g'(x)| = \left|\frac{1}{2}(x+3)^{-3/2}\right| \leq \frac{1}{2 \cdot 2^{3/2}} < 1$$

- En el intervalo [0,1], la raíz a calcular será de la ecuación $x^2(x+3) = 1$. Se puede considerar $g(x) = \sqrt{(x+3)^{-1}}$ que aplica el intervalo $[0, 1]$ dentro del intervalo $[0, 1]$. En este caso,
$$|g'(x)| = \left|\frac{1}{2}(x+3)^{-3/2}\right| \leq \frac{1}{2 \cdot 3^{3/2}} < 1$$

En la escena interactiva 3.5, se muestra paso a paso cómo generar el código de una función Matlab/Octave: `puntoFijo(g,x0, tol, maxiter)` que tenga como argumentos la función `g`, el punto inicial `x0`, la tolerancia `tol` y el número máximo de iteraciones `maxiter`.

Creada esta función, se calculan los tres ceros considerando una tolerancia por ejemplo de 10^{-6} y 100 iteraciones como máximo, escribiendo el siguiente código en la ventana de comandos.

```
%Raiz en el intervalo [-3,-2]
g=@(x) x.^3-3;tol=10^-6;
puntoFijo(g,-2.5, tol, 100)
%Raíz en el intervalo [-1,0]
g=@(x) -sqrt((x+3)^-1); puntoFijo(g,-1.5, tol, 100)
%Raíz en el intervalo [0,1]
g=@(x) -sqrt((x+3)^-1); puntoFijo(g,0.5, tol, 100)
```

Interactivo 3.5. Programación guiada del método del punto fijo

3.6 Método de Newton

Este método aproxima la solución de la ecuación $f(x) = 0$ mediante una sucesión x_k construida utilizando la expresión iterativa siguiente a partir de un valor inicial x_0:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad k \geq 0$$

La idea de este método se basa en la aproximación lineal que proporciona la recta tangente. Una vez obtenido x_k, se calcula x_{k+1} utilizando la aproximación lineal de $f(x)$ mediante la recta tangente que pasa por el punto $(x_k, f(x_k))$:

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k)$$

Se toma x_{k+1} como la intersección de esta recta con el eje de abscisas, es decir, la solución de

$$f(x_k) + f'(x_k)(x - x_k) = 0$$

Video 3.4. Método de Newton

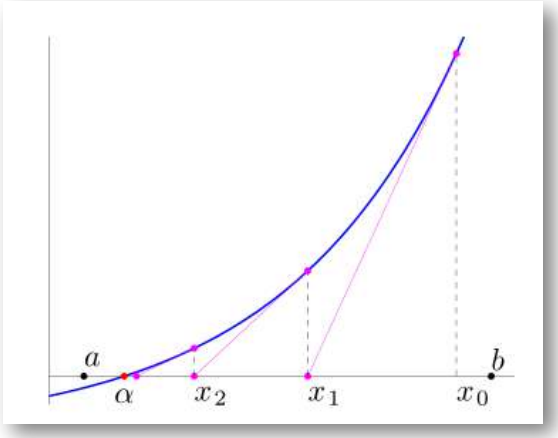


Figura 3.7. Representación del método de Newton

Ejemplo 3.13. Obtener cinco iteraciones del método de Newton para encontrar la solución de la ecuación no lineal

$$x - cos(x) = 0$$

Se considera $f(x) = x - cos(x)$ siendo $f'(x) = 1 + sen(x)$.
Se toma el intervalo $[0, 1]$ como el intervalo $[a, b]$ donde $f(a)f(b) < 0$. Además, en ese intervalo la derivada es estrictamente positiva por lo que la función es creciente y solo hay un cero. Como punto x_0, se considera el punto medio del intervalo $[0, 1]$.

$x_0 = 0.5$

$x_1 = x_0 - \frac{x_0 - cos(x_0)}{1 + sen(x_0)}$ 0.755222417105636

$x_2 = x_1 - \frac{x_1 - cos(x_1)}{1 + sen(x_1)}$ 0.739141666149879

$x_3 = x_2 - \frac{x_2 - cos(x_2)}{1 + sen(x_2)}$ 0.739085133920807

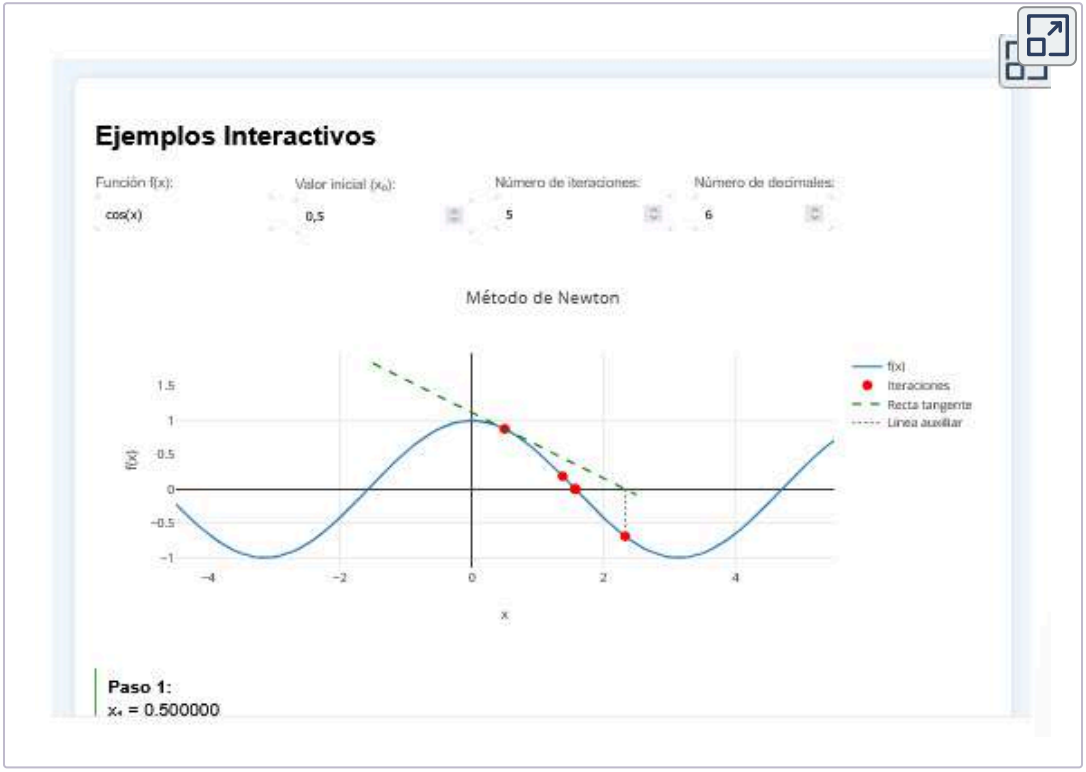
$x_4 = x_3 - \frac{x_3 - cos(x_3)}{1 + sen(x_3)}$ 0.739085133215161

$x_5 = x_4 - \frac{x_4 - cos(x_4)}{1 + sen(x_4)}$ 0.739085133215161

El código Matlab/Octave a utilizar para calcular las cinco iteraciones es el siguiente:

```
x=0.5
for k=1:5
    x=x-(x-cos(x))/(1+sin(x))
end
%Se puede ver que a partir de la cuarta iteración
%el resultado se estabiliza
```

Ejemplo 3.14. Con ayuda de la herramienta siguiente, realiza las primeras iteraciones del método de Newton considerando distintas funciones derivables e introduciendo el punto inicial.



Interactivo 3.6. Ejemplos del método de Newton

3.6.1 Convergencia

El siguiente resultado establece que bajo ciertas condiciones sobre la función f, el método de Newton es convergente localmente con orden de convergencia cuadrático (de orden 2).

Convergencia local. Sea f un función de $\mathbb{R}$ en $\mathbb{R}$ derivable con f' continua. Si que $f(\overline{x}) = 0$ y $f'(\overline{x}) \neq 0$, entonces existe $\varepsilon > 0$ tal que si

$$x_o \in (\overline{x} - \varepsilon, \overline{x} + \varepsilon)$$

la sucesión generada por el método de Newton, $\{x_k\}_{k=0}^{\infty}$ está bien definida (esto es, está en el intervalo anterior)

$$\lim_{k\to\infty} x_k = \overline{x}$$

Convergencia cuadrática. Además, si f es dos veces derivable con f'' continua, existe una constante $C > 0$ tal que:

$$|x_{k+1} - \overline{x}| \leq C|x_k - \overline{x}|^2 \quad \forall k \geq 0$$

Ejemplo 3.15. Calcula las raíces de $e^x + 2^{-x} + 2cos(x) = 6$ en $[0, 1]$.

Se define la función y su derivada y se aplica el proceso iterativo del método de Newton.

```
f=@(x) exp(x)+2.^-x+2*cos(x)-6
df=@(x) exp(x)-log(2)*2.^-x-2*sin(x)
%Se busca un intervalo [a, b] con f(a)f(b) negativo
%Se toma [1,2] y se aplica el Método de Newton
x=1.5
x=x-f(x)/df(x)
%Esta última instrucción se repetiría hasta
%que x estabilice las 16 cifras decimales
```

En la siguiente tabla se muestran las primeras seis iteraciones viendo que se estabiliza a partir de la sexta iteración.

x_0	1.500000000000000		
x_1	1.956489721124210	x_4	1.829383614494166
x_2	1.841533061042061	x_5	1.829383601933849
x_3	1.829506013203651	x_6	1.829383601933849

Se puede observar, que en este caso, se duplica, aproximadamentemente, la cantidad de decimales de cada iteracion (convergencia cuadrática).

Para calcular la diferencia entre dos iteraciones se podría utilizar el siguiente código Matlab/Octave:

```
f=@(x) exp(x)+2^-x+2*cos(x)-6
df=@(x) exp(x)-2^-x*log(2)-2*sin(x)
x=1.5
x1=x-f(x)/df(x),error=abs(x-x1),x=x1;
%Se repetiría esta última línea de código
%para seguir iterando hasta conseguir
%la tolerancia deseada
```

Alternativamente, si se quisiera ir guardando los distintos valores de cada iteración en un vector, el código podría ser el siguiente:

```
f=@(x) exp(x)+2^-x+2*cos(x)-6
df=@(x) exp(x)+2^-x*log(2)-2*sin(x)
k=1; x(k)=1.5
x(k+1)=x(k)-f(x(k))/df(x(k))
error=abs(x(k)-x(k+1))
k=k+1;
%Se repetirían las tres últimas líneas de código
%para realizar una nueva iteración
```

Ejemplo 3.16. Obtén la raíz del polinomio $p(x) = x^3 + 3x^2 + 2$ que se encuentra en el intervalo $[-4, -2]$.

Se puede considerar como punto de inicio el punto medio del intervalo $x_0 = -3$. Dado que se trata de un polinomio, se modifica el código de los ejemplos anteriores para utilizar los comandos `polyval` y `polyder` que, respectivamente, evalúan y derivan un polinomio definido a partir del vector de sus coeficientes.

```
p=[1 3 0 2]
dp=polyder(p) %polyder deriva el polinomio
%Se comprueba que hay cambio de signo en [-4,-2]
polyval(p,-4)*polyval(p,-2) %polyval evalúa un
polinomio
x=-3
%Se aplica el método de Newton
x=x-polyval(p,x)/polyval(dp,x)
%Se repite este último paso para iterar
%sol=-3.195823345445647
```

Repitiendo el proceso con $x_0 = 1$, se puede ver que no converge, las iteraciones oscilan entre valores negativos y positivos. Esto ocurre porque el método de Newton converge localmente.

3.6.2 Test de parada

Para dejar de iterar, se pueden considerar distintos test de parada:

1. Verificar que la diferencia entre dos iteraciones consecutivas es menor que un cierto valor ϵ

$$|x_{k+1} - x_k| < \varepsilon \quad \text{error absoluto}$$

$$|x_{k+1} - x_k| < \varepsilon |x_{k+1}| \quad \text{error relativo}$$

Se pueden combinar ambos errores considerando

$$|x_{k+1} - x_k| < \varepsilon \max\{1, |x_{k+1}|\}$$

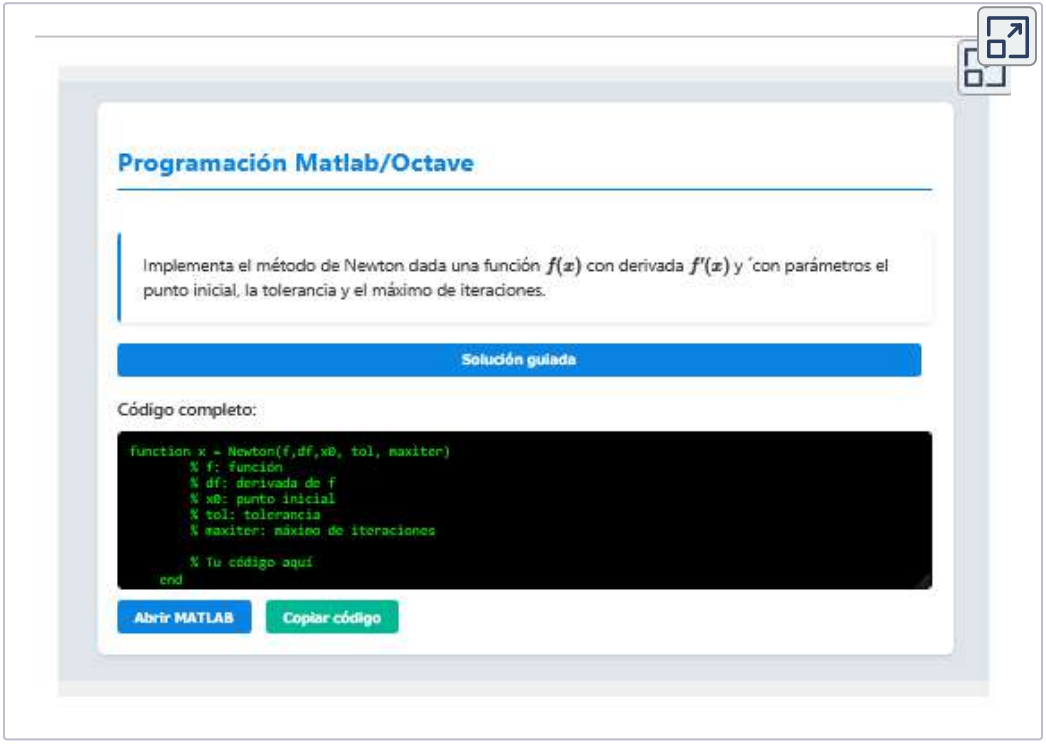
2. Dado que se busca $f(x) = 0$, se puede dejar de iterar cuando $f(x_{k+1})$ sea pequeño, verificando si

$$|f(x_{k+1})| < \epsilon_f$$

Para funciones sencillas, se puede considerar $\varepsilon_f = \varepsilon_M^{3/4}$.

Ejemplo 3.17. Construye una función Matlab/Octave que calcule la raíz de una función aplicando el método de Newton a partir de la función `f`, su derivada `df`, el punto inicial `x0`, el máximo de iteraciones `maxiter` y la tolerancia `tol`.

En el interactivo siguiente se muestra cómo generar paso a paso el código Matlab/Octave de la función `Newton(f,df, x0, tol, maxiter)` que utiliza como método de parada que la diferencia entre dos iteraciones consecutivas sea menor que la tolerancia fijada.



Interactivo 3.7. Programación guiada del método de Newton

3.6.3 Método de Newton combinado con bisección

Para este método, se parte de un intervalo $[a, b]$ en el que $f(a)f(b) < 0$ y se toma como punto inicial $x_0 = \frac{a+b}{2}$.

En cada iteración se busca una nueva aproximación x_{k+1} y se actualizan a y b como se indica a continuación:

- Si $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ está en $[a, b]$ se acepta este valor, en caso contrario se utiliza bisección, es decir, $x_{k+1} = \frac{a_k+b_k}{2}$.
- Se considera $[a_{k+1}, b_{k+1}]$ para la siguiente iteración, como el intervalo $[a_k, x_{k+1}]$ o $[x_{k+1}, b_k]$ según haya cambio de signo de f en los extremos.

Debe tenerse en cuenta que no se aplicará el método de Newton cuando el denominador $f'(x_k)$ sea muy pequeño (overflow), o cuando el cociente $\frac{f(x_k)}{f'(x_k)}$ sea muy grande. Es decir, **se utilizará el método de Newton** cuando

$$\left| \frac{f(x_k)}{f'(x_k)} \right| < \frac{1}{\epsilon_M}$$

Esta última condición se implementa de la forma siguiente y se comprueba antes de aplicar cada paso del método de Newton:

$$|f'(x_k)| > \epsilon_M |f(x_k)|$$

Se muestra a continuación el algoritmo aplicado a un ejemplo.

Ejemplo 3.18. Encuentra la raiz de $e^x + 2^{-x} + 2cos(x) - 6 = 0$ en $[1, 2]$ utilizando el método de Newton combinado con bisección.

Se calcula la función y su derivada y el punto inicial para empezar a iterar con el método de Newton.

```
f=@(x) exp(x)+2.^-x+2*cos(x)-6
df=@(x) exp(x)-log(2)*2.^-x-2*sin(x)
f(1)*f(2)  %Se comprueba cambio de signo
em=eps/2
a=1;b=2;fa=f(a);fb=f(b);x=(a+b)/2;fx=f(x);
[a x b],[fa fx fb]
```

Punto inicial		
a	x	b
1	1.5	2
$f(a)$	$f(x)$	$f(b)$
-1.701113559804675	-1.023283135733256	0.806762425836365

Para la siguiente iteración se debe considerar el nuevo intervalo o bien $[a, x]$, si $f(a) \cdot f(x) < 0$, o bien $[x, b]$ en caso contrario. A partir de los datos de la tabla anterior se toma $[a, b] = [1.5, 2]$ que es donde hay cambio de signo. Antes de obtener la siguiente iteración se comprueba si la derivada toma valores pequeños.

Iteración 1

```
a=x;fa=fx;m=df(x);
if abs(m)>em*abs(fx),x=x-fx/m;[a x b],end
fx=f(x); [fa fx fb]
```

Iteración 1		
a	x	b
1.500000000000000	1.956489721124211	2.000000000000000
$f(a)$	$f(x)$	$f(b)$
-1.023283135733256	0.579701373274924	0.806762425836365

El valor de x obtenido está en $[a, b]$, como además el cambio de signo de f se produce en $[a, x]$, se toma para la siguiente iteración $[a, b] = [a, x]$.

Iteración 2

```
b=x;fb=fx;m=df(x);
if abs(m)>em*abs(fx),x=x-fx/m,end
fx=f(x);[fa fx fb]
```

Iteración 2		
a	x	b
1.500000000000000	1.841533061042061	1.956489721124211
$f(a)$	$f(x)$	$f(b)$
−1.023283135733256	0.050340951614865	0.579701373274924

El valor de x obtenido está en $[a, b]$, como f cambia de signo en $[a, x]$, se toma este intervalo para la siguiente iteración $[a, b] = [a, x]$.

Iteración 3

```
b=x;fb=fx;m=df(x);
if abs(m)>em*abs(fx),x=x-fx/m,end
fx=f(x);[a x b],[fa fx fb]
```

Iteración 3		
a	x	b
1.500000000000000	1.829506013203651	1.841533061042061
$f(a)$	$f(x)$	$f(b)$
−1.023283135733256	0.0005021213225915	0.050340951614865

De nuevo, el valor de x está en $[a, b]$. Como f cambia de signo en $[a, x]$, se toma $[a, b] = [a, x]$ para la siguiente iteración.

Iteración 4

```
b=x;fb=fx;m=df(x);
if abs(m)>em*abs(fx),x=x-fx/m,end
fx=f(x);[a x b],[fa fx fb]
```

Iteración 4		
a	x	b
1.500000000000000	1.829383614494166	1.829506013203651
$f(a)$	$f(x)$	$f(b)$
−1.023283135733256	0.000000051516139	0.000502121322591

De nuevo, el valor de x obtenido está en $[a, b]$. Como el cambio de signo de f es en $[a, x]$ se considera para la siguiente iteración $[a, b] = [a, x]$.

Iteración 5

```
b=x;fb=fx;m=df(x);
if abs(m)>em*abs(fx),x=x-fx/m,end
fx=f(x);[a x b],[fa fx fb]
```

Iteración 5		
a	x	b
1.500000000000000	**1.829383601933849**	1.829383614494166
$f(a)$	$f(x)$	$f(b)$
−1.023283135733256	0.000000000000001	0.0000000515161391

La solución es $x = 1.829383601933849$. En este ejemplo, siempre el valor de x obtenido ha estado en el intervalo por lo que en cada iteración se ha aplicado Newton.

Ejemplo 3.19. Calcula la raiz de $e^x + 2^{-x} + 2cos(x) - 6 = 0$ que se encuentra entre $[-3, -2]$ utilizando el método de Newton combinado con bisección.

Se definen los datos del problema.

```
f=@(x) exp(x)+2^(-x)+2*cos(x)-6
df=@(x) exp(x)-log(2)*2^(-x)-2*sin(x)
f(-3)*f(-2)
a=-3;fa=f(a);b=-2;fb=f(b);x=(a+b)/2;fx=f(x);em=eps/2;
[a x b],[f(a) f(x) f(b)]
```

a	x	b
-3.000000000000000	-2.500000000000000	-2.000000000000000
$f(a)$	$f(x)$	$f(b)$
0.069802075166974	-1.863347982977588	-2.696958389857672

El cambio de signo ocurre en $[a, x]$, por lo que ese será el intervalo en la siguiente iteración. Se modifica el código del ejemplo anterior para comprobar, en cada paso, si el valor obtenido x está en $[a, b]$; en tal caso se acepta, y si no, se aplica el método de bisección.

Iteración 1

```
b=x;fb=fx;dfx=df(x);
if abs(dfx)>em*abs(fx)
  x=x-fx/dfx;
  if a<=x && b>=x
    disp('Se aplica Newton');fx=f(x);
  else
    disp('Se aplica Biseccion');x=(a+b)/2;fx=f(x);
  end
else
  x=(a+b)/2;fx=f(x);
end
[a x b],[fa fx fb]
```

Iteración 1		
a	x	b
-3.000000000000000	-2.750000000000000	-2.500000000000000
$f(a)$	$f(x)$	$f(b)$
0.069802075166974	-1.057505574028503	-1.863347982977588

Repitiendo el código en cada iteración y actualizando previamente el intervalo $[a, b]$ según se produzca cambio de signo de f en los extremos de $[a, x]$ o de $[x, b]$, se obtiene la solución en 6 iteraciones. A continuación, se muestran los resultados de cada paso.

Iteración 2: Bisección		
a	x	b
-3.000000000000000	-2.875000000000000	-2.750000000000000
$f(a)$	$f(x)$	$f(b)$
0.069802075166974	-0.536899807501486	-1.057505574028503

Iteración 3: Newton		
a	x	b
-3.000000000000000	-2.994267548648236	-2.875000000000000
$f(a)$	$f(x)$	$f(b)$
0.069802075166974	0.040014356224199	-0.536899807501486

Iteración 4: Newton		
a	x	b
-2.994267548648236	-2.986542066999646	-2.875000000000000
$f(a)$	$f(x)$	$f(b)$
0.040014356224199	0.000174552940576	-0.536899807501486

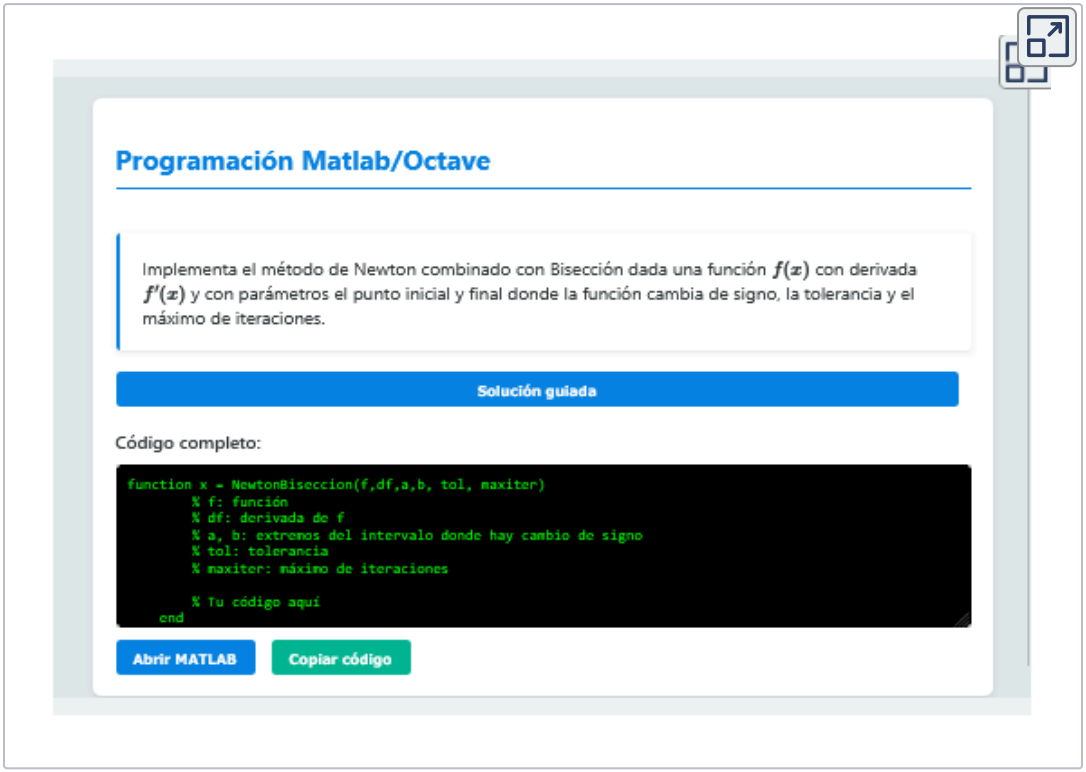
Iteración 5: Newton		
a	x	b
-2.994267548648236	2.986508070038639	-2.875000000000000
$f(a)$	$f(x)$	$f(b)$
0.000174552940576	0.000000003371666	-0.536899807501486

Iteración 6: Newton		
a	x	b
-2.986508070038639	$\mathbf{-2.986508069381928}$	-2.875000000000000
$f(a)$	$f(x)$	$f(b)$
0.000000003371666	0	-0.536899807501486

Después de las 6 iteraciones, el cero de f en el intervalo $[-3,-2]$ es -2.986508069381928

Ejemplo 3.20. Escribe el código Matlab/Octave para programar el método de Newton combinado con bisección de una función `f` derivable con derivada `df`, en un intervalo donde la función cambie de signo, $[a,b]$, con una tolerancia `tol` establecida y un máximo de iteraciones `maxiter` permitidas: `NewtonBisección(f,df, a, b, tol, maxiter)`

Con la herramienta siguiente se muestra cómo generar paso a paso el código Matlab/Octave de la función `NewtonBiseccion(f,df, a, b,tol, maxiter)` que utiliza como método de parada que la diferencia entre dos iteraciones consecutivas sea menor que la tolerancia fijada.



Interactivo 3.8. Programación guiada del método de Newton combinado con bisección

El método de Newton tiene el problema de que podría darse "overflow" si se divide por un número muy pequeño, o dada su convergencia local, podría converger a otra raíz distinta de la buscada o incluso no converger. Además, al requerir la derivada de la función podría ser difícil de obtener por ejemplo en el caso de ciertas funciones como las definidas por integrales o por series infinitas.

En el siguiente apartado se muestra un método que no requiere la derivada de la función cuyo cero se quiere calcular.

3.7 Método de la secante

Este método considera la aproximación que da la secante en intervalos pequeños en lugar de la recta tangente para aproximar la raíz.

El algoritmo parte de dos aproximaciones iniciales x_o y x_1 y en el paso $k+1$, calcula x_{k+1} a partir de x_{k-1} y x_k utilizando la siguiente expresión iterativa

$$x_{k+1} = x_k - \frac{f(x_k)}{m_k} \qquad m_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Esta expresión se justifica considerando la recta secante que pasa por los puntos $(x_{k-1}, f(x_{k-1}))$ y $(x_k, f(x_k))$ y obteniendo el punto de corte de dicha recta con el eje $y = 0$. La ecuación de la recta es

$$y = f(x_k) + m_k(x - x_k) \qquad m_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

y su punto de intersección con $y = 0$ será el valor de x_{k+1}

$$0 = f(x_k) + m_k(x - x_k) \rightarrow \; x_{k+1} = x_k - \frac{f(x_k)}{m_k}$$

Video 3.5. Método de la secante

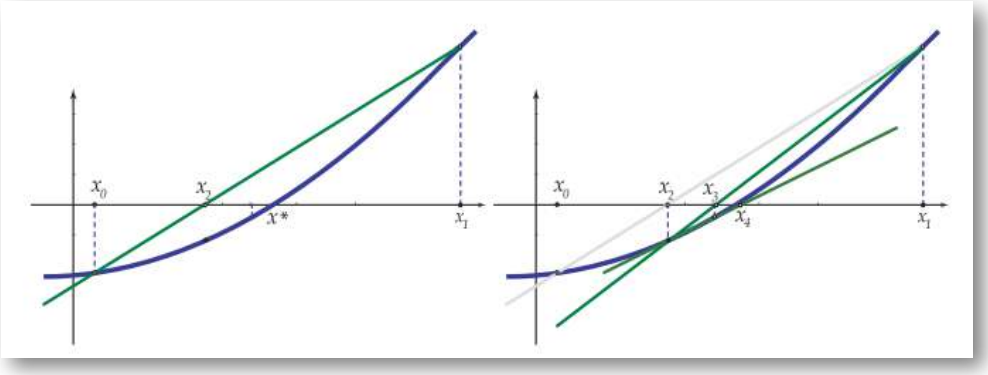
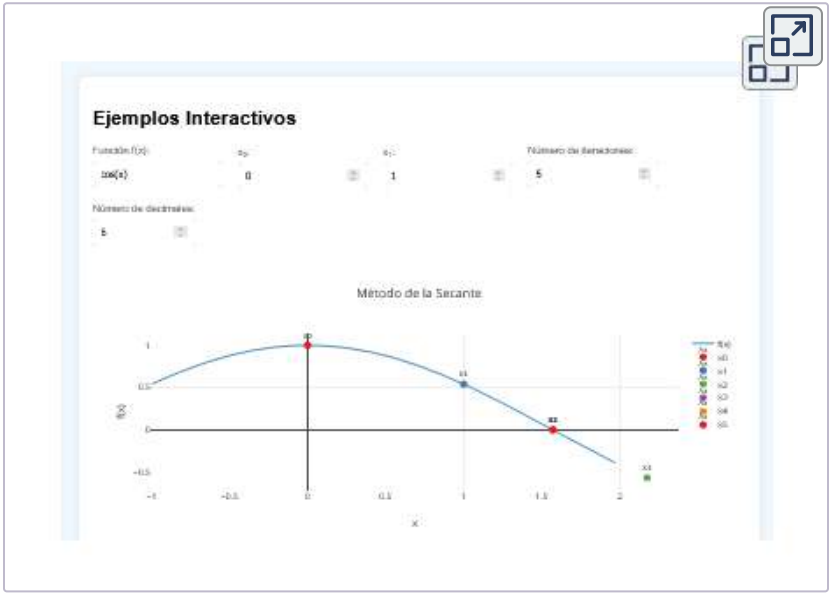


Figura 3.8. Representación del método de la secante

Ejemplo 3.21. Calcula las 5 primeras iteraciones del método de la secante para obtener el cero de la función $f(x) = cos(x)$ considerando como valores iniciales $x_0 = 0$ y $x_1 = 1$.

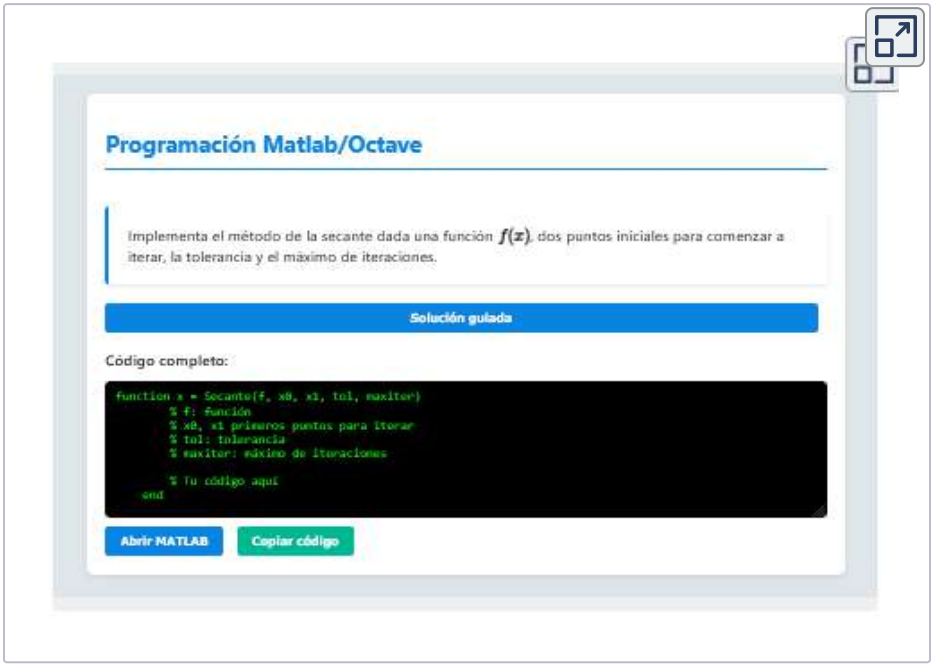
Con la ayuda del siguiente interactivo puede calcular las tres primeras iteraciones del método de la secante comenzando en los puntos 0 y 1.



Interactivo 3.9. Ejemplos del método de la secante

Ejemplo 3.22. Escribe una función Matlab/Octave para aplicar el método de la secante que tenga como argumentos la función `f`, los puntos iniciales `x0` y `x1`, la tolerancia `tol` y el número máximo de iteraciones permitidas `maxiter` .

Pulsando en el botón **solución guiada** del interactivo siguiente, se muestra la programación de la función `Secante(f, x0, x1, tol, maxiter)`.



Interactivo 3.10. Programación guiada del método de la secante

3.7.1 Convergencia

Este método tiene convergencia local si la función f es derivable cerca de la raíz, si la raíz es simple y si las aproximaciones iniciales están suficientemente próximas a ella. Su orden de convergencia es el número áureo, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$, lo que implica una convergencia superlineal, aunque inferior a la del método de Newton. Si las aproximaciones iniciales están demasiado alejadas de la raíz o si esta es múltiple, el método no asegura la convergencia.

3.7.2 Método de la secante combinado con bisección

Al igual que en el caso del método de Newton, resulta conveniente implementar este método combinado con el de bisección, para garantizar la convergencia en situaciones en las que la aproximación secante no sea válida o segura.

En el algoritmo, se parte de un intervalo $[a, b]$ donde $f(a)f(b) < 0$. Se toma $x_0 = a$ y $x_1 = b$.

En cada iteración, se busca una nueva aproximación x_{k+1} y se actualiza a y b como se indica a continuación:

- Si $x_{k+1} = x_k - \frac{f(x_k)}{m_k}$ está en $[a_k, b_k]$ se acepta este valor, en caso contrario se utiliza bisección, es decir, $x_{k+1} = \frac{a_k+b_k}{2}$
- Se modifica el intervalo $[a_{k+1}, b_{k+1}]$ tomando o bien $[a_k, x_{k+1}]$ o bien $[x_{k+1}, b_k]$ según que haya cambio de signo de f en los extremos del intervalo.

Tampoco se aplicará el método de la secante cuando el denominador m_k sea muy pequeño o $|f(x_k)/m_k| > 1/\epsilon_M$. Se verificará por tanto que se cumple la siguiente condición antes de aplicar el método de la secante:

$$|m_k| > \epsilon_M |f(x_k)|$$

Los tests de parada a utilizar son los mismos que en el caso del método de Newton-Bisección.

Ejemplo 3.23. La función $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ está definida para $x \geq 0$. Determina el valor de x tal que $erf(x) = 0.5$ utilizando la función de Matlab erf.

Se considera en primer lugar un intervalo donde la función cambie de signo, por ejemplo $[a, b] = [0, 1]$.

Se define $x_0 = 0$ y $x_1 = 1$ para empezar a iterar en el método de la secante y se calcula el siguiente valor x utilizando el método de la secante comprobando si está en $[0, 1]$.

```
em=eps/2;format long
f=@(x) erf(x)-0.5
f(0)*f(1)
a=0;b=1;fa=f(a);fb=f(b);x0=a;fx0=fa;x1=b;fx1=fb;
%Se calcula siguiente punto por el mét. de la secante
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
%El valor x está en [a,b], no se necesita bisección
[a x b],[f(a) fx f(b)]
```

Iteración 1		
a	x	b
0	0.593330401707401	1.000000000000000
$f(a)$	$f(x)$	$f(b)$
-0.500000000000000	0.098584503929305	0.342700792949715

El cambio de signo se produce en $[a, x]$ por lo que se toma este intervalo como el nuevo $[a, b]$. Se actualiza x_0 y x_1 que serán respectivamente x_1 y x.

```
b=x;x0=x1;x1=x;fx0=f(x0);fx1=f(x1);
% Calculo del siguiente punto
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
[a x b],[f(a) fx f(b)]
% Se comprueba que x está en [a,b]
```

Iteración 2		
a	x	b
0	0.429099981968989	0.593330401707401
$f(a)$	$f(x)$	$f(b)$
-0.500000000000000	-0.043957753989566	0.342700792949715

El punto x está dentro del intervalo $[a, b]$. El cambio de signo se produce en $[x, b]$, se toma este intervalo como el nuevo $[a, b]$. Se actualiza x_0 y x_1 que serán respectivamente x_1 y x para la siguiente iteración.

```
a=x;x0=x1;x1=x;fx0=f(x0);fx1=f(x1);
% Calculo del siguiente punto
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
[a x b],[f(a) fx f(b)]
%Se comprueba que está en [a,b]
```

Iteración 3		
a	x	b
0.429099981968989	0.479746018406641	0.593330401707401
$f(a)$	$f(x)$	$f(b)$
-0.043957753989566	0.002522030582461	0.098584503929305

El punto x está dentro del intervalo $[a, b]$. El cambio de signo se produce ahora en $[a, x]$, se toma este intervalo como el nuevo $[a, b]$. Se actualiza x_0 y x_1 que serán respectivamente x_1 y x para la siguiente iteración.

```
b=x;x0=x1;x1=x;fx0=f(x0);fx1=f(x1);
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
[a x b],[f(a) fx f(b)]
%Se comprueba que está en [a,b]
```

Iteración 4

a	x	b
0.429099981968989	0.476997923639157	0.479746018406641
$f(a)$	$f(x)$	$f(b)$
−0.043957753989566	0.000055407570768	0.002522030582461

El punto x se encuentra dentro del intervalo $[a, b]$. De nuevo, el cambio de signo de f se produce en $[a, x]$. Se toma este intervalo como el nuevo $[a, b]$. Se actualiza x_0 y x_1 que serán respectivamente x_1 y x.

```
b=x;x0=x1;x1=x;fx0=f(x0);fx1=f(x1);
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
[a x b],[f(a) fx f(b)]
%Se comprueba que está en [a,b]
```

Iteración 5

a	x	b
0.429099981968989	0.476936276206905	0.476997923639157
$f(a)$	$f(x)$	$f(b)$
−0.043957753989566	−0.000000074435110	0.000055407570768

El cambio de signo de f se produce en $[x, b]$. Se toma este intervalo como el nuevo $[a, b]$. Se actualiza x_0 y x_1 que serán respectivamente x_1 y x.

```
a=x;x0=x1;x1=x;fx0=f(x0);fx1=f(x1);
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
[a x b],[f(a) fx f(b)]
% Se comprueba que está en [a,b]
```

Iteración 6

a	x	b
0.476936193389100	0.476936276206905	0.476997923639157
$f(a)$	$f(x)$	$f(b)$
$1.0e-04*$ -0.000744351095761$	$1.0e-04*$ 0.000000021886937	$1.0e-04*$ 0.554075707681623

```
b=x;x0=x1;x1=x;fx0=f(x0);fx1=f(x1);
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;fx=f(x);
end
[a x b],[f(a) fx f(b)]
%Se comprueba que está en [a,b]
```

El valor de x está en $[a, b]$. Como el cambio de signo de f se produce en $[a, x]$, este intervalo será el nuevo intervalo $[a, b]$ para la siguiente iteración.

Iteración 7		
a	x	b
0.476936193389100	**0.476936276204470**	0.476936276206905
$f(a)$	$f(x)$	$f(b)$
$1.0e-07*$ -0.744351095760543$	$1.0e-07*0$	$1.0e-07*$ 0.000021886936707

Después de estas iteraciones, se obtiene el punto 0.476936276204470 como aproximación de la raíz.

Ejemplo 3.24. Dada la matriz

$$A(x) = \begin{pmatrix} x & 2 & -3 & 5 & 0 & x^2 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & x & x & 0 & 5 & -2 \\ 1 & -1 & 1 & -1 & 2x & 0 \\ 1 & 0 & 1 & 0 & x^2 & -2 \\ 0 & 0 & 1 & 0 & \cos(x) & -x \end{pmatrix}$$

determinar un valor de x tal que -2 sea un valor propio.

Se recuerda que un valor propio λ de una matriz cuadrada A es un escalar para el que existe un vector no nulo v de forma que $Av = \lambda v$, es decir, debe cumplirse, que existe v de forma que $(A - \lambda I)v = 0$. Esto significa que la ecuación a considerar en este ejemplo es $f(x) = det(A(x) + 2I) = 0$ siendo I la matriz identidad 6x6.

Dada la dificultad de calcular la derivada de $f(x)$, se utilizará el método de la secante.

Se define la función f y se comprueba que en el intervalo $[1, 2]$ hay cambio de signo.

```
%det(A-landa*I) landa=-2 valor propio
%f(x)=det(A(x)+2I)
f=@(x) det([x+2 2 -3 5 0 x^2;1 3 1 1 1 1;
0 x x+2 0 5 -2; 1 -1 1 1 2*x 0;
1 0 1 0 (x^2)+2 -2;0 0 1 0 cos(x) -x+2])
%Se buscan cambios de signo
% f(0)<0
% f(1)<0
% f(2)>0
% Hay otros intervalos donde la función cambia de
% signo pero solo piden calcular un valor de x
```

Se considera el intervalo $[a, b] = [1, 2]$ y los puntos iniciales $x_0 = 1, x_1 = 2$ para empezar a iterar . Se calcula el punto x_2 si la aproximación por el método de la secante se encuentra en el intervalo $[a, b]$, o se utiliza bisección en caso contrario.

```
format long
em=eps/2;
a=1;b=2;x0=a;x1=b;fx0=f(x0);fx1=f(x1);
m=(fx1-fx0)/(x1-x0);
%Se aplica el método de la secante si
%la pendiente no es próxima a cero
if abs(m)>em*abs(fx1)
    x=x1-fx1/m
end
[a x b]
%Se observa que x está dentro del intervalo [a,b]
fx=f(x);[fa fx fb]
```

Iteración 1		
a	x	b
1.000000000000000	1.264336373802500	2.000000000000000
$f(a)$	$f(x)$	$f(b)$
$1.0e-02*$ -0.389735559513635$	$1.0e-02*$ 0.168432357189391	$1.0e-02*$ 1.084656912121114

El cambio de signo de f es en $[a, x]$ por lo que este intervalo será el nuevo intervalo $[a, b]$ para la siguiente iteración. Se actualizan x_0 y x_1 que serán, respectivamente, x_1 y x.

```
b=x;fb=f(b);x0=x1;fx0=fx1;x1=x;fx1=fx;
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m
end
%En este caso x está dentro del intervalo [a,b]
[a x b],fx=f(x);[fa fx fb]
```

Iteración 2		
a	x	b
1.000000000000000	1.184570415928323	1.264336373802500
$f(a)$	$f(x)$	$f(b)$
-38.973555951363515	0.507113188549775	16.843235718939074

Se vuelve a producir un cambio de signo en $[a, x]$ por lo que ejecutando el mismo código anterior se obtienen los siguientes valores.

Iteración 3		
a	x	b
1.000000000000000	1.182094285551603	1.184570415928323
$f(a)$	$f(x)$	$f(b)$
-38.973555951363515	-0.008881043100172	0.507113188549775

En este caso hay cambio de signo en $[x, b]$ por lo que el nuevo intervalo $[a, b]$ es $[x, b]$. Se actualizan x_0 y x_1 que serán, respectivamente, x_1 y x.

```
a=x;fa=f(a);x0=x1;fx0=fx1;x1=x;fx1=fx;
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m
    [a x b]
end
%En este caso x está dentro del intervalo [a,b]
[a x b],fx=f(x);[fa fx fb]
```

Iteración 4		
a	x	b
1.182094285551603	1.182136903509806	1.184570415928323
$f(a)$	$f(x)$	$f(b)$
-0.008881043100172	0.000004066033751	0.507113188549775

En este caso hay cambio de signo en $[a, x]$ por lo que el nuevo intervalo $[a, b] = [a, x]$. Se actualizan x_0 y x_1 que serán, respectivamente, x_1 y x.

```
b=x;fb=f(b);x1=x;fx1=fx;
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m
end
%En este caso x está dentro del intervalo [a,b]
[a x b],fx=f(x);[fa fx fb]
```

Iteración 5		
a	x	b
1.182094285551603	1.182136884006832	1.182136903509806
$f(a)$	$f(x)$	$f(b)$
−0.008881043100172	0.000000000032507	0.000004066033751

Como vuelve a haber cambio de signo en el intervalo $[a, x]$ se ejecuta el mismo código anterior.

Iteración 6		
a	x	b
1.182094285551603	1.182136884006832	1.182136903509806
$f(a)$	$f(x)$	$f(b)$
−0.008881043100172	0.000000000000003	0.000000000032507

Nuevamente vuelve a haber cambio de signo en $[a, x]$ por lo que se ejecuta el último código otra vez.

Iteración 7		
a	x	b
1.182136884006676	**1.182136884006676**	1.182136884006832
$f(a)$	$f(x)$	$f(b)$
$1.0e-10*$ 0.000025575661434	$1.0e-10*$ 0.000025575661434	$1.0e-10*$ 0.325066656820725

En cada iteración el valor de x calculado por el método de la secante ha cumplido que está en el intervalo $[a, b]$ por lo que no se ha aplicado bisección en ninguna iteración.

Ejemplo 3.25. Aplicando el método de la secante bisección calcula el cero de la función $f(x) = e^x + 2^{-x} + 2cos(x) - 5$ en el intervalo $[-5, 0]$ con una tolerancia 10^{-7}.

Se comienza viendo si hay cambio de signo de f en $[-5, 0]$. Como $f(-5)f(0) < 0$, se considera $[a, b] = [-5, 0]$, y se toma $x_0 = -5$, $x_1 = 0$ los puntos iniciales para aplicar el método de la secante.

```
$Se establecen los datos del problema
f=@(x) exp(x)+2.^-x+2*cos(x)-5;
em=eps/2;
format long
a=-5;b=0;fa=f(a);fb=f(b);
x0=a;x1=b;fx0=fa;fx1=fb;
```

Se calcula el siguiente término de la sucesión, $x = x_2$, aplicando secante si el valor obtenido se encuentra en $[a, b]$ o, en caso contrario, utilizando bisección.

```
%Código cálculo siguiente término
m=(fx1-fx0)/(x1-x0);
if abs(m)>em*abs(fx1)
    x=x1-fx1/m;
    if a<=x & b>=x
        disp('Se aplica secante')
    else
        disp('Se aplica biseccion')
        x=(a+b)/2;
    end
    fx=f(x);
    [a x b],[f(a) fx f(b)]
else
    disp('División por cero')
end
```

Iteración 1		
a	x	b
-5.000000000000000	-0.174983869789607	0
$f(a)$	$f(x)$	$f(b)$
27.574062317925538	-1.062118961781644	-1.000000000000000

Como hay cambio de signo en $[a, x]$ este será el nuevo intervalo $[a, b]$. Se modifica el intervalo y se cambia el valor de x_0 por x_1 y x_1 por x para la próxima iteración.

```
b=x;
x0=x1;fx0=fx1;
x1=x;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 2. Se aplica bisección		
a	x	b
-5.000000000000000	-2.587491934894803	-0.174983869789607
$f(a)$	$f(x)$	$f(b)$
27.574062317925538	-0.615010653972961	-1.062118961781644

De nuevo el cambio de signo de f es en $[a, x]$, por lo que este será el nuevo intervalo $[a, b]$. Se modifica el intervalo, se cambia el valor de x_0 por x_1 y x_1 por x y se ejecuta después el código para calcular el siguiente término.

```
b=x;x0=x1;fx0=fx1;x1=x;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 3. Se aplica bisección		
a	x	b
-5.000000000000000	-3.793745967447402	-2.587491934894803
$f(a)$	$f(x)$	$f(b)$
27.574062317925538	7.301512320864664	-0.615010653972961

Ahora el cambio de signo de f es en $[x, b]$, se cambia el intervalo $[a, b]$ y los valores de x_0 y x_1.

```
a=x;x0=x1;x1=x;fx0=fx1;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 4. Se aplica secante		
a	x	b
-3.793745967447402	-2.681202151333973	-2.587491934894803
$f(a)$	$f(x)$	$f(b)$
7.301512320864664	-0.309376060449428	-0.615010653972961

El intervalo donde cambia de signo f es $[a, x]$, se cambia el intervalo $[a, b]$ y se actualiza x_0 y x_1 para la próxima iteración.

```
b=x;x0=x1;x1=x;fx0=fx1;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 5. Se aplica secante		
a	x	b
-3.793745967447402	-2.726426099664577	-2.681202151333973
$f(a)$	$f(x)$	$f(b)$
7.301512320864664	-0.146504190440662	-0.309376060449428

De nuevo el intervalo donde cambia de signo f es $[a, x]$, se cambia el intervalo $[a, b]$ y se actualiza x_0 y x_1 para la próxima iteración.

```
b=x;x0=x1;x1=x;fx0=fx1;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 6. Se aplica secante		
a	x	b
-3.793745967447402	-2.767105303128926	-2.726426099664577
$f(a)$	$f(x)$	$f(b)$
7.301512320864664	0.008859811280272	-0.146504190440662

El cambio de signo f es en $[x, b]$, se cambia el intervalo $[a, b]$ y se actualiza x_0 y x_1 para la próxima iteración.

```
a=x;x0=x1;x1=x;fx0=fx1;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 7. Se aplica secante		
a	x	b
-2.767105303128926	-2.764785524662011	-2.726426099664577
$f(a)$	$f(x)$	$f(b)$
0.008859811280272	-0.000229260174994	-0.146504190440662

El cambio de signo de f es $[a, x]$, se cambia el intervalo $[a, b]$ y se actualiza x_0 y x_1 para la próxima iteración.

```
b=x;x0=x1;x1=x;fx0=fx1;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 8. Se aplica secante		
a	x	b
-2.767105303128926	-2.764844038099813	-2.764785524662011
$f(a)$	$f(x)$	$f(b)$
0.008859811280272	-0.000000343381642	-0.000229260174994

De nuevo, el cambio de signo de f es en $[a, x]$, se cambia el intervalo $[a, b]$ y se actualiza x_0 y x_1 para la próxima iteración.

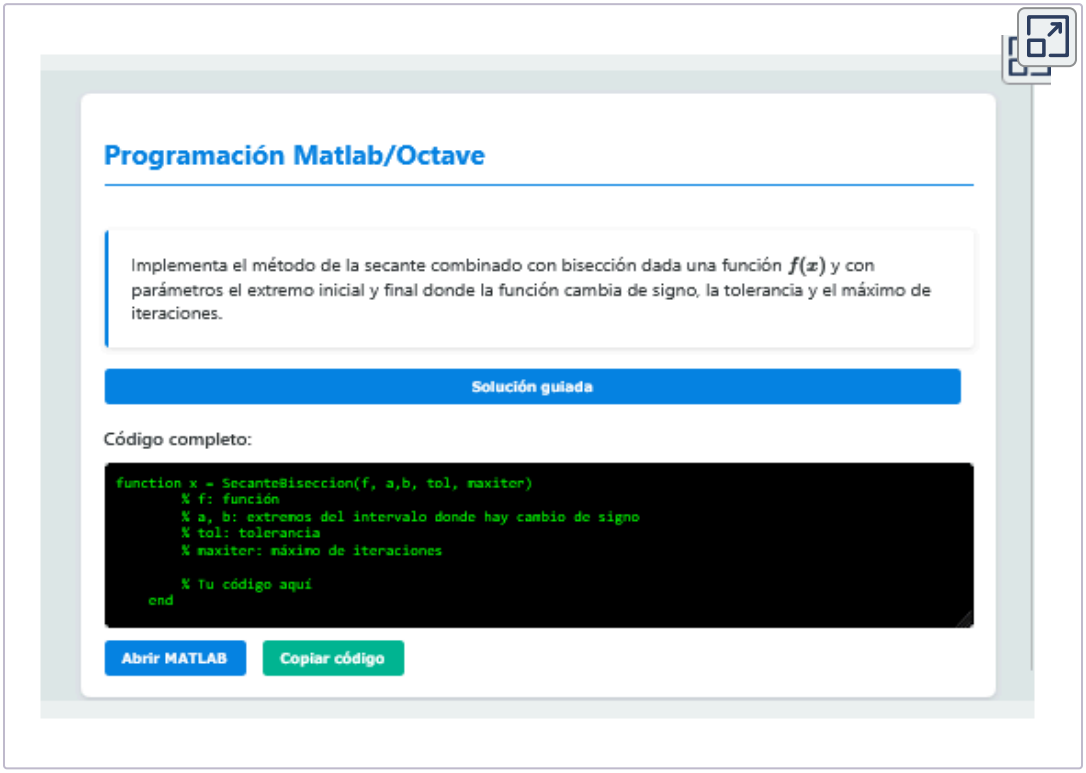
```
b=x;x0=x1;x1=x;fx0=fx1;fx1=fx;
%Ejecutar después el código para calcular
%el siguiente término
```

Iteración 9. Se aplica secante		
a	x	b
-2.767105303128926	$\mathbf{-2.764844125871619}$	-2.764844038099813
$f(a)$	$f(x)$	$f(b)$
0.008859811280272	0.000000000013341	-0.000000343381642

Después de las iteraciones realizadas, la raiz de la función f en $[-5, 0]$ con la tolerancia indicada es -2.764844125871619.

Ejemplo 3.26. Escribe el código Matlab/Octave para programar el método de la secante combinado con bisección a partir de la función `f`, dos puntos imiciales donde la función cambie de signo `a` y `b`, la tolerancia `tol` y `maxiter` el máximo de iteraciones permitidas: `SecanteBiseccion(f, a, b, tol, maxiter)`

Con la herramienta siguiente se muestra cómo generar paso a paso el código Matlab/Octave de la función `SecanteBiseccion(f, a, b, tol, maxiter)` que utiliza como método de parada que la diferencia entre dos iteraciones consecutivas sea menor que la tolerancia fijada.



Interactivo 3.11. Programación guiada del método de la secante combinado con bisección

3.8 Raíces de un polinomio

En este apartado, se analizará el problema de encontrar todas las raíces (reales y complejas) de un polinomio, es decir, se presentará un método para calcular todas las soluciones de

$$a_o + a_1 x + a_2 x^2 + ... + a_n x^n = 0$$

Se recuerdan los siguientes resultados que aplican a los polinomios:

- **Teorema Fundamental del Álgebra**: Un polinomio de grado n tiene exactamente n raíces en el conjunto de los números complejos, donde cada una de ellas debe contarse con su multiplicidad.
- Si un polinomio tiene coeficientes reales, siempre que tenga una raiz compleja no real, también tendrá como raíz su conjugada.
- Si $p(x) = a_o + a_1 x + a_2 x^2 + ... + a_n x^n$, todas las raíces satisfacen

$$\frac{|a_0|}{b + |a_0|} \leq \overline{x} \leq 1 + \frac{c}{|a_n|}$$

siendo

$$b = \max\{|a_1|,\ |a_2|, ..., |a_n|\}$$

$$c = \max\{|a_0|,\ |a_1|, ..., |a_{n-1}|\}$$

Aunque inicialmente se ha descrito el método de Newton para calcular las raíces de una ecuación $f(x) = 0$ siendo f una función real de variable real, el método también es válido para encontrar raíces de funciones complejas. Además, como la derivada de un polinomio es fácilmente calculable, el método de Newton resulta adecuado para calcular las raíces de un polinomio.

Con el objetivo de calcular todas las raíces de un polinomio, surge el problema de cómo encontrar nuevas raíces una vez que se ha obtenido una mediante el método de Newton, evitando que este vuelva a converger a la ya conocida.

Una primera idea sería elegir otro punto de inicio, sin embargo, esto no garantizaría que el método converja necesarimente a otra raíz.

Otra posibilidad, podría ser considerar el polinomio resultado de dividir $p(x)$ entre $x - \alpha$ siendo α la raíz calculada previamente y aplicar el método de Newton a este nuevo polinomio. Este método, conocido como deflacción, presenta también problemas:

1. Por un lado, la operación de división conlleva errores de redondeo.
2. Por otro, el método de Newton devuelve α, una aproximación de la raíz exacta, $\alpha*$, por lo que el cociente $\frac{p(x)}{x-\alpha}$ no coincide con $\frac{p(x)}{x-\alpha*}$ y, en consecuencia, no tienen por qué tener las mismas raíces. Por ejemplo, $p(x) = x^2 + 1.99x + 1.01$ y $q(x) = x^2 + 2x + 1$ son polinomios con coeficientes muy próximos y sus raíces no están próximas.

El método que se utiliza para encontrar una nueva raíz de un polinomio, cuando se han calculado ya otras previamente, es el Método de Maehly.

El método de Maehly consiste en, una vez calculadas algunas raíces del polinomio $p(x)$,

$$\{\alpha_1, \alpha_2, ..., \alpha_m\}$$

aplicar el método de Newton a la función f_m sin realizar la división de polinomios

$$f_m(x) = \frac{p(x)}{(x-\alpha_1)(x-\alpha_2)...(x-\alpha_m)}$$

Derivando f_m y operando, se puede llegar a la siguiente expresión que es la que se considera directamente:

$$\frac{f_m(x)}{f'_m(x)} = \frac{p(x)}{p'(x) - p(x)\sum_{j=1}^{m}\frac{1}{x-\alpha_j}}$$

Por lo tanto, el método iterativo de Maehly consiste en aplicar el método de Newton aplicado a la función f_m de la siguiente manera

$$x_o \in \mathbb{C}$$

$$x_{k+1} = x_k - \frac{p(x_k)}{p'(x_k) - p(x_k)\sum_{j=1}^{m}\frac{1}{x_k-\alpha_j}} \qquad k \geq 0$$

En los ejercicios prácticos se usará un programa del profesor Eduardo Casas (ficheros Matlab) para realizar una iteraciòn del método de Maehly. Se puede descargar el fichero Matlab itraiz.m.

Para aplicar este método, se debe tener en cuenta lo siguiente:

1. A la hora de calcular la primera raiz, el algoritmo se reduce a aplicar el método de Newton al polinomio $p(x)$:

$$x_o \in \mathbb{C}$$

$$x_{k+1} = x_k - \frac{p(x_k)}{p'(x_k)} \qquad k \geq 0$$

2. Si los coeficientes del polinomio son todos reales y x_0 es real, la sucesion x_k será de números reales y no podrá converger a una raíz compleja. Por lo tanto, deberá tomarse x_0 complejo si se quiere encontrar una raíz compleja.

3. Como se ha visto anteriormente, el método de Newton converge cuadráticamente hacia una raíz α si $p'(\alpha) \neq 0$. Si se observa que los valores de x_k convergen, pero lentamente, se puede deber a que se esté aproximando a una raíz múltiple o que haya raíces muy próximas. En este caso se aplica el método de Newton a la derivada del polinomio comenzando en el último punto x_k calculado en las iteraciones sobre p.

   Si la convergencia siguiera siendo lenta, entonces se aplicaría el método de Newton a la segunda derivada para ver si la raíz es doble, o a derivadas sucesivas en el caso en el que la multiplicidad sea mayor.

   Si se calcula una raíz α de $p'(x)$ (o de derivadas sucesivas de p) y se observa que

$$|p(\alpha)| \gg |p'(\alpha)|$$

   entonces se debe concluir que α no es raíz de p, lo que ocurre es que p posee dos o más raíces muy próximas. En el caso de que haya raíces próximas entonces la convergencia será lenta y, en general, no se podrá obtener una precisión alta en la solución.

4. Si el polinomio tiene coeficientes reales y $\alpha = a + bi$ es una raiz compleja entonces su conjugada $\overline{\alpha} = a - bi$ también es raíz del polinomio. Además, si α tuviera multiplicidad m, entonces $\overline{\alpha}$ tendría la misma multiplicidad.

En los siguientes ejemplos se verá cómo aplicar el método de Maehly para encontrar todas las raíces de un polinomio utilizando el fichero itraiz.m que realiza una iteración de este método. Los parámetros de los que depende son los coeficientes del polinomio cuya raíz se quiere calcular, los coeficientes del polinomio derivada, el vector con las raíces ya calculadas y el punto a partir del cual se va a iterar.

Ejemplo 3.27. Calcula todas las raices del polinomio $p(x) = 8x^7 + 26x^6 + 54x^5 + 27x^4 + 128x^3 - 576x^2 + 864x - 432 = 0$.

```
%Se define el polinomio a partir de sus coeficientes
p=[8 26 54 27 -128 -576 -864 -432]
%Se calcula su derivada
dp=polyder(p)
%Se inicializa el vector donde se almacenarán
%las raíces cuando se vayan obteniendo
v=[],x=i;
format long
x=itraiz(p,dp,v,x)
%Se hacen iteraciones hasta encontrar la primera raíz
%aplicando x=itraiz (p,dp,v,x) hasta que x se
%estabilice
%Se comprueba con: polyval(p,x)
```

Tras iterar se ha obtenido: `x= -1.056847723228969 + 0.262522264027478i`

```
%Como la raiz obtenida x es compleja se incluye
%en v tanto x como su conjugada
v=[x;conj(x)]
%Se aplica el método con el vector v modificado
x=i;x=itraiz (p,dp,v,x)
%Se itera hasta que x se estabilice
%aplicando x=itraiz (p,dp,v,x)
```

Tras iterar se obtiene:
`x= -1.540661467764130 + 1.474052474547216i`

```
%Raiz compleja, se incluye en v tanto x como su
conjugada
v=[v;x;conj(x)]
x=i;x=itraiz(p,dp,v,x)
%Se itera hasta que x se estabilice
%aplicando x=itraiz (p,dp,v,x)
```

Tras iterar se obtiene:

```
x= -0.059269229873907 + 2.202319797999819i
```

```
%La raíz es compleja, se incluye en v
%tanto x como su conjugada
v=[v;x;conj(x)]
%Se aplica el método para encontrar una nueva raíz
x=i;x=itraiz(p,dp,v,x)
%Se itera hasta encontrar la raiz
%Se comprueba que es raíz del polinomio
```

Tras iterar se obtiene la raíz real:

```
x= 2.063556841734012
```

```
%La raíz es real, se incluye en v
v=[v;x]
```

Se termina de iterar al haber encontrado las siete raíces del polinomio:

```
v =
-1.056847723228969 + 0.262522264027478i
-1.056847723228969 - 0.262522264027478i
-1.540661467764130 + 1.474052474547216i
-1.540661467764130 - 1.474052474547216i
-0.059269229873907 + 2.202319797999819i
-0.059269229873907 - 2.202319797999819i
2.063556841734012 + 0.000000000000000i
```

Ejemplo 3.28. Calcula todas las raíces del polinomio $p\left(x\right)=25x^{8}+85x^{7}+181x^{6}+238x^{5}+226x^{4}+142x^{3}+61x^{2}+13x+1$.

En este caso, se verá que el polinomio tiene raíces múltiples, se muestran todas las iteraciones.

```
p=[25 85 181 238 226 142 61 13 1]
```

```
p = 1x9
    25    85   181   238   226   142    61    13     1
```

```
dp=polyder(p) %para la derivada
```

```
dp = 1x8
         200         595        1086        1190         904
```

```
v=[] %para guardar las raíces
```

```
v =

     []
```

```
x=i %numero complejo
```

```
x =
  0.000000000000000 + 1.000000000000000i
```

```
format long
x=itraiz (p,dp,v,x)
```

```
x =
 -0.098224852071006 + 0.924260355029586i
```

```
x=itraiz (p,dp,v,x)
```

```
x =
 -0.190709670418108 + 0.868192402497058i
```

```
x=itraiz (p,dp,v,x)
```

```
x =
```

Ejemplo 3.29. Calcula todas las raíces del polinomio $p(x) = 3x^7 + 5x^6 + 21x^5 + 35x^4 + 48x^3 + 80x^2 + 36x + 60$.

```
p=[3 5 21 35 48 80 36 60]
```

```
p = 1×8
     3     5    21    35    48    80    36    60
```

```
dp=polyder(p) %para la derivada
```

```
dp = 1×7
    21    30   105   140   144   160    36
```

```
v=[] %para guardar las raíces
```

```
v =

     []
```

```
x=i %numero complejo
```

```
x =
  0.000000000000000 + 1.000000000000000i
```

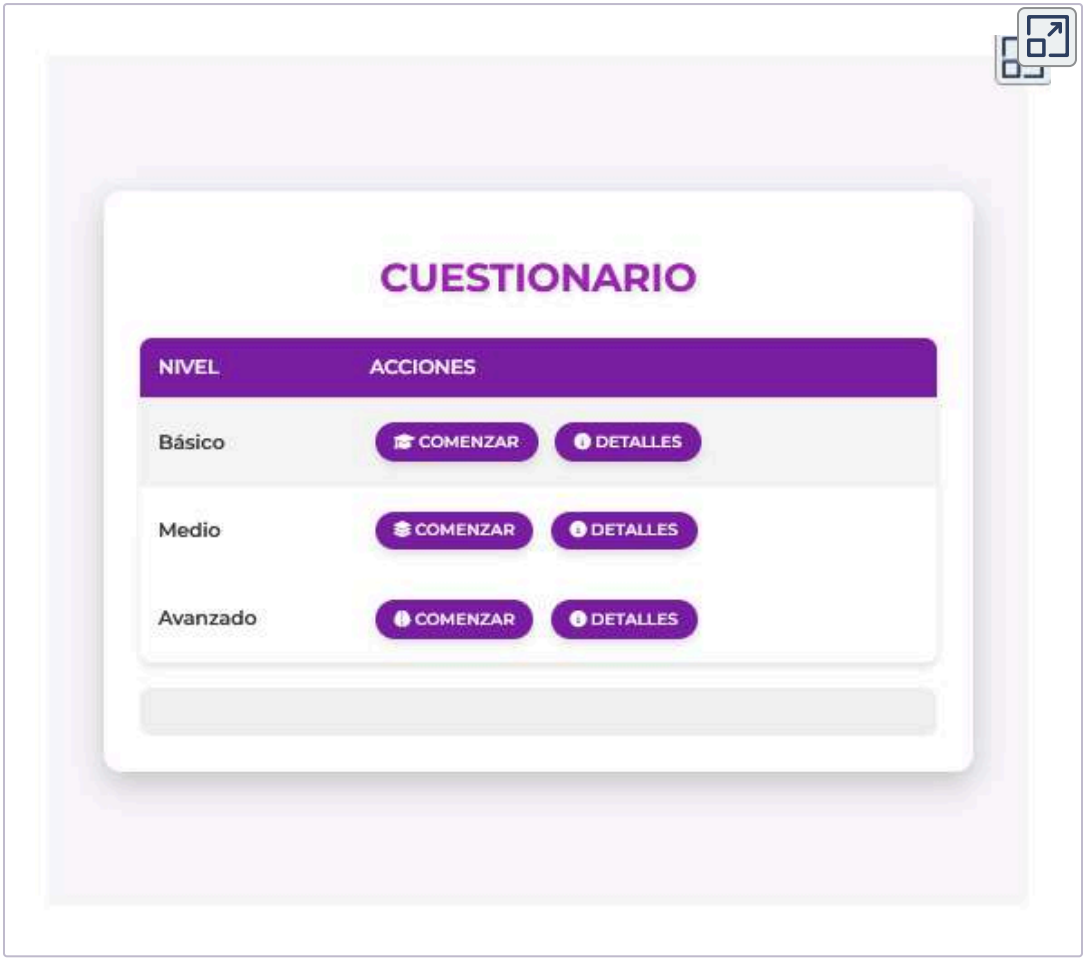
```
format long
x=itraiz (p,dp,v,x)
```

```
x =
 -0.019505851755527 + 1.209362808842653i
```

```
x=itraiz (p,dp,v,x)
```

```
v =
```

3.9 Autoevaluación



Interactivo 3.12. Actividad de autoevaluación

3.10 Ejercicios

1 Encuentra una raíz de $x = \frac{1}{2}sen(x) + \frac{1}{4}$ en el intervalo $[0, \pi/2]$ utilizando el método del punto fijo con un error absoluto de 10^{-16}. Determina previamente que la raíz es única en dicho intervalo.

Solución

2 Resuelve $f(x) = x^2 - e^{-x} = 0$, en el intervalo $[0, 1]$ usando el método del punto fijo considerando el problema $x = g(x)$ con $g(x)$ las funciones siguientes:

- $g(x) = \sqrt{(e^{-x})}$
- $g(x) = -\log(x^2)$

Empezando la iteración con $x_0 = 0.5$, realiza 100 iteraciones y analiza los resultados. Dibuja en el mismo gráfico la función f y las dos elecciones de g con la recta $y = x$ junto con la localización de la raíz.

Solución

3 Encuentra una raíz de $x^3 - x - 2 = 0$ en el intervalo $[1, 2]$ utilizando el método de bisección y el método del punto fijo. Compara ambos métodos.

Solución

4 Implementa el método de la bisección para resolver la ecuación no lineal $f(x) = \frac{1}{\pi}\int_0^x e^{t^2}dt - \frac{3}{4} = 0$ con un error absoluto $\epsilon_a = 10^{-10}$
Nota: Utiliza el comando `integral` para definir la función f.

Solución

5 En la mecánica celeste, la ecuación de Kepler para la anomalía excéntrica E de una órbita elíptica es:

$$E - e\,\sin(E) \;-\; M \;=\; 0,$$

donde el parámetro **excentricidad** e y la **anomalía media** M se conocen. Se toma $e = 0.8$ y $M = 1.0$ (en radianes). Aplica el método de Newton para aproximar la solución E, partiendo de la aproximación inicial $E_0 = M$. Explica el comportamiento de las iteraciones y determina E con una tolerancia de 10^{-10}.

Solución

6 Un puente colgante sigue la ecuación de la catenaria:

$$y(x) = a\cosh\!\left(\tfrac{x}{a}\right) + C.$$

El punto más bajo del cable está en $x = 0$, y se quiere que ese punto esté a 20 m sobre el terreno (es decir, $y(0) = 20$). Las torres, situadas a $x = \pm 100$, tienen una altura de 100 m (medida desde el mismo nivel del terreno).

En consecuencia, si $y(0) = a\cosh(0) + C = a + C = 20$, se obtiene $C = 20 - a$. Además, en $x = 100$ se debe cumplir

$$y(100) = a\cosh\!\left(\tfrac{100}{a}\right) + C = 100.$$

Reemplazando $C = 20 - a$, se llega a la ecuación no lineal para a:

$$a\,\cosh\!\left(\tfrac{100}{a}\right) \;+\; (20 - a) \;=\; 100 \quad\Longleftrightarrow\quad a\left[\cosh\!\left(\tfrac{100}{a}\right) - 1\right] \;=\; 80.$$

Determina a con tolerancia 10^{-10} usando el método de Newton.

Solución

7 Una empresa enfrenta una demanda exponencial: el precio de venta $P(x)$ (en unidades monetarias por producto) cuando produce x unidades viene dado por

$$P(x) = 200\, e^{-0.01\,x}.$$

Su costo de producción es

$$C(x) = 50 + 10\,x + 0.05\,x^2.$$

Los ingresos son $R(x) = x\,P(x) = 200\,x\,e^{-0.01\,x}$, y el beneficio $\pi(x)$ es $R(x) - C(x)$. Como la condición de primer orden para el máximo conduce a una ecuación trascendental

$$\frac{d\pi}{dx} = 0 \iff 200\,e^{-0.01x}\big(1 - 0.01x\big) \;-\; \big(10 + 0.1x\big) = 0,$$

no se resuelve de forma cerrada. Se pide usar el método de Newton para hallar x^* que maximiza $\pi(x)$, con tolerancia 10^{-10}.

Solución

8 Considere la función

$$f(x) = 0.2\,\sin(16x) \;-\; x \;+\; 1.75.$$

Se sabe que $f(x)$ cambia de signo en el intervalo $[1, 2]$, de modo que existe una raíz real $x^* \in [1, 2]$. Comprueba que si se intenta iniciar el método de Newton en $x_0 = 1$, las iteraciones divergen. En cambio, aplicando el método de bisección en $[1, 2]$ se aproxima la solución.

Solución

9 Se supone que la oscilación de una estructura, dotada de un sistema de amortiguación, ante un movimiento oscilatorio, viene dada por $y(t) = 10e^{-t/2}\cos(2t)$. ¿En qué instante t la posición de la estructura es $y(t) = 4$?

Solución

10 Se puede calcular la raíz n-ésima de un número a resolviendo la ecuación $x^n - a = 0$. Implementa el método de Newton para obtener $\sqrt[3]{345}$ con 7 cifras decimales exactas, partiendo del valor inicial $x_0 = 2$.

Solución

11 Encuentra las dos raices reales de la ecuación:

$$200 \;=\; 100\,\cosh\left(x/0.5\right) \;+\; 20\,\sinh\left(x/0.5\right)$$

en el intervalo $[-2, 2]$. Utiliza un error absoluto de $10^{'} - 10$.

Solución

12 La función de error $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ está definida para $x \geq 0$. Determina el valor de x tal que $erf(x) = 0.5$.
Nota: Utiliza la función de Matlab `erf` para evaluar las integrales.

Solución

13 En un intercambiador de calor, la temperatura de salida T_s satisface la ecuación:

$$T_s = T_e + (T_i - T_e)e^{-\frac{UA}{mc_p}}$$

donde $T_e = 20^\circ C$ es la temperatura exterior, $T_i = 80^\circ C$ es la temperatura inicial, $U = 200\,W/(m^2 \cdot K)$ es el coeficiente global de transferencia de calor, $m = 2\,kg/s$ es el flujo másico, A es el área de intercambio térmico (superficie) del intercambiador de calor y $c_p = 4186\,J/(kg \cdot K)$ es el calor específico. Si se desea una temperatura de salida de $T_s = 35^\circ C$, ¿cuál debe ser el área A?

Solución

14 La velocidad v de un paracaidista que cae está dada por $v = \frac{mg}{c}(1 - e^{-ct/m})$. Para un paracaidista con coeficiente de arrastre $c = 15\,\text{kg/s}$, calcula la masa m de modo que la velocidad sea $v = 126\,\text{km/h}$ en $t = 9\,\text{s}$. Considera $g = 9.81 m/s^2$.

Solución

15 Dada la ecuación $2e^x - x^2 - 1 = 0$:

a. Demuestra que tiene una única solución en el intervalo $[-2, 0]$.

b. Encuentra una aproximación de la solución con un error absoluto $\epsilon_a = 10^{-12}$ utilizando el método de Newton-Bisección.

Solución

16 En un canal trapezoidal, la profundidad crítica y_c satisface la ecuación:

$$\frac{Q^2}{g} = \frac{(by + my^2)^3}{b + 2y\sqrt{1 + m^2}}$$

donde $Q = 2\,m^3/s$ es el caudal, $g = 9.81\,m/s^2$ es la gravedad, $b = 1\,m$ es el ancho del fondo, y $m = 1$ es la pendiente lateral. Encuentra y_c.

Solución

17 Se considera la ecuación $\det(A(x)) = 0$, donde x es un número real y $A(x)$ es la matriz:

$$A(x) = \begin{pmatrix} \cos(x) & 1 & 2 & 3 \\ 0 & 1 & 0 & 1 \\ x & 2 & 3 & 4 \\ \sin(x) & x & 1 & \cos(x) \end{pmatrix}$$

Encuentra una raíz de $f(x) = \det(A(x)) = 0$ en el intervalo $[0, 1]$.

Solución

18 Para la función $f(x) = xe^x - 1$:

a. Demuestra que tiene una única raíz en el intervalo $[0, 1]$.

b. Estudia las condiciones de convergencia del método del punto fijo para las siguientes transformaciones:
   - $g_1(x) = e^{-x}$
   - $g_2(x) = \log(1/x)$
   - $g_3(x) = x - \frac{xe^x - 1}{e^x + xe^x}$

c. ¿Cuál de estas transformaciones garantiza convergencia más rápida?

Solución

19 Utiliza el método de la secante para resolver $f(x) = 100x^5 - 3995x^4 + 79700x^3 - 794004x^2 + 3160075x - 0$ en el intervalo $[17, 22.2]$ con un error absoluto y relativo de 10^{-7}. Realiza los cálculos considerando como puntos iniciales $x_0 = 17$ y $x_1 = 22.2$ y observa que no converge. Sin embargo, comprueba que sí converge si $x_0 = 22.2$ y $x_1 = 17$. ¿Qué justificación se puede dar?

Solución

20 Utiliza el método de Maehly para hallar todas las soluciones de las siguientes ecuaciones:

a. $x^6 + 3x^5 - 28x^4 + 8x^3 - 8x^2 + 4 = 0$

b. $x^7 + 26x^6 - 54x^5 - 27x^4 + 128x^3 - 576x^2 + 864x - 432 = 0$

c. $x^4 - 12x^3 + 47x^2 - 60x + 24 = 0$

¿Cuántas raíces tiene cada ecuación? Comprueba los resultados con el comando `roots` de Matlab.

Solución

RESUMEN

El cápitulo se centra en la resolución aproximada de ecuaciones escalares no lineales. Se analizan métodos iterativos para encontrar soluciones de ecuaciones de la forma $f(x) = 0$, destacando técnicas como el método de bisección, el método de Newton-Raphson y el método de la secante.

También se analiza la convergencia de estos métodos, la rapidez con la que se aproximan a la solución y las condiciones necesarias para su aplicación.

Además, se discuten estrategias para la localización y separación de raíces, haciendo hincapié en la importancia de una elección adecuada del intervalo inicial y de las aproximaciones iniciales para garantizar la precisión en la obtención de soluciones numéricas.

Los aspectos clave en el estudio de los métodos numéricos de este tema son:

1. **Introducción**. Se plantea la necesidad de resolver ecuaciones no lineales de la forma $f(x) = 0$ que, en muchos casos, no admiten soluciones analíticas exactas, lo que motiva el uso de métodos numéricos para obtener soluciones aproximadas.
2. **Generalidades**. Se definen conceptos fundamentales como raíces o ceros de una función, multiplicidad de una raíz y las fases del proceso de resolución: localización, separación y aproximación de raíces.

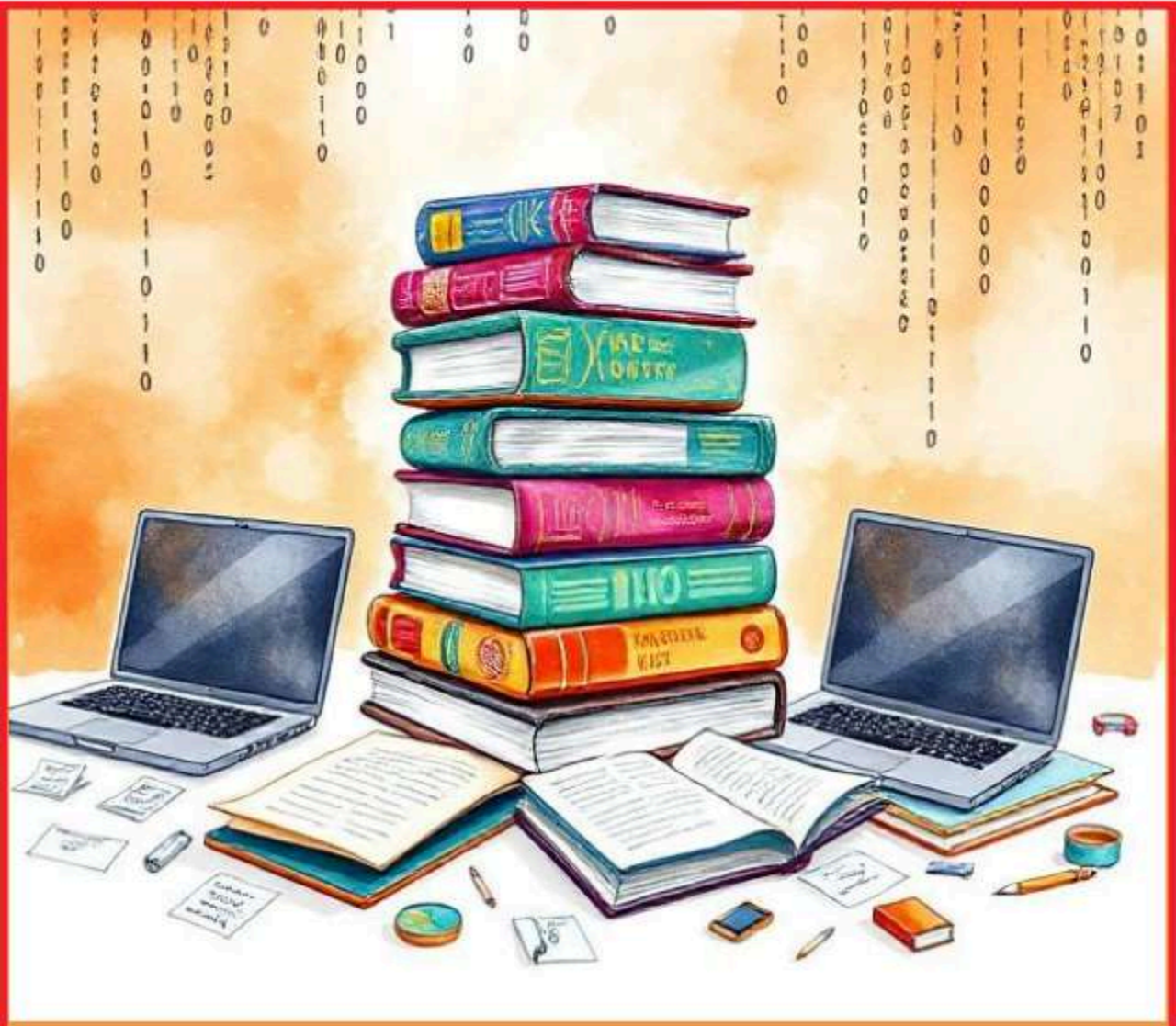
3. **Método del punto fijo**. El método de punto fijo es un procedimiento iterativo que, partiendo de una aproximación inicial, aplica repetidamente una función g para encontrar un valor x que satisfaga $g(x) = x$.

4. **Método de la bisección**. Se presenta este método como una técnica que, partiendo de un intervalo donde la función cambia de signo, divide repetidamente el intervalo en mitades para aproximar la raíz con una precisión deseada.

5. **Método de Newton-Raphson** Se introduce este método iterativo que utiliza la derivada de la función para aproximar la raíz, ofreciendo una convergencia rápida bajo condiciones adecuadas, pero que requiere una buena estimación inicial y que la derivada no sea nula en las proximidades de la raíz.

6. **Método de la secante**. Se describe este método que no requiere el cálculo explícito de la derivada, sino que aproxima la pendiente mediante la cuerda que une dos puntos. Este método es útil cuando la derivada de la función cuyos ceros se quieren obtener es difícil de calcular.

7. **Convergencia de los métodos**. Se analizan las condiciones bajo las cuales los métodos iterativos convergen a una solución, la velocidad de convergencia y cómo la elección de las aproximaciones iniciales influye en el proceso.



4 APROXIMACIÓN DE FUNCIONES POR POLINOMIOS

Capítulo IV

Aproximación de funciones por polinomios

4.1 Introducción

El ajuste por polinomios, en el contexto de los métodos numéricos, consiste en determinar un polinomio de grado adecuado que reproduzca con el menor error posible un conjunto de datos, ya sea obtenido mediante muestreo, experimentación o proveniente de una función analítica. Al expresar la información mediante una fórmula polinómica, se facilitan operaciones como el cálculo de valores intermedios, la interpolación y la predicción de comportamientos futuros.

Se supone por ejemplo que se tiene una tabla que relaciona la viscosidad dinámica del agua con la temperatura

T °C	0	5	10	20
μ_o	1.787	1.519	1.307	1.002

¿Cómo estimar la viscosidad a una temperatura de 7.5°? Una opción es encontrar un polinomio que pase por los puntos de la tabla y estimar después el valor de 7.5 con él.

En este capítulo se presentarán dos técnicas para aproximar una función mediante un polinomio: la interpolación, que ajusta el polinomio pasando exactamente por los datos, y la aproximación por mínimos cuadrados, que busca minimizar el error global cuando no es posible (o deseable) que el polinomio pase por todos los puntos. Para más información, se puede consultar [9] y [6].

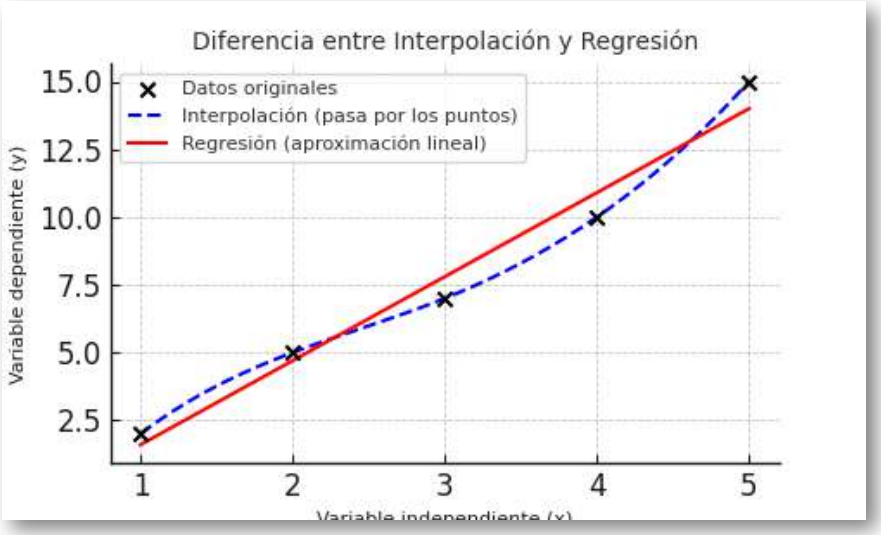


Figura 4.1. Comparación interpolación vs. regresión

4.2 Interpolación de Lagrange

El método de interpolación de Lagrange es una técnica que permite encontrar un polinomio que pasa exactamente por un conjunto de puntos dados.

Dados $n+1$ puntos distintos $x_0, x_1, ..., x_n$ en el intervalo $[a, b]$, el polinomio de Lagrange[5] de una función f definida en $[a, b]$ es el polinomio de grado menor o igual a n que cumple $p(x_i) = f(x_i)$ para cada i entre 0 y n. A los puntos $x_0, x_1, ..., x_n$ se les llama nodos de interpolación.

[5] Joseph Louis Lagrange (1736-1813) fue uno de los más grandes matemáticos de su tiempo. Nació en Italia pero se nacionalizó Francés. Hizo grandes contribuciones en todos los campos de la matemática y también en mecánica. Su obra principal es la "Mécanique analytique"(1788). En esta obra de cuatro volúmenes, se ofrece el tratamiento más completo de la mecánica clásica desde Newton y sirvió de base para el desarrollo de la física matemática en el siglo XIX.

El polinomio de interpolación de Lagrange posee una única solución y viene dado mediante la fórmula:

$$p(x) = \sum_{i=0}^{n} f(x_i)\, l_i(x)$$

donde $\{l_i\}_{i=0}^{n}$ son los polinomios base de Lagrange asociados a los nodos $\{x_i\}_{i=0}^{n}$ que están definidos de la manera siguiente:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

El video siguiente muestra un ejemplo práctico que calcula los polinomios base de Lagrange con tres nodos.

Video 4.1. Método de Lagrange

Ejemplo 4.1. Calcula los polinomios base de Lagrange asociados a los nodos $x_0 = -1, x_1 = 1$ y $x_2 = 2$. y el polinomio que interpola en dichos nodos a la función f real en $[-1, 2]$ que satisface $f(x_0) = 2$, $f(x_1) = 1$ y $f(x_2) = 1$.

Los datos son:

$$x_o = -1 \qquad x_1 = 1 \qquad x_2 = 2$$

$$f(x_o) = 2 \qquad f(x_1) = 1 \qquad f(x_2) = 1$$

Los polinomios base de Lagrange son:

$$\begin{aligned} l_o(x) &= \tfrac{x-1}{-1-1} \cdot \tfrac{x-2}{-1-2} \\ &= \tfrac{1}{6}\left(x^2 - 3x + 2\right) = \tfrac{1}{6}x^2 - \tfrac{1}{2}x + \tfrac{1}{3} \\ l_1(x) &= \tfrac{x-(-1)}{1-(-1)} \cdot \tfrac{x-2}{1-2} \\ &= -\tfrac{1}{2}(x+1)(x-2) = -\tfrac{1}{2}x^2 + \tfrac{1}{2}x + 1 \\ l_2(x) &= \tfrac{x-(-1)}{2-(-1)} \cdot \tfrac{x-1}{2-1} \\ &= \tfrac{1}{3}(x+1)(x-1) = \tfrac{1}{3}x^2 - \tfrac{1}{3} \end{aligned}$$

El polinomio interpolador es:

$$p(x) = f(x_o)l_0(x) + f(x_1)l_1(x) + f(x_2)l_2(x) =$$

$$p(x) = 2\,l_0(x) + l_1(x) + l_2(x) =$$

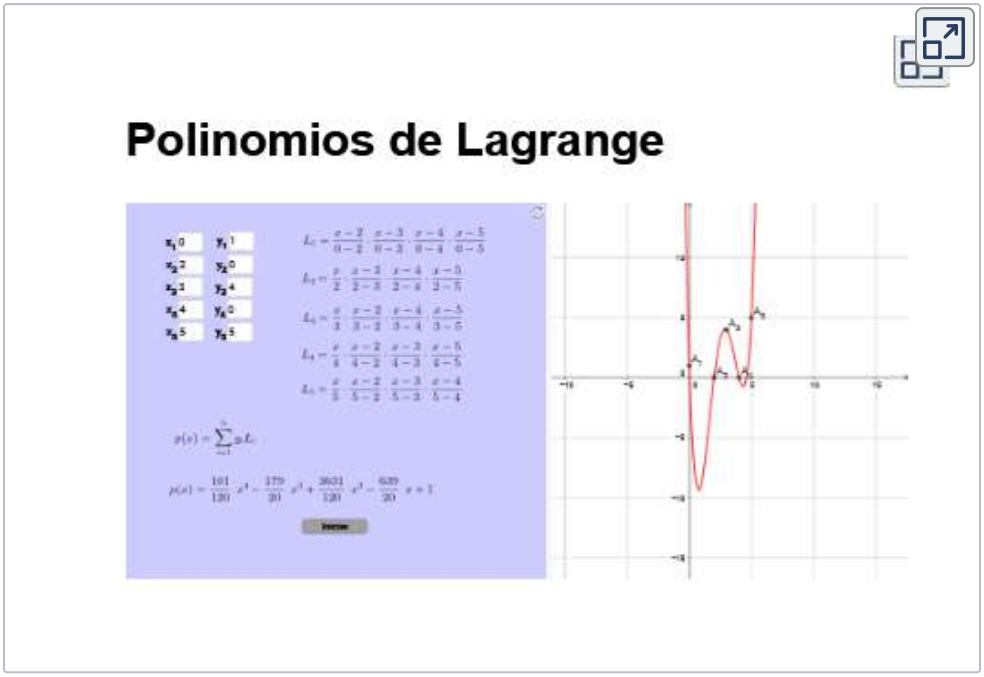
$$= \frac{1}{6}x^2 - \frac{1}{2}x + \frac{4}{3}$$

Se puede comprobar que el polinomio coincide con la función en los nodos (ver figura 4.2):

$$p(-1) = 2 \qquad p(1) = 1 \qquad p(2) = 1$$

Figura 4.2. Polinomio interpolador del ejemplo 4.1

Ejemplo 4.2. Calcula el polinomio de Lagrange con 5 nodos distintos introduciendo sus valores y comprobando su resultado en la herramienta interactiva siguiente.



Interactivo 4.1. Polinomios de Lagrange. Autor: William Jiménez

Ejemplo 4.3. Genera un código Matlab/Octave para construir el polinomio de Lagrange a partir de los polinomios base interpolando un conjunto de datos dados. Considera dos vectores x, y para incluir las coordenadas de los nodos de interpolación.

```
function p=polinomioLagrange(x,y)
% Se parte de un vector x e y de igual
% dimensión en los que se incluye las abscisas
% y ordenadas de los nodos
n = length(x);
% Se inicializa el polinomio resultante
p = zeros(1, n);
for i = 1:n
    %Contrucción de li
    % Índices de todos los puntos salvo el i-ésimo
    indices = [1:i-1, i+1:n];
    % Construye el numerador de li
    li_num = poly(x(indices));
    % Calcula el denominador de li
    li_denom = polyval(li_num, x(i));
    % Obtiene el polinomio base Li(t)
    li = li_num/li_denom;
    % Suma y_i·li(t) al polinomio total
    p = p + y(i) * li;
end
end
```

Para obtener el polinomio del ejercicio 4.1 habría que escribir:

```
x=[-1,1,2];y=[2,1,1];
format rat
p=polinomioLagrange(x,y)
```

La fórmula para calcular el polinomio de Lagrange a partir de los polinomios base posee un interés teórico, pero existen otras formas de obtener en la práctica el polinomio interpolador. Una de ellas se muestra a continuación.

Se consideran los nodos $x_0, x_1, ..., x_n$. Como se quiere calcular $p(x) = a_o + a_1 x + ... + a_n x^n$ de forma que $p(x_i) = f(x_i)$, se tendrá que cumplir que:

$$\begin{pmatrix} x_o^n & x_o^{n-1} & ... & x_0 & 1 \\ x_1^n & x_1^{n-1} & ... & x_1 & 1 \\ ... & ... & ... & ... & ... \\ x_{n-1}^n & x_{n-1}^{n-1} & ... & x_{n-1} & 1 \\ x_n^n & x_n^{n-1} & ... & x_n & 1 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \\ ... \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} f(x_o) \\ f(x_1) \\ ... \\ f(x_{n-1}) \\ f(x_n) \end{pmatrix}$$

El método consiste en construir la matriz A de orden $(n+1)$, el vector b de $\mathbb{R}^n$ y resolver el sistema $Aa = b$, donde a es el vector de los coeficientes del polinomio. La matriz A es invertible[6] y el problema de interpolación de Lagrange tiene una única solución.

Ejemplo 4.4. Calcula con Matlab/Octave el polinomio interpolador de grado 2 de una función que verifica $f(-1) = 2$, $f(1) = 1$ y $f(2) = 1$.

```
x=[-1 1 2]';b=[2 1 1]';n=2;
A=ones(n+1,1);t=A;
for k=1:n
    t=t.*x; A=[t,A];
end
a=A\b   %Solución p(x)=0.1667 x^2 - 0.5 x + 1.3333
%Comprobación
polyval(a,x)
```

En Matlab, el comando `vander` halla la matriz A de Vandermonde a partir de los nodos x_i. También puede obtenerse la matria A con el código `x.^(n:-1:0)`.

[6] La matriz A es de Vandermonde ya que presenta una progresión geométrica en cada fila. Esta matriz es invertible si y solo si todos los x_i son distintos entre sí.

```
x=[-1 1 2]';A=vander(x)
%También A=x^(2:-1:0)
```

Ejemplo 4.5. Interpola con Matlab la función $f(x) = e^x$ en el intervalo $[0, 1]$ con polinomios de grado 5, 10, 15. Evalúa la función y el polinomio en 500 puntos del intervalo y determina cuál es el valor máximo del valor absoluto de la diferencia entre la función y el polinomio en esos puntos para hacer una estimación del error de la aproximación.

El código a utilizar es el siguiente:

```
%Con n=5
n=5;a=0;b=1;x=linspace(a,b,n+1)';
fx=f(x);
A=[x ones(n+1,1)];t=x;
for iter=1:n-1
   t=t.*x;
   A=[t A];
end
p=A\fx;
%Se representa la función y el polinomio
s=linspace(a,b,500)';
fs=f(s);ps=polyval(p,s);
plot(s,[ps fs])
% En valor absoluto el máximo de la diferencia
% entre el polinomio y la función en los puntos x
max(abs(ps-fs))
% Considerando n=10 la diferencia es
% 1.794120407794253e-13
% Considerando n=15 la diferencia es
% 1.110223024625157e-14
```

En el ejemplo anterior, se ha obtenido una estimación del error evaluando la función y el polinomio en puntos del intervalo y calculando su valor máximo, en el siguiente apartado se da una expresión de la cota del error de la interpolación.

4.2.1 Error en la interpolación

El error de interpolación mide la diferencia entre la función real y el polinomio interpolante, proporcionando una estimación cuantitativa de la precisión de la aproximación.

COTA DEL ERROR. Si f es $n+1$ veces derivable en el intervalo $[a,b]$ con $f^{(n+1}$ función continua, una cota del error de interpolación de Lagrange de f asociada a los nodos $\{x_i\}_{i=0}^{n}$ viene dada por la expresión

$$|f(x)-p(x)| \leq \frac{\max\limits_{c\in[a,b]} \left|f^{(n+1}(c)\right|}{(n+1)!} \left|\prod_{j=0}^{n}(x-x_j)\right|$$

Ejemplo 4.6. Sea f en el intervalo $[-1,2]$ cumpliendo $f(-1)=2$, $f(1)=1$ y $f(2)=1$ y p el polinomio interpolador de Lagrange de f en esos puntos. Calcula una cota del error de interpolación en $[-1,2]$ escribiéndola en términos de la derivada de f que corresponda.

Como el polinomio de interpolación es de grado $n=2$, se llama M al valor absoluto máximo de f''' en el intervalo. Se tendrá que para cada x en $[-1,2]$ se cumple,

$$|f(x)-p(x)| \leq \frac{M}{3!}|(x-x_o)(x-x_1)(x-x_2)|$$

$$|f(x)-p(x)| \leq \frac{M}{3!}|(x+1)(x-1)(x-2)|$$

Para obtener una cota en el intervalo $[-1,2]$, se puede considerar el polinomio $q(x)=(x+1)(x-1)(x-2)$, calcular después los ceros de su derivada y tomar finalmente el mayor valor de $|q|$ en ellos.

```
format long
q=poly([-1 1 2]);dq=polyder(q);
r=roots(dq);%Máximos y mínimos de q
v=polyval(q,r);max(abs(v)) %2.112611790922380
```

Una cota del error en funcion de M será entonces:

$$M \cdot 2.112611790922380/6$$

Ejemplo 4.7. Considera la función $f(x) = sen(x)$ y el polinomio interpolador de Lagrange en los nodos $0, \pi/2$ y π. Calcula la cota del error de interpolación de Lagrange en el punto $\pi/4$.

En este caso, el polinomio de interpolación de Lagrange que se obtiene es

$$p\left(x\right) = -0.4053x^2 + 1.2732x$$

Según la expresión del error, en un punto x entre 0 y π se tendrá:

$$|sen\left(x\right) - p\left(x\right)| \leqslant \frac{\max\limits_{c\in[0,\pi]} \left|f'''\left(c\right)\right|}{3!} \left|x-0\right| \left|x-\frac{\pi}{2}\right| \left|x-\pi\right|$$

$$\leq \frac{1}{3!} \left|x-0\right| \left|x-\frac{\pi}{2}\right| \left|x-\pi\right|$$

Sustituyendo x por $\pi/4$ se encontraría una cota de valor: 0.2422. Para encontrar una cota en todo el intervalo, se considera el polinomio $q(x) = x(x-\pi/2)(x-\pi)$ y se calculan sus máximos y mínimos, esto es, las raíces de la derivada. El valor máximo en valor absoluto de q en estos puntos será una cota de $|q\left(x\right)|$.

```
format long
q=poly([0 pi/2 pi]);dq=polyder(q);
r=roots(dq);v=polyval(q,r);max(abs(v))
%1.491790182328260
```

Una cota del error en $[0, \pi]$ sería entonces,

$$|sen\,(x) - p\,(x)| \leq \frac{1}{3!}\,|x-0|\left|x-\frac{\pi}{2}\right||x-\pi| \leq \frac{1.492}{6} \approx 0.248632$$

Dado que en este ejemplo se conoce la función, se calculará una estimación del error de la aproximación entre la función y el polinomio interpolante evaluando en varios puntos y calculando el máximo de la diferencia entre f y p en todos ellos.

```
format long
%a es el polinomio interpolante
v=linspace(0,pi,300);
max(abs(polyval(a,v)-sin(v)))
%0.248631697054710
```

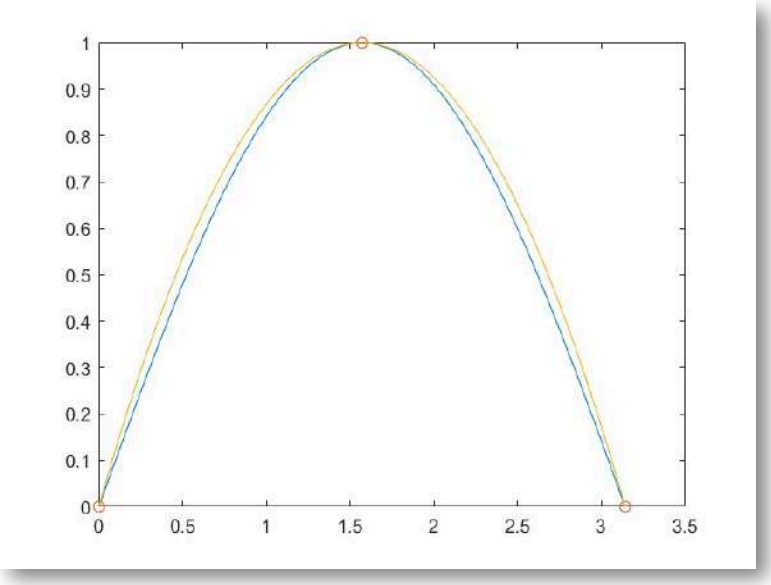


Figura 4.3. Función y polinomio de Lagrange interpolador del ejemplo 4.7

Ejemplo 4.8. Interpola la función $f(x) = log(x)$, logaritmo neperiano, en los nodos $0.99, 1, 1.1$ utilizando polinomios de Lagrange y obtén después aproximaciones de $log(x)$ en los puntos 0.95678 y 1.03597. Compara los valores calculados con los exactos. Da una cota del error de la aproximación en $[0.99, 1.1]$.

Se escribe el código para resolver el problema.

```
% Función log(x)
format long
f=@(x) log(x);
% Grado del polinomio
n=2;
% Nodos: 0.9, 1, 1.1
x=[0.9 1 1.1]';fx=f(x);
A=[x ones(n+1,1)];t=x;
for k=1:n-1
    t=t.*x;
    A=[t A];
end
p=A\fx
% El valor de la aproximación en los puntos pedidos
valor1=polyval(p,0.95678)
valor2=polyval(p,1.03597)
%Las diferencias con los valores que da Matlab es
log(0.95678)-valor1  %1.218237642829495e-04
log(1.03597)-valor2  %-1.022619105114989e-04
% Comprobación
polyval(p,x)-f(x)
% Cota de error
q=poly(x);dq=polyder(q);r=roots(dq);
v=polyval(q,r);M=max(abs(v))
% Derivada tercera de f: 2/x^3 el máximo es 2/(0.9^3)
% El error es M*2/(6*0.9^2)
error=M/(3*0.9^3)
% 1.759945950892248e-04
```

4.2.2 Nodos de Chebyshev

La elección de nodos que permite minimizar el valor de la expresión,

$$\max_{x\in[a,b]}\left|\prod_{j=0}^{n}(x-x_j)\right|$$

que aparece en la cota de error de interpolación de Lagrange, son los puntos de Chebyshev[7].

Estos nodos en [0, 1] son las raíces del $n+1$-ésimo polinomio de Chebyshev que se define de forma recurrente:

$$T_o(x)=1 \qquad T_1(x)=x$$

$$T_k(x)=2xT_{k-1}(x)-T_{k-2}(x) \qquad k\geq 2$$

Se puede demostrar que

$$T_{n+1}(x)=\cos\left((n+1)\,arc\cos(x)\right)$$

Nodos de Chebyshev. Los nodos de Chebyschev en $[a,b]$ se calculan como

$$x_i=\frac{a+b}{2}+\frac{b-a}{2}\cos\left(\frac{(2i+1)\,\pi}{2n+2}\right) \qquad 0\leq i\leq n$$

[7] Pafnuti Lvóvich Tchebyschev (1821 - 1894). El más prominente miembro de la escuela de matemáticas de St. Petersburg. Hizo investigaciones en Mecanismos, Teoría de la Aproximación de Funciones, Teoría de Números y Teoría de Probabilidades. Sin embargo, escribió acerca de muchos otros temas: formas cuadráticas, construcción de mapas, cálculo geométrico de volúmenes, etc.

Ejemplo 4.9. Considerando el intervalo $[-3, 3]$ y tomando el polinomio de interpolación de grado $n = 8$, calcula con Matlab/Octave los nodos de Chebyshev (9 nodos) y represéntalos en el plano.

```
n=8;a=-3;b=3;
%Para obtener los nodos en un vector
x=(a+b)/2+(b-a)/2*cos((2*(0:n)+1)*pi/(2*n+2));
plot(x,zeros(n+1),'o')
```

En la siguiente figura se muestran la distribución de los nodos de Chebyshev y se observa que no están igualmente distribuidos en el intervalo $[-3, 3]$, se concentran más en los extremos.

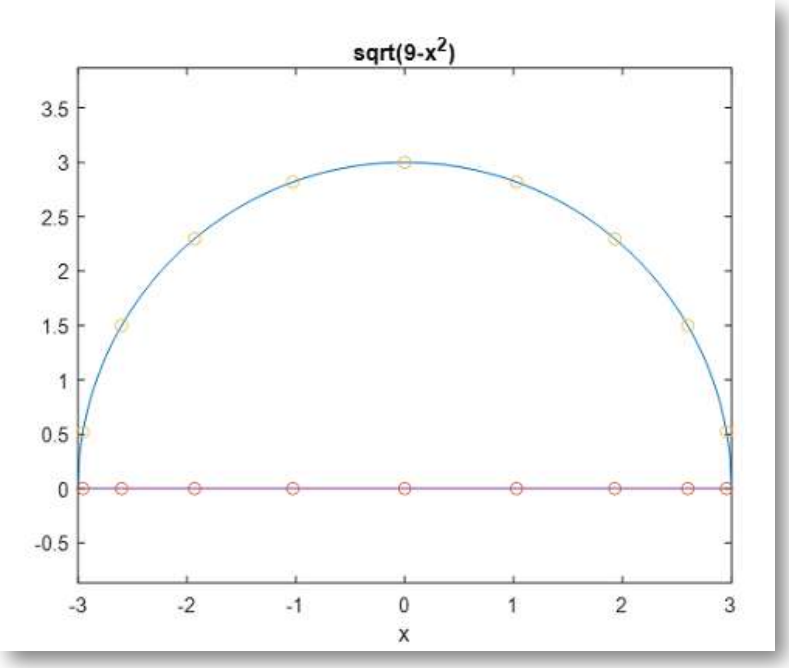


Figura 4.4. Representación de 9 nodos de Chebyshev en el intervalo [-3,3]

Se presenta seguidamente, cuál sería la cota del error si en la interpolación de Lagrange se consideran los nodos de Chevyshev.

Si $\{x_j\}_{j=0}^{n}$ son los nodos de Chebyshev en $[a,b]$, se cumple:

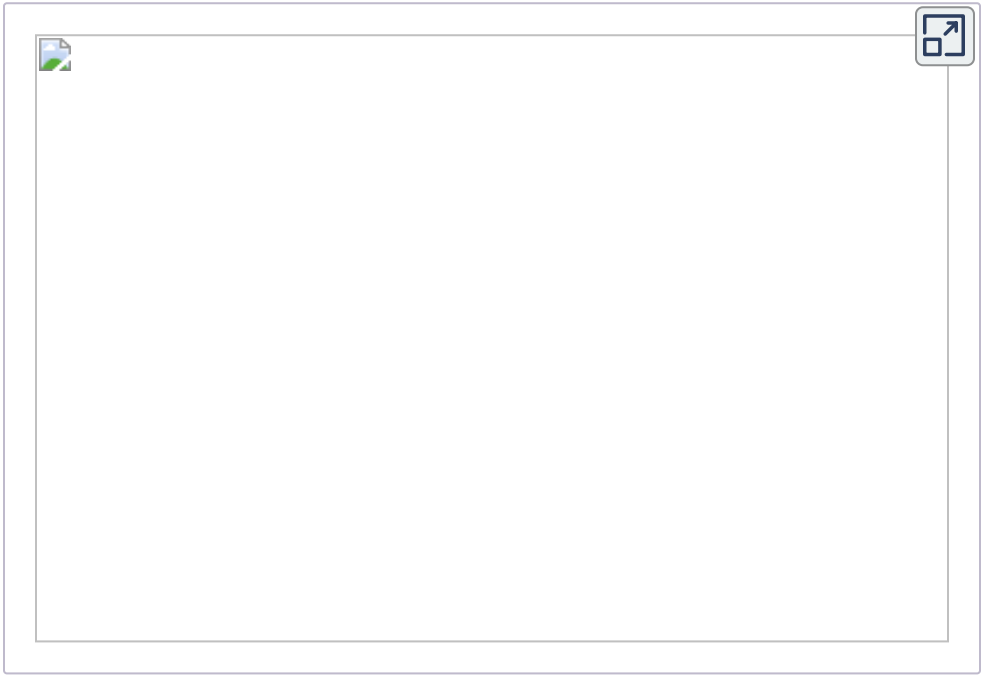
$$\max \left| \prod_{j=0}^{n} (x - x_j) \right| = \frac{(b-a)^{n+1}}{2^{2n+1}}$$

por lo que la cota del error de Lagrange será:

$$|f(x) - p(x)| \leq \frac{\max\limits_{c \in [a,b]} \left| f^{(n+1)}(c) \right|}{(n+1)!} \frac{(b-a)^{n+1}}{2^{2n+1}}$$

Ejemplo 4.10. Interpola la función $f(x) = sen(x)$ en los nodos $x_i = i\pi/10$ para $i = 0, \ldots, 10$ utilizando polinomios de Lagrange y da una cota del error. Considera después 11 nodos de Chebyshev en el intervalo $[0, \pi]$ y analiza el error de la aproximación.

Representa la función y los dos polinomios obtenidos.



Ejemplo 4.11. Dada la función de Runge $f(x) = \frac{1}{1+25x^2}$, halla el polinomio de Lagrange de grado menor o igual a 20 considerando puntos igualmente espaciados en $[-1, 1]$. Representa gráficamente la función y el polinomio interpolador.

```
n=20;% Número de nodos
f=@(x) 1./(1+25*x.^2);
u=linspace(-1,1);
plot(u,f(u))
%polinomio
x=linspace(-1,1,n)'
b=f(x)
m=n-1;
A=x.^(m:-1:0)
p=A\b
hold on
plot(u,polyval(p,u))
hold off
```

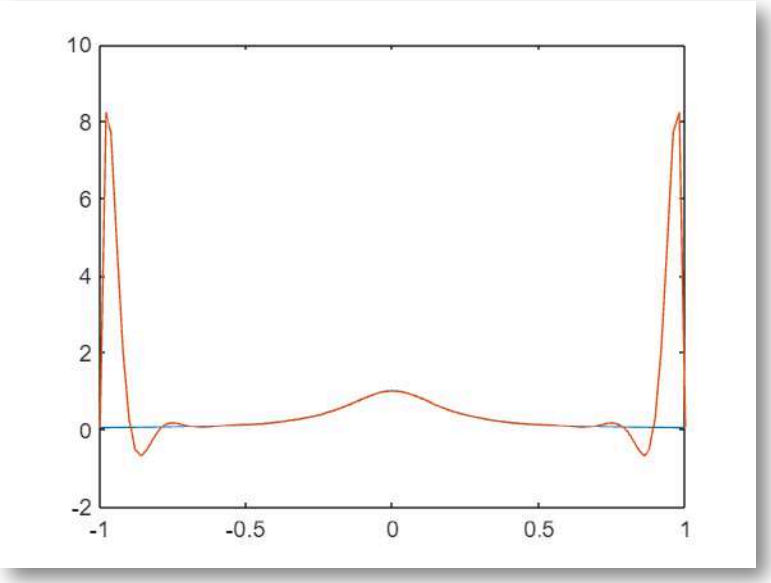


Figura 4.5. Fenómeno Runge

Ejemplo 4.12. Dada la función de Runge $f(x) = \frac{1}{1+25x^2}$, halla los polinomios interpoladores de Lagrange de grado 20 considerando los nodos de Chebyshev en el intervalo $[-1, 1]$. Representa gráficamente la función y los polinomios interpoladores.

```
n=20;%Grado polinomio
f=@(x) 1./(1+25*x.^2);
x=linspace(-1,1);
plot(x,f(x))
%Nodos
x=cos((2*(0:n)+1)*pi/(2*n+2))';%21 nodos
%polinomio
b=f(x)
A=x.^(n:-1:0)
p=A\b
hold on
plot(x,polyval(p,x))
hold off
```

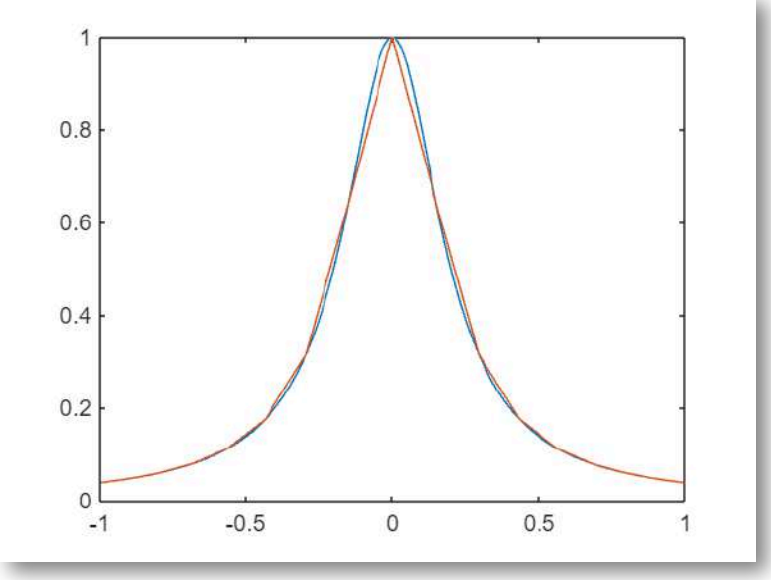


Figura 4.6. Fenómeno Runge: nodos de Chevishev

4.2.3 Diferencias divididas de Newton

El método de diferencias divididas de Newton es una técnica que permite construir el polinomio de interpolación de forma incremental, se evita recalcular desde cero cuando se añade un nuevo nodo. Gracias a su estructura recursiva, solo es necesario actualizar los términos nuevos, lo que facilita su aplicación en problemas donde se agregan puntos progresivamente.

A partir de los datos, $\{x_0, x_1, ..., x_n\}$, se construye una tabla de diferencias divididas:

- **Nivel 0:** los valores originales $f[x_i] = y_i$
- **Nivel 1:** diferencias divididas de primer orden:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

- **Nivel k:** diferencias divididas de orden k:

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

El polinomio interpolador en forma de Newton se escribe de la forma siguiente:

$$\begin{aligned} P(x) = {} & f[x_0] \\ & + f[x_0, x_1]\,(x - x_0) \\ & + f[x_0, x_1, x_2]\,(x - x_0)(x - x_1) \\ & \vdots \\ & + f[x_0, x_1, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i). \end{aligned}$$

Ejemplo 4.13. Se consideran los puntos: $(x_0, y_0) = (1, 3)$, $(x_1, y_1) = (2, 5)$ y $(x_2, y_2) = (4, 2)$. Calcula el polinomio de Lagrange utilizando el método de diferencias divididas.

La tabla de diferencias divididas a partir de los datos del problema será:

		Δ_1	Δ_2
x_i	$f[x_i] = y_i$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$
1	3	$\frac{5-3}{2-1} = 2$	$\frac{-1.5-2}{4-1} = -7/6$
2	5	$\frac{2-5}{4-2} = -1.5$	
4	2	–	

De la tabla se obtiene:

- $f[x_0] = 3$
- $f[x_0, x_1] = 2$
- $f[x_0, x_1, x_2] = -7/6$

Por tanto, el polinomio de interpolación es:

$$P(x) = 3 + 2 \cdot (x - 1) - \frac{7}{6}(x - 1)(x - 2) =$$

$$= -\frac{7}{6}x^2 + \frac{11}{2}x - \frac{4}{3}$$

Se puede comprobar que $P(1) = 3$, $P(2) = 5$ y $P(4) = 2$.

4.3 Interpolación de Hermite

Los polinomios de Hermite extienden la interpolación de Lagrange al incluir no solo los valores de la función en ciertos puntos, sino también sus derivadas. Aseguran que la interpolación no solo pase por los puntos dados, sino que también respete las derivadas en esos puntos.

Sea f una función real definida en el intervalo $[a,b]$, $x_0, x_1, ..., x_k$ son $k+1$ puntos distintos del intervalo $[a,b]$ y $\alpha_0, \alpha_1, ..., \alpha_k$ son $k+1$ enteros no negativos. El problema de interpolación de Hermite consiste en determinar un polinomio de grado menor o igual que $n = k + \alpha_0 + \alpha_1 + ... + \alpha_k$ cumpliendo

$$p(x_o) = f(x_o) \quad p'(x_o) = f'(x_o) \quad ... \quad p^{(\alpha_o}(x_o) = f^{(\alpha_o}(x_o)$$

$$p(x_1) = f(x_1) \quad p'(x_1) = f'(x_1) \quad ... \quad p^{(\alpha_1}(x_1) = f^{(\alpha_1}(x_1)$$

$$...$$

$$p(x_k) = f(x_k) \quad p'(x_k) = f'(x_k) \quad ... \quad p^{(\alpha_k}(x_k) = f^{(\alpha_k}(x_k)$$

Nota: En el caso particular en que $\alpha_i = 1$ para cada i, entonces el problema se denomina interpolación de Hermite clásica.

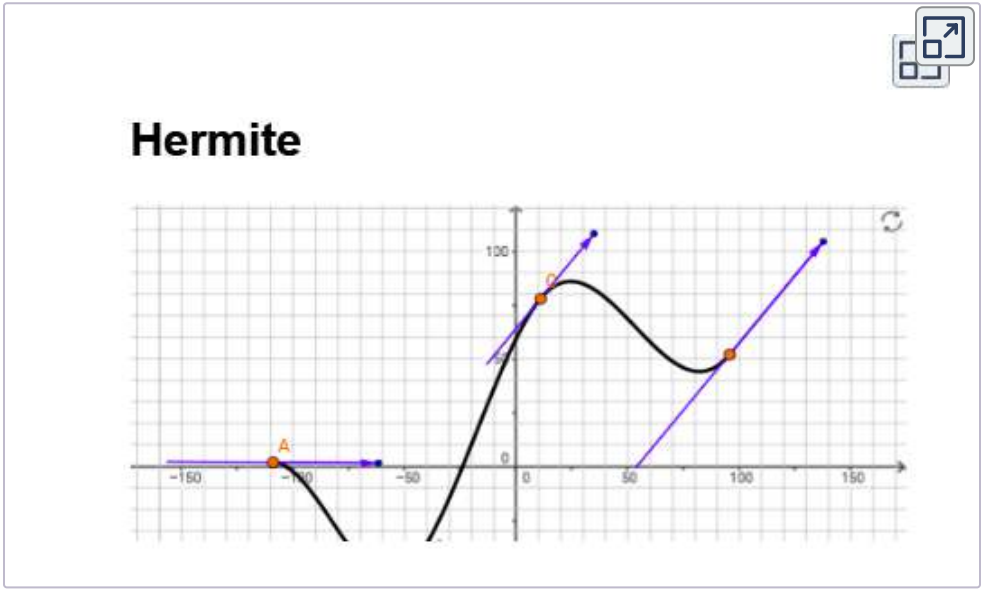
Es importante que en cada nodo x_i que se imponga una condición sobre la derivada de orden α_i, se fije tambien una condición sobre cada derivada inferior a α_i y el valor de la función. En otro caso, los problemas serían de interpolación de Birkhoff que no tienen solución en todos los casos.

El problema de interpolación de Hermite posee una única solución.

En el siguiente video se muestra un ejemplo de interpolación de Hermite.

Video 4.2. Ejemplo de interpolación de Hermite con dos nodos

Ejemplo 4.14. Con la siguiente herramienta interactiva se puede ver cómo varía el polinomio según se modifiquen los cuatro nodos y los valores de las derivadas en cada uno de ellos.



Interactivo 4.2. Polinomio de Hermite con cuatro nodos

Ejemplo 4.15. Escribe el sistema de ecuaciones a resolver para obtener el polinomio interpolador de Hermite cumpliendo que coincide con f y su derivada en los puntos 0 y 1 siendo:

- $f(0) = 1 \quad f'(0) = 1$
- $f(1) = 2 \quad f'(1) = 3$

Determina el polinomio sin utilizar Matlab/Octave.

Dado que se deben cumplir cuatro condiciones, el polinomio será de grado 3, de la forma:

$$p(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_o$$

y su derivada es

$$p'(x) = 3a_3 x^2 + 2a_2 x + a_1$$

Se buscan los coeficientes cumpliendo el siguiente sistema de ecuaciones:

$$\begin{aligned} p(0) &= a_3 0^3 + a_2 0^2 + a_1 0 + a_o = 1 \\ p(1) &= a_3 1^3 + a_2 1^2 + a_1 1 + a_o = 2 \\ p'(0) &= 3a_3 0^2 + 2a_2 0 + a_1 = 1 \\ p'(1) &= 3a_3 1^2 + 2a_2 1 + a_1 = 3 \end{aligned}$$

Entonces, el sistema a resolver para obtener los coeficientes del polinomio es,

$$\begin{aligned} a_o &= 1 \\ a_3 + a_2 + a_1 + a_o &= 2 \\ a_1 &= 1 \\ 3a_3 + 2a_2 + a_1 &= 3 \end{aligned}$$

Resolviendo el sistema se obtiene: $a_0 = 1$, $a_1 = 1$, $a_2 = -2$, $a_3 = 2$ y el polinomio de Hermite buscado es

$$p(x) = 2x^3 - 2x^2 + x + 1$$

Ejemplo 4.16. Tomando como nodos $x_0 = 0$ y $x_1 = 1$, obtén el polinomio de interpolación de Hermite que interpola a la función $f(x) = cos(\pi x/2)$ en la forma siguiente:

$$p(x_i) = f(x_i) \quad p'(x_i) = f'(x_i) \quad p''(x_o) = f''(x_o)$$

Se define la función, su derivada y los nodos del ejemplo.

```
format long
f=@(x) cos(pi*x/2);
df=@(x) sin(pi*x/2)*pi/2;
d2f=@(x) -cos(pi*x/2)*pi^2/4;
x=[0;1];
```

El polinomio a encontrar es de grado menor o igual a 4, es decir, de la forma

$$\begin{aligned} p(x) &= a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_o \\ p'(x) &= 4a_4x^3 + 3a_3x^2 + 2a_2x + a_1 \\ p''(x) &= 12a_4x^2 + 6a_3x + 2a_2 \end{aligned}$$

Se define la matriz del sistema a resolver.

```
A1=[0 0 0 0 1;1 1 1 1 1];
A2=[0 0 0 1 0;4 3 2 1 0];
A3=[0 0 2 0 0];
A=[A1;A2;A3];
```

Se resuelve el sistema de ecuaciones asociado.

```
%El término indendiente es
b=[f(x);df(x);d2f(0)];
p=A\b   %p=
[3.337095776658726;-3.103395226522557;-1.233700550136
170;0;1]
```

4.3.1 Error de interpolación

A continuación, se presenta la expresión del error que permite estimar la precisión de la interpolación de Hermite.

COTA DEL ERROR. Dado el problema de interpolación de Hermite definido en el apartado 4.3, si la función f es $n+1$ veces derivable en el intervalo $[a,b]$ y además es una función continua, entonces una cota del error de interpolación de Hermite viene dada por la expresión:

$$|f(x)-p(x)| \leq \frac{\max\limits_{c\in[a,b]} \left|f^{(n+1}(c)\right|}{(n+1)!} \left|\prod_{j=0}(x-x_j)^{\alpha_j+1}\right|$$

Ejemplo 4.17. Dada la función $f(x)=e^{x/2}$ y los nodos $x_0=0, x_1=\pi/2$ y $x_2=\pi$, calcula el polinomio de Hermite que verifica $p(x_i)=f(x_i)$ y $p'(x_i)=f'(x_i)$. Da una cota del error de la aproximación.

Se define la función, su derivada y los nodos del ejemplo.

```
format long
f=@(x) exp(x/2);
df=@(x) exp(x/2)/2
x=[0;pi/2;pi];
```

El grado del polinomio a calcular es de grado menor o igual a 5. Se considera el término independiente del sistema a resolver.

```
dx=[f(x);df(x)]
```

Se escribe en Matlab/Octave la matriz del sistema de ecuaciones a resolver.

```
A=[x ones(3,1)];t=x;
for k=1:4
    t=t.*x;A=[t A];
end
A
B=[2*x ones(3,1) zeros(3,1)];t=x;
for k=3:5
    t=t.*x;B=[k*t B];
end
B
```

Otra forma de generar la matriz podría ser con el siguiente código

```
A=x^(5:-1:0);
B=(5:-1:0).*x.^[4:-1:0 0];
```

Los coeficientes del polinomio se obtienen resolviendo un sistema de ecuaciones.

```
p=[A;B]\dx
%Comprobación
polyval(p,x)-f(x)
dp=polyder(p);polyval(dp,x)-df(x)
```

Se calcula una cota del error de la aproximación.

```
q=poly(x);q=conv(q,q);
dq=polyder(q);r=roots(dq);
v=polyval(q,r);
M=max(abs(v));
error=M*exp(pi/2)/((2^6)*factorial(6))
```

Ejemplo 4.18. Dada la función $f(x) = log(1+x)$, calcula el polinomio de interpolación de Hermite $p(x)$ que satisface

$$p(x_i) = f(x_i) \quad , \quad p'(x_i) = f'(x_i) \quad , \quad x_i = i/5 \quad 0 \leq i \leq 5$$

Da una cota del error.

Se define la función y los nodos y luego se establece la matriz del sistema.

```
format long
f=@(x) log(1+x);df=@(x) 1./(1+x);
x=linespace(0,1,6);
```

El polinomio a determinar es de grado menor o igual que 11 ya que se tienen 12 condiciones. Sus coeficientes serán la solución del sistema de ecuaciones cuya matriz y término independiente son los siguientes:

```
A1=x^(10:-1:0);
A2=(10:-1:0).*x.^[9:-1:0 0];
A=[A1;A2];b=[f(x);df(x)];
```

Se resuelve el sistema para obtener los coeficientes del polinomio.

```
p=A\b
%Comprobación
polyval(p,x)-f(x)
dp=polyder(p);polyval(dp,x)-df(x)
```

Para calcular la cota del error, se tiene en cuenta que

$$f'(x) = (1+x)^{-1} \quad f''(x) = -(1+x)^{-2} \quad f'''(x) = 2(1+x)^{-3}$$

$$f^{iv}(x) = -3 \cdot 2(1+x)^{-4} \quad \ldots \; f^{(12}(x) = 11!(1+x)^{-11}$$

Se calcula una cota del error de la aproximación, aplicando la expresión vista y teniendo en cuenta que el máximo de la derivada en $[0,1]$ es $11!$.

$$|f(x)-p(x)| \leq \frac{\max\left|f^{(12)}(x)\right|}{12!}\left|\prod_{i=0}^{n}\left(x-\frac{i}{5}\right)^2\right|$$

$$\leq \frac{11!}{12!}\left|x^2\left(x-\frac{1}{5}\right)^2\left(x-\frac{2}{5}\right)^2\left(x-\frac{3}{5}\right)^2\left(x-\frac{4}{5}\right)^2(x-1)^2\right|$$

```
q=poly(x);q=conv(q,q);
dq=polyder(q);r=roots(dq);
v=polyval(q,r);
M=max(abs(v));
%La cota de la derivada dividido el factorial de 12
%es 1/12
error=M/12
```

4.4 Mínimos cuadrados

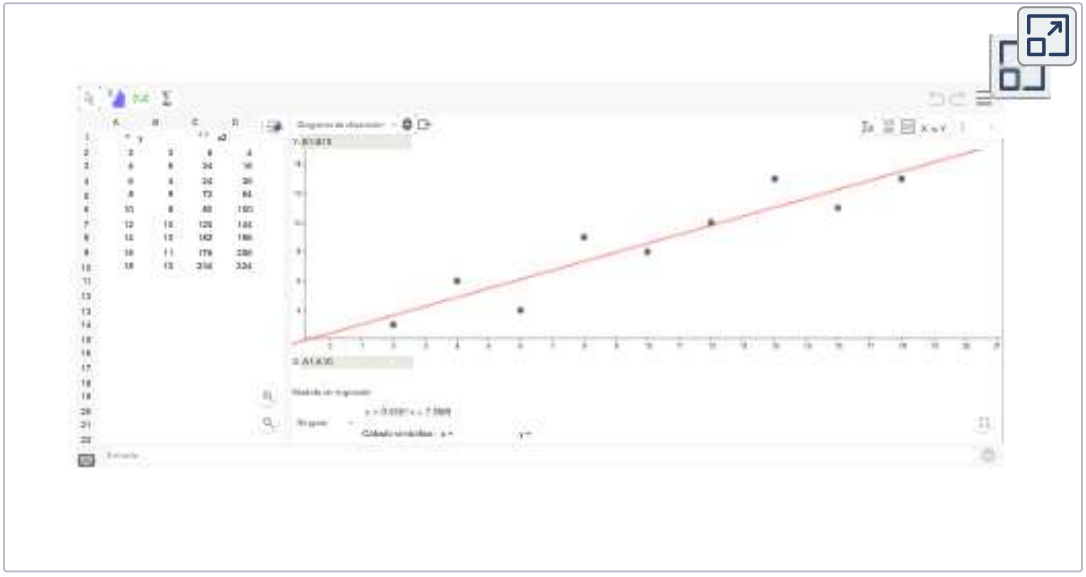
En los métodos de interpolación, aumentar el número de puntos no garantiza una mejor aproximación. De hecho, al utilizar un número elevado de nodos, los cálculos pueden volverse computacionalmente costosos y generar errores de redondeo significativos..

Como se verá con el siguiente método, cuando se dispone de abundante información sobre la función en ciertos puntos, en lugar de incrementar notablemente el grado del polinomio interpolador, puede ser más ventajoso emplear el método de mínimos cuadrados.

La idea del método de mínimos cuadrados consiste en encontar un polinomio de grado k a partir de un conjunto de m puntos de forma que el polinomio tenga grado inferior al que se podría obtener pasando por todos ellos minimizando el error que existe entre el valor del polinomio y las ordenadas de los puntos.

El siguiente vídeo, junto con la herramienta interactiva, ilustra esta idea aplicando el método al ajuste de una recta.

Video 4.3. Ajuste a una recta por mínimos cuadrados



Interactivo 4.3. Ajuste a una recta por mínimos cuadrados

Como ejemplo, la figura siguiente representa el ajuste de una rectaa los puntos $(-1, 3)$, $(2, -1)$, $(3, 3)$, $(4, 2)$ y $(5, 0)$ utilizando el método de mínimos cuadrados.

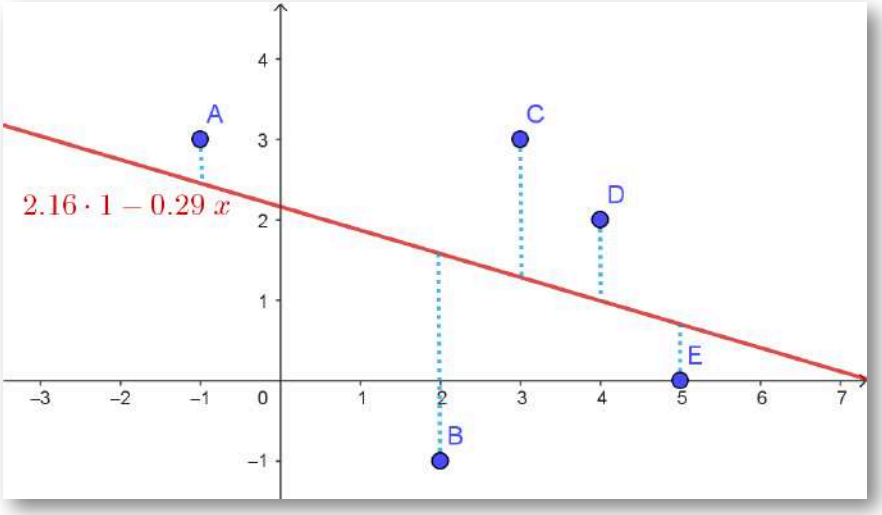


Figura 4.7. Ajuste lineal por mínimos cuadrados

Considerando los mismos puntos y el ajuste por mínimos cuadrados a un polinomio de grado 2, se obtendría la curva que se muestra en la figura siguiente.

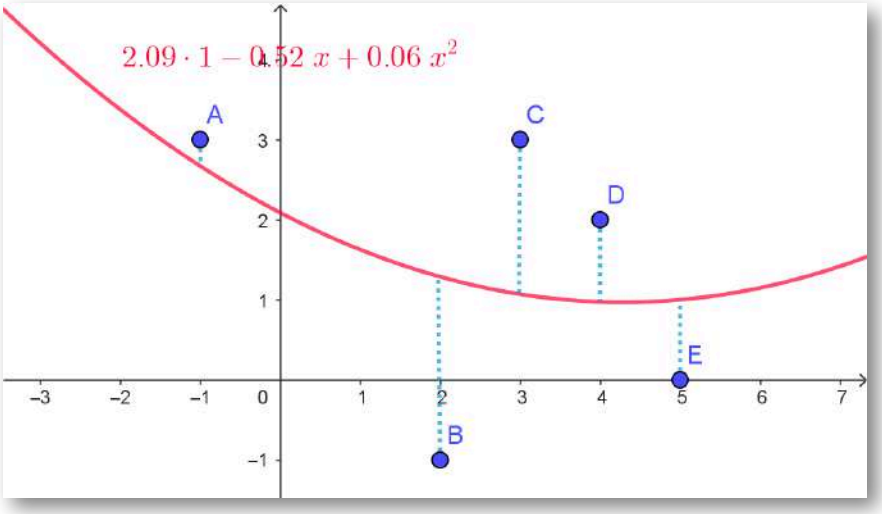


Figura 4.8. Ajuste cuadrático por mínimos cuadrados

Se describe seguidamente la formulación del método de mínimos cuadrados.

Dados $n+1$ puntos distintos $x_0, x_1, ..., x_n$ en el intervalo $[a, b]$, $n+1$ números reales positivos $\{\omega_0, \omega_1, ..., \omega_n\}$, f una función definida en $[a, b]$ y un entero $1 \leq k \leq n$, el problema de mínimos cuadrados consiste en resolver el siguiente problema

$$(P) \quad \underset{p \in P_k}{\text{minimizar}} \sum_{j=0}^{n} \omega_j \left(p\left(x_j\right) - f\left(x_j\right)\right)^2$$

donde P_k denota el espacio de los polinomios de grado menor o igual k.

La razón de incluir los pesos ω_i tiene sentido, por ejemplo, para ajustar el polinomio a los valores más precisos de f o para mejorar la precisión en unas regiones específicas de un intervalo.

4.4.1 Formulación algebraica del problema

Se muestra cómo se puede formular algebraicamente el problema de encontrar el polinomio p de grado menor o igual a k que minimice la siguiente expresión

$$\sum_{j=0}^{n} \omega_j \left(p\left(x_j\right) - f\left(x_j\right)\right)^2$$

Se considera para ello un polinomio p de grado k,

$$p\left(x\right) = a_o + a_1 x + a_2 x^2 + .. + a_k x^k$$

y las siguientes matrices

$$A = \begin{pmatrix} \sqrt{\omega_0}\, x_0^k & \sqrt{\omega_0}\, x_0^{k-1} & \dots & \sqrt{\omega_0}\, x_0 & \sqrt{\omega_0} \\ \sqrt{\omega_1}\, x_1^k & \sqrt{\omega_1}\, x_1^{k-1} & \dots & \sqrt{\omega_1}\, x_1 & \sqrt{\omega_1} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \sqrt{\omega_n}\, x_n^k & \sqrt{\omega_n}\, x_n^{k-1} & \dots & \sqrt{\omega_n}\, x_n & \sqrt{\omega_n} \end{pmatrix}$$

$$a = \begin{pmatrix} a_k \\ a_{k-1} \\ a_{k-2} \\ \vdots \\ a_0 \end{pmatrix} \qquad b = \begin{pmatrix} \sqrt{\omega_o}\, f\left(x_o\right) \\ \sqrt{\omega_1}\, f\left(x_1\right) \\ \vdots \\ \sqrt{\omega_n}\, f\left(x_n\right) \end{pmatrix}$$

Se cumple que

$$Aa - b = \begin{pmatrix} \sqrt{\omega_o}\left(p\left(x_o\right) - f\left(x_o\right)\right) \\ \sqrt{\omega_1}\left(p\left(x_1\right) - f\left(x_1\right)\right) \\ \vdots \\ \sqrt{\omega_n}\left(p\left(x_n\right) - f\left(x_n\right)\right) \end{pmatrix}$$

Teniendo en cuenta la norma euclídea

$$\|Aa - b\|^2 = \sum_{j=0}^{n} \omega_j (p\left(x_j\right) - f\left(x_j\right))^2$$

los la solución del problema del problema:

$$(P2) \quad \underset{x \in \mathbb{R}^{k+1}}{\text{minimizar}} \|Ax - b\|^2$$

serán los coeficientes del polinomio p que se busca.

Se trata de encontrar por tanto el vector x, que se corresponde con los coeficientes del polinomio, que haga mínimo la norma del vector de residuos $Ax - b$, es decir, la desviación global entre la aproximación, Ax, y los datos observados b.

En los ejemplos se considerarán todos los pesos iguales a 1, por lo que el problema se reduce a minimizar la siguiente expresión::

$\min \|Ax-b\|^2$ donde

$$Ax-b=$$

$$=\underbrace{\begin{pmatrix} x_0^k & x_0^{k-1} & \dots & x_0 & 1 \\ x_1^k & x_1^{k-1} & \dots & x_1 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ x_n^k & x_n^{k-1} & \dots & x_n & 1 \end{pmatrix}}_{(n+1)\times(k+1)} \underbrace{\begin{pmatrix} a_k \\ a_{k-1} \\ \dots \\ a_o \end{pmatrix}}_{(k+1)\times 1} - \underbrace{\begin{pmatrix} f(x_o) \\ f(x_1) \\ \dots \\ f(x_n) \end{pmatrix}}_{(n+1)\times 1}$$

siendo $n+1$ el número de nodos y k el grado del polinomio.

Ejemplo 4.19. Formula algebraicamente el problema de ajustar en el sentido de mínimos cuadrados un polinomio de grado 2 de la forma $y = a_o + a_1 x + a_2 x^2$ a un conjunto de observaciones: $(x_0, y_0), (x_1, y_1)$,..., (x_n, y_n) con $n > 2$.

El problema de mínimos cuadrados consiste en encontrar a_o, a_1, a_2, coeficientes del polinomio $p(x) = a_2 x^2 + a_1 x + a_0$, de forma que sea mínima la siguiente expresión

$$\sum_{i=0}^{n} \left(y_i - \left(a_o + a_1 x_i + a_2 x_i^2\right)\right)^2$$

Para escribir el problema de forma matricial, se considera

$$A = \begin{pmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ \dots & \dots & \dots \\ x_n^2 & x_n & 1 \end{pmatrix} \quad x = \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} \quad b = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_n \end{pmatrix}$$

El problema se reduce a buscar el vector x de $\mathbb{R}^3$ solución del problema $\min \|Ax-b\|$.

Ejemplo 4.20. Imagina un muelle colgado verticalmente dentro de un cilindro abierto por los extremos a altura 0, por encima de la apertura superior. La longitud del muelle es desconocida porque su extremo inferior está dentro del cilindro.

Se desea determinar L, y al mismo tiempo la constante del muelle usando la ley de Hooke: $F = k(h - L) = kh - kL$ donde F denota la fuerza aplicada al muelle y h es la distancia desde el extremo superior del muelle. Para ello, se disponen de los siguientes datos,

h	6	7	10
F	3	8	12

Se llama $a_1 = k$ y $a_0 = -kL$ y se busca la recta $F(h) = a_1 h + a_2 0$ que se ajuste a los datos. El sistema resultante es,

$$6a_1 + a_0 = 3$$

$$7a_1 + a_0 = 8$$

$$10a_1 + a_0 = 12$$

En forma matricial, se podría escribir de la forma $Ax = b$ siendo

$$A = \begin{pmatrix} 6 & 1 \\ 7 & 1 \\ 10 & 1 \end{pmatrix} \quad x = \begin{pmatrix} a_1 \\ a_0 \end{pmatrix} \quad b = \begin{pmatrix} 3 \\ 8 \\ 12 \end{pmatrix}$$

El problema de mínimos cuadrados consiste en encontrar el vector x de $\mathbb{R}^2$ de forma que sea mínima la expresiónn: $\|Ax - b\|$.

4.4.2 Resolución del problema mediante la factorización QR

En este apartado se verá cómo resolver el problema siguiente,

$$(P2) \quad \underset{x \in \mathbb{R}^{k+1}}{\text{minimizar}} \, \|Ax - b\|^2$$

utilizando la descomposición QR de la matriz A de orden $(n+1)\text{x}(k+1)$ siendo k el grado del polinomio y $n+1$ el número de puntos. El método se describe a continuación.

1. Se realiza la factorización QR de A donde Q es ortogonal, es decir, $Q^TQ = QQ^T = I$ y tiene dimensiones $(n+1)\text{x}(n+1)$ y R es una matriz triangular superior $(n+1)\text{x}(k+1)$

2. Se consideran en Q y R las submatrices Y y $\widehat{R}$:

   - $Q = [Y\,Z]$ con Y matriz $(n+1)\text{x}(k+1)$ y Z matriz $(n+1)\text{x}(n-k)$
   - $R = \begin{pmatrix} \widehat{R} \\ 0 \end{pmatrix}$ siendo $\widehat{R}$ es una matriz triangular superior $(k+1)\text{x}(k+1)$ y 0 es una matriz formada por ceros de orden $(n-k)\text{x}(k+1)$

3. Los coeficientes del polinomio solución
   $$a = (a_k, a_{k-1}, ..., a_1, a_0)^T$$
   se obtienen resolviendo el sistema de ecuaciones lineales
   $$\widehat{R}x = Y^Tb$$
   y el residuo es
   $$\text{Res} = \left\|Z^Tb\right\|$$

A continuación, se justifica el punto 3.

Como Q es ortogonal se cumple

$$\|Qv\|^2 = (Qv)^T (Qv) = v^T \left(Q^T Q\right) v = v^T I v = v^T v = \|v\|^2$$

Por lo tanto,

$$\|Ax - b\|^2 = \|QRx - b\|^2 = \left\|Q\left(Rx - Q^T b\right)\right\|^2 =$$

$$\left\|Rx - Q^T b\right\|^2 = \left\| \begin{pmatrix} \widehat{R} \\ O \end{pmatrix} x - \begin{pmatrix} Y^T \\ Z^T \end{pmatrix} b \right\|^2 = \left\| \begin{pmatrix} \widehat{R}x - Y^T b \\ -Z^T b \end{pmatrix} \right\|^2 =$$

$$\left\| \widehat{R}x - Y^T b \right\|^2 + \| Z^T b \|^2$$

El mínimo valor del problema se obtiene para el valor de x que hace cero el término

$$\left\| \widehat{R}x - Y^T b \right\|^2$$

De la expresión anterior se deduce que la solución del problema (P_2) se obtiene de forma equivalente calculando el valor x solución del sistema

$$\widehat{R}x = Y^T b$$

donde si k es el grado del polinomio buscado y QR es la factorización de A, la matriz $\widehat{R}$ es la submatriz de R con las primeras $k+1$ filas y la matriz Y es la submatriz de Q considerando las primeras $k+1$ columnas.

Hay que hacer notar que la matriz $\widehat{R}$ es invertible debido a que los nodos $\{x_j\}_{j=0}^n$ son todos distintos. Por lo tanto, la solución del problema de mínimos cuadrados (P2) existe y es única.

Ejemplo 4.21. Considerando el ejemplo anterior de la ley de Hooke, se seguirá el método visto de descomposición QR para obtener la recta que resuelve el problema de mínimos cuadrados con los datos

h	6	7	10
F	3	8	12

A partir de los datos, se tiene

$$A = \begin{pmatrix} 6 & 1 \\ 7 & 1 \\ 10 & 1 \end{pmatrix} \quad x = \begin{pmatrix} a_1 \\ a_0 \end{pmatrix} \quad b = \begin{pmatrix} 3 \\ 8 \\ 12 \end{pmatrix}$$

Para encontrar la descomposición QR de la matriz A se puede utilizar el código Matlab siguiente,

```
[Q,R]=qr([6 1;7 1;10 1])
```

Las matrices obtenidas son,

$$Q = \begin{pmatrix} -0.4411 & -0.6777 & -0.5883 \\ -0.5147 & -0.3460 & 0.7845 \\ -0.7352 & 0.6488 & -0.1961 \end{pmatrix}$$

$$R = \begin{pmatrix} -13.6015 & -1.6910 \\ 0 & -0.3749 \\ 0 & 0 \end{pmatrix}$$

Como se trata de una recta, el valor de k es 1. Las matrices Y, $\widehat{R}$ son

$$Y = \begin{pmatrix} -0.4411 & -0.6777 \\ -0.5147 & -0.3460 \\ -0.7352 & 0.6488 \end{pmatrix} \quad \widehat{R} = \begin{pmatrix} -13.6015 & -1.6910 \\ 0 & -0.3749 \end{pmatrix}$$

El sistema a resolver es entonces, $\widehat{R}x = Y^T b$.

Es decir,

$$\begin{pmatrix} -13.6015 & -1.6910 \\ 0 & -0.3749 \end{pmatrix} \begin{pmatrix} a_1 \\ a_0 \end{pmatrix}$$

$$= \begin{pmatrix} -0.4411 & -0.5147 & -0.7352 \\ -0.6777 & -0.3460 & 0.6488 \end{pmatrix} \begin{pmatrix} 3 \\ 8 \\ 12 \end{pmatrix}$$

Operando,

$$\begin{pmatrix} -13.6015 & -1.6910 \\ 0 & -0.3749 \end{pmatrix} \begin{pmatrix} a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} -14.2632 \\ 2.9847 \end{pmatrix}$$

La solución de este sistema es $a_1 = 2.0385$ y $a_o = -7.9615$

Las submatrices Y y $\widehat{R}$ se obtienen a partir de Q y R con el código siguiente.

```
k=1;
Y=Q(:,1:k+1)'
RG=R(1:k+1,:)
```

Finalmente, se resuelve el sistema.

```
fx=[3;8;12];
p=RG\Y*fx
```

Para dibujar los puntos y la recta con Matlab/Octave se puede escribir el siguiente código.

```
h=[6;7;10];
ph=polyval(p,h);
plot(h,fx,'o')
hold on
plot(h,ph)
hold off
```

Ejemplo 4.22. Dada la función de Runge $f(x) = \frac{1}{1+x^2}$, halla el polinomio de grado menor o igual que 20 que mejor aproxima la función en el sentido de mínimos cuadrados en $[-5, 5]$, tomando como datos los valores de $f(x)$ en 100 nodos igualmente espaciados.

```
f=@(x) 1./(1+x.^2);
n=99;k=20;a=-5;b=5;
x=linspace(a,b,n+1)';fx=f(x);
A=[ x ones(n+1,1)];t=x;
for iter=1:k-1
    t=t.*x;
    A=[t A];
end;
[Q,R]=qr(A);
p=R(1:k+1,:)\Q(:,1:k+1)'*fx;
%Representamos la función
s=linspace(a,b,1001)';
fs=f(s);ps=polyval(p,s);
plot(s,[ps fs])
```

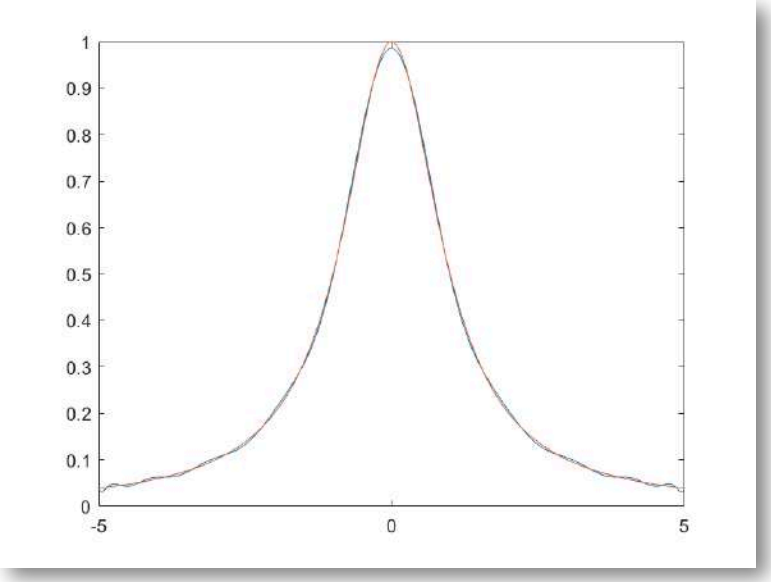
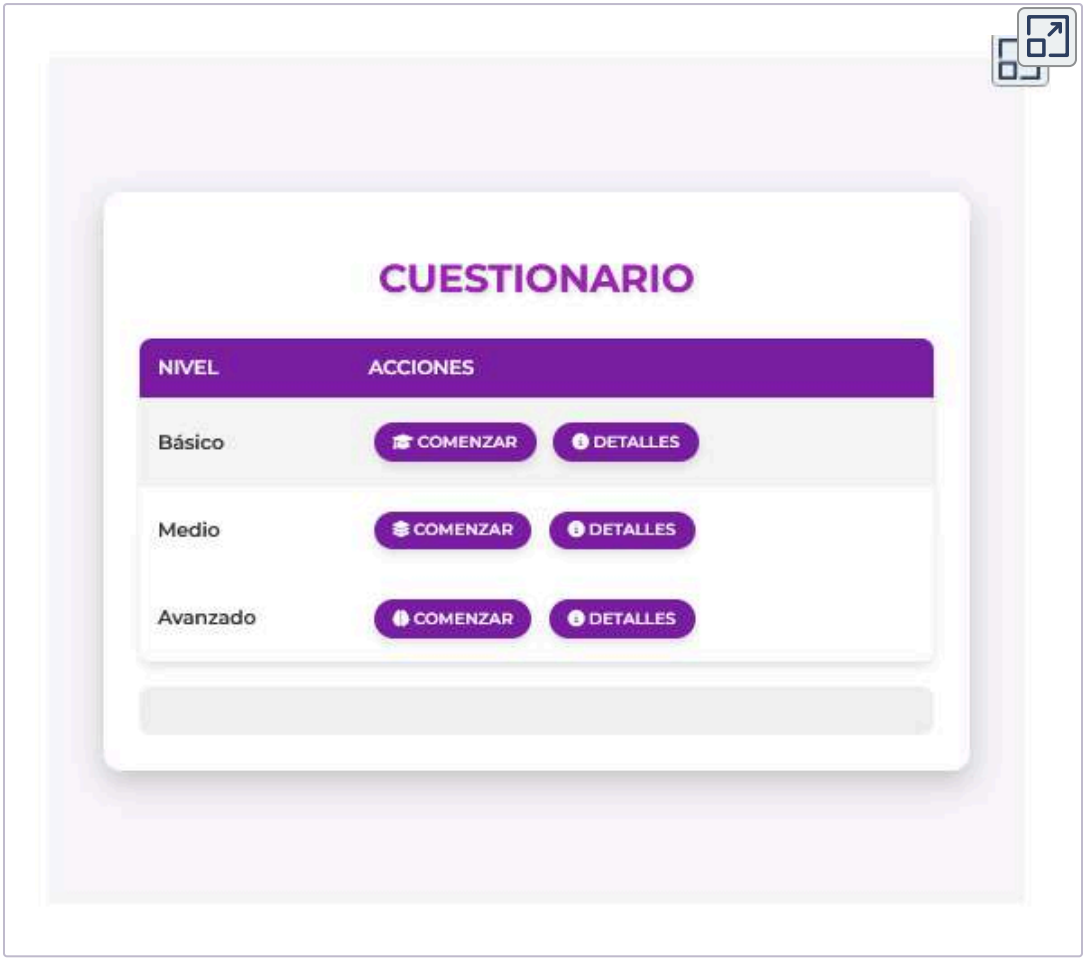


Figura 4.9. Representación de la aproximación del ejemplo

4.5 Autoevaluación



Interactivo 4.4. Actividad de autoevaluación

4.6 Ejercicios

1 Calcula el polinomio interpolador de grado 2 de una función que verifica: $f(-1) = 2 \quad f(1) = 1 \quad f(2) = 1$. Calcula este polinomio a partir de los polinomios base de Lagrange y resolviendo el sistema de ecuaciones.

Solución

2 Interpola con MATLAB la función $f(x) = e^{-x^2}$ en el intervalo $[0, 1]$ con polinomios de grado 5, 10, y 15. Evalúa la función y el polinomio en 500 puntos del intervalo y encontrar el máximo de la diferencia entre la función y el polinomio en esos puntos.

Solución

3 Se considera la función $f\,(x) = \cos(x)$ en el intervalo $[0,\ \pi]$.

a. Escribe el sistema que permitiría obtener el polinomio de interpolación de Lagrange de la función $f\,(x)$ tomando como nodos $x_o = 0, x_1 = \pi/2, x_2 = \pi$.

b. Calcula con Matlab el polinomio interpolador de Lagrange considerando 8 nodos equidistantes en el intervalo $[0,\ \pi]$ siendo el primer nodo 0 y π el último. Dibuja la función y el polinomio interpolador calculado.

c. ¿Cuál sería una cota del error de aproximación del polinomio obtenido en el apartado anterior en el intervalo $[0,\ \pi]$?

d. Considera 8 nodos de Chebyshev en $[0,\ \pi]$ y escribe su valor. Determina el polinomio de interpolación de la función en este intervalo considerando estos nodos. Dibuja la función y el polinomio interpolador.

e. ¿Cuál sería una cota del error de aproximación del polinomio obtenido en el apartado anterior en el intervalo $[0,\ \pi]$?

Solución

4 La siguiente tabla proporciona el calor específico de la plata (en grados Kelvin) a diversas temperaturas:

T (K)	8	10	12	14	16	18
C(T)	0.0236	0.0475	0.083	0.1736	0.202	0.25

Halla el polinomio interpolador y da un valor aproximado del calor específico a 15 K. Representa gráficamente los datos y el polinomio interpolador.

Solución

5 Se considera la función $f(x) = cos(x)$ en el intervalo $[0, \pi]$. Considera el siguiente problema de Hermite, donde p es el polinomio interpolador.

$$\begin{array}{lll} f(0) = p(0) & f'(0) = p'(0) & f''(0) = p''(0) \\ f\left(\frac{\pi}{2}\right) = p\left(\frac{\pi}{2}\right) & f'\left(\frac{\pi}{2}\right) = p'\left(\frac{\pi}{2}\right) & \\ f(\pi) = p(\pi) & & \end{array}$$

Escribe el sistema que habría que resolver para hallar el polinomio p y la expresión que daría una cota del error de la interpolación de Hermite en cada x de $[0, \pi]$.

Solución

6 Dada la función $f(x) = \log(1 + x)$, calcula el polinomio de interpolación de Hermite $p(x)$ que satisface:

- $p(x_i) = f(x_i)$
- $p'(x_i) = f'(x_i)$
- $x_i \in \{0, 0.5, 1\}$

Da una cota del error de interpolación y compara el resultado con el máximo en valor absoluto de la diferencia entre la función y el polinomio tomando 500 puntos del intervalo $[0, 1]$.

Solución

7 Dada la función $f(x) = \sin(\pi x)$, calcula el polinomio de interpolación $p(x)$ que resuelve los siguientes problemas ya que:

a. $p(x_i) = f(x_i)$, $p'(x_i) = f'(x_i)$ donde x_i son 10 puntos igualmente espaciados en el intervalo $[0, 1]$.

b. $p(x_i) = f(x_i)$, donde p es un polinomio de grado 10 siendo x_i son los nodos de Chebyshev en el intervalo $[0, 1]$.

Representa gráficamente ambos polinomios junto con la función $f(x)$.

Solución

8 Ajusta una línea recta a los valores x e y en el sentido de mínimos cuadrados, usando la siguiente tabla:

x	1	2	3	4	5	6	7
y	0.5	2.5	2	4	3.5	6	5.5

Determina el polinomio interpolador que se ajusta a estos valores y grafica las soluciones junto con el conjunto (x, y) en la misma figura.

Solución

9 Dada la función $f(x) = e^{-x^2}$, halla el polinomio de grado menor o igual a 20 que mejor aproxima la función en el sentido de mínimos cuadrados en $[-5, 5]$, tomando como datos los valores de $f(x)$ en nodos igualmente espaciados.

Solución

10 Dada la función $f(x) = \frac{e^{-x^2}\sin(x)}{x^3+x^2+1}$ calcula el polinomio p de grado 15 mejor aproximación de f en el sentido de mínimos cuadrados, tomando como datos 100 puntos $(x_i, f(x_i))$ estando x_i igualmente espaciados en el intervalo $[0, 2\pi]$.

Solución

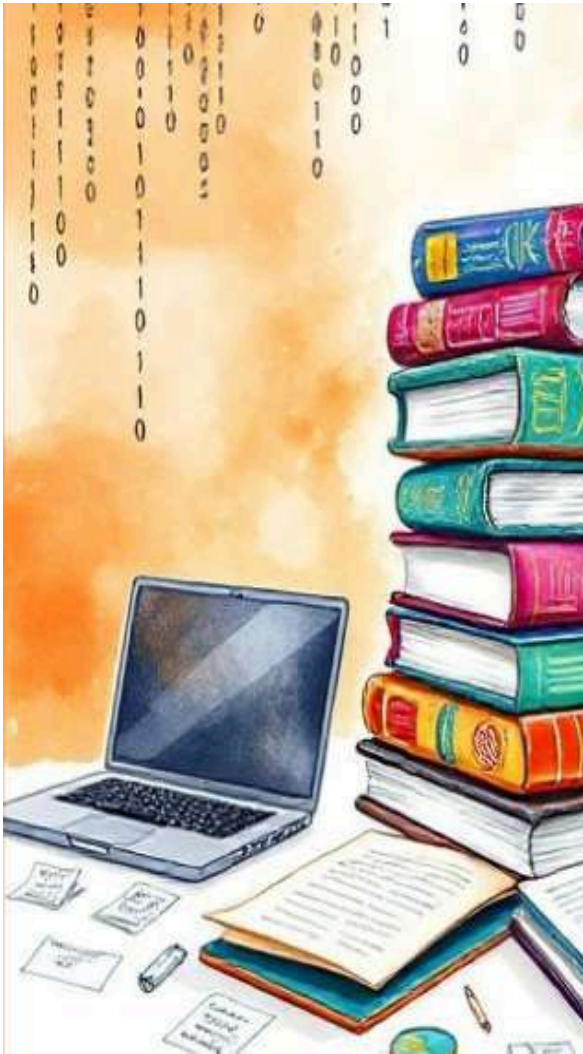
RESUMEN

Este capítulo aborda la aproximación de funciones de una variable mediante polinomios, presentando dos técnicas principales: la interpolación y la aproximación por mínimos cuadrados. En la interpolación, se destacan los métodos de Lagrange y Hermite, que permiten construir polinomios que pasan exactamente por un conjunto de puntos dados. Por otro lado, la aproximación por mínimos cuadrados se utiliza cuando se dispone de una gran cantidad de datos y se busca un polinomio que minimice el error global, sin que necesariamente pase por todos los puntos.

El capítulo también aborda la elección óptima de nodos, como los nodos de Chebyshev, para mejorar la precisión de la interpolación y evitar problemas como el fenómeno de Runge. Además, se discuten las limitaciones de los métodos de interpolación en casos de funciones con comportamientos complejos,

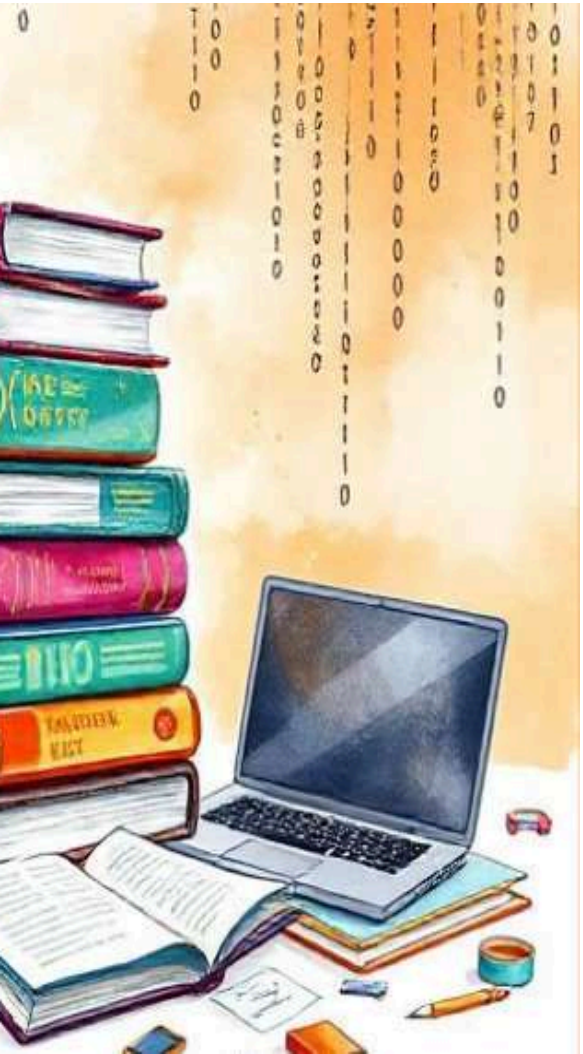
Los aspectos clave estudiados en este tema son:

1. **Introducción.** Se presenta la necesidad de aproximar funciones desconocidas o complejas mediante polinomios, utilizando técnicas como la interpolación y la aproximación por mínimos cuadrados.
2. **Interpolación de Lagrange.** Se introduce el polinomio de Lagrange como una herramienta para construir un polinomio de grado n que pasa por $n+1$ puntos dados, garantizando una única solución y proporcionando una fórmula explícita para su cálculo.



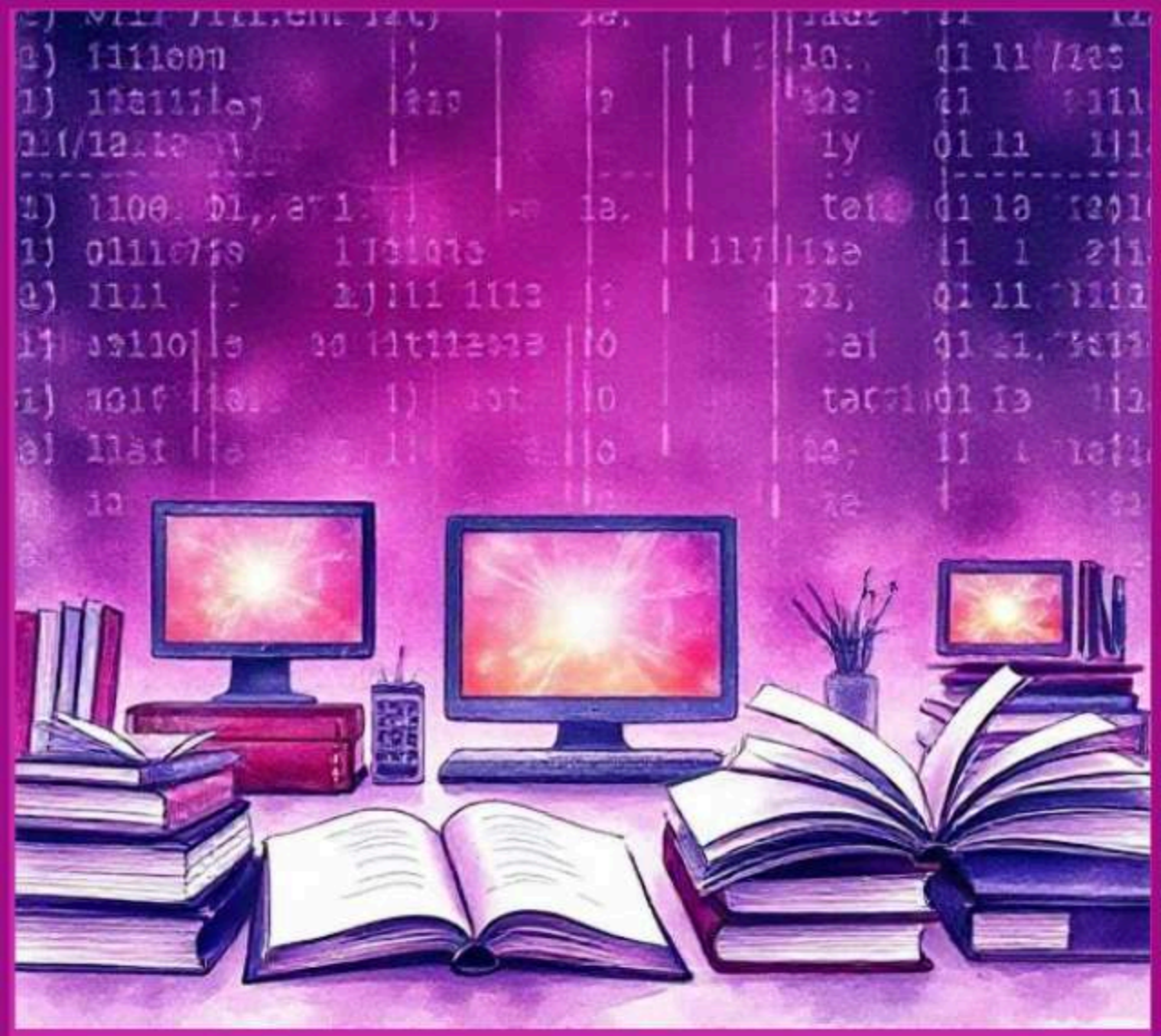
3. **Método de mínimos cuadrados.** Se presenta la técnica de aproximación por mínimos cuadrados, que permite ajustar un polinomio de grado menor que el número de puntos disponibles, minimizando el error global y proporcionando una mejor aproximación en presencia de datos con ruido o inconsistencias.

4. **Errores en la interpolación.** Se analizan los errores asociados a la interpolación polinómica, incluyendo el fenómeno de Runge, que ilustra cómo, aumentar el número de nodos, no siempre mejora la aproximación pudiendo empeorarla en algunos casos.



5. **Nodos de Chebyshev.**Se discute la elección óptima de nodos para la interpolación, destacando que los nodos de Chebyshev minimizan el error de interpolación.

6. **Aplicaciones prácticas.** Se proporcionan ejemplos y ejercicios propuestos que ilustran la aplicación de las técnicas de interpolación y aproximación por mínimos cuadrados.



5 INTEGRACIÓN NUMÉRICA

Capítulo V

Integración numérica

5.1 Introducción

Los métodos numéricos de integración sirven para aproximar el valor de una integral definida de una función en un intervalo con una precisión dada cuando no es posible resolverla de forma exacta o analítica. El interés de estas técnicas puede venir motivado por distintas razones.

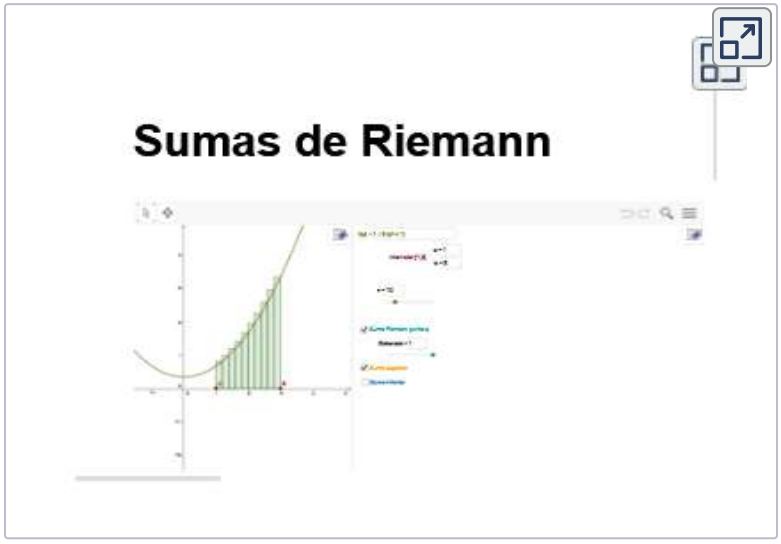
1. La posibilidad de aplicar la Regla de Barrow se restringe a una pequeña parte de funciones que tienen primitivas que se puedan escribir a partir de operaciones algebraicas de funciones elementales. Muchas funciones, aún siendo continuas, no tienen primitivas elementales como, por ejemplo, e^{-x^2}, utilizada en la distribución normal o $\frac{e^{-x}}{x}$, en procesos de decaimiento y teoría de colas.

2. Aunque existiera un procedimiento para encontrar la primitiva, obtenerla puede resultar en muchos casos muy complicado. Se puede pensar por ejemplo en las funciones racionales o en aquellas que requieren de cambios de variable más o menos ingeniosos.

3. Por otro lado, en muchas aplicaciones sólo se tiene una tabla de valores de la función o la posibilidad de obtener su valor en un número finito de puntos. En estos casos, el cálculo aproximado permite estimar magnitudes que se expresan mediante integrales definidas.

Estas razones obligan a encontrar métodos alternativos para calcular de forma aproximada el valor de una integral definida en un intervalo. Por razones históricas, el cálculo aproximado de integrales se denomina cuadratura dejando la denominación de integración para la resolución de ecuaciones diferenciales.

Como es conocido, las sumas de Riemann permiten aproximar el valor de la integral definida de una función integrable. Se verá en este tema otras fórmulas utilizando polinomios interpoladores.



Interactivo 5.1. Sumas de Riemann

Se comienza viendo un ejemplo de aplicación. La gestión sostenible del agua, incluida en el ODS 6 [8], requiere controlar el caudal de los ríos para garantizar su uso resposable. Sin embargo, el caudal no siempre se mide directamente en todo el curso del río, sino que se calcula a partir de datos como la velocidad del agua en diferentes puntos y las características de la sección transversal del río.

[8] Objetivos de Desarrollo Sostenible

Para calcular el caudal específico de agua que pasa por una sección transversal del río, se debe integrar la velocidad del agua a lo largo de la profundidad del río, obteniendo el volumen que pasa por cada metro de ancho del cauce. La fórmula para ello es la siguiente,

$$Q = \int_a^b v\left(x\right) dx$$

donde Q es el caudal específico, $v(x)$ es la velocidad del agua en el punto x de la seccion transversal y a y b son los límites de la sección transversal. En la siguiente tabla se recogen unos datos para calcular el valor aproximado de Q con alguna técnica que se mostrará en este capítulo.

Profundidad (m)	Velocidad (m/s)
0	0.0
1	1.2
2	2.1
3	2.4
4	2.0
5	1.5
6	0.0

Tabla 5.1. Velocidad del agua en función de la profundidad

La figura 5.1 muestra la velocidad del agua en función de la profundidad. El área bajo la curva es el valor a calcular.

Figura 5.1. Velocidad del agua en función de la profundidad

Si se utiliza por ejemplo la fórmulas de cuadratura compuesta de los trapecios, que se verá en este tema, se podría obtener un valor aproximado del caudal o volumen que fluye, Q, de la forma siguiente,

$$Q \approx \frac{h}{2}\left[v(x_0) + 2\sum_{i=1}^{5} v(x_i) + v(x_6)\right] \approx 9.2$$

donde:

- x_i: Profundidad i-ésima.
- h: Ancho del intervalo entre los puntos de profundidad ($h = x_{i+1} - x_i$). En los datos de la tabla $h = 1$.
- $v(x_i)$: Velocidad del agua en el punto de profundidad x_i.
- x_0 y x_6: Extremos de la profundidad medida.

5.2 Fórmulas de cuadratura interpolatorias

Para encontrar una aproximación de la integral definida de f en un intervalo $[a,b]$, se considerará la integración de un polinomio interpolador de f en distintos nodos del intervalo.

Así, si $p(x)$ es un polinomio interpolador de f en $n+1$ nodos,

$$\{x_i\}_{i=0}^{n} \subset [a,b] \qquad p\left(x_i\right) = f\left(x_i\right) \;\; i = 0, 1, ..., n$$

una aproximación del valor de la integral de f en $[a,b]$ será:

$$\int_a^b f\left(x\right) dx \approx \int_a^b p\left(x\right) dx$$

Considerando el polinomio de interpolación de Lagrange, definido de la forma siguiente:

$$p\left(x\right) = \sum_{i=0}^{n} f\left(x_i\right) l_i\left(x\right) \quad , \quad l_i\left(x\right) = \prod_{j=0 \;\; j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \quad 0 \leq i \leq n$$

se obtendría,

$$\int_a^b p\left(x\right) dx = \sum_{i=0}^{n} f\left(x_i\right) \int_a^b l_i\left(x\right) dx =$$

$$= \left(b-a\right) \sum_{i=0}^{n} f\left(x_i\right) \underbrace{\left(\frac{1}{b-a} \int_a^b l_i\left(x\right) dx \right)}_{w_i}$$

La aproximación será

$$\int_a^b f(x)\,dx \approx (b-a)\sum_{i=0}^{n} w_i f(x_i) \qquad siendo$$

$$w_i = \frac{1}{b-a}\int_a^b l_i(x)\,dx$$

Este tipo de cuadratura se dice que es una fórmula interpolatoria. Los puntos w_i se denominan pesos y los puntos x_i son los nodos de la fórmula de cuadratura.

Ejemplo 5.1. Se considera la función $f(x) = e^{-x^2}$, el intervalo $[0,1]$ y los 4 nodos siguientes $x_0 = 0, x_1 = 1/3, x_2 = 2/3$ y $x_3 = 1$. Calcula la fórmula de cuadratura.

Se tendrá

$$\int_0^1 e^{-x^2}dx \approx \omega_o e^0 + \omega_1 e^{-1/9} + \omega_2 e^{-4/9} + \omega_3 e^{-1}$$

donde

$$w_0 = \int_0^1 l_o(x)\,dx \qquad l_o = \frac{(x-1/3)(x-2/3)(x-1)}{(-1/3)(-2/3)(-1)}$$

$$w_1 = \int_0^1 l_1(x)\,dx \qquad l_1 = \frac{(x-0)(x-2/3)(x-1)}{1/3\cdot(-1/3)(-2/3)}$$

$$w_2 = \int_0^1 l_2(x)\,dx \qquad l_2 = \frac{(x-0)(x-1/3)(x-1)}{2/3\cdot(1/3)(-1/3)}$$

$$w_3 = \int_0^1 l_3(x)\,dx \qquad l_3 = \frac{(x-0)(x-1/3)(x-2/3)}{1\cdot(2/3)(1/3)}$$

Se obtendría

$$w_o = \frac{1}{8} \quad w_1 = \frac{3}{8} \quad w_2 = \frac{3}{8} \quad w_3 = \frac{1}{8}$$

y, por tanto,

$$\int_0^1 e^{-x^2}\,dx \approx \frac{1}{8}e^0 + \frac{3}{8}e^{-1/9} + \frac{3}{8}e^{-4/9} + \frac{1}{8}e^{-1}$$

Ejemplo 5.2. Demostrar que la fórmula $\int_0^2 f(x)\,dx \approx 3f(0) + f(2)$ no es de tipo interpolatorio.

En efecto, la fórmula dada se podría escribir

$$\int_0^2 f(x)\,dx \approx 3f(0) + f(2) = (2-0)\left[\frac{3}{2}f(0) + \frac{1}{2}f(2)\right]$$

siendo los nodos $x_0 = 0$ y $x_1 = 2$ y los pesos $\omega_0 = 3/2$ y $\omega_1 = 1/2$.
Si fuera de tipo interpolatorio, se tendría que cumplir $\omega_0 = 3/2$ pero

$$\omega_0 = \frac{1}{2}\int_0^2 l_o(x)\,dx = \frac{1}{2}\int_0^2 \frac{x-x_1}{x_o-x_1}\,dx = \frac{1}{2}\int_0^2 \frac{x-2}{0-2}\,dx = \frac{1}{2} \neq \frac{3}{2}$$

Por lo tanto, la fórmula no es de tipo interpolatorio.

De cara a obtener los pesos, resulta de utilidad el resultado siguiente.

Los pesos son independientes del intervalo siempre que se elijan adecuadamente los nodos, esto es, estén relacionados por la transformación de $[a,b]$ en $[c,d]$ dada de la siguiente forma,

$$\begin{aligned} [a,b] &\to [c,d] \\ x &\to t = c + \frac{d-c}{b-a}(x-a) \end{aligned}$$

Este resultado significa que si se cambia de intervalo $[a,b]$ a $[c,d]$ y se consideran los nodos x_i en $[a,b]$ y t_i en $[c,d]$ relacionados de la forma siguiente:

$$t_i = c + \frac{d-c}{b-a}(x_i - a)$$

se tendrá que los pesos de la fórmula interpolatoria en ambos casos coinciden.

En efecto,

$$\widehat{\omega_i} = \frac{1}{d-c}\int_c^d \widehat{l_i}(t)\,dt = \frac{1}{b-a}\int_a^b l_i(x)\,dx = \omega_i$$

donde

$$\widehat{l_i}(t) = \prod_{j=0\ j\neq i}^{n} \frac{t-t_j}{t_i-t_j} \quad 0 \le i \le n$$

$$l_i(x) = \prod_{j=0\ j\neq i}^{n} \frac{x-x_j}{x_i-x_j} \quad 0 \le i \le n$$

En el siguiente interactivo se muestra dos intervalos, $[a, b]$ y $[c, d]$ y cómo está vinculado el punto x en $[a, b]$, en rojo, con el correspondiente punto t en $[c, d]$ que se muestra en naranja considerando la correspondencia anterior.

Interactivo 5.2. Mapeo entre dos intervalos $[a, b]$ y $[c, d]$

Ejemplo 5.3. Teniendo en cuenta el último resultado y también el ejemplo 5.1, escribir la fórmula de cuadratura en cualquier intervalo $[a, b]$ considerando cuatro nodos equiespaciados siendo el primero a y el último b.

Por el resultado anterior, los pesos ω_o, ω_1, ω_2, ω_3 coinciden con los obtenidos para el intervalo $[0, 1]$ considerando $0, 1/3, 2/3$ y 1. Teniendo en cuenta el ejemplo 5.1, se ha visto que estos valores son

$$w_o = \frac{1}{8} \quad w_1 = \frac{3}{8} \quad w_2 = \frac{3}{8} \quad w_3 = \frac{3}{8}$$

Por lo tanto, la fórmula de cuadratura para una función f y un intervalo $[a, b]$ tomando cuatro nodos equiespaciados e incluyendo los extremos es

$$\int_a^b f(x)\,dx \approx (b-a)\left[\tfrac{1}{8}f(a) + \tfrac{3}{8}f\left(a+\tfrac{b-a}{3}\right) + \right.$$
$$\left. + \tfrac{3}{8}f\left(a+\tfrac{2(b-a)}{3}\right) + \tfrac{1}{8}f(b)\right]$$

Se mostrará más adelante, que esta fórmula de cuadratura, recibe el nombre de fórmula 3/8 de Newton-Cotes.

Posteriormente, se verá que el cálculo de los pesos ω_i se puede obtener de otra forma más sencilla sin requerir integrar los polinomios base de Lagrange.

5.2.1 Grado de exactitud

Para calcular la calidad de una fórmula de cuadratura se define su grado de precisión o exactitud.

Una fórmula de cuadratura tiene grado de exactitud $n > 0$ si calcula exactamente la integral para cada polinomio de grado menor o igual que n, pero no es exacta para al menos un polinomio de grado $n + 1$.

Esto significa que para cada polinomio $q(x)$ de grado menor o igual que n, el valor de la integral y el valor que devuelve la fórmula de cuadratura coinciden, esto es, se cumple

$$\int_a^b q(x)\,dx = (b-a)\sum_{i=0}^{n} \omega_i\, q(x_i)$$

Una fórmula de cuadratura de tipo interpolatorio

$$\int_a^b f(x)\,dx \approx (b-a)\sum_{i=0}^{n} w_i f(x_i)$$

tiene grado de exactitud mayor o igual a n.

Ejemplo 5.4. Comprueba que la fórmula

$$\int_0^4 f(x)\,dx \approx 2f(1) + 2f(3)$$

tiene orden de exactitud o precisión 1.

Se verá que es exacta de grado $n = 1$. Se puede comprobar que es exacta para $f(x) = 1$,

$$\int_0^4 1\,dx = 4 = 2 + 2$$

Es exacta también para $f(x) = x$

$$\int_0^4 x\,dx = \frac{16}{2} = 2 + 2 \cdot 3$$

Por lo tanto, es exacta para todo polinomio de grado menor o igual que 1. Sin embargo, deja de ser exacta para $f(x) = x^2$

$$\int_0^4 x^2\,dx = \frac{64}{3} \neq 2 \cdot 1^2 + 2 \cdot 3^2 = 20$$

Por lo que su grado de precisión es exactamente uno.

5.3 Fórmulas de cuadratura de Newton-Cotes

Se analizará en este apartado las fórmulas de cuadratura de Newton-Cotes que son de tipo interpolatorio donde los nodos $\{x_0, x_1, ..., x_n\}$ son equidistantes. Se considerarán únicamente las fórmulas cerradas de Newton-Cotes que son aquellas en las que los nodos incluyen los extremos del intervalo donde integrar, esto es, $[a, b] = [x_0, x_n]$.

Se llama fórmula de Newton-Cotes, también fórmula cerrada de Newton-Cotes, a aquella fórmula de cuadratura numérica de tipo interpolatorio, donde los nodos vienen dados por la expresión

$$x_j = a + jh \ , \ 0 \leq j \leq n \quad , \quad h = \frac{b-a}{n}$$

Es decir, se tienen $n + 1$ nodos equidistantes

$$a = x_o < x_1 < x_2 < ... < x_n = b$$

$$h = x_j - x_{j-1} \quad j = 1, ..., n$$

Interactivo 5.3. Intervalo $[a, b]$ con $n + 1$ nodos equidistantes

Dados los nodos x_i para $i = 0, 1, ..., n$, y teniendo en cuenta la exactitud de la fórmula para polinomios de grado menor o igual que n, los pesos de las fórmulas de Newton-Cotes se pueden determinar resolviendo el siguiente sistema de $n+1$ ecuaciones con $n+1$ incógnitas:

$$\int_a^b x^k dx = (b-a) \sum_{j=0}^{n} \omega_j x_j^k \qquad k = 0, 1, ..., n$$

Este sistema tiene una única solución ya que la matriz de coeficientes es de Vandermonde.

Como este método puede ser numéricamente inestable, no es aconsejable para valores de n grandes.

Ejemplo 5.5. Deduce la fórmula de cuadratura cerrada de Newton-Cotes en $[a, b]$ con cuatro nodos. Escribe el sistema de ecuaciones que permitiría calcular los pesos. Compara con el resultado del ejemplo 5.3.

El sistema a resolver es el siguiente,

$$f(x) = 1 \qquad \frac{1}{b-a} \int_a^b dx = \omega_o + \omega_1 + \omega_2 + \omega_3$$

$$f(x) = x \qquad \frac{1}{b-a} \int_a^b x dx = \omega_o x_o + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3$$

$$f(x) = x^2 \qquad \frac{1}{b-a} \int_a^b x^2 dx = \omega_o x_0^2 + \omega_1 x_1^2 + \omega_2 x_2^2 + \omega_3 x_3^2$$

$$f(x) = x^3 \qquad \frac{1}{b-a} \int_a^b x^3 dx = \omega_o x_0^3 + \omega_1 x_1^3 + \omega_2 x_2^3 + \omega_3 x_3^3$$

En forma matricial, el sistema se puede escribir,

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ x_o & x_1 & x_2 & x_3 \\ x_0^2 & x_1^2 & x_2^2 & x_3^2 \\ x_0^3 & x_1^3 & x_2^3 & x_3^3 \end{pmatrix} \begin{pmatrix} \omega_o \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} = \begin{pmatrix} 1 \\ (b-a)/2 \\ (b-a)^2/3 \\ (b-a)^3/4 \end{pmatrix}$$

Para obtener el valor de los pesos se puede considerar por ejemplo el intervalo $[0, 3]$ y los nodos $x_0 = 0, x_1 = 1, x_2 = 2$ y $x_3 = 3$ ya que, como se ha visto anteriormente, el valor de los pesos es el mismo siempre que en un intervalo se consideren como se indicaba para $[a, b]$.

Resolviendo el sistema, se obtiene $\omega_0 = 1/8$, $\omega_1 = 3/8$, $\omega_2 = 3/8$ y $\omega_3 = 1/8$. El siguiente código Matlab/Octave permite resolver el sistema.

```
x=[0 1 2 3];
format rational
A=x.^[0;1;2;3];
%También x.^([0:3]')
b=[1;3/2;3;27/4];
pesos=A\b
```

5.3.1 Cálculo de las fórmulas de cuadratura Newton-Cotes cerradas

Se muestra a continuación las primeras fórmulas de Newton-Cotes tomando diferentes números de nodos x_0, x_1, ..., x_n. Para ello, se utilizará que el cálculo de los pesos no depende del intervalo considerado siempre que se tomen los nodos equidistantes en dicho intervalo y se consideren los extremos del intervalo de integración.

5.3.1.1 Primera fórmula de Newton-Cotes (n=1)

En este caso se considerarán dos nodos en el intervalo $[a, b]$, esto es, $x_0 = a$ y $x_1 = b$. La fórmula de cuadratura será de la forma,

$$\int_a^b f(x)\,dx \approx (b-a)\left[\omega_o f(a) + \omega_1 f(b)\right]$$

Para calcular los nodos se puede tomar por ejemplo $[a, b] = [0, 1]$ y aplicar que la fórmula de cuadratura es exacta al menos de grado $n = 1$, es decir, es exacta para $f(x) = 1$ y para $f(x) = x$. Resolviendo el sistema 2x2, se obtiene $\omega_o = \omega_1 = \frac{1}{2}$.

La fórmula de cuadratura para $n = 1$ es

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\left[f(a) + f(b)\right]$$

Esta fórmula de Newton-Cotes de dos puntos se llama fórmula del trapecio.

Observad que el polinomio interpolador que pasa por dos puntos es la recta que pasa por dichos puntos, de ahí el nombre de la fórmula de cuadratura.

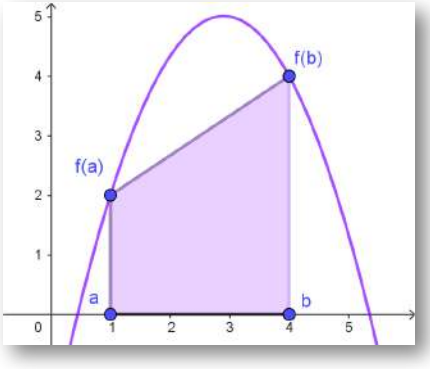
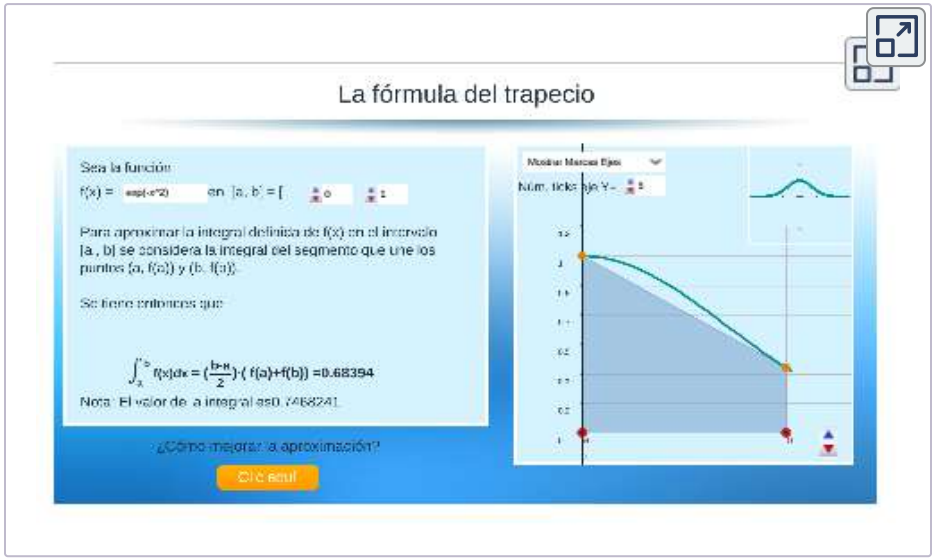


Figura 5.2. Regla del trapecio

En la siguiente escena interactiva, se muestra la aproximación que da la regla de los trapecios, según la función y el intervalo que se introduzca como datos.



Interactivo 5.4. Método del trapecio. Escena tomada de la unidad Integración Numérica. Proyecto Un100. Autora: Elena E. Álvarez

5.3.1.2 Segunda fórmula de Newton-Cotes (n=2)

En este caso, el número de nodos equidistantes en $[a, b]$ es 3, y sérán los puntos siguientes: $x_o = a$, $x_1 = \frac{a+b}{2}$ y $x_2 = b$ y la fórmula de cuadratura es

$$\int_a^b f(x)\,dx \approx (b-a)\left[\omega_o f(a) + \omega_1 f\left(\frac{a+b}{2}\right) + \omega_2 f(b)\right]$$

Para determinar los nodos, se puede considerar el intervalo $[a, b] = [-1, 1]$ y resolver el sistema de 3 ecuaciones con tres incógnitas que resulta de aplicar que la fórmula de cuadratura es exacta para $f(x) = 1$, $f(x) = x$ y $f(x) = x^2$.

Los valores de los pesos obtenidos serían $\omega_o = \omega_2 = \frac{1}{6}, \omega_1 = \frac{2}{3}$.

La fórmula de cuadratura para $n = 2$ es

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Esta regla se llama regla de Simpson.

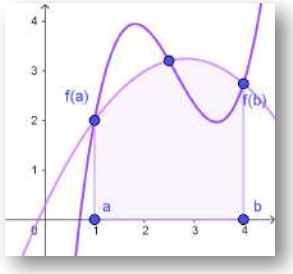
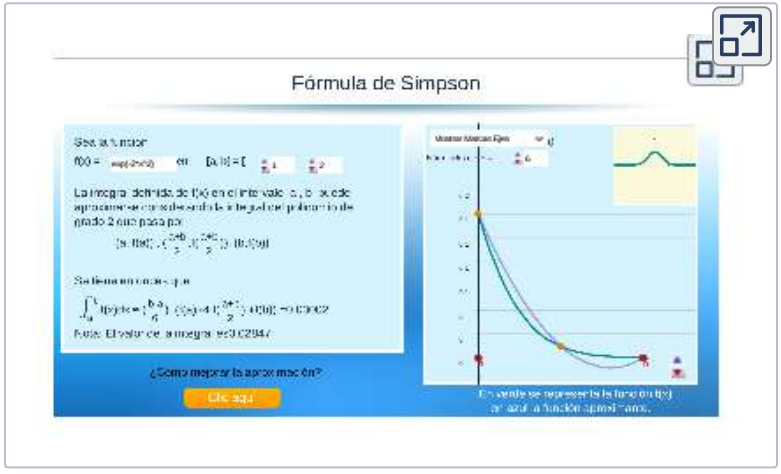


Figura 5.3. Regla de Simpson

En la siguiente escena interactiva se puede analizar las aproximaciones de la cuadratura introduciendo la función y el intervalo. Se puede ver que la aproximación es mejor cuanto menor es la longitud del intervalo.



Interactivo 5.5. Regla de Simpson. Escena tomada de la unidad Integración Numérica. Proyecto Un100. Autora: Elena E. Álvarez

5.3.1.3 Tercera fórmula de Newton-Cotes (n=3)

En este caso el número de nodos equidistantes que se considera en el intervalo $[a,b]$ es 4: $x_0 = a, x_1 = a + \frac{b-a}{3}, x_2 = a + 2\frac{b-a}{3}$ y $x_3 = b$ y la fórmula de cuadratura es

$$\int_a^b f(x)\,dx \approx (b-a)\left[\omega_0 f(a) + \omega_1 f\left(\frac{2a+b}{3}\right) + \right.$$

$$\left. + \omega_2 f\left(\frac{a+2b}{3}\right) + \omega_3 f(b)\right]$$

Para determinar los nodos se considera el intervalo $[0,3]$ y se resuelve el sistema de 4 ecuaciones con 4 incógnitas asociado, obteniendo $\omega_o = \omega_3 = \frac{1}{8}, \omega_1 = \omega_2 = \frac{3}{8}$.

La fórmula de cuadratura para $n = 3$

$$\int_a^b f(x)\,dx \approx$$

$$\approx \frac{(b-a)}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right]$$

Esta fórmula se llama regla de los tres octavos.

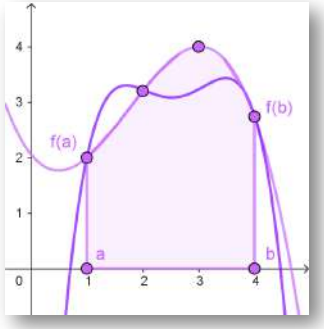
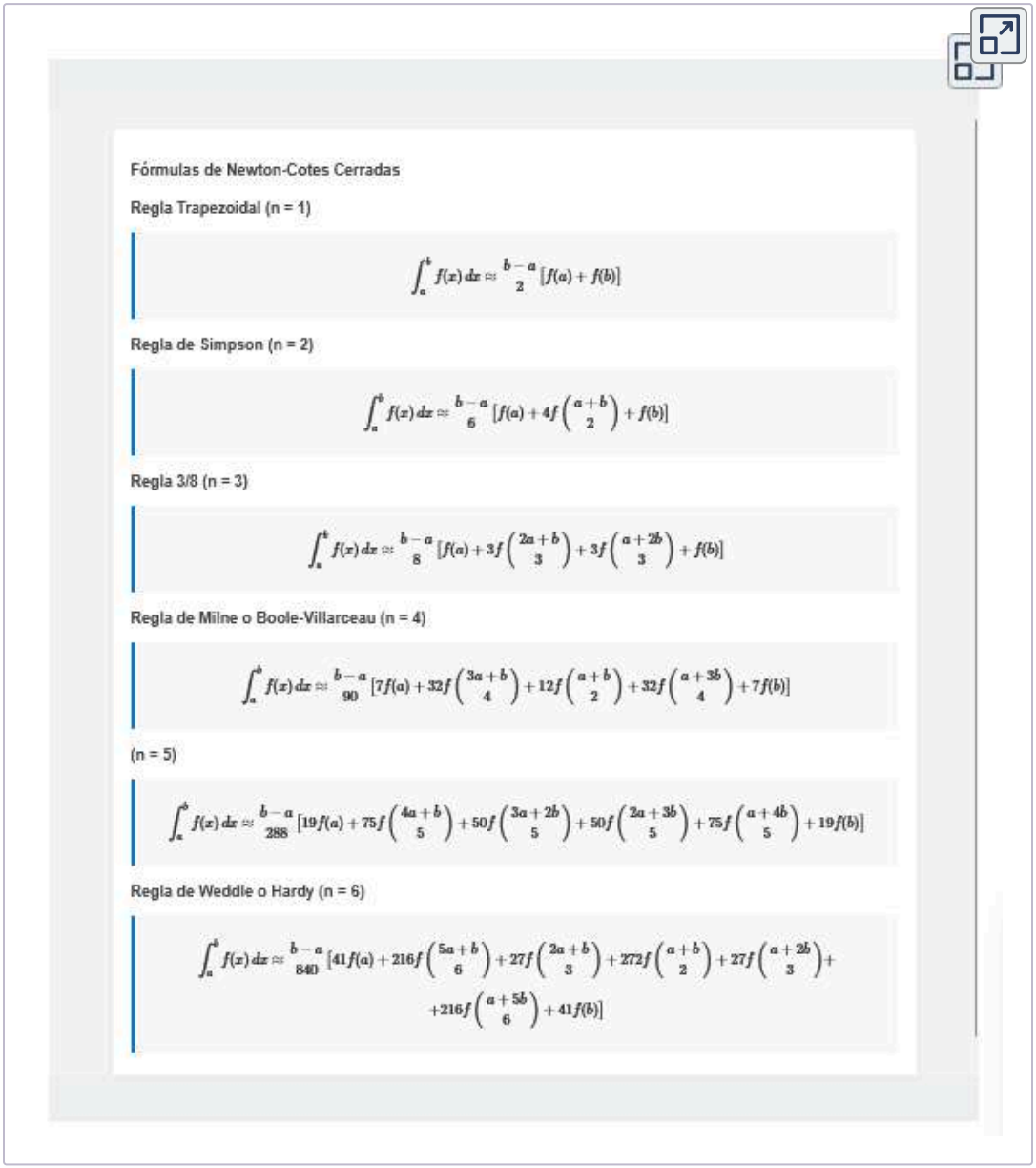


Figura 5.4. Regla de los tres octvavos

Se puede observar que los pesos en estas fórmulas de cuadratura son simétricos, es decir, el primero es igual al último, el segundo es igual al penúltimo, etc.

De forma análoga, se pueden obtener las fórmulas de Newton-Cotes cerradas para n desde 1 hasta 6 que se muestran a continuación.



Fórmulas de Newton-Cotes Cerradas

Regla Trapezoidal (n = 1)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}[f(a)+f(b)]$$

Regla de Simpson (n = 2)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

Regla 3/8 (n = 3)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{8}\left[f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right]$$

Regla de Milne o Boole-Villarceau (n = 4)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{90}\left[7f(a)+32f\left(\frac{3a+b}{4}\right)+12f\left(\frac{a+b}{2}\right)+32f\left(\frac{a+3b}{4}\right)+7f(b)\right]$$

(n = 5)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{288}\left[19f(a)+75f\left(\frac{4a+b}{5}\right)+50f\left(\frac{3a+2b}{5}\right)+50f\left(\frac{2a+3b}{5}\right)+75f\left(\frac{a+4b}{5}\right)+19f(b)\right]$$

Regla de Weddle o Hardy (n = 6)

$$\int_a^b f(x)\,dx \approx \frac{b-a}{840}\left[41f(a)+216f\left(\frac{5a+b}{6}\right)+27f\left(\frac{2a+b}{3}\right)+272f\left(\frac{a+b}{2}\right)+27f\left(\frac{a+2b}{3}\right)+\right.$$
$$\left.+216f\left(\frac{a+5b}{6}\right)+41f(b)\right]$$

Ejemplo 5.6. Obtén con Matlab la aproximación de la integral $\int_0^{\pi} e^{senx\ \cos x} dx$ utilizando la regla de los tres octavos ($n = 3$).

Considerando $f(x) = e^{senx\ \cos x}$, el valor que aproxima a la integral es

$$\int_0^{\pi} e^{senx \cos x} dx \approx \frac{\pi}{8}\left(f(0) + 3f\left(\frac{\pi}{3}\right) + 3f\left(\frac{2\pi}{3}\right) + f(\pi)\right)$$

```
format long
f=@(x) exp(sin(x).*cos(x));
a=0;b=pi;
%Regla tres octavos
h=(b-a)/3;nodos=a:h:b
coef=[1 3 3 1];
valor=sum(f(nodos).*coef)*(b-a)/8  %3.365958987799939
```

Ejemplo 5.7. Escribe una función Matlab/Octave que calcule, dada una función f y el extremo inferior y superior del intervalo, la aproximación mediante la fórmula de los tres octavos. Calcula con ella la aproximación de la integral del ejemplo anterior.

```
function valor=TresOctavos(f,a,b)
    h=(b-a)/3;
    nodos=a:h:b
    coef=[1 3 3 1];
    valor=sum(f(nodos).*coef)*(b-a)/8;
end
```

Para llamar a la función, se utilizará el siguiente código,

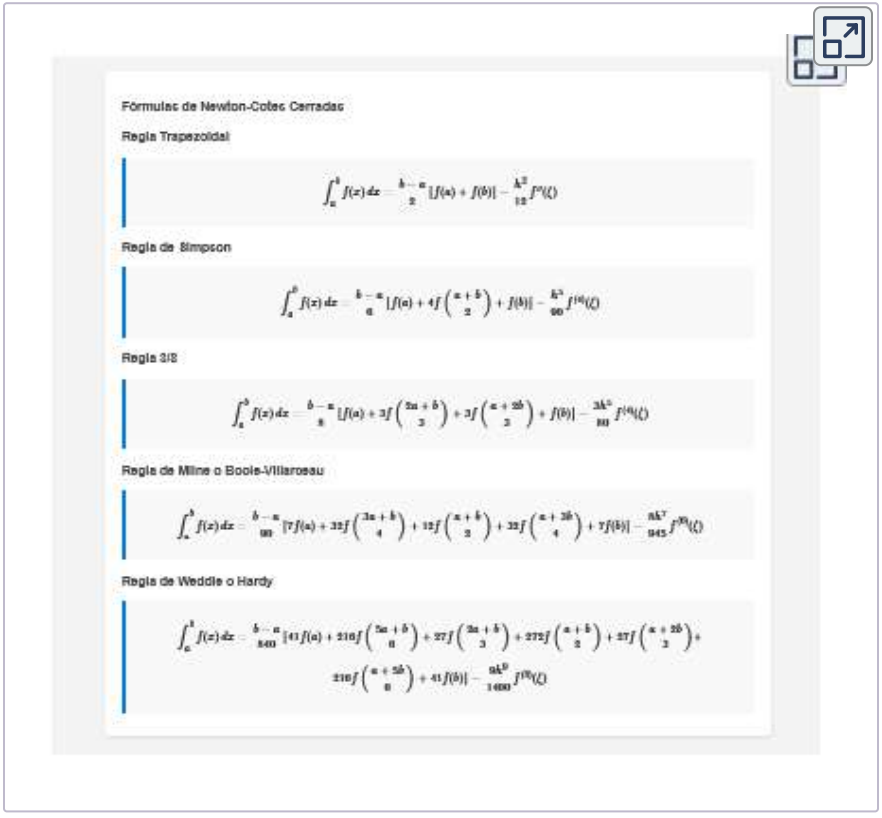
```
fun=@(x) exp(sin(x).*cos(x));
aprox=TresOctavos(fun,0,pi)
```

5.3.2 Error en la fórmula de cuadratura de Newton-Cotes.

En este apartado se analizará el error de la aproximación, es decir, la diferencia entre el valor exacto dado por la integral y el valor aproximado que se obtiene con la fórmula de Newton-Cotes

$$E = \int_a^b f(x)\,dx - (b-a)\sum_{j=0}^{n} \omega_j f(x_j)$$

A continuación se presenta la expresión del término del resto para algunas de las fórmulas cerradas de Newton-Cotes vistas en el apartado anterior, considerando $h = \frac{b-a}{n}$. Algunas de estas expresiones del resto se demostrarán en el apartado siguiente.



Se puede observar que el error que se comete en la aproximación para las distintas fórmulas de cuadratura, depende de la separación entre los nodos, h. Además, como la potencia a la que está elevado h es mayor cuanto mayor es el número de nodos, al disminuir el valor de h es esperable que el error sea menor.

Además, teniendo en cuenta la expresión del error, se puede comprobar que el grado de precisión de las fórmulas cerradas de Newton-Cotes es $n+1$ cuando n es par y es n cuando n es impar.

Ejemplo 5.8. Dada la integral $\int_0^{\pi/2} e^{sen(x)}dx$, calcula el valor aproximando utilizando la regla del trapecio. Obtén también una cota del error de dicha aproximación.

En este caso, $f(x)=e^{sen(x)}$, $a=0, b=\pi/2$ y la aproximación utilizando la fórmula de los trapecios será

$$\int_0^{\pi/2} e^{sen(x)}dx \approx \frac{\pi}{4}\left[f\left(0\right)+f\left(\pi/2\right)\right]=\frac{\pi}{4}\left[e^{sen(0)}+e^{sen(\pi/2)}\right]\approx 2.9203$$

Se cumple,

$$E=\int_0^{\pi/2} e^{sen(x)}dx-\frac{\pi}{4}\left[f\left(0\right)+f\left(\pi/2\right)\right]=-\frac{\pi^3}{8\cdot 12}f''\left(\xi\right)\quad \xi\in[0,1]$$

Las derivadas de f son

$$f'\left(x\right)=\cos\left(x\right)e^{sen(x)}$$

$$f''\left(x\right)=\cos^2\left(x\right)e^{sen(x)}-sen\left(x\right)e^{sen(x)}$$

Por lo tanto, una cota sencilla de calcular de la derivada segunda podría ser: $|f''(x)| \leq 2e$. Utilizando la expresión del error, una cota de la aproximación será entonces,

$$|E| = \left| -\frac{\pi^3}{96} f''(\xi) \right| \leq \frac{\pi^3 e}{48} \approx 0.642$$

La cota obtenida es muy grande, por lo que no permite determinar la bondad de la aproximación. Matlab/Octave da como aproximación utilizando el comando `integral` el valor 3.104379017855555.

Ejemplo 5.9. Encuentra una cota del error de la aproximación de la integral $\int_0^1 \log(1+e^x)\,dx$ con la regla de los trapecios.

Se tiene $f(x) = \log(1+e^x)$, $a = 0$ y $b = 1$. Dado que

$$f'(x) = \frac{e^x}{1+e^x}$$

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

Teniendo en cuenta que la derivada segunda tiene un máximo en el punto 0, una cota del error podría ser

$$\left| -\frac{h^3}{12} f''(\xi) \right| \leq \frac{1}{12 \cdot 4} \qquad \xi \in [0,1]$$

Este valor es aproximadamente 0.0208. El valor de la aproximación sería:

$$\int_0^1 \log(1+e^x)\,dx \approx \frac{1}{2}(f(0) + f(1)) = \frac{\log 2 + \log(1+e)}{2} \approx 1.0032$$

El valor que devuelve Matlab/Octave como aproximación utilizando el comando `integral`es 0.983819037020661.

5.3.3 Deducción de la expresión del error de la fórmula de cuadratura

El siguiente resultado da una expresión del error asociado a una fórmula de cuadratura de Newton-Cotes cerrada.

Se considera $h = \frac{b-a}{n}$ y $\pi(t) = t(t-1)\ldots(t-n)$.

- Si n es impar y la función f es $n+1$ veces derivable con $f^{(n+1}$ continua, entonces existe $\xi \in [a,b]$ de forma que el error de la fórmula viene dada por

$$E = \frac{h^{n+2}}{(n+1)!} f^{(n+1}(\xi) \int_0^n \pi(t)\, dt$$

- Si n es par y la función f es $n+2$ veces derivable con $f^{(n+2}$ continua, entonces existe $\xi \in [a,b]$ de forma que el error de la fórmula viene dada por

$$E = \frac{h^{n+3}}{(n+2)!} f^{(n+2}(\xi) \int_0^n t\pi(t)\, dt$$

Según este teorema, si n es par, el grado de precisión de la fórmula de Newton-Cotes es $n+1$. Cuando n es impar, el grado de precisión es n.

La expresión del error se puede escribir de la siguiente manera;

$$E = \begin{cases} C_n h^{n+2} f^{(n+1}(\xi) & \text{si n impar} \\ C_n h^{n+3} f^{(n+2}(\xi) & \text{si n par} \end{cases}$$

donde la constante C_n tiene el siguiente valor:

$$C_n = \begin{cases} \dfrac{1}{(n+1)!} \displaystyle\int_0^n t\,(t-1)\,(t-2)\ldots(t-n)\,dt & \text{si n impar} \\ \dfrac{1}{(n+2)!} \displaystyle\int_0^n t^2\,(t-1)\,(t-2)\ldots(t-n)\,dt & \text{si n par} \end{cases}$$

Por lo tanto, el error de la fórmula de cuadratura de Newton Cotes se puede escribir como $E = C_n h^{m+1} f^{(m}(\xi)$ siendo

- $m = \begin{cases} n+1 \; si \; n \; impar \\ n+2 \; si \; n \; par \end{cases}$
- C_n una constante dada por la expresión anterior
- $\xi \in [a,b]$.

Se determinará seguidamente el valor de las constantes C_n para las primeras fórmulas de Newton-Cotes.

5.3.3.1 Error en la fórmula del trapecio ($n = 1$)

Se considera la fórmula de Newton-Cotes para $n = 1$, que es impar. Se toma $m = n + 1 = 2$. Se tiene que

$$E = \int_a^b f\,(x)\,dx - \frac{b-a}{2}\,(f\,(a) + f\,(b))$$

$$E = C_1 h^3 f''\,(\xi)$$

Para determinar C_1, se considera la función $f(x) = (x-a)^m = (x-a)^2$.

Para esta función se tiene que por un lado se cumple

$$\int_a^b f(x)dx = \int_a^b (x-a)^2 dx = \frac{1}{3}(b-a)^3 = \frac{h^3}{3}$$

y por otro

$$\frac{b-a}{2}\left(f(a)+f(b)\right) = \frac{b-a}{2}\left(0+(b-a)^2\right) = \frac{(b-a)^3}{2} = \frac{h^3}{2}$$

Por lo tanto,

$$C_1 h^3 f''(\xi) = \frac{h^3}{3} - \frac{h^3}{2}$$

como $f''(\xi) = 2$, se tiene,

$$2C_1 = \frac{1}{3} - \frac{1}{2} = -\frac{1}{6}$$

$$C_1 = -\frac{1}{12}$$

Se cumple entonces

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\left[f(a)+f(b)\right] - \frac{1}{12}h^3 f''(\xi)$$

para algún valor de ξ en $[a,b]$.

5.3.3.2 Error en la fórmula de Simpson

Se considera la fórmula de Newton-Cotes para $n = 2$ que es par, se toma $m = n + 2 = 4$. Se tiene que

$$E = \int_a^b f(x)\,dx - \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$
$$E = C_2 h^5 f''(\xi)$$

Para determinar C_2 se considera la función $f(x) = (x-a)^m = (x-a)^4$. Se tiene que por un lado se cumple

$$\int_a^b (x-a)^4 dx = \frac{1}{5}(b-a)^5 = \frac{(2h)^5}{5}$$

y por otro

$$\frac{b-a}{6}\left[0 + 4 \cdot h^4 + (2h)^4\right] = \frac{2h^5}{6} \cdot 20$$

Por lo tanto,

$$C_2 h^5 f^{iv}(\xi) = \frac{2^5 h^5}{5} - \frac{20h^5}{3}$$

es decir,

$$4!C_2 = \frac{2^5}{5} - \frac{20}{3} = -\frac{4}{15} \quad \Rightarrow \quad C_2 = -\frac{1}{90}$$

Se cumple entonces

$$\int_a^b f(x)\,dx = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] + E$$

$$E = -\frac{1}{90}h^5 f^{(iv)}(\xi) \text{ para algún valor de } \xi \in [a,b]$$

5.3.3.3 Error en la fórmula de cuadratura 3/8

Se considera la fórmula de Newton-Cotes para $n = 3$, que es impar y se toma $m = n + 1 = 4$. Se tiene que

$$E = \int_a^b f(x)\,dx -$$

$$\frac{(b-a)}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right]$$

$$E = C_3 h^5 f^{(4}(\xi)$$

Para determinar C_3 se considera la función $f(x) = (x-a)^m = (x-a)^4$. Se cumplirá

$$\int_a^b (x-a)^4 dx = \frac{1}{5}(b-a)^5 = \frac{(3h)^5}{5}$$

$$\frac{3h}{8}\left[0 + 3 \cdot h^4 + 3 \cdot (2h)^4 + (3h)^4\right] = \frac{99h^5}{2}$$

Por lo tanto,

$$E = C_3 h^5 f^{(4}(\xi) = \frac{-9}{10}h^5$$

$$4!C_3 = \frac{-9}{10} \quad \Rightarrow \quad C_3 = -\frac{3}{80}$$

Se cumple

$$\int_a^b f(x)\,dx =$$

$$\frac{(b-a)}{8}\left[f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right]+E$$

$$E=-\frac{3}{80}h^5 f^{(4}(\xi) \text{ para algún valor de } \xi\in[a,b]$$

5.4 Fórmulas compuestas

Cuando la distancia entre los nodos aumenta, el error que se comete, de forma general, también aumenta considerablemente. Para conseguir una mejor aproximación, se puede dividir el intervalo $[a,b]$ en N subintervalos y utilizar una fórmula de cuadratura en cada uno de los subintervalos. Por linealidad de la integral, se tendrá

$$a=\alpha_o<\alpha_1<\ldots<\alpha_N=b \quad , \quad \int_a^b f(x)\,dx=\sum_{i=1}^{N}\int_{\alpha_{i-1}}^{\alpha_i} f(x)\,dx$$

La aproximación de las fórmulas compuestas consiste en aplicar una fórmula de cuadratura en cada subintervalo $[\alpha_{i-1},\alpha_i]$.

5.4.1 Regla del trapecio compuesta

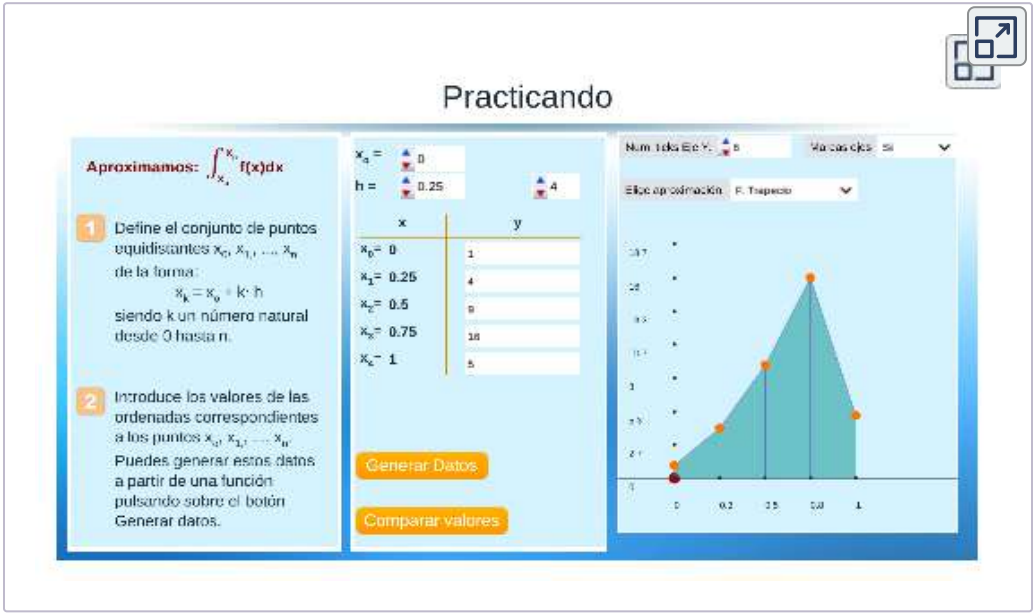
En este caso, la fórmula quedaría de la forma siguiente, teniendo en cuenta que la longitud de cada subintervalo es $\frac{b-a}{N}$.

$$\int_a^b f(x)\,dx\approx\frac{(b-a)}{2N}\sum_{i=1}^{N}\left[f(\alpha_{i-1})+f(\alpha_i)\right]$$

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{2N}\left[f(\alpha_0) + 2f(\alpha_1) + \ldots + 2f(\alpha_{N-1}) + f(\alpha_N)\right]$$

La cuadratura del trapecio compuesta consiste en aproximar una integral en $[a, b]$ utilizando N trapecios. En primer lugar, se divide el intervalo en N subintervalos de longitud $h = \frac{b-a}{N}$. Despues de aplicar en cada uno de ellos la fórmula de cuadratura de los trapecios se obtiene

$$\int_a^b fx)dx \approx \frac{h}{2}\left[f(a) + 2f(a+h) + 2f(a+2h) + \ldots + f(b)\right]$$



Interactivo 5.6. Fórmulas compuestas. Escena tomada de Integración Numérica. Proyecto Un100.. Autora: Elena E. Álvarez

Ejemplo 5.10. Utilizando la regla del trapecio compuesta con $N = 5$, aproxima el valor de la siguiente integral $\int_1^{\pi/2} \sqrt{x}\log(\sin(x))\,dx$.

Se define una función que calcule la regla de los trapecios compuesta.

```
function valor=TrapecioCompuesta1(f,a,b,N)
%Aproxima la integral de f en el intervalo [a,b]
h = (b-a)/N;x=linspace(a,b,N+1);I = 0;
for k = [1:N]
    I = I + f(x(k))+f(x(k+1));
end
valor=I*h/2;
end
```

También se podría escribir esta función de la forma siguiente.

```
function valor = TrapecioCompuesta(f, a, b, N)
%Aproxima la integral de f en el intervalo [a,b]
%usando la regla compuesta de los trapecios
 h = (b-a)/N;x=linspace(a,b,N+1); %Tambien: x=a:h:b;
 valor = h/2*(f(a)+f(b))+h*sum(f(x(2:end-1)));
end
```

Para utilizar la función y calcular de forma aproximada la integral pedida:

```
f=@(x) sqrt(x).*log(sin(x))
%Regla compuesta de los trapecios con N=5
TrapecioCompuesta(f,1,pi/2,5)
%-0.034825244439641
%Con comando integral Matlab: -0.034221721637133
```

Una cota del error en la aproximación utilizando la regla compuesta del trapecio considerando N subintervalos será

$$\frac{M \cdot N}{12}\left(\frac{b-a}{N}\right)^3 = \frac{M(b-a)^3}{12N^2} \quad con\ M = \max_{c\in(a,b)} |f''(c)|$$

Ejemplo 5.11. Considera la integral $\int_0^{\pi} e^{sen(x)}dx$ usando la regla del trapecio compuesta considerando $N = 8$ intervalos. Calcula el valor aproximado y una cota del error de dicha aproximación.

Una cota del error es:

$$\frac{M(b-a)^3}{12N^2} = \frac{M\,\pi^3}{12 \cdot 8^2} \quad siendo\, M = \max_{c\in[0,\,\pi]} |\, f''(c)\,|$$

Teniendo en cuenta que,

$$f'(x) = -e^{senx}\cos x$$

$$f''(x) = e^{senx}\cos^2 x - e^{senx}senx$$

aplicando la desigualdad triangular y que el seno y coseno toman como máximo el valor 1, se tiene $M \leq 2e$. Una cota del error de aproximación será entonces

$$\frac{\pi^3}{12 \cdot 8^2} 2e \approx 0.2194$$

El valor aproximado de la integral sería:

$$\int_0^{\pi} e^{senx}dx \approx \frac{\pi}{16}\left[f(0) + 2f(h) + \ldots + 2f(\pi - h) + f(\pi)\right]$$

tomando $h = \pi/8$. Para realizar el cálculo, el código Matlab/Octave a utilizar es el siguiente.

```
f=@(x) exp(sin(x));
%Regla compuesta de los trapecios con N=8
valor=TrapecioCompuesta(f,0,pi,8)
%6.183057933280413
%Con comando integral Matlab: 6.208758035711110
```

Ejemplo 5.12. Encuentra el número de subintervalos, N, para que el error de cuadratura con la regla del trapecio compuesta en el caso de las siguientes integrales sea menor que 10^{-4}.

$$(a) \quad \int_0^2 e^{x^2} dx \qquad (b) \quad \int_0^1 \cos\left(\pi x^2\right) dx$$

Apartado a) Si se llama M a la cota de la derivada segunda de $f(x) = e^{x^2}$ en el intervalo $[0, 2]$, lo que nos piden es encontrar N de forma que

$$\frac{(b-a)^3}{12N^2} M \leq 10^{-4}$$

Esto es

$$\frac{(b-a)^3}{12 \cdot 10^{-4}} M \leq N^2 \Leftrightarrow N \geq \sqrt{\frac{(b-a)^3}{12 \cdot 10^{-4}} M}$$

Se tiene $a = 0, b = 2$ y $M = 18e^4$ ya que

$$f'(x) = 2xe^{x^2}$$
$$f''(x) = 2e^{x^2} + 4x^2 e^{x^2} = \left(2 + 4x^2\right) e^{x^2}$$

Para conseguir el error pedido, bastaría tomar tomar $N = 2560$ y el valor de la integral sería:

```
f=@(x) exp(x.^2);
%Regla compuesta de los trapecios con N=2560
valor=TrapecioCompuesta(f,0,2,2560)
%16.452638873526805
%Con comando integral Matlab: 16.452627765507231
```

Apartado b). Si se llama M a la cota de la derivada segunda de $f(x) = cos(\pi x^2)$ en el intervalo $[0, 1]$, lo que nos piden es encontrar N de forma que

$$\frac{(b-a)^3}{12N^2} M \leq 10^{-4}$$

Se tiene $a = 0, b = 1$ y $M = 2\pi + 4\pi^2$ ya que

$$f'(x) = sen\left(\pi x^2\right) 2\pi x$$
$$f''(x) = -\cos\left(\pi x^2\right) (2\pi x)^2 + sen\left(\pi x^2\right) 2\pi$$

Luego,

$$N \geq \sqrt{\frac{1^3}{12 \cdot 10^{-4}} (2\pi + 4\pi^2)} \approx 195.28$$

Por lo tanto, se puede tomar $N = 196$ para obtener la aproximación con el error dado.

```
f=@(x) cos(pi*x.^2);
%Regla compuesta de los trapecios con N=196
valor=TrapecioCompuesta(f,0,1,196)
%0.373982833304262
%Con comando integral Matlab: 0.373982833415732
```

5.4.2 Regla de Simpson compuesta

En este caso, si se divide el intervalo en N subintervalos y se aplica la regla de Simpson compuesta a cada uno de ellos, se tendrá:

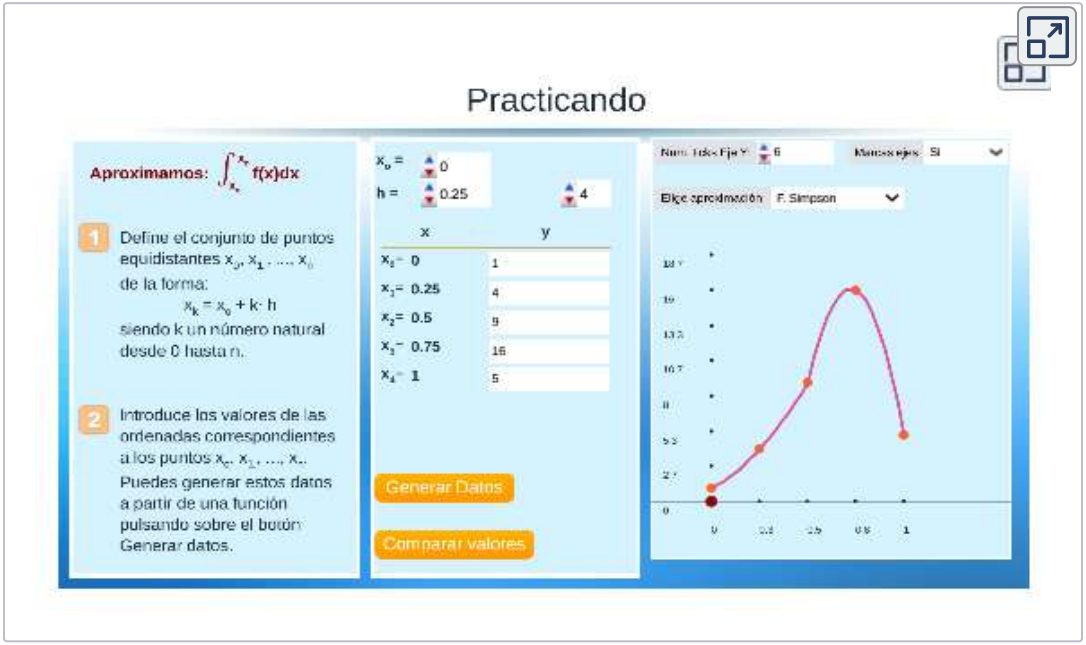
$$\int_a^b f(x)\, dx = \sum_{i=1}^{N} \left(\int_{\alpha_{i-1}}^{\alpha_i} f(x)\, dx \right)$$

$$\approx \sum_{i=1}^{N} \left(\frac{(b-a)}{6N} \left[f(\alpha_{i-1}) + 4f\left(\frac{\alpha_{i-1} + \alpha_i}{2} \right) + f(\alpha_i) \right] \right)$$

La cuadratura de Simpson compuesta consiste en aproximar una integral en $[a,b]$ dividiendo el intervalo en N subintervalos de longitud $h=\frac{b-a}{N}$ y aplicar en cada uno de ellos la regla de Simpson. En este caso se precisarían $2N+1$ nodos, equidistantes a distancia $h/2$, de la forma $x_i = a + ih/2$, $0 \leq i \leq 2N$ y la fórmula de cuadratura será

$$\int_a^b f(x)dx \approx \frac{h}{6}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots$$

$$\ldots + 4f(x_{2N-1}) + f(x_{2N})]$$



Interactivo 5.7. Fórmulas compuestas. Escena tomada de Integración Numérica. Proyecto Un100. Autora: Elena E. Álvarez

Ejemplo 5.13. Aproxima utilizando la regla de Simpson compuesta con $N=5$ el valor de la siguiente integral $\int_1^{\pi/2} \sqrt{x}\log(\sin(x))\,dx$.

Se define una función que calcule la regla de los trapecios compuesta.

```
function valor = SimpsonCompuesta(f, a, b, N)
%Aproxima la integral de f en el intervalo [a,b]
h=(b-a)/N;
x=linspace(a,b,2*N+1); %También x=a:h/2:b
valor=(h/6)* (f(x(1))+ 2*sum(f(x(3:2:end-2)))+
4*sum(f(x(2:2:end)))+f(x(end)))
end
```

Para utilizar la función y calcular de forma aproximada la integral pedida

```
f=@(x) sqrt(x).*log(sin(x))
%Regla compuesta de los trapecios con N=10
SimpsonCompuesta(f,1,pi/2,5) %-0.034221742037540
%Con comando integral Matlab: -0.034221721637133
```

Ejemplo 5.14. Para simular las características de los frenos de disco, se tuvo que aproximar numéricamente la temperatura exterior promediada del área, T, en el cojín del freno, basándose para ello en la expresión

$$T = \frac{\int_{r_e}^{r_o} T(r)\, r\theta_p dr}{\int_{r_e}^{r_o} r\theta_p dr}$$

donde $r_e = 0.308$ pies, $r_0 = 0.478$ pies y $\theta_p = 0.7051$ radianes. Si las temperaturas se obtuvieron en varios puntos del disco como muestra la tabla, se pide calcular la aproximación de T.

r (pies)	0.308	0.325	0.342	0.359	0.376	0.393	0.410	0.427	0.444	0.461	0.478
T(r) (°F)	640	794	885	943	1034	1064	1114	1152	1204	1222	1239

Tabla 5.2. Tabla de valores de la temperatura en función de los radios

Considerando los datos del ejercicio se trata de calcular la siguiente expresión

$$T = \frac{\int_{0.308}^{0.478} 0.7051\, T(r)\, r\, dr}{\int_{0.308}^{0.478} 0.7051\, r\, dr}$$

Haciendo los cálculos con Matlab/Octave la temperatura es aproximadamente $20.603°$F.

```
r=[0.308 0.325 0.342 0.359 0.376 0.393 0.410 0.427
0.444 0.461 0.478];
T=[640 794 885 943 1034 1064 1114 1152 1204 1222
1239]
%El número de subintervalos es N=5
%El número de nodos es 11=2*N+1 con N=5 subintervalos
%Aplicando la fórmula de Simpson compuesta se
%obtienen los siguientes valores
a=0.308;b=0.478;N=5;h=(b-a)/N;%Longitud de cada
intervalo
%donde se aplica Simpson
f=0.7051*T.*r;
I1=(h/6)* (f(1)+ 2*sum(f(3:2:end-2))+
4*sum(f(2:2:end))+f(end));
g=0.7051*r;
I2=(h/6)* (g(1)+ 2*sum(g(3:2:end-2))+
4*sum(g(2:2:end))+f(end));
%El valor de la temperatura es
T=I1/I2 %20.602700006958784
```

La función Matlab/Octave que se han definido para obtener la aproximación de una integral definida mediante una fórmula de cuadratura compuesta puede adaptarse para que en lugar de la función f se consideren los vectores x, $f(x)$ y pueda aplicarse cuando los datos vengan tabulados.

Si se considera M el máximo del valor absoluto de la derivada cuarta de la función en el intervalo (a, b), una cota del error en la fórmula de cuadratura de Simpson considerando N subintervalos es,

$$|E| \leq \frac{N \cdot M}{90}\left(\frac{b-a}{2N}\right)^5 = \frac{M(b-a)^5}{180(2N)^4}$$

Ejemplo 5.15. Considera la integral $\int_0^{\pi} e^{sen(x)}dx$ y calcula el valor aproximado usando la regla compuesta de Simpson tomando $N = 8$ subintervalos. Da una cota del error de dicha aproximación.

Para calcular el valor aproximado de la integral se considera el siguiente código.

```
f=@(x) exp(sin(x));
valor=SimpsonCompuesta(f,0,pi,8)
%6.208757412287802
%Con comando integral Matlab: 6.208758035711110
```

Para aplicar la expresión de la cota del error, se tiene $a = 0, b = \pi, N = 8$. El valor de M se calcula acotando la derivada cuarta de la función $f(x) = e^{sin(x)}$

$$\begin{aligned} f'(x) &= -\cos x\ e^{senx} \\ f''(x) &= e^{senx}\left(-senx + \cos^2 x\right) \\ f'''(x) &= e^{senx}\left(2\sin 2x + \cos^2 x\right) \\ f^{iv}(x) &= e^{senx}\left(-\cos^3 x - 2\cos(2x)\right) \end{aligned}$$

La derivada cuarta es menor que $3e$, por lo que una cota del error es

$$|E| \leq \frac{M\pi^3}{180(2\cdot 8)^4} \leq \frac{3e\pi^3}{180(2\cdot 8)^4} \approx 2.14345\text{e}-05$$

Ejemplo 5.16. Calcula la siguiente integral

$$\int_0^1 \left(1 + sen\left(x^2\right)\right)\, dx$$

utilizando la cuadratura de los trapecios compuesta dividiendo el intervalo en $N = 4$ subintervalos. Da una cota del error de la aproximación.

El valor aproximado de la integral se puede calcular con el siguiente código

```
f=@(x) (1+sin(x.^2));
valor=SimpsonCompuesta(f,0,1,4)
%1.310248532388182
%Con comando integral Matlab: 1.310268301723381
```

En este caso, se tiene $a = 0, b = 1, N = 4$. La derivada cuarta de la función $f(x) = 1 + sen(x^2)$ es

$$\begin{aligned}
f'\left(x\right) &= 2x\cos\left(x^2\right)\\
f''\left(x\right) &= 2\cos\left(x^2\right) - 4x^2 sen\left(x^2\right)\\
f'''\left(x\right) &= -12x\, sen\left(x^2\right) - 8x^3\cos\left(x^2\right)\\
f^{iv}\left(x\right) &= sen\left(x^2\right)\left(16x^4 - 12\right) - 48x^2\cos\left(x^2\right)
\end{aligned}$$

Teniendo en cuenta que $sen(x)$ y $cos(x)$ son menores que 1 y que $|16x^4 - 12|$ y $|48x^2|$ son menores que 4 y 48 respectivamente en el intervalo $[0, 1]$, la derivada cuarta puede acotarse por

$$M = \max_{x\in[0,1]} \left|f^{iv}\left(x\right)\right| \leqslant 52$$

Se puede obtener un valor más ajustado calculando el valor máximo de la derivada cuarta en este intervalo que es aproximadamente 28.4283. Una cota del error sería entonces:

$$|E| \leq \frac{52}{180(2\cdot 4)^4} \approx 7.052951388888888\mathrm{e}-05$$

A continuación, se incluyen algunos ejemplos en los que a partir de ciertos datos tabulados, se realizan distintas aproximaciones del valor de una integral utilizando fórmulas simples o compuestas de cuadratura.

Ejemplo 5.17. Estima el área de un canal subterráneo que transporta aire del medio ambiente al interior de una nave, dicha sección es simétrica a su eje horizontal de 90 cm de longitud, se tiene como dato el registro de 5 medidas verticales espaciadas cada 15 cm, tal como indica la siguiente tabla.

d	0	15	30	45	60	75	90
Medida en cm.	0	13	14	16	18	15	0

Se pide:

- Considerando toda la información, estima el área usando la regla compuesta del método del trapecio.
- Estima el área usando la fórmula compuesta del método de cuadratura 3/8.
- Estima el área usando la fórmula compuesta de Simpson.
- Se sabe que el área es $0.241356m^2$, ¿qué fórmula de las utilizadas anteriormente es mejor considerando el error absoluto y relativo?

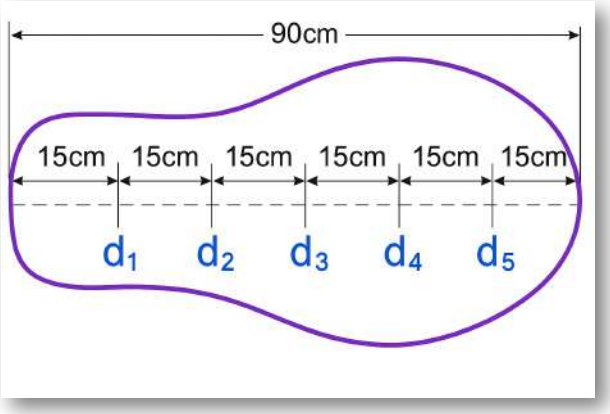


Figura 5.5. Datos ejemplo

Usando la regla compuesta del método del trapecio, se considera el siguiente código.

```
%Se utiliza la fórmula del trapecio compuesta
x=[0 15 30 45 60 75 90];
d=[0 13 14 16 18 15 0];
%Subintervalos N=6
h=15;%Longitud de cada subintervalo: 90/6
valor=(d(1)+2*sum(d(2:6))+d(7))*h/2
%Otra forma de obtener este valor
%aplicando trapecio a cada intervalo
I=0;
for k=1:6
    I=I+(d(k)+d(k+1));
end
I*h/2
```

Para obtener el valor pedido, se debe tener en cuanta que la región es simétrica y que el valor pedido se solicita en m^2.

```
%Simetría de la región de integración
areaA=2*valor;
%Como el resultado debe estar en m^2
areaTotalA=areaA*10^-4  %0.228000000000000
```

El valor del área es aproximadamente $0.228 m^2$. Utilizando la fórmula 3/8, el área es $0.2385\ m^2$.

```
x=[0 15 30 45 60 75 90];d=[0 13 14 16 18 15 0];
%Subintervalos 2 (en cada uno cuatro puntos)
lonI=45; I1=d(1)+3*d(2)+3*d(3)+d(4);
I2=d(4)+3*d(5)+3*d(6)+d(7);
I=(I1+I2)*lonI/8
%Simetría
areaB=2*I;
%Pasado a m^2
areaTotalB=areaB*10^-4  %0.2385
```

Utilizando la fórmula de Simpson, el código para calcular el área es el siguiente.

```
x=[0 15 30 45 60 75 90]; d=[0 13 14 16 18 15 0];
h=30;%Subintervalos 3 (en cada uno tres puntos)
I1=(d(1)+4*d(2)+d(3))*h/6;
I2=(d(3)+4*d(4)+d(5))*h/6;
I3=(d(5)+4*d(6)+d(7))*h/6;
valor=I1+I2+I3;
areaC=2*valor;%Simetría
areaTotalC=areaC*10^-4 %0.24
```

Se analiza el error absoluto y relativo considerando el valor exacto dado.

```
areaExacta=0.241356;
format long
errorAbs=abs([areaTotalA areaTotalB areaTotalC]-
areaExacta)
errorRel=abs([areaTotalA areaTotalB areaTotalC]-
areaExacta)/areaExacta
% errorRelativo1=abs((areaExacta-
areaTotalA)/areaExacta)
% errorRelativo2=abs((areaExacta-
areaTotalB)/areaExacta)
% errorRelativo3=abs((areaExacta-
areaTotalC)/areaExacta)
```

Los errores calculados, dan como mejor resultado la regla de Simpson.

```
%Error absoluto
0.013356   0.002856   0.001356
%Error relativo
0.055337344006364   0.011833142743499
0.005618256848804
```

Ejemplo 5.18. Se considera:

- C(z) es la concentración de una sustancia a profundidad z.
- A(z) es el área de la sección transversal a profundidad z en m^2.
- H es la profundidad total.

La concentración promedio de una sustancia que varía con la profundidad se puede calcular de la forma siguiente,

$$c_m = \frac{\int_0^H C(z)\,A(z)\,dz}{\text{volumen total}} = \frac{\int_0^H C(z)\,A(z)\,dz}{\int_0^H A(z)\,dz}$$

Se pide determinar la concentración promedio para $H = 16$ considerando los siguientes datos:

z (m)	0	4	8	12	18
$C(z)\,(g/m^3)$	10.2	8.5	7.4	5.2	4.1
$A(z)$	9,82	5.11	1,96	0.393	0

Utiliza al menos tres reglas de cuadratura.

```
%Cálculo utilizando la fórmula de los
%trapecios compuesta
z=[0 4 8 12 16];
c=[10.2 8.5 7.4 5.2 4.1];
A=[9.82 5.11 1.96 0.393 0];
%Utilizando la fórmula de los trapecios compuesta (3
subintervalos)
h=4;
masa1=
(c(1)*A(1)+2*c(2)*A(2)+2*c(3)*A(3)+2*c(4)*A(4)+c(5)*A
(5))*h/2;
```

```
vol1=(A(1)+2*A(2)+2*A(3)+2*A(4)+A(5))*h/2;
concentracion=masa1/vol1
%Cálculo utilizando Simpson compuesta (2
subintervalos)
h=8;
I1=c(1)*A(1)+4*c(2)*A(2)+c(3)*A(3);
I2=c(3)*A(3)+4*c(4)*A(4)+c(4)*A(5);
masa2=(I1+I2)*h/6;
I3=A(1)+4*A(2)+A(3);
I4=A(3)+4*A(4)+A(5);
vol2=(I3+I4)*h/6;
concentracion2=masa2/vol2
%Fórmula utilizando n=4 (5 nodos) Regla de Millne
h=16;
pesos=[7 32 12 32 7];
masa3=sum(pesos.*c.*A)*h/90;
vol3=sum(pesos.*A)*h/90;
concentracion3=masa3/vol3
```

Ejemplo 5.19. Un bloque se desplaza sobre una superficie plana por efecto de una fuerza cuya magnitud y el ángulo que forma con la horizontal, varían en función de la posición que va tomando el bloque.

Debido a restricciones experimentales, solo se ha registrado la siguiente información:

x (m)	0	2.5	5	10	12.5	15	20	25	30
$F(x)$ (Newton)	0	6	9	13	13.5	14	10.5	12	5
$\theta(x)$ (radianes)	0.5	0.76	1.4	0.75	0.57	0.9	1.3	1.48	1.5

Calcula el trabajo (W) realizado por el bloque en su recorrido considerando toda la información recogida en la tabla.

Se trata de calcular la integral siguiente:

$$W = \int_{0}^{30} F(x) \cos(\theta(x))\, dx$$

Teniendo en cuenta la separación de los nodos, la integral entre 0 y 30 se puede descomponer integrando entre 0 y 5, entre 5 y 10, entre 10 y 15 y entre 15 y 30. Se utilizará:

- En 0 y 5, se considerará la regla de Sipmson 1/3
- En 5 y 10, se considerará la regla del trapecio
- En 10 y 15, se considerará la regla de Simpson 1/3
- En 15 y 30, se considerará la regla 3/8

```
%Datos
x=[0 2.5 5 10 12.5 15 20 25 30];
F=[0 6 9 13 13.5 14 10.5 12 5];
theta=[0.5 0.76 1.4 0.75 0.57 0.9 1.3 1.48 1.5];
%Función a integrar
fxint=F.*cos(theta);
%En [0, 5] Regla de Simpson
lon1=5;
I1=(fxint(1)+4*fxint(2)+fxint(3))*lon1/6
%En [5,10] Regla del Trapecio
lon2=5;
I2=(fxint(3)+fxint(4))*lon2/2
%En [10, 15] Regla de Simpson 1/3
lon3=5;
I3=(fxint(4)+4*fxint(5)+fxint(6))*lon3/6
%En [15, 30] Regla 3/8
lon4=15;
I4=(fxint(6)+3*fxint(7)+3*fxint(8)+fxint(9))*lon4/8
%El Trabajo en [0, 30]
W=I1+I2+I3+I4
%Solución W=135.3398
```

5.5 Métodos adaptativos

En Los métodos compuestos se considera que la longitud de cada subintervalo es siempre igual, por lo que no se tiene en cuenta el comportamiento de la función. En este apartado se mostrarán métodos en los que se realizan subdivisiones del intervalo con mayor error estimado hasta alcanzar una tolerancia establecida. Así, dependiendo de la función, puede ser conveniente utilizar subintervalos más pequeños donde haya más variabilidad de la función.

> En estos métodos adaptativos se estima el error cometido en la aproximación de la integral en un intervalo de forma que permita decidir si hay que dividirlo para conseguir una determinada precisión.

5.5.1 Cuadratura adaptativa trapecios

En primer lugar, se calcula la aproximación I_1 de la integral de f en $[a, b]$ mediante la cuadratura de los trapecios.

Además, se considera I_2 la aproximación de la integral utilizando la regla del trapecio compuesta con $N = 2$ siendo E el error de esta aproximación.

Se tendrá:

$$\int_a^b f(x)\,dx = I_1 - \frac{(b-a)^3}{12} f''(\xi_1)$$

$$\int_a^b f(x)\,dx = I_2 - \frac{(b-a)^3}{12 \cdot 4} f''(\xi_2) = I_2 + E$$

Si se considera I_2 como valor de la integral, y se supone que $f''(\xi_1) \approx f''(\xi_2)$, restando las dos expresiones anteriores, una estimación del error sería

$$|I_2 - I_1| \approx |E - 4E| = |3E|$$

$$|E| \approx \left|\frac{1}{3}(I_2 - I_1)\right|$$

Teniendo en cuenta esta estimación del error, se puede establecer un método recursivo para calcular la integral de f en $[a, b]$ con un error inferior a un valor dado. El método consiste en aplicar los siguientes pasos.

- Se calcula I_1 e I_2, es decir, la aproximación de la integral utilizando respectivamente la fórmula de los trapecios y la fórmula de los trapecios compuesta con $N = 2$.
- Si la estimación del error, E, es menor que la tolerancia establecida, se devuelve el valor de I_2 con la estimación del error.
- Si esta estimación del error es mayor que la tolerancia, se repite el proceso en cada subintervalo $[a, \frac{a+b}{2}]$ y $[\frac{a+b}{2}, b]$ permitiendo en cada uno de ellos solo la mitad de la tolerancia.

Se puede observar que una estimación del error si se tomara como valor de la integral el valor de I_1 es decir, utilizando la cuadratura de los trapecios en $[a, b]$, sería $\left|\frac{4}{3}(I_2 - I_1)\right|$.

En esta estrategia la tolerancia se reparte equitativamente entre todos los nuevos subintervalos, sin embargo, cómo se verá más adelante en el método general, se puede establecer otra estrategia en el que se controla el error globalmente y se va subdividiendo en cada paso aquel subintervalo que tiene mayor error.

Ejemplo 5.20. Conocida la sección transversal $A(z)$ de un lago a partir de la profundidad z, se puede calcular el volumen de agua a profundidad H mediante la siguiente expresión:

$$volumen(H) = \int_0^H A\,(z)\,dz$$

Calcula el volumen para $H = 16$ utilizando la fórmula de cuadratura de los trapecios y la fórmula de cuadratura compuesta de los trapecios con $N = 2$ a partir de los siguientes datos dando una estimación del error de cada aproximaicón.

z (m)	0	4	8	12	16
$A(z)$	9.82	5.11	1.96	0.393	0

Con el siguiente código se cálcula en la variable I2 el volumen utilizando la fórmula compuesta de los trapecios con $N = 2$. Se calcula también la aproximación con la fórmula simple de los trapecios en $I1$.

```
A=[9.82 5.11 1.96 0.393 0];
disp('Cálculos aproximados de la integral')
I2=(A(1)+2*A(3)+A(5))*8/2  %54.960000000000001
I1=(A(1)+A(5))*16/2    %78.560000000000002
```

Se calculan las estimaciones del error en cada caso.

```
%Considerando I2 como valor de la integral
estimE2=abs(I2-I1)/3
%7.866666666666667
%Considerando I1 como valor de la integral
estimE1=abs(I2-I1)*4/3
%31.466666666666669
```

Ejemplo 5.21. Calcula

$$\int_0^1 \frac{\log(1+x)}{1+x^2}\,dx$$

con un error inferior a 10^{-3} aplicando paso a paso el método adaptativo de los trapecios.

Se define la función integrando y el error permitido.

```
f = @(x) log(1 + x) ./ (1 + x.^2);
a = 0; b = 1; tol = 1e-3;
```

Se define una función Matlab para calcular la fórmula de cuadratura de los trapecios para $N = 2$ de f en $[a, b]$, dando también la estimación del error que se obtendría al considerar este valor como aproximación. Además, según la estimación del error, se muestra un mensaje indicando si éste está por debajo de la tolerancia establecida o si, en caso contrario, es necesario subdividir el intervalo para lograrlo.

```
function [S,estE]=PasoTrapAdaptativo(f,a,b,tol)
    I1=Trapecio(f,a,b);
    xm=(a+b)/2;
    I2=Trapecio(f,a,xm)+Trapecio(f,xm,b);
    S=I2;
    estE=abs(I1-I2)/3;
    if estE>tolerancia
        disp('Dividir el intervalo')
    else
        disp('Estimación inferior a la permitida')
    end
end
```

Para aplicar el método adaptativo, se considera el intervalo $[0, 1]$ y se analiza, con ayuda de la función anterior, si la estimación de la aproximación es inferior a la tolerancia, `tol`.

```
[S,est]=PasoTrapAdaptativo(f,0,1,tol);
%Se debe dividir el intervalo [0,1]
```

Como no se cumple que el error sea menor que la tolerancia, se divide el intervalo $[0,1]$ y se repite el proceso en $[0,0.5]$ y $[0.5,1]$ buscando que el error estimado sea inferior a tol/2 en cada intervalo.

Se repite el proceso para calcular la integral en $[0,0.5]$ con error inferior a tol/2.

```
%[0, 0.5] tol/2
[S,est]=PasoTrapAdaptativo(f,0,0.5,tol/2);
%Se debe dividir el intervalo [0, 0.5]
```

De nuevo, hay que dividir el intervalo $[0,0.5]$ y repetir el proceso con los intervalos $[0,0.25]$ y $[0.25,0.5]$ buscando ahora que la estimación sea inferior a tol/4.

Se analiza $[0,0.25]$.

```
%[0, 0.25] tol/4
[S,est]=PasoTrapAdaptativo(f,0,0.25,tol/4);
%Se debe dividir el intervalo [0, 0.25]
```

Se debe dividir el intervalo $[0,0.25]$ y repetir el proceso con los intervalos $[0,0.125]$ y $[0.125,0.25]$ considerando tol/8.

```
%[0, 0.125] tol/8
[S,est]=PasoTrapAdaptativo(f,0,0.125,tol/8);
%Se cumple
```

Dado que sí se cumple con la tolerancia establecida, se considera el valor `S` como la aproximación de la integral en $[0,0.125]$.

```
val1=S %0.007398388926458
```

Se analiza $[0.125, 0.25]$ y se comprueba que también se cumple con la tolerancia permitida, así que se considera el valor de la integral en ese intervalo como S.

```
%[0.125, 0.25] tol/8
[S,est]=PasoTrapAdaptativo(f,0.125,0.25,tol/8);
%Se cumple
val2=S   %0.020563003600857
```

Hasta este momento se tiene calculado el valor de la integral en $[0, 0.25]$ con la tolerancia adecuada. Se analiza $[0.25, 0.5]$ con tol/4.

```
%[0.25, 0.5] tol/4
[S,est]=PasoTrapAdaptativo(f,0.25,0.5,tol/4);
%Se debe dividir el intervalo [0.25 0.5]
```

Como el error no es inferior, se divide el intervalo $[0.25, 0.5]$ buscando que el valor de la integral en $[0.25, 0.375]$ y $[0.375, 0.5]$ tenga un error inferior a tol/8.

```
%[0.25, 0.375] tol/8
[S,est]=PasoTrapAdaptativo(f,0.25,0.375,tol/8);
%Se cumple
val3=S  %0.030771575660209
%[0.375, 0.5] tol/8
[S,est]=PasoTrapAdaptativo(f,0.375,0.5,tol/8);
%Se cumple
val4=S %0.037899052702097
```

Se ha calculado el valor de la integral en $[0.25, 0.5]$ con la tolerancia adecuada. Se analiza $[0.5, 1]$ con una estimación del error inferior a tol/2.

```
%[0.5, 1] tol/2
[S,est]=PasoTrapAdaptativo(f,0.5,1,tol/2);
%Se debe dividir el intervalo [0.5 1]
```

Como el error no es inferior, se considera el valor de la integral en $[0.5, 0.75]$ y $[0.75, 1]$ con error inferior a tol/4.

Se analiza $[0.5, 0.75]$.

```
%[0.5, 0.75] tol/4
[S,est]=PasoTrapAdaptativo(f,0.5,0.75,tol/4);
%Se debe dividir el intervalo [0.5 0.75]
```

Como el error no es inferior, se considera el valor de la integral en $[0.5, 0.625]$ y $[0.625, 1]$ con tol/8.

```
%[0.5, 0.625] tol/8
[S,est]=PasoTrapAdaptativo(f,0.5,0.625,tol/8);
%Se cumple
val5=S  %0.042235620867100
%[0.625, 0.75] tol/8
[S,est]=PasoTrapAdaptativo(f,0.625,0.75,tol/8);
%Se cumple
val6=S  %0.044309421526304
```

Se ha calculado el valor de la integral en $[0.5, 0.75]$ con la tolerancia adecuada. Se analiza $[0.75, 1]$ con tol/4.

Se analiza $[0.75, 1]$.

```
%[0.75, 1] tol/4
[S,est]=PasoTrapAdaptativo(f,0.75,1,tol/4);
%Se cumple
val7=S  %0.088548748833623
```

Por lo tanto, el valor de la integral será la suma de los valores obtenidos en

- $[0, 0.125]$, $[0.125, 0.25]$, $[0.25, 0.375]$, $[0.375, 0.5]$, $[0.5, 0.625]$ y $[0.625, 0.75]$ (error estimado tol/8 en cada intervalo).
- $[0.75, 1]$ (error estimado tol/4).

```
disp('El valor de la integral es')
val1+val2+val3+val4+val5+val6+val7
%0.271725812116648
%Cálculo dado con Matlab/Octave
integral(f,0,1) % 0.272198261287950
```

En la siguiente imagen se representa las distintas divisiones realizadas en el ejemplo.

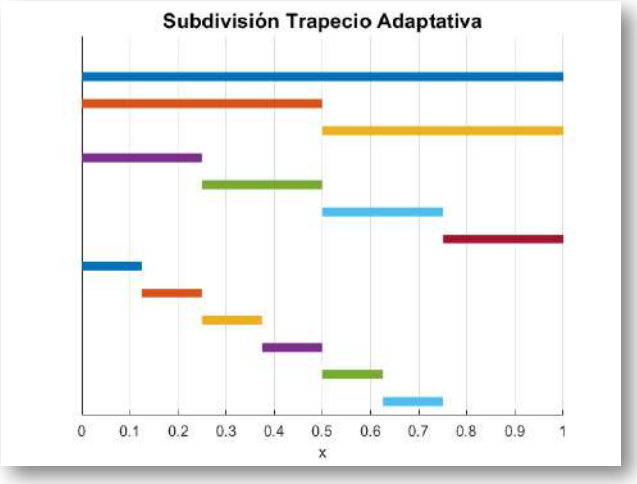


Figura 5.6. División realizada del intervalo [0,1]

Ejemplo 5.22. Programa el método recursivo de la fórmula de cuadratura adaptiva de los trapecios y calcula, con un error menor que 10^{-5}, el valor de la integral siguiente

$$\int_0^1 \left(1 + sen\left(x^2\right)\right)\, dx$$

Se da a continuación el código del programa del método recursivo de la fórmula de cuadratura adaptativa que tendrá como argumentos, la función a integrar, el extremo inferior y superior del intervalo de integración, y la tolerancia permitida en la aproximación.

```
function valor=TrapecioAdaptativa(f,a,b,tol)
    %Aproxima la integral de f en el intervalo [a,b]
    %usando la regla del trapecio adaptativa
    %en forma recursiva
    %Trapecios en [a,b]
    I1=(f(a)+f(b))*(b-a)/2;
    %Punto medio del intervalo
    xm=(a+b)/2;
    %Trapecios en [a,xm] y en [xm,b]
    I2=(f(a)+f(xm))*(xm-a)/2+(f(xm)+f(b))*(xm-a)/2;
    %También: (f(a)+2*f(xm)+f(b))*(xm-a)/2;
    %Estimación del error considerando I2
    $como valor aproximado de la integral
    error=abs(I1-I2)/3;
    if error<tol
        valor=I2;
    else
        %Si la estimación no es la deseada
        %se aplica el proceso en [a, xm] y [xm, b]
        %con error inferior en cada uno a tol/2
        valor= TrapecioAdaptativa(f,a,xm,tol/2)+
        TrapecioAdaptativa(f,xm,b,tol/2);
    end
end
```

Para calcular el valor de la integral, se deberá escribir el siguiente código.

```
f=@(x) 1+sin(x.^2);
TrapecioAdaptativa(f,0,1,10^-5)
%Devuelve el valor 1.310271922110889
%Con comando integral de Matlab: 1.310268301723381
```

5.5.2 Cuadratura adaptativa Simpson

De forma análoga, llamando I_1 al cálculo de la integral de $f(x)$ en $[a,b]$ con la fórmula de cuadratura de Simpson e I_2 al cálculo con la fórmula de cuadratura de Simpson compuesta con $N=2$, se tendrá,

$$\int_a^b f(x)\,dx = I_1 - \frac{h^5}{90} f^{iv}(\xi_1)$$

$$\int_a^b f(x)\,dx = I_2 - \frac{h^5}{90\cdot 2^4} f^{iv}(\xi_2) = I_2 + E$$

siendo $h=(b-a)/2$.

Si se supone que $f^{(iv}(\xi_1) \approx f^{(iv}(\xi_2)$, restando las dos expresiones anteriores, se puede considerar que una estimación del error considerando como valor de la integral I_2 será

$$|I_2 - I_1| \approx |E - 16E| = |15E|$$

$$|E| \approx \left|\frac{1}{15}(I_2 - I_1)\right|$$

Teniendo en cuenta la estimación anterior del error, el método recursivo para obtener una aproximación de la integral con una cierta tolerancia, consiste en aplicar los siguientes pasos.

- Se calcula I_1 que es la aproximación de la integral de f en $[a,b]$ utilizando la regla de Simpson e I_2 la fórmula compuesta de Simpson para $N=2$.

- Se estima el error, E. Si es menor que la tolerancia, se devuelve el valor de I_2 junto con la estimación del error calculada.
- Si esta estimación del error es mayor que la tolerancia, se repite el proceso en cada subintervalo $[a, \frac{a+b}{2}]$ y $[\frac{a+b}{2}, b]$ permitiendo en cada uno de ellos solo la mitad de la tolerancia.

Una estimación del error considerando como valor la regla de Simpson en un intervalo $[a, b]$, es decir, tomando como aproximación I_1, sería $\left|\frac{16}{15}(I_2 - I_1)\right|$.

Ejemplo 5.23. Conocida la sección transversal $A(z)$ de un lago a partir de la profundidad z, se puede calcular el volumen de agua a profundidad H mediante la siguiente expresión:

$$volumen(H) = \int_0^H A\,(z)\,dz$$

A partir de los datos siguientes, se pide calcular la aproximación del volumen para $H = 16$ empleando la cuadratura compuesta de Simpson con $N = 2$.

z (m)	0	4	8	12	16
$A(z)$	9.82	5.11	1.96	0.393	0

Dar una estimación del error que se comete al considerar como volumen para $H = 16$ esta aproximación.

Para el cálculo aproximado del volumen utilizando la fórmula compuesta de Simpson con $N = 2$ se utiliza el siguiente código.

```
A=[9.82 5.11 1.96 0.393 0];
I2=(A(1)+4*A(2)+4*A(4)+2*A(3)+A(5))*8/6   %47.6693
%También:
%(A(1)+4*A(2)+A(3))*8/6+(A(3)+4*A(4)+A(5))*8/6
```

Una estimación del error será:

```
I1=(A(1)+4*A(3)+A(5))*16/6  %47.0933
%Considerando I2 como valor de la integral
estimE2=abs(I2-I1)/15   %0.0384
%De considerar I1 como valor de la integral
estimE1=abs(I2-I1)*16/15   %0.6144
```

Ejemplo 5.24. Calcula

$$\int_0^1 \frac{\log(1+x)}{1+x^2}dx$$

con un error inferior a 10^{-5} aplicando el método adaptativo de Simpson paso a paso.

Se define la función integrando y el error permitido.

```
f = @(x) log(1 + x) ./ (1 + x.^2);
a = 0; b = 1;
tol = 1e-5;
```

Se define una función Matlab para calcular la fórmula de cuadratura de Simpson para $N = 2$ de f en $[a, b]$, dando también la estimación del error que se obtendría al considerar ese valor como aproximación. Asimismo, tras comparar la estimación del error con la tolerancia establecida, se imprime un mensaje en pantalla que informa si el error está dentro del margen permitido o, de lo contrario, se debe dividir el intervalo para cumplir con dicha tolerancia.

```
function [S,estE]=PasoSimponAdaptativo(f,a,b,tol)
    I1=Simpson(f,a,b);
    xm=(a+b)/2;
    I2=Simpson(f,a,xm)+Simpson(f,xm,b);
    S=I2;
    estE=abs(I1-I2)/15;
    if estE>tolerancia
        disp('Dividir el intervalo')
    else
        disp('Estimación inferior a la permitida')
    end
end
```

Se considera en primer lugar el intervalo $[0, 1]$ y se estima el error para ver si es inferior a la tolerancia establecida, `tol`.

```
[S,est]=PasoSimponAdaptativo(f,0,1,tol)
%No se cumple que sea menor que tol
```

Como no se cumple, se divide el intervalo $[0, 1]$ y se repite el proceso en $[0, 0.5]$ y $[0.5, 1]$ buscando que el error estimado sea inferior a tol/2.

Se calcula la integral en $[0, 0.5]$ con error inferior a tol/2.

```
[S,est]=PasoSimponAdaptativo(f,0,0.5,tol/2)
%No se cumple que es inferior a tol/2
```

Se divide el intervalo $[0, 0.5]$ en dos subintervalos, $[0, 0.25]$ y $[0.25, 0.5]$, repitiendo el proceso considerando el error inferior a tol/4.

Se analiza $[0, 0.25]$.

```
%Se ve si en [0,0.25] el error es menor que tol/4
[S,est]=PasoSimponAdaptativo(f,0,0.25,tol/4)
%Se cumple que es inferior a tol/4
```

En este caso, se toma S como el valor de la integral en $[0, 0.25]$

```
valor1=S
%Se obtiene en [0, 0.25]: 0.028074368412078
```

Se analiza $[0.25, 0.5]$.

```
%Se ve si en [0,25,0.5] el error es menor que tol/4
[S,est]=PasoSimponAdaptativo(f,0.25,0.5,tol/4)
%Se cumple que es inferior a tol/4
```

Como el error es inferior, se considera S como el valor de la integral en $[0.25, 0.5]$

```
valor2=S
%Resultado en [0.25, 0.5]: 0.068795007885837
```

Se analiza $[0.5, 1]$

```
[S,est]=PasoSimponAdaptativo(f,0.5,1,tol/2)
%Se cumple que es inferior a tol/2
```

Como el error es inferior, se considera el valor de la integral en $[0.5, 1]$ el dado por S.

```
valor3=S
%Resultado en [0, 0.5]: 0.175328138427274
```

Por lo tanto, el valor de la integral será la suma de los valores obtenidos en

- $[0.0.25]$ y $[0.25, 0.5]$ (error estimado tol/4 en cada uno de los intervalos).
- $[0.5, 1]$ (error estimado tol/2).

```
disp('El valor de la integral es')
valor1+valor2+valor3 %0.272197514725190
%Calculo dado con Matlab/Octave
integral(f,0,1) %0.272198261287950
```

En la siguiente imagen se representa las distintas divisiones realizadas en el ejemplo.

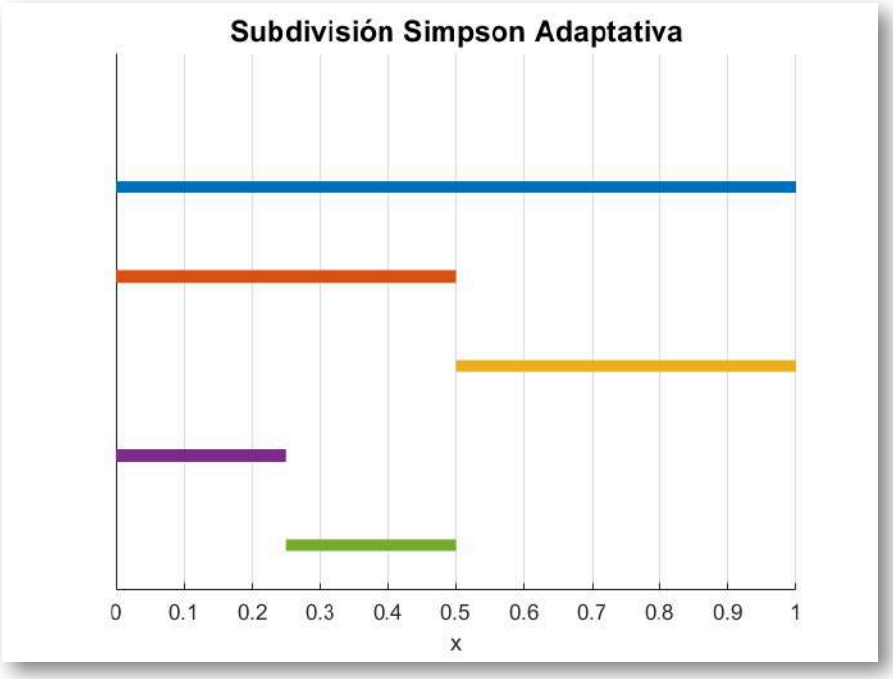


Figura 5.7. División realizada del intervalo [0,1]

Ejemplo 5.25. Programa de forma recursiva la fórmula de cuadratura adaptiva de Simpson y calcula, con un error menor que 10^{-5}, el valor de

$$\int_0^1 \left(1 + sen\left(x^2\right)\right)\, dx$$

En primer lugar se considera la función que permite obtener de forma recursiva la aproximación de la integral con una cierta tolerancia aplicando el método recursivo adaptativo de Simpson. Se utiliza la función `Simpson` que calcula la aproximación de la integral de una función en un intervalo mediante esta fórmula de cuadratura.

```
function valor=SimpsonAdaptativa(f,a,b,tol)
 %Simpson en [a,b]
 I1=Simpson(f,a,b);
 %Simpson en [a,puntoMedio] y [puntoMedio,b]
 %Punto medio
 xm=(a+b)/2;
 I2=Simpson(f,a,xm)+Simpson(f,xm,b);
 %Estimación del error al considerar
 %I2 como aproximación de la integral
 E=abs(I2-I1)/15;
 if E<tol
     valor=I2;
 else
     %Si la estimación del error es superior
     %a la tolerancia se aplica el mismo
     %proceso en [a,xm] y [xm,b] con tol/2
     J1=SimpsonAdaptativa(f,a,xm,tol/2);
     J2=SimpsonAdaptativa(f,xm,b,tol/2);
     valor=J1+J2;
 end
end
```

Para aproximar la integral pedida, bastará escribir el siguiente código.

```
f=@(x) 1+sin(x.^2);
SimpsonAdaptativa(f,0,1,10^-5)
%Devuelve: 1.310264088438636
```

Ejemplo 5.26. Calcula $\int_0^{\pi} e^{\text{sen}x \cos x}\, dx$ con error inferior a 10^{-12}.

Se define la función a integrar y se ejecuta `SimpsonAdaptativa` considerando $a = 0, b = 1$ y $tol = 10^-12$.

```
f=@(x) exp(sin(x).*cos(x));
SimpsonAdaptativa(f,0,pi,10^-12)
%Devuelve el valor 3.341031544735790
```

Ejemplo 5.27. Calcula $\int_0^{4} e^{-x^2}\, dx$ con error inferior a 10^{-12}.

Se define la función a integrar y se ejecuta `SimpsonAdaptativa` considerando $a = 0, b = 4$ y $tol = 10^-12$.

```
f=@(x) exp(-x.^2);
SimpsonAdaptativa(f,0,4,10^-12)
%Devuelve el siguiente valor 0.886226911789732
```

Si bien la recursividad puede ser una herramienta útil para resolver este problema, se debería refinar el código para evitar el desbordamiento.

5.5.3 Método adaptativo general

En este apartado se verá cómo encontrar una estimación del error cuando se utiliza una fórmula de Newton-Cotes considerando dos pasos diferentes h y $h/2$ y una estrategia para aplicar este método adaptativo en general.

Si se llama I a la aproximación de la integral

$$\int_a^b f(x)\,dx = I + E \ \text{ con } E = C_n h^{m+1} f^{(m}(\xi)$$

siendo

$$m = \begin{cases} n+1 \ si \ n \ impar \\ n+2 \ si \ n \ par \end{cases}$$

Aplicando la misma fórmula a los intervalos $[a, \frac{a+b}{2}]$ y $[\frac{a+b}{2}, b]$ se tendrá,

$$\int_a^{\frac{a+b}{2}} f(x)\,dx = I_1 + E_1 = I_1 + C_n \left(\frac{h}{2}\right)^{m+1} f^{(m}(\xi_1)$$

$$\xi_1 \in \left[a, \frac{a+b}{2}\right]$$

$$\int_{\frac{a+b}{2}}^b f(x)\,dx = I_2 + E_2 = I_2 + C_n \left(\frac{h}{2}\right)^{m+1} f^{(m}(\xi_2)$$

$$\xi_2 \in \left[\frac{a+b}{2}, b\right]$$

Entonces,

$$I + E = I_1 + I_2 + C_n \frac{h^{m+1}}{2^m} \left[\frac{f^{(m}(\xi_1) + f^{(m}(\xi_2)}{2} \right]$$

Si $[a, b]$ es pequeño,

$$I + E \approx I_1 + I_2 + \underbrace{C_n \frac{h^{m+1}}{2^m} f^{(m}(\xi)}_{E/2^m}$$

$$I + E \approx I_1 + I_2 + \frac{E}{2^m}$$

$$\frac{2^m - 1}{2^m} E \approx I_1 + I_2 - I$$

$$E \approx \frac{2^m}{2^m - 1} (I_1 + I_2 - I)$$

En consecuencia, si se utiliza la fórmula de Newton-Cotes de orden n para aproximar la función en el intervalo $[a, b]$, tomando $m = n + 1$ si n es impar y $m = n + 2$ si n es par y considerando las aproximaciones

- I en el intervalo $[a, b]$
- I_1 en el intervalo $[a, \frac{a+b}{2}]$
- I_2 en el intervalo $[\frac{a+b}{2}, b]$

se cumple que el error E de la aproximación en $[a, b]$ es

$$|E| \approx \left| \frac{2^m}{2^m - 1} (I_1 + I_2 - I) \right|$$

Por lo tanto, fijada una precisión ϵ deseada de la integral, se describe la estrategia para obtener la aproximación con la precisión deseada.

1. Se elige una fórmula de Newton-Cotes para aproximar la integral en $[a, b]$.

2. Se subdivide el intervalo $[a, b]$ en dos subintervalos de la misma longitud y se calculan las aproximaciones de las integrales I_1 e I_2 en ambos intervalos, con las correspondientes estimaciones de los errores E_1 y E_2. Se definen los vectores $I = [I_1,\ I_2]$ y $E = [E_1,\ E_2]$

3. Si la suma de los errores, sum(E), es inferior a ϵ, se toma la suma de las aproximaciones I como valor aproximado de la integral y se habrá terminado. Si no es así, $sum(E) \geq \epsilon$, entonces se va al paso siguiente.

4. Se busca el error E_k más grande de los contenidos en el vector E. Se subdivide el intervalo asociado a E_k en dos mitades y se calculan las aproximaciones de la integral en cada una de las mitades y los correspondientes errores. Se elimnan I_k y E_k y se pone en su lugar los dos valores de la integral calculados y los correspondientes errores. Se vuelve al paso 3.

Para practicar con este procedimiento, se puede utilizar el fichero iehardy.m que a su vez utiliza la función hardy.m, que devuelve la aproximación de la integral utilizando la fórmula de Hardy. Estas dos funciones han sido desarrolladas por el profesor Eduardo Casas (ficheros Matlab).

```
[itg error]=iehardy(a,b,f)
%Devuelve tanto la estimación de la integral como
%el error cometido. Requiere que la
%función f se programe vectorialmente.
```

Ejemplo 5.28. Calcula la integral $\int_0^{\pi} e^{senx\ \cos x} dx$ con un error inferior a $\epsilon_a = 10^{-12}$ considerando la regla de Hardy.

Se definen los datos del problema.

```
a=0;b=pi;
format long
f=@(x) exp(sin(x).*cos(x));
error=10^-12;
```

Se divide el intervalo en dos partes y se aplica la fórmula de Newton-Cotes de Hardy. Se tendrá que

```
x=[a (a+b)/2 b];
[i1 e1]=iehardy(x(1),x(2),f);
[i2 e2]=iehardy(x(2),x(3),f);
E=[e1 e2];I=[i1 i2];
error=sum(E)
itg=sum(I)
```

Si el error es menor que ϵ_a se habrá terminado y el valor de la integral es itg. Como en este caso no es así, se considera el subintervalo con mayor error. Ejecutando la orden,

```
[s k]=max(E)
```

El valor que se obtiene es $k = 1$, Se continúa repitiendo el proceso en el intervalo $[x(k), x(k+1)]$ hasta conseguir la precisión considerada.

Los vectores x, E, I van creciendo según se van tomando subdivisiones en los intervalos.

```
xn=(x(k)+x(k+1))/2;
x=[x(1:k) xn x(k+1:end)];
[i1 e1]=iehardy(x(k),xn,f);
[i2 e2]=iehardy(xn,x(k+2),f);
E=[E(1:k-1) e1 e2 E(k+1:end)]
I=[I(1:k-1) i1 i2 I(k+1:end)];
itg=sum(I)
error=sum(E)
```

Se repite el proceso 26 veces obteniendo que el valor de la integral es 3.341031544735853.

Nota: El valor que devuelve matlab con el comando `integral(f,0,pi)` es 3.341031544735852.

Si se considera el error absoluto en la precisión de la solución el algoritmo se detiene si $E < \epsilon_a$. Si se usa el error relativo, el algoritmo se detenderá si $E < \epsilon_r |I|$. A menudo se utiliza un test de parada que combina ambos errores:

$$E < \epsilon_a + \epsilon_r |I|$$

Usualmente, si la integral es un número grande, el algoritmo se detiene cuando el error relativo es inferior a ϵ_r, mientras que si $|I|$ es pequeño, el algoritmo finaliza cuando el error absoluto es inferior a ϵ_a.

Se muestra a continuación, el código general a ejecutar con Matlab/Octave.

```
%ea error absoluto
%er error relativo
%Se definen los extremos del intervalo
%y el punto medio
x=[a (a+b)/2 b]
%Se aplica la cuadratura de Hardy en [a, xm]
[i1 e1]=iehardy(x(1),x(2),f)
%Se aplica la cuadratura de Hardy en [xm, b]
[i2 e2]=iehardy(x(2),x(3),f)
%Se almacenan los valores de las integrales y
%la estimación de los errores
I=[i1 i2];E=[e1 e2];
error=sum(E);
%Se comprueba si el error es menor que
%el error absoluto considerado
if error<ea
    itg=sum(I);
    [itg error]
else
    %Se considera el subintervalo con mayor error
    [s,k]=max(E)
end
parar=0
%Se repite el proceso hasta conseguir la
%precisión dada
while parar==0
    xn=(x(k)+x(k+1))/2
    x=[x(1:k) xn x(k+1:end)]
    [i1 e1]=iehardy(x(k),xn,f);
    [i2 e2]=iehardy(xn,x(k+2),f);
    E=[E(1:k-1) e1 e2 E(k+1:end)];
    I=[I(1:k-1) i1 i2 I(k+1:end)];
    error=sum(E);
    if error < ea+er*abs(itg)
        itg=sum(I);
        [itg error]
        parar=1;
    else
        [s,k]=max(E);
    end
end
```

5.6 Cálculo de Integrales impropias

En este apartado, se verá cómo proceder cuando el intervalo $[a, b]$ no es acotado o cuando la función f tiene una singularidad en un punto x_0 en $[a, b]$ de forma que $\lim_{x \to x_o} |f(x)| = +\infty$. En el primer caso, se denominan integrales impropias de primera especie y en el segundo integrles impropias de segunda especie.

5.6.1 Integrales impropias de primera especie

Para estas integrales se considera que los límites de integración a o b o ambos, no son finitos.

TIPO I. Se consideran en este caso integrales del tipo

$$\int_a^{\infty} f(x)\, dx = \lim_{b \to \infty} \int_a^b f(x)\, dx$$

con f una función continua para la cual la integral es convergerte, esto es, el límite anterior existe. En consecuencia, se tendrá que

$$\lim_{b \to \infty} \int_b^{\infty} f(x)\, dx = 0$$

Teniendo esto en cuenta, el valor aproximado de la integral con un error inferior a un valor fijado, ϵ, se obtendrá siguiendo los siguientes pasos.

Paso 1. En primer lugar se hallará un valor de b de forma que

$$\left| \int_b^{\infty} f(x)\, dx \right| < \frac{\varepsilon}{2}$$

Paso 2. Posteriormente, se buscará una aproximación de la intengral Itg de forma que

$$\left| \int_a^b f(x)\,dx - Itg \right| < \frac{\varepsilon}{2}$$

TIPO II. Procediendo de forma análoga, se podrá calcular las integrales de la forma

$$\int_{-\infty}^{a} f(x)\,dx$$

TIPO III. En el caso de una integral de la forma

$$\int_{-\infty}^{\infty} f(x)\,dx$$

se tendrá en cuenta que

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_{-\infty}^{0} f(x)\,dx + \int_{0}^{\infty} f(x)\,dx$$

Se procederá entonces buscando una aproximación de cada una de ellas con un error menor que $\epsilon/2$. Alternativamente se pueden buscar números $a < 0$ y $b > 0$ de forma que

$$\left| \int_{-\infty}^{a} f(x)\,dx \right| < \frac{\varepsilon}{4} \qquad \left| \int_{b}^{\infty} f(x)\,dx \right| < \frac{\varepsilon}{4} \qquad \left| \int_{a}^{b} f(x)\,dx \right| < \frac{\varepsilon}{2}$$

Ejemplo 5.29. Calcula $\int_0^\infty e^{-x}\cos^2(x)\,dx$ considerando $e_a = 10^{-12}$.

Esta integral es impropia de primera especie, del tipo I. Dado que

$$\int_0^\infty e^{-x}\cos^2(x)\,dx = \underbrace{\int_0^b e^{-x}\cos^2(x)\,dx}_{=I_1} + \underbrace{\int_b^\infty e^{-x}\cos^2(x)\,dx}_{=I_2}$$

se calculará el valor de la integral, siguiendo los siguientes pasos:

- En primer lugar, se busca b de forma que

  $$|I_2| \leq \frac{10^{-12}}{2}$$

  Teniendo en cuenta que

  $$\left|\int_b^\infty e^{-x}\cos^2 x dx\right| \leqslant \int_b^\infty \left|e^{-x}\cos^2 x\right| dx \leqslant \int_b^\infty e^{-x}dx = \frac{1}{e^b}$$

  basta obtener b cumpliendo

  $$\frac{1}{e^b} < \frac{10^{-12}}{2}$$

  Puede considerarse cualquier b cumpliendo $\log\left(2\cdot 10^{12}\right) \leq b$, por ejemplo, $b = 28.4$.

- Una vez obtenido b, se calcula I_1 con un error menor que $10^{-12}/2$.

El valor de la integral siguiendo este proceso es 0.599999999999007.

```
f=@(x) exp(-x).*cos(x).^2;
SimpsonAdaptativa(f,0,28.5,10^-12/2)
%Valor 0.599999999999479
```

Ejemplo 5.30. Calcula $\int_0^\infty e^{-x}\log(2+\text{sen}x)\,dx$ considerando $e_a = 10^{-12}$

En este caso, se considera $f(x) = e^{-x}\log(2+\text{sen}x)$. Se escribe la integral pedida de la forma siguiente,

$$\int_0^\infty f(x)\,dx = \underbrace{\int_0^b f(x)\,dx}_{I_1} + \underbrace{\int_b^\infty f(x)\,dx}_{I_2}$$

Se busca b cumpliendo

$$\left|\int_b^\infty e^{-x}\log(2+\text{sen}x)\,dx\right| \leqslant \int_b^\infty e^{-x}\log 3\,dx = \frac{\log 3}{e^b} \leqslant \frac{10^{-12}}{2}$$

Despejando el valor de b se debe cumplir,

$$b \geqslant \log\left(2\log 3\cdot 10^{12}\right) \approx 11.978886494563813$$

Se toma $b = 11.98$ y se calcula I_1 con un error menor que $\frac{10^{-12}}{2}$.

```
f=@(x) exp(-x).*log(2+sin(x));
SimpsonAdaptativa(f,0,11.98,10^-12/2)
%Valor 0.902221166208351
```

La solución que da Matlab con el comando `integral` es 0.902221166208359

5.6.2 Integrales impropias de segunda especie

En este tipo de integrales la función f es continua en $[a,b]$ salvo en un punto x_0 del intervalo donde

$$\lim_{x\to x_0}|f(x)|=+\infty$$

Caso 1 Si el punto $x_0=a$ será

$$\int_a^b f(x)\,dx=\lim_{\delta\searrow 0}\int_{a+\delta}^a f(x)\,dx$$

Se procede de la siguiente forma:

Se busca δ cumpliendo

$$\left|\int_a^{a+\delta} f(x)\,dx\right|<\frac{\varepsilon}{2}$$

y luego una aproximación verificando

$$\left|\int_{a+\delta}^b f(x)\,dx\right|<\frac{\varepsilon}{2}$$

Caso 2 Si el punto $x_0=b$ será

$$\int_a^b f(x)\,dx=\lim_{\delta\searrow 0}\int_a^{b-\delta} f(x)\,dx$$

Se procede de la misma forma que en el caso anterior

Caso 3 Si x_0 es un punto interior se define esta integral impropia de la forma siguiente:

$$\int_a^b f(x)\,dx = \lim_{\delta \searrow 0} \int_a^{x_o-\delta} f(x)\,dx + \lim_{\delta \searrow 0} \int_a^{x_o+\delta} f(x)\,dx$$

Se busca un valor de $\delta > 0$ verificando

$$\left| \int_{x_o-\delta}^{x_o} f(x)\,dx \right| + \left| \int_{x_o}^{x_o+\delta} f(x)\,dx \right| < \frac{\varepsilon}{2}$$

Después se buscan aproximaciones de las integrales cumpliendo

$$\left| \int_a^{x_o-\delta} f(x)\,dx \right| < \frac{\varepsilon}{4} \qquad \left| \int_{x_o+\delta}^{b} f(x)\,dx \right| < \frac{\varepsilon}{4}$$

Ejemplo 5.31. Calcula

$$\int_0^1 \frac{\cos(x)}{2\pi \operatorname{sen}(\sqrt{x})}\,dx$$

considerando $e_a = 10^{-12}$.

En este caso la singularidad está en el punto 0 por lo que se deberá buscar un valor δ de forma que si se considera

$$\int_0^1 f(x)\,dx = \underbrace{\int_0^{\delta} f(x)\,dx}_{I_1} + \underbrace{\int_{\delta}^1 f(x)\,dx}_{I_2}$$

las dos integrales I_1 e I_2 cumplan que en valor absoluto son menores que $\epsilon/2$.

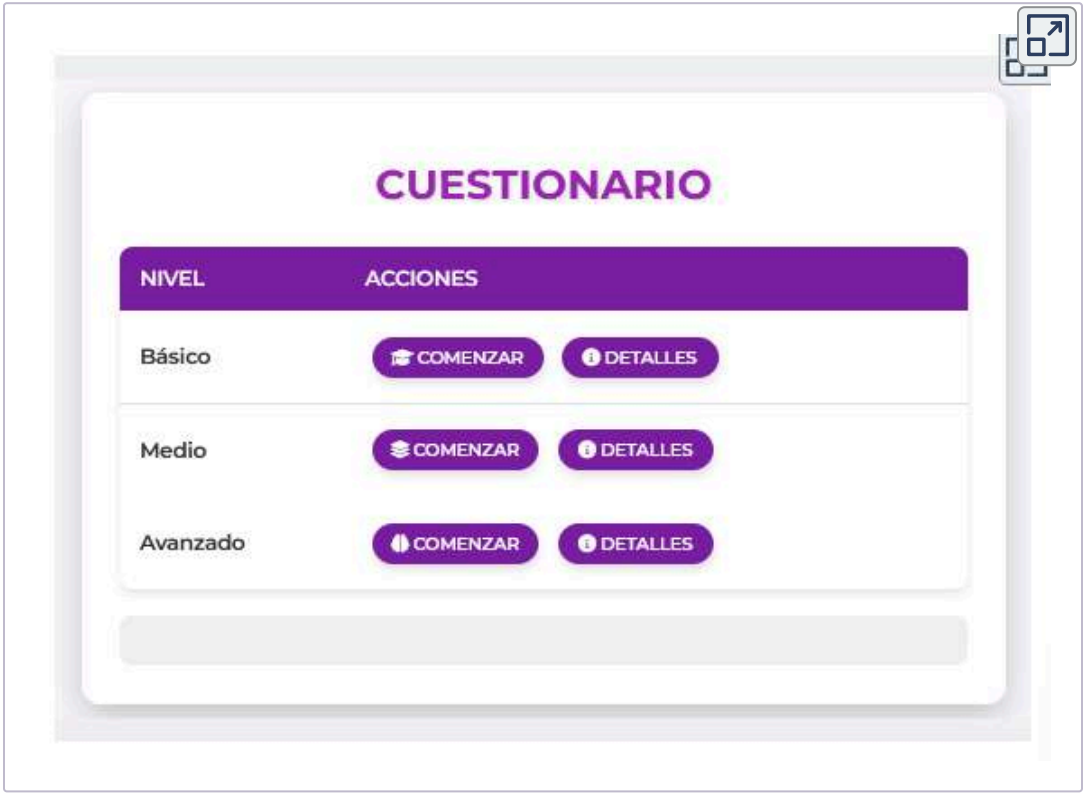
Como $sen(x)$ es mayor que $2x/\pi$ en $[0, \pi/2]$ en el entorno de 0 se tiene que se cumple,

$$\left| \int_0^{\delta} \frac{\cos x}{2\pi \ \ sen\left(\sqrt{x}\right)} \right| \leqslant \int_0^{\delta} \frac{dx}{\pi \ \pi\sqrt{x}} = \frac{\sqrt{\delta}}{2\pi^2} \leqslant \frac{10^{-12}}{2}$$

El valor de δ puede ser cualquier valor menor que $2.4352272758500 61e-23$. Con este valor se calcula I_2 para que sea menor que $\epsilon/2$.

Solución Matlab: 0.302993744656398

5.7 Autoevaluación



Interactivo 5.8. Actividad de autoevaluación

5.8 Ejercicios

1 Se considera la función $f(x) = e^{-2x}$, el intervalo $[0, 1]$, y los nodos: $x_0 = 0$, $x_1 = 1/3$, $x_2 = 2/3$, $x_3 = 1$. Calcula la fórmula de cuadratura a partir de los polinomios base de Lagrange.

Solución

2 Comprueba si las siguientes fórmulas son de tipo interpolatorio:

a. $\int_0^1 f(x)dx \approx f(0) + f(2/3)$

b. $\int_0^1 f(x)dx \approx f(0) + f(1/3) + f(2/3) + f(1)$

Solución

3 Obtén con MATLAB la aproximación de la integral:

$$\int_0^{\pi} e^{\cos(x)} \sin(x)dx$$

utilizando:

a. La regla de los tres octavos ($n = 3$).

b. La regla de Hardy ($n = 6$).

Solución

4 Utiliza la fórmula de los trapecios compuesta dividiendo el intervalo $[0, \pi]$ en tres subintervalos para aproximar la integral $\int_0^{\pi} \sin(x)dx$. Calcula cuántos subintervalos serían necesarios para que el error cumpla $E \leq 0.5 \times 10^{-8}$.

Solución

5 Aproxima el valor de π mediante la integral:

$$\int_0^1 \frac{4}{1+x^2}dx = 4\ arctg\,(1) = \pi$$

utilizando las fórmulas compuestas del trapecio y de Simpson dividiendo el intervalo $[0, 1]$ respectivamente en 1024 y 512 subintervalos donde aplicar la fórmula de cuadratura correspondiente (en ambos casos con 1025 nodos).

Solución

6 Calcula una cota del error al aproximar $\int_0^{0.2} \sin(x)e^{x^2}\,dx$ utilizando la regla de los trapecios.

Solución

7 Calcula el área limitada por la gráfica de la función

$$y = \frac{\sin(x) - 0.2}{1 + x},$$

y la recta $y = 0$ para los valores x entre x_0 y 1 siendo x_0 el punto de corte de la curva y el eje x en el intervalo $[0, 1]$. Para calcular x_0 se debe utilizar el método de Newton con una tolerancia 10^{-10} y para calcular la integral se debe utilizar Simpson compuesta con una tolerancia de 10^{-8}.

Solución

8 Calcula el valor aproximado de la integral:

$$\int_1^{\pi/2} \log(\sin(x))dx,$$

utilizando:

a. La regla del trapecio compuesta con 10 intervalos.

b. Un error absoluto inferior a 10^{-10}.

Solución

9 Calcula las siguientes integrales con un error inferior al indicado utilizando la fórmula adaptativa de los trapecios:

a. $\int_0^{\pi} e^{\cos(x)} \sin(x)dx$, con $E_a \leq 10^{-12}$.

b. $\int_0^4 e^{2x}dx$, con $E_a \leq 10^{-12}, E_r \leq 10^{-10}$.

c. $\int_0^{\pi} (1 + x^2)e^{\sin(x)}dx$, con $E_a \leq 10^{-12}$.

Nota: Las dos primeras integrales se pueden calcular de forma exacta.

Solución

10 Calcula el valor aproximado de las siguientes integrales:

a. $\int_0^\infty e^{-x^2}\cos(x)dx$, con $E_a \leq 10^{-12}$.

b. $\int_0^\infty \log(2+x)e^{-x}dx$, con $E_a \leq 10^{-12}$.

c. $\int_0^\infty \frac{\sin(x)}{1+x^3}dx$, con $E_a \leq 10^{-8}$.

d. $\int_0^\infty \frac{e^{-x^4}}{1+x}dx$, con $E_a \leq 10^{-12}$.

Solución

RESUMEN

En este capítulo, se aborda la integración numérica, una herramienta fundamental para calcular integrales cuando las soluciones analíticas no son posibles o resultan dificiles de obtener. Se describen métodos numéricos y se evalúa su precisión. Además, se estudian métodos específicos para tratar integrales impropias y se analizan los errores asociados a estas aproximaciones.

Los principales aspectos tratados en este tema son los siguientes.

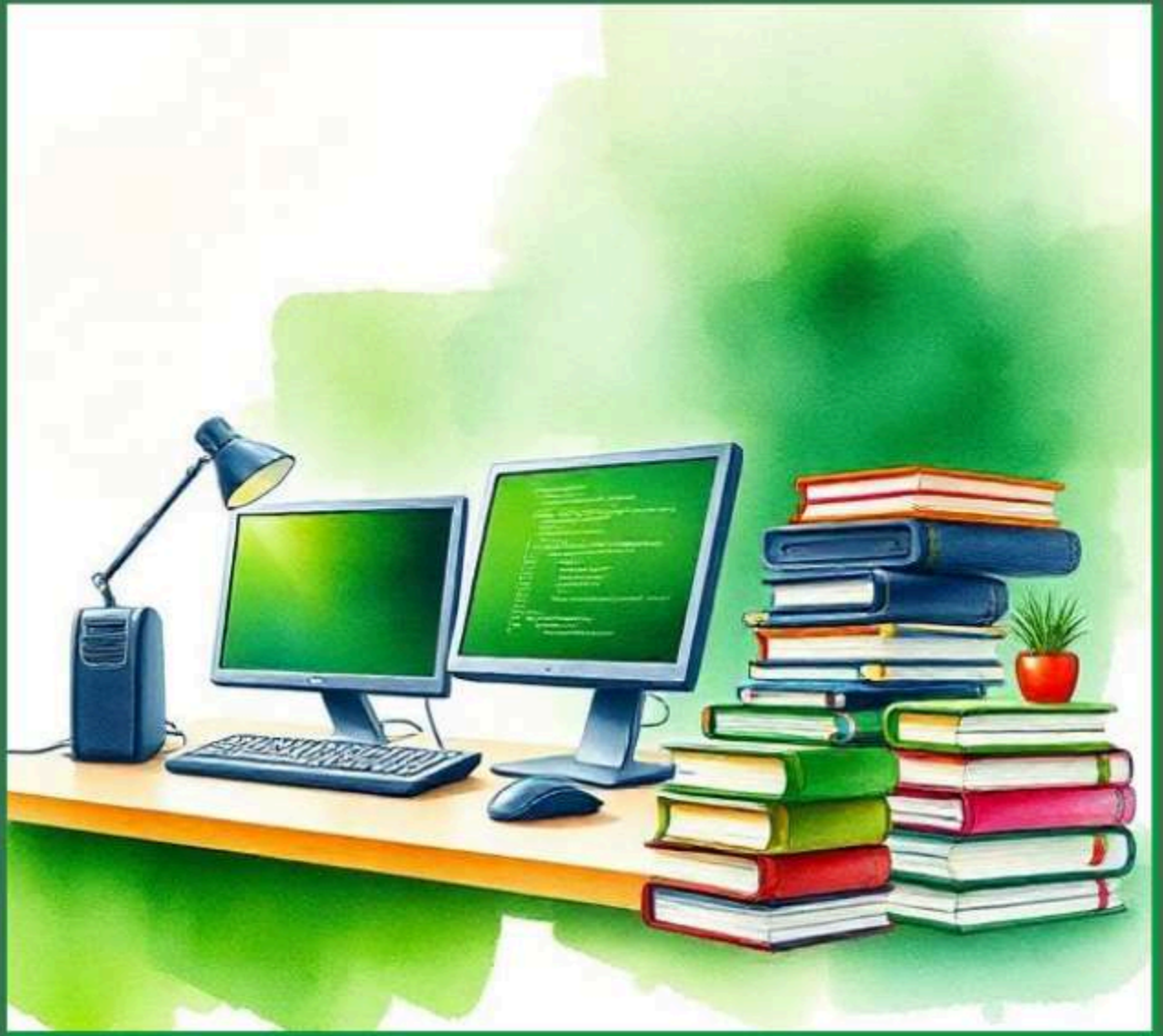
1. **Fórmulas de cuadratura interpolatorias**
   Se muestran las fórmulas de cuadratura que aproximan la integral de una función en un intervalo mediante la interpolación polinómica en un número de nodos.

2. **Fórmulas de Newton-Cotes**
   En particular, se analizan las fórmulas de Newton-Cotes donde los puntos se consideran equidistantes y se incluyen los extremos del intervalo. Se incluyen ejemplos donde se muestra su utilidad y se analizan los errores asociados.

3. **Fórmulas compuestas**
   Se extienden las fórmulas de Newton-Cotes a subintervalos más pequeños, mejorando la precisión para integrales de funciones complejas. Además, se introducen métodos adaptativos que ajustan dinámicamente los intervalos según la función.

4. **Cálculo de integrales impropias**
   Se presentan métodos para evaluar integrales impropias diferenciando entre las de primera y segunda especie. En el primer caso el intervalo de integración no es acotado y en el segundo la función presenta una singularidad dentro del intervalo de integración.

5. **Ejercicios y autoevaluaciones**
   El capítulo incluye ejemplos prácticos y ejercicios para aplicar y comprender los conceptos, así como autoevaluaciones para reforzar el aprendizaje.



6 INTEGRACIÓN NUMÉRICA DE ECUACIONES DIFERENCIALES

Capítulo VI

Integración numérica de ecuaciones diferenciales

6.1 Introducción

En Ingeniería, el comportamiento de un sistema suele describirse mediante ecuaciones que relacionan la tasa de cambio de una magnitud con la propia magnitud y otras variables asociadas. De hecho, muchas leyes físicas se establecen en términos de razones de cambio, como ocurre en fenómenos relacionados con la caída de un cuerpo o la variación de la temperatura, entre otros. Estas relaciones se representan comúnmente mediante ecuaciones diferenciales ordinarias (EDOs), unas ecuaciones que vinculan una función de una única variable independiente con sus derivadas, describiendo formalmente cómo cambia dicha función respecto de esa variable.

Dado que las EDOs suelen ser complejas y a menudo no tienen solución explícita o resulta difícil de obtener, se necesita recurrir a métodos numéricos para aproximarlas.

En este capítulo se estudiará la resolución numérica de dos problemas asociados a las ecuaciones diferenciales ordinarias:

1. el problema de Cauchy o problema de valor inicial.
2. el problema de contorno.

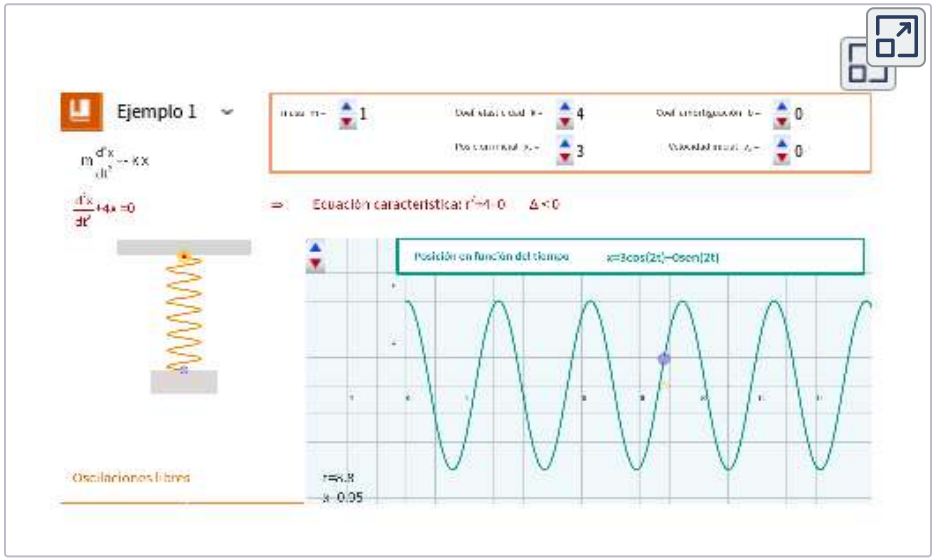
En un problema de valor inicial las condiciones se establecen sobre el instante inicial y en un problema de contorno, las condiciones se imponen tanto en el extremo inicial como en el final de un intervalo. Para más información, se puede consultar [9] y [6].

Como ejemplo de ecuación diferencial, se muestra en el interactivo 6.1 un ejemplo de aplicación para el análisis de un sistema masa-resorte. Este sistema está compuesto por una masa m unida a un resorte que ejerce una fuerza proporcional al desplazamiento, según la ley de Hooke. Si además se considera una fuerza de amortiguamiento proporcional a la velocidad, se obtiene la siguiente ecuación diferencial de segundo orden:

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

donde:

- $x(t)$ es el desplazamiento de la masa en función del tiempo,
- m es la masa,
- c es el coeficiente de amortiguamiento,
- k es la constante del resorte.



Interactivo 6.1. Sistema muelle-resorte. Escena tomada de Red Digital Descartes Autora: Elena E. Álvarez.

En el siguiente ejemplo, las ecuaciones diferenciales modelizan al vaciado de un tanque. Asumiendo que el flujo de salida obedece la ley de Torricelli, la velocidad de vaciado es proporcional a la raíz cuadrada de la altura del líquido. Esto lleva a una ecuación diferencial de la forma:

$$\frac{dh}{dt} = -k\sqrt{h}$$

donde:

- $h(t)$ representa la altura del líquido en el tanque en función del tiempo,
- k es una constante positiva que depende del área del orificio de salida y de la aceleración debida a la gravedad.

Interactivo 6.2. Vaciado de un tanque. Escena tomada de Red Digital Descartes Autora: Elena E. Álvarez.

6.2 Formulación del problema de valor inicial

Se abordan en primer lugar los problemas de valor inicial (PVI), que plantean una ecuación diferencial ordinaria junto a la condición de que la función solución adopte un valor concreto en un instante dado, de modo que dicha función cumpla simultáneamente la ecuación y esa condición inicial.

El problema de valor inicial, PVI, puede escribirse de forma general como

$$\begin{cases} y'(t) = f(t, y(t)) & t \in [t_o, t_f] \\ y(t_o) = y_0 \end{cases}$$

siendo

$$-\infty < t_0 < t_f < \infty$$

$$f : [t_o, t_f] \, \mathrm{x} \mathbb{R}^n \to \mathbb{R}^n$$

En su forma general, se considera

$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{pmatrix} \qquad f(t, y(t)) = \begin{pmatrix} f_1(t, y_1, y_2, ..., y_n) \\ f_2(t, y_1, y_2, ..., y_n) \\ \vdots \\ f_n(t, y_1, y_2, ..., y_n) \end{pmatrix}$$

$$\begin{aligned} y_1' &= f_1(t, y_1, y_2, ..., y_n) \\ y_2' &= f_2(t, y_1, y_2, ..., y_n) \\ &\vdots \\ y_n' &= f_n(t, y_1, y_2, ..., y_n) \end{aligned}$$

Esta forma general permite escribir muchos problemas. Se muestran algunos ejemplos.

Ejemplo 6.1. Considera el siguiente problema

$$\begin{cases} y''(t) + y(t) = 1 & t \in [0, 2\pi] \\ y(0) = 1,\ y'(0) = 1 \end{cases}$$

Se pide escribir este problema de valor inicial en su forma general.

Llamando

$$y_1 = y \qquad y_2 = y'$$

se puede escribir

$$\begin{cases} y_1' = y_2 \\ y_2' = 1 - y_1 \\ y_1(0) = 1, \quad y_2(0) = 1 \end{cases}$$

Considerando

$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} \qquad f(t, y(t)) = \begin{pmatrix} y_2 \\ 1 - y_1 \end{pmatrix}$$

la ecuación diferencial puede escribirse de la forma

$$\begin{pmatrix} y_1'(t) \\ y_2'(t) \end{pmatrix} = \begin{pmatrix} y_2 \\ 1 - y_1 \end{pmatrix}$$

con las siguientes condiciones en el punto inicial $t_0 = 0$

$$y(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Ejemplo 6.2. Escribe el siguiente problema en su forma general

$$\begin{cases} y''(t) = 2ty(t) + t^2 & t \in [0, 2\pi] \\ y(0) = 1,\ y'(0) = 0 \end{cases}$$

Llamando

$$y_1 = y \qquad y_2 = y'$$

se puede escribir

$$\begin{cases} y_1' = y_2 \\ y_2' = 2ty_1 + t^2 \\ y_1(0) = 1, \quad y_2(0) = 0 \end{cases}$$

Considerando

$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} \qquad f(t, y(t)) = \begin{pmatrix} y_2 \\ 2ty_1 + t^2 \end{pmatrix}$$

la ecuación diferencial puede escribirse de la forma

$$\begin{pmatrix} y_1'(t) \\ y_2'(t) \end{pmatrix} = \begin{pmatrix} y_2 \\ 2ty_1 + t^2 \end{pmatrix}$$

con las siguientes condiciones en el punto inicial

$$y(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Ejemplo 6.3. Se considera la ecuación diferencial de segundo orden de un sistema mecánico vibratorio sometido a una fuerza externa senoidal

$$\frac{d^2x}{dt^2} + w^2x = F_o sen\,(\gamma t)$$

estando las unidades en el sistema internacional. Se considera la posición de reposo y los valores $\omega = 2, F_0 = 20$ y $\gamma = 0.5$. Escribe la ecuación diferencial en la forma general.

La ecuación diferencial es

$$\frac{d^2x}{dt^2} + 4x = 20sen\,(0.5t)$$

Considerando $x_1 = x\,(t) \quad x_2 = x'\,(t)$, se podrá escribir

$$\begin{pmatrix} x_1'\,(t) \\ x_2'\,(t) \end{pmatrix} = \begin{pmatrix} x_2 \\ -4x_1 + 20sen(0.5t) \end{pmatrix}$$

Las condiciones en el punto inicial son $\begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

6.3 Existencia y unicidad

Para asegurar que un problema de valor inicial (PVI) tenga una única solución, se considera el siguiente teorema de existencia y unicidad.

Dado el problema

$$\begin{cases} y'\,(t) = f\,(t, y\,(t)) & t \in [t_o, t_f] \\ y\,(t_o) = y_0 \end{cases}$$

con $f : [t_o, t_f]\,\mathrm{x}\mathbb{R}^n \to \mathbb{R}^n$ continua y acotada en un dominio que contenga a (t_0, y_o) entonces el problema tiene al menos una solución.

Para que la solución sea única se puede exigir que la función cumpla la condición de Lipschitz para la variable y, es decir, exista $L > 0$ de forma que para todo t en $[t_0, t_f]$

$$\|f(t, y_1) - f(t, y_2)\| \leq L \|y_1 - y_2\|$$

6.4 Métodos de Runge-Kutta

En este apartado se estudiarán los métodos de Runge-Kutta de un solo paso, donde la solución aproximada se obtiene progresivamente de un punto al siguiente a partir de la información de la derivada en puntos intermedios dentro de cada intervalo, sin emplear los valores de pasos previos. Así, dado el problema

$$y'(t) = f(t, y(t)) \qquad t \in [t_o, t_f]$$

la integración numérica del problema consiste en determinar unos instantes

$$t_o < t_1 < t_2 < \ldots < t_N = t_f \quad h_n = t_{n+1} - t_n$$

y valores

$$\{y_o,\ y_1,\ y_2, \ldots,\ y_N\}$$

de forma que a partir de

$$y(t_n) \approx y_n$$

se pueda obtener la aproximación de y_{n+1}.

En la imagen siguiente se muestra, para una ecuación diferencial ordinaria de primer orden, la solución analítica de un problema de valor inicial y la solución aproximada con 7 pasos considerando el intervalo $[0, 4]$. El método numérico deberá indicar cómo obtener cada punto de la solución numérica a partir del anterior. En el caso del gráfico, el avance al siguiente punto utiliza la pendiente en el punto anterior.

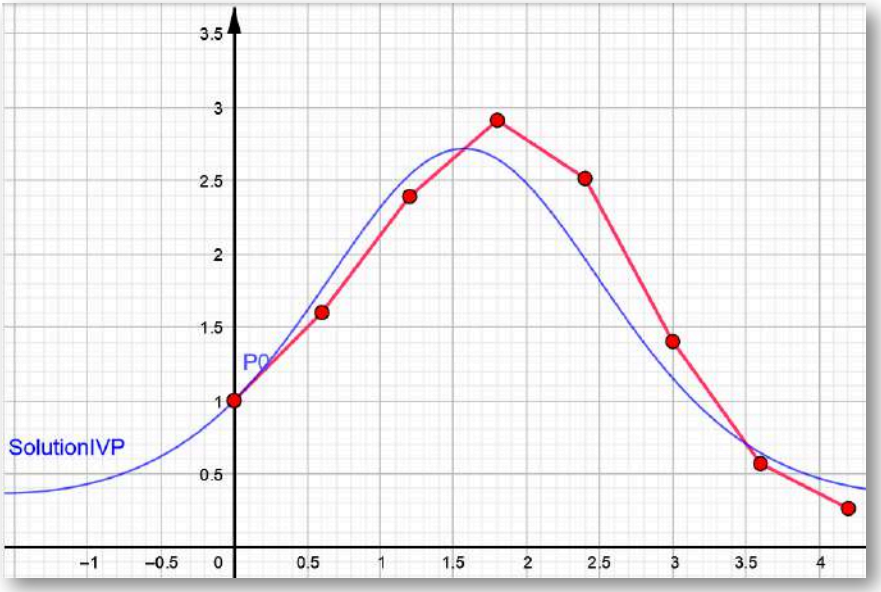


Figura 6.1. Representación de la solución exacta y aproximada de un PVI

A continuación, se presentarán distintos métodos de Runge-Kutta[9] [10], una familia de métodos numéricos ampliamente utilizados para resolver problemas de valor inicial en ecuaciones diferenciales ordinarias.

[9] Carl David Tolmé Runge (30 de agosto de 1856 - 3 de enero de 1927) fue un matemático y físico alemán conocido por su trabajo en análisis numérico y ecuaciones diferenciales. Nació en Bremen, Alemania, y estudió matemáticas y física en las universidades de Múnich y Berlín, obteniendo su doctorado en 1880. Runge es reconocido por desarrollar, junto con Martin Kutta, los métodos de Runge-Kutta para resolver ecuaciones diferenciales ordinarias de manera numérica. También realizó importantes contribuciones a la espectroscopía y la teoría de aproximación. Fue profesor en la Universidad de Hannover y, más tarde, en la Universidad de Gotinga.

[10] Martin Wilhelm Kutta (3 de noviembre de 1867 - 25 de diciembre de 1944) fue un matemático alemán conocido principalmente por su trabajo conjunto con Carl Runge en el desarrollo de los métodos de Runge-Kutta para la solución numérica de ecuaciones diferenciales ordinarias. Nacido en Pitschen (entonces parte de Prusia), estudió en las universidades de Breslavia y Múnich, donde trabajó con destacados matemáticos de la época. Además de su contribución a los métodos de Runge-Kutta, Kutta realizó estudios en mecánica y aerodinámica, desarrollando la teoría de sustentación conocida como el "teorema de Kutta-Joukowski". Fue profesor en la Universidad Técnica de Stuttgart.

Su idea principal es aproximar la solución mediante una combinación ponderada de evaluaciones de la función derivada en puntos intermedios dentro de un intervalo. Estas evaluaciones permiten calcular el siguiente valor de la solución con más exactitud, sin necesidad de derivadas superiores.

6.4.1 Método de Euler

El método de Euler[11] es el procedimiento más sencillo para aproximar la solución de un problema de valor inicial. Se analiza cómo obtener a partir de $y(t_n) \approx y_n$ el valor y_{n+1}. Dado que,

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f\left(t, y\left(t\right)\right) dt$$

se puede usar la fórmula de cuadratura con un nodo para obtener un valor aproximado de la integral de la forma siguiente

$$y(t_{n+1}) \approx y_{n+1} = y_n + h_n f\left(t_n, y_n\right)$$

La idea básica por tanto consiste en avanzar desde un valor conocido paso a paso, utilizando la pendiente de la función para estimar el siguiente valor.

En la siguiente figura se muestra gráficamente cómo obtener para un paso constante $h = 1$ el punto (t_1, y_1) a partir de (t_0, y_0) y (t_2, y_2) a partir de (t_1, y_1) conocidas las derivadas en estos dos puntos.

[11] Leonhard Euler (1707–1783), nacido en Basilea, Suiza, es considerado uno de los matemáticos más influyentes de la historia. Sus aportaciones en cálculo infinitesimal, análisis matemático, teoría de números y mecánica sentaron las bases para el desarrollo de múltiples disciplinas científicas.

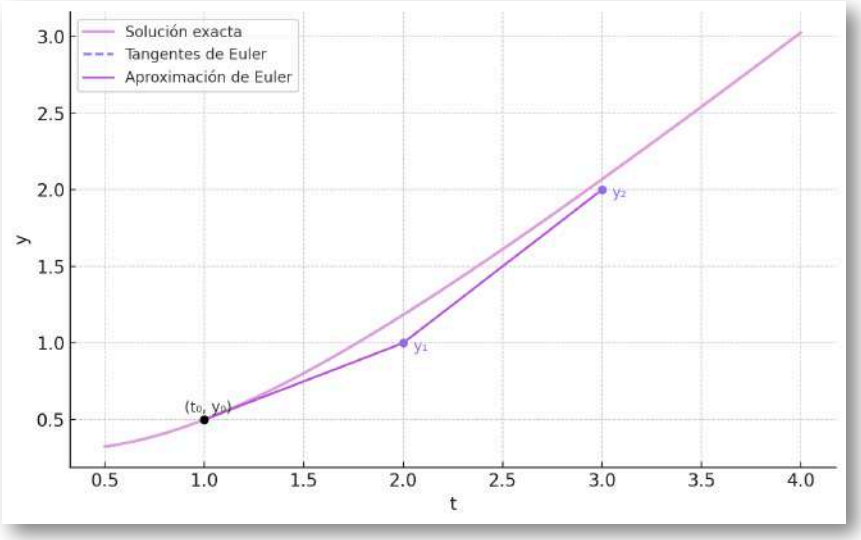


Figura 6.2. Primeros pasos del método de Euler.

Video 6.1. Método de Euler

Ejemplo 6.4. Considera el problema de valor inicial siguiente

$$\frac{dy}{dt} = 0.7y - t^2 + 1 \quad , \quad y\,(1) = 1 \quad , \quad t \in [1, 2]$$

Encuentra la solución aproximada por el método de Euler con 10 pasos tomando una longitud del paso $h = \frac{2-1}{10}$.

En este caso, $f(t,y) = 0.7y - t^2 + 1$, el intervalo es $[1,2]$ y la condición inicial es $y(1) = 1$.

Se calculan los cuatro primeros pasos de la aproximación de la curva solución del PVI. Se tiene $t_0 = 1, y_0 = y(1) = 1$.

$$t_1 = 1.1 \quad y\,(1.1) \approx y_1 = y_0 + 0.1 \cdot f\,(1,\ y_0) \approx 1.07$$

$$t_2 = 1.2 \quad y\,(1.2) \approx y_2 = y_1 + 0.1 \cdot f\,(1.1,\ y_1) \approx 1.1239$$

$$t_3 = 1.3 \quad y\,(1.3) \approx y_3 = y_2 + 0.1 \cdot f\,(1.2,\ y_2) \approx 1.15857$$

$$t_4 = 1.4 \quad y\,(1.4) \approx y_4 = y_3 + 0.1 \cdot f\,(1.3,\ y_3) \approx 1.17067$$

...

Nótese que en cada paso se necesita el valor $y(t_n)$, por lo que cuanto más cercano sea el valor y_n al real, $y(t_n)$, más preciso será el método. Así, para calcular y_1 se utiliza el valor y_0, para calcular y_2 se sustituye el valor exacto $y(t_1)$ desconocido por su valor aproximado y_1, para calcular y_3, se sustituye el valor $y(t_2)$ por su valor aproximado y_2, y así sucesivamente.

Se muestra seguidamente el código Matlab para obtener los cálculos de la solución aproximada del PVI anterior con 10 pasos.

```
clear all
f=@(t,y) 0.7*y-t.^2+1;   %Función
a=1;b=2;y0=1;   %Intervalo
N=10;  %Pasos (N+1 puntos)
h=(b-a)/N;   %Longitud del paso
t(1)=a;y(1)=y0;
for n=1:N
    t(n+1)=t(n)+h;
    % dy/dt=f(t,y)
    y(n+1)=y(n)+h*f(t(n),y(n));
end
[t' y']
```

t	1	1.1	1.2	1.3	1.4
y	1.00000000	1.07000000	1.12390000	1.15857300	1.17067311

t	1.5	1.6	1.7
y	1.1566202277	1.112583643639000	1.034464498693729

t	1.8	1.9	2.0
y	0.917877013602290	0.758128404554450	0.550197392873262

Para dibujar tanto los valores obtenidos como la solución exacta en una misma gráfica en Matlab, se añadiría al código anterior el siguiente.

```
%Representación de los valores aproximados obtenidos
plot(t,y,'o')
xlabel(['Metodo de Euler con N=',num2str(N),'
pasos'])
hold on
%Solución simbólica dada por Matlab
%Se define la función en simbólico
syms s(u)
%Se define la ecuación diferencial
eqn=diff(s)==f(u,s)
%Se define la condición inicial
cond=s(a)==y0
%Se resuelve con el comando dsolve
sol(u)=dsolve(eqn,cond)
%Representación de la solucion exacta
t1=linspace(a,b,50);
plot(t1,sol(t1))
hold off
```

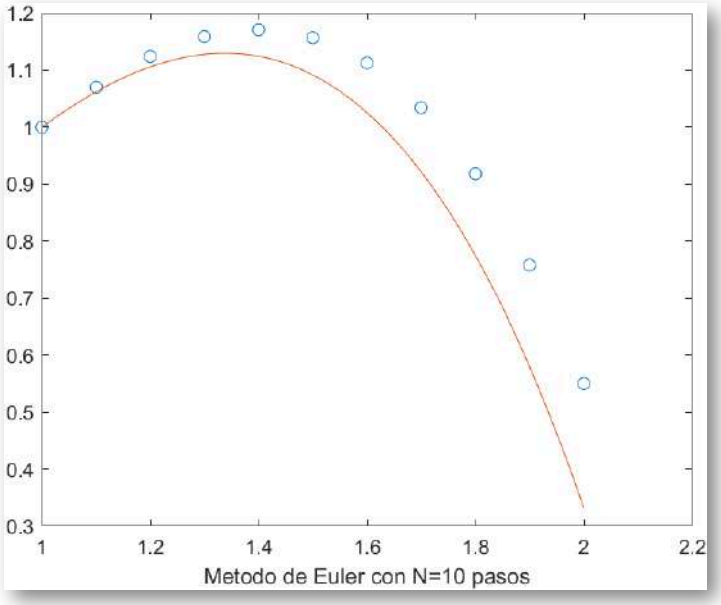


Figura 6.3. Gráfica de la solución exacta y aproximada

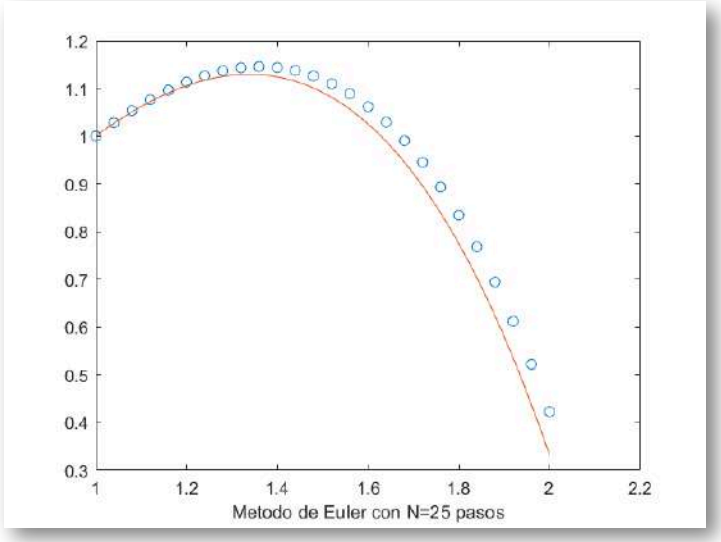
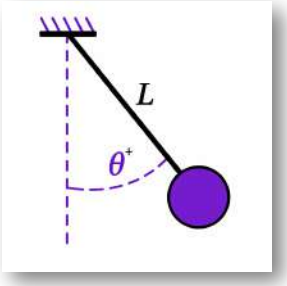


Figura 6.4. Solución aproximada con 25 pasos

Ejemplo 6.5. El modelo matemático de un péndulo es $\theta''(t) + \frac{g}{L} sen(\theta(t)) = 0$. Se considera que la bola es lanzada desde $\theta(0) = \pi/6$ rad con una velocidad inicial de $\theta'(0) = -4rad/s$, que $g = 9.81m/s^2$ y $L = 1m$. Se desea calcular la posición y velocidad angular en el primer segundo usando un paso de 0.5 segundos utilizando el método de Euler.



En forma general, se puede escribir el problema,

$$y_1 = \theta \qquad y_2 = \theta'$$
$$y = \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ -\frac{g}{L} sen(y_1) \end{pmatrix} \quad y(0) = \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} \frac{\pi}{6} \\ -4 \end{pmatrix}$$

Escribiendo el cálculo con tres cifras decimales, se tendrá

$$y(0.5) = y(0) + hf(0, y(0))$$
$$y(0.5) = \begin{pmatrix} \frac{\pi}{6} \\ -4 \end{pmatrix} + h \begin{pmatrix} -4 \\ -9.81 sen\left(\frac{\pi}{6}\right) \end{pmatrix} = \begin{pmatrix} -1.476 \\ -6.452 \end{pmatrix}$$

$$y(1) = y(0.5) + hf(0.5, y(0.5))$$
$$y(1) = \begin{pmatrix} -1.476 \\ -6.452 \end{pmatrix} + h \begin{pmatrix} -6.452 \\ -9.81 sen(-1.476) \end{pmatrix} = \begin{pmatrix} -4.703 \\ -1.569 \end{pmatrix}$$

```
f=@(t,y) [y(2);-9.81*sin(y(1))]
y0=[pi/6;-4];h=0.5;t0=0;
%Valor en 0.5
y1=y0+h*f(t0,y0)
%Valor en 1
t1=t0+h; y2=y1+h*f(t1,y1)  %Posición -4.703 rad,
velocidad -1569 rad/s
```

Ejemplo 6.6. La ecuación diferencial siguiente se utiliza para modelar la deflexión del mástil de un bote sujeto a la fuerza del viento

$$\frac{d^2y}{dz^2} = \frac{f(z)}{2EI}(L-z)^2$$

donde $E = 1.25 \cdot 10^8$ es el módulo de elasticidad, $I = 0.05$ es el momento de inercia, $L = 30$ es la longitud del mástil y f la fuerza del viento considerando que se trabaja en el sistema internacional de unidades. Se sabe que la fuerza del viento varía con la altitud de acuerdo a la siguiente relación

$$f(z) = \frac{200z}{5+z}e^{-\frac{z}{15}}$$

Calcula la deflexión para $z = 1.5$, si para $z = 0$ se tiene $y = 0, y'(0) = 0$ utilizando el método de Euler considerando un paso $h = 0.1$.

En este caso, al ser la ecuación diferencial de orden dos,

$$\frac{d^2y}{dz^2} = \frac{200z}{2 \cdot 1.25 \cdot 10^8 \cdot 5 \cdot 10^{-2}(5+z)}e^{-\frac{z}{15}}(30-z)^2$$

$$\frac{d^2y}{dz^2} = \frac{z}{6.25 \cdot 10^4(5+z)}e^{-\frac{z}{15}}(30-z)^2$$

el problema se escribe como un sistema de ecuaciones. Se considera

$$y_1 = y \qquad y_2 = y'$$

y se tiene

$$\begin{pmatrix} y'_1 \\ y'_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ \frac{z}{6.25 \cdot 10^4(5+z)}e^{-\frac{z}{15}}(30-z)^2 \end{pmatrix} \qquad \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Se calcula la deflexión para 1.5 utilizando el método de Euler con un paso $h = 0.1$, dando por tanto $N = 15$ pasos.

Aplicando el método de Euler con N pasos, se tendrá que $h = 0.6/N$. El método de Euler, $y(n+1) = y(n) + hf(t(n), y(n))$, aplicado a cada componente de y será

$$
\begin{aligned}
y_1(n+1) &= y_1(n) + h\, y_2(n) \\
y_2(n+1) &= y_2(n) + h \left[\frac{z(n)}{6.25 \cdot 10^4 (5 + z(n))} e^{-\frac{z(n)}{15}} (30 - z(n))^2 \right] \\
z(n+1) &= z(n) + h
\end{aligned}
$$

En el código, se trabaja vectorialmente teniendo en cuenta que tanto y como f son vectores adaptando el código del ejemplo 6.4.

```
clear all
%Definición de la función
f=@(z,y) [y(2);z.*exp(-z/15)*10^-4.*(30-z).^2./(6.25*
(5+z))]
h=0.1; % Longitud del paso
N=15; % Número de pasos
format long
z(1)=0;
y(:,1)=[0;0];
for n=1:N
    z(n+1)=z(n)+h;
    y(:,n+1)=y(:,n)+h*f(z(n),y(:,n));
end
%Resultados
[z' y(1,:)' y(2,:)']
%Representación de la solución
plot(z,y,'*')
legend('y1 aprox', 'y2 aprox')
```

La deflexión para 1.5 es $y(16) = 0.001052773045811$ y su velocidad es 0.002241031532091.

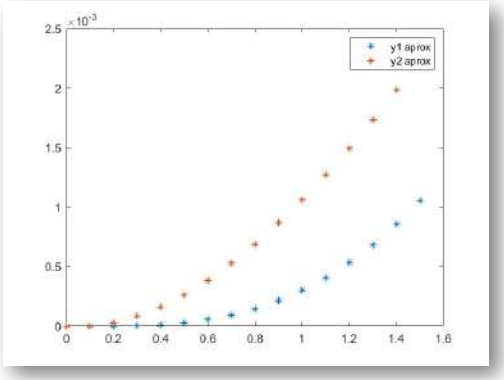


Figura 6.5. Solución para $N = 15, h = 0.1$

Ejemplo 6.7. Se considera el sistema mecánico que se muestra en la figura 6.6. con las siguientes ecuaciones de movimiento

$$
\begin{aligned}
&J_1\theta_1'' + (k_1 + k_2)\,\theta_1 - k_2\theta_2 = 0\\
&J_2\theta_2'' - k_2\theta_1 + k_2\theta_2 = 0\\
&k_2 = 2,\ \ k_1 = 1,\ \ \ J_2 = 2,\ \ \ J_1 = 1
\end{aligned}
$$

Se sabe que en el instante inicial ($t = 0$ s) los discos están en la posición $\theta_1 = 0.1$ radianes y $\theta_2 = 0.2$ radianes, respectivamente y la velocidad angular inicial de ambos discos es de 0 radianes por segundo. Se pide:

- Plantea el sistema de ecuaciones de primer orden a resolver.
- Escribe el algoritmo de Euler para realizar N pasos.
- Aproxima las posiciones y velocidades angulares de los discos en $t = 0.6$ seg. tomando $h = 0.1$.

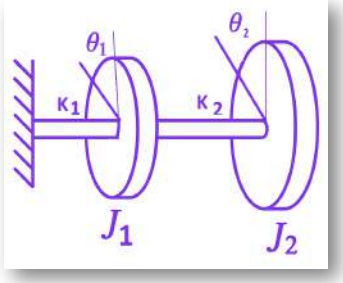


Figura 6.6. Sistema mecánico

Las ecuaciones a resolver son las siguientes,

$$\theta_1^{''} + 3\theta_1 - 2\theta_2 = 0$$
$$2\theta_2^{''} - 2\theta_1 + 2\theta_2 = 0$$

Considerando $y_1 = \theta_1, y_2 = \theta_2, y_3 = \theta_1^{'}, y_4 = \theta_2^{'}$, el sistema a resolver es:

$$\begin{pmatrix} y_1' \\ y_2' \\ y_3' \\ y_4' \end{pmatrix} = \begin{pmatrix} y_3 \\ y_4 \\ -3y_1 + 2y_2 \\ y_1 - y_2 \end{pmatrix} \qquad \begin{pmatrix} y_1(0) \\ y_2(0) \\ y_3(0) \\ y_4(0) \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0 \\ 0 \end{pmatrix}$$

Aplicando el método de Euler con N pasos, se tendrá que $h = 0.6/N$. El método de Euler, $y(n+1) = y(n) + hf(t(n), y(n))$, aplicado a cada componente de y será

$$\begin{aligned} y_1(n+1) &= y_1(n) + h\, y_3(n) \\ y_2(n+1) &= y_2(n) + h\, y_4(n) \\ y_3(n+1) &= y_3(n) + h\,(-3y_1(n) + 2y_2(n)) \\ y_4(n+1) &= y_4(n) + h\,(y_1(n) - y_2(n)) \\ t(n+1) &= t(n) + h \end{aligned}$$

En el código, se trabaja vectorialmente teniendo en cuenta que tanto y como f son vectores. Se definen los datos del problema.

```
clear all
%Definición de la función
f=@(t,y) [y(3);y(4);-3*y(1)+2*y(2);y(1)-y(2)]
h=0.1; %Longitud del paso
N=6; %Número de pasos
format long
t(1)=0;
y(:,1)=[0.1;0.2;0;0];
```

Se calculan los distintos pasos para obtener la solución aproximada.

```
for n=1:N
    t(n+1)=t(n)+h;
    y(:,n+1)=y(:,n)+h*f(t(n),y(:,n));
end
%Resultados
[t' y(1,:)' y(2,:)' y(3,:)'  y(4,:)']
%Representación de la solución
plot(t,y,'*')
legend('theta1 aprox', 'theta2 aprox', 'theta1''
aprox', 'theta2'' aprox')
```

Se muestra a continuación los datos obtenidos en una tabla teniendo en cuenta que las componentes primera, segunda, tercera y cuarta de y representan, respectivamente, $\theta_1, \theta_2, \theta_1'$ y θ_2'

t	θ_1	θ_2	θ_1'	θ_2'
0	0.1	0.2	0	0
0.1	0.10000000	0.20000000	0.01000000	-0.01000000
0.2	0.10100000	0.19900000	0.02000000	-0.02000000
0.3	0.10300000	0.19700000	0.02950000	-0.02980000
0.4	0.10595000	0.19402000	0.03800000	-0.03920000
0.5	0.10975000	0.19010000	0.04501900	-0.04800700
0.6	0.11425190	0.18529930	0.05011400	-0.05604200

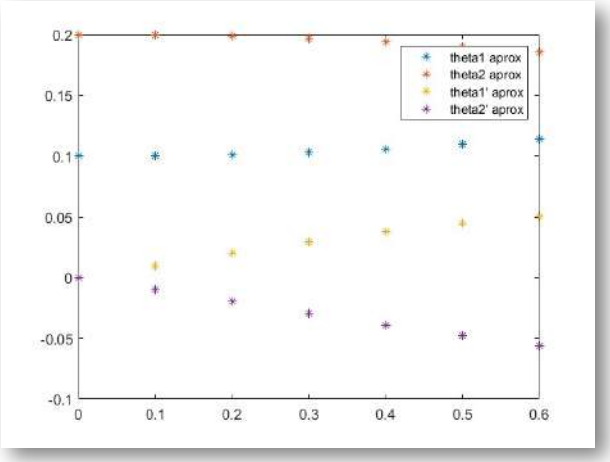
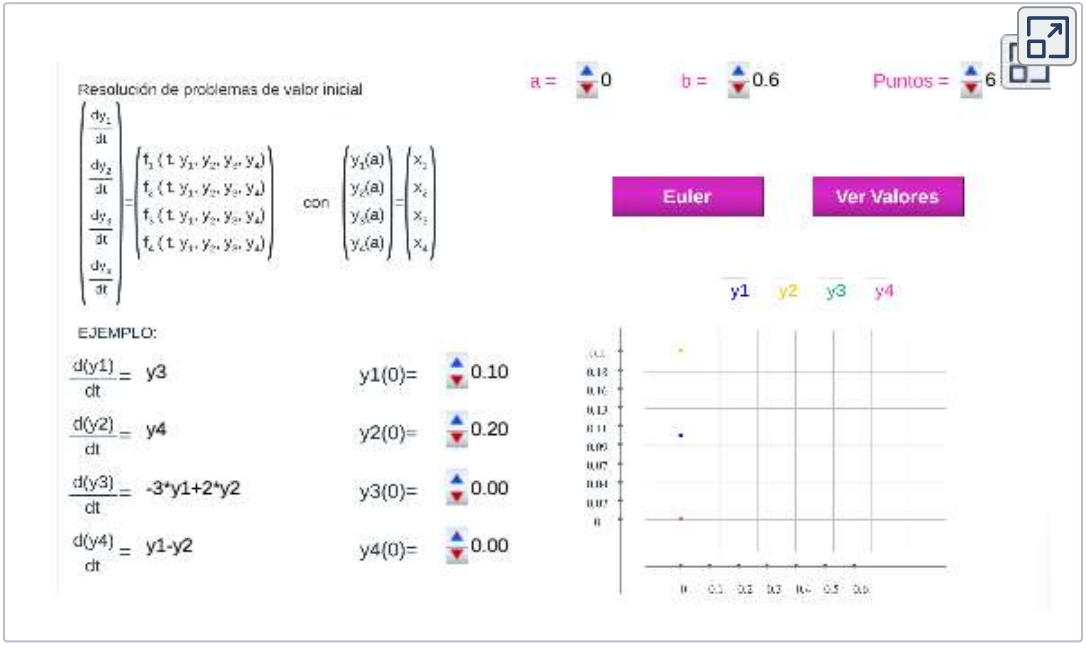


Figura 6.7. Solución del problema

El interactivo muestra cómo evoluciona la aproximación de la solución (en azul) al aumentar el número de pasos, y permite ajustar tanto la EDO como las condiciones iniciales.



Interactivo 6.3. Método de Euler

6.4.2 Método de Heun

Para mejorar la precisión frente al esquema de Euler explícito, se puede recurrir a una regla de cuadratura que incorpore información en ambos extremos del intervalo como es el método de Heun [12].

Partiendo de la igualdad exacta

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f\left(t, y\left(t\right)\right) dt$$

se aproxima la integral mediante la regla del trapecio que para una función g se expresa

$$\int_{t_n}^{t_n+h_n} g\left(t\right) dt \approx \frac{h_n}{2}\left(g\left(t_n\right) + g\left(t_{n+1}\right)\right)$$

Si se toma $g(t) = f(t, y(t))$ y se sustituye $y_n \approx y\left(t_n\right)$, se obtiene

$$y_{n+1} = y_n + \frac{h_n}{2}\left[f\left(t_n, y_n\right) + f\left(t_{n+1}, y(t_{n+1})\right)\right]$$

Como $y(t_{n+1})$ aparece en el término derecho, se reemplaza por su predicción por Euler

$$y_{n+1} = y_n + \frac{h_n}{2}\left[f\left(t_n, y_n\right) + f\left(t_n + h_n, \underbrace{y_n + h_n f\left(t_n, y_n\right)}_{Euler}\right)\right]$$

[12] Carl Friedrich Heun (1859-1932) fue un matemático alemán conocido por sus estudios sobre ecuaciones diferenciales de segundo orden y por desarrollar las funciones de Heun, que generalizan las funciones hipergeométricas. Su trabajo tiene aplicaciones en física teórica, especialmente en mecánica cuántica y relatividad.

Por lo tanto, el valor de y_{n+1} se obtiene a partir del punto (t_n, y_n) considerando la media ponderada de la pendiente en ese punto, y la pendiente en el punto que se obtendría aplicando el método de Euler, $(t_n + h_n, y_n + h_n f(t_n, y_n))$.

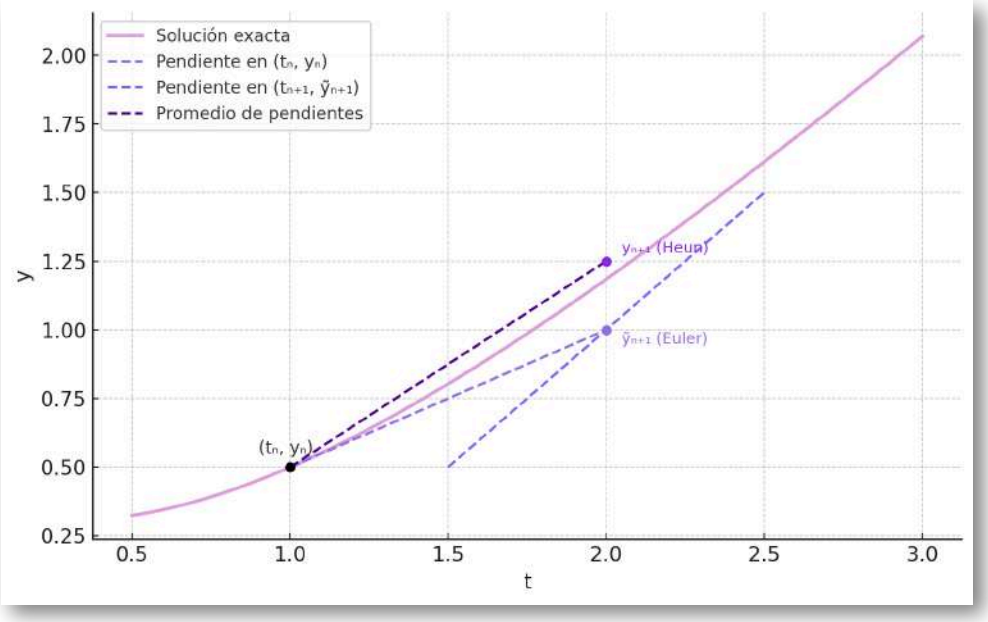


Figura 6.8. Calculo del primer paso por el método de Heun

Ejemplo 6.8. Considera el problema de valor inicial siguiente:

$$\frac{dy}{dt} = t - y \qquad y\,(1) = 0.5$$

Tomando un paso $h = 0.8$, obtener el valor aproximado de $y(1.8)$ utilizando el método de Heun.

En este caso se tiene que $f(t, y) = t - y, t_0 = 1$ e $y_0 = 0.5$. Tomando $h = 0.8$, en el punto $t = 1.8$ la solución aproximada se obtendrá como el promedio entre la pendiente en el punto inicial $(1, 0.5)$ y el previsto por el método de Euler para $t = 1.8$.

$$y_{euler} = 0.5 + 0.8 \cdot f\,(1,\, 0.5) = 0.5 + 0.8\,(1 - 0.5) = 0.9 \rightarrow (1.8, 0.9)$$

El valor aproximado utilizando el método de Heun es

$$y\,(1.8) = 0.5 + 0.8 \cdot \frac{1}{2}\,[f\,(1,\, 0.5) + f\,(1.8, 0.9)] =$$

$$= 0.5 + \frac{1}{2}\,[0.5 + 0.9] = 1.2$$

En cada paso, considerando h_n constante e igual a h, este método podría escribirse de la forma siguiente:

$$K_1 = f\,(t_n, y_n)$$

$$K_2 = f\,(t_n + 1 \cdot h, y_n + 1 \cdot hK_1)$$

$$y_{n+1} = y_n + h\left[\frac{1}{2}\,K_1 + \frac{1}{2}\,K_2\right]$$

asociando este algoritmo al siguiente tablero

0	0	0
1	1	0
	1/2	1/2

Video 6.2. Método de Heun

Ejemplo 6.9. Considera el siguiente PVI:

$$y' = (t-y)/2 \quad , \quad t \in [0,3] \quad y(0) = 1$$

Calcula por el método de Heun con 30 pasos la solución aproximada y comparar con la solución exacta del PVI que es $y(t) = 3e^{-t/2} - 2 + t$.

A continuación se muestra el código Matlab para calcular los pasos del método de Heun.

```
clear all
f=@(t,y) (t-y)/2   %Función
t0=0;tf=3;   %Intervalo
N=30;   %Número de Pasos
h=(tf-t0)/N; %Longitud del paso
%Condición inicial
t(1)=t0;y(1)=1;
%Se realizan N pasos
for n=1:N
    t(n+1)=t(n)+h;
    %Método de Heun
    k1=f(t(n),y(n));
    k2=f(t(n)+h,y(n)+h*k1);
    y(n+1)=y(n)+(k1+k2)*h/2;
end
%Se representan los valores obtenidos
plot(t,y,'*')
%Se representa la solución exacta
hold on
t1=linspace(t0,tf,50);
plot(t1,3*exp(-t1/2)-2+t1)
hold off
%Se comparan los valores obtenidos entre
%la solución exacta y la obtenida
yexac=3*exp(-t/2)-2+t;
[y' yexac'  (y-yexac)']
```

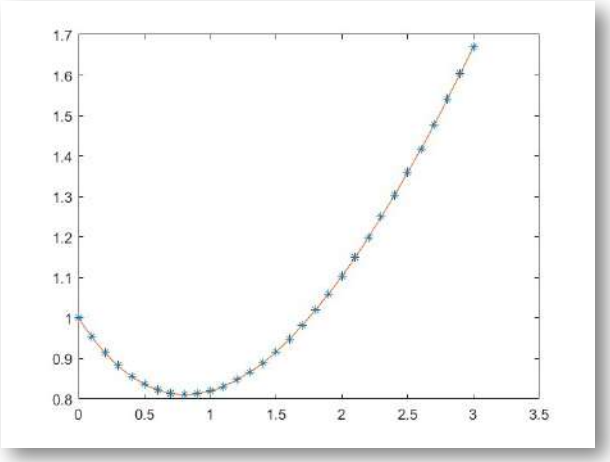


Figura 6.9. Representación de la solución de la EDO

Ejemplo 6.10. Calcula y dibuja la solución del problema

$$y'' + 7seny + 0.1\cos t = 0 \quad t \in [0,5]$$

$$y(0) = 0 \qquad y'(0) = 1$$

Llamando $y_1 = y,\ y_2 = y'$, se tiene

$$\begin{aligned} y_1' &= y_2 & y_2' &= -7seny - 0.1\cos t \\ y_1(0) &= 0 & y_2(0) &= 1 \end{aligned}$$

En este caso,

$$\begin{pmatrix} y_1'(t) \\ y_2'(t) \end{pmatrix} = \begin{pmatrix} y_2 \\ -7sen(y_1) - 0.1cos(t) \end{pmatrix} \qquad \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

El código para resolver la EDO aplicando el método de Heun con matlab considerando $N = 30$ pasos se muestra a continuación.

```
clear all
%Función
f=@(t,y) [y(2);-7*sin(y(1))-0.1*cos(t)]
t0=0;tf=5;%Intervalo
N=30;%Pasos
h=(tf-t0)/N; %Longitud del paso
%Condición inicial
t(1)=t0;y0=[0;1];y(:,1)=y0;
for n=1:N %
    t(n+1)=t(n)+h;
    k1=f(t(n),y(:,n));
    k2=f(t(n)+h,y(:,n)+h*k1);
    y(:,n+1)=y(:,n)+(k1+k2)*h/2;
end
%Representación de la solución aproximada obtenida
plot(t,y,'*')
hold on
legend('y_1','y_2')
xlabel(['Metodo de Heun con N=',num2str(N),' pasos'])
```

En la primera componente de la solución está la aproximación de $y(t)$, en la segunda, la aproximación de la derivada.

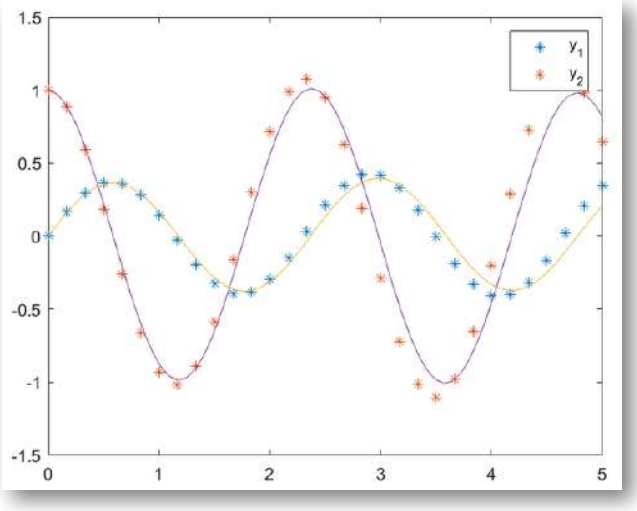


Figura 6.10. Representación de la solución de la EDO

6.4.3 Método general de Runge Kutta

Dado el PVI siguiente

$$y'(t) = f(t, y(t)) \qquad t \in [t_o, t_f]$$
$$y(t_o) = y_0$$

se muestra cómo calcular, $y(t_{n+1})$ a partir de $y(t_n)$. Para ello, se considera la siguiente igualdad

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_n+h_n} y'(t)\, dt$$

Teniendo en cuenta que en el PVI se cumple $y' = f(t, y(t))$

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt$$

la aproximación de la integral de $f(t, y(t))$ se puede obtener utilizando una fórmula de cuadratura de m nodos

$$y(t_{n+1}) \approx y_n + h_n \sum_{i=1}^{m} b_i f(t_n + c_i h_n, y(t_n + c_i h_n))$$

donde b_i son los pesos y los c_i son los números entre 0 y 1 que definen los nodos

$$t_{n,i} = t_n + c_i h_n$$

en el intervalo $[t_n, t_{n+1}]$.

En la imagen se representa un ejemplo con $m = 6$ donde se muestran los valores c_i y $t_{n,i}$.

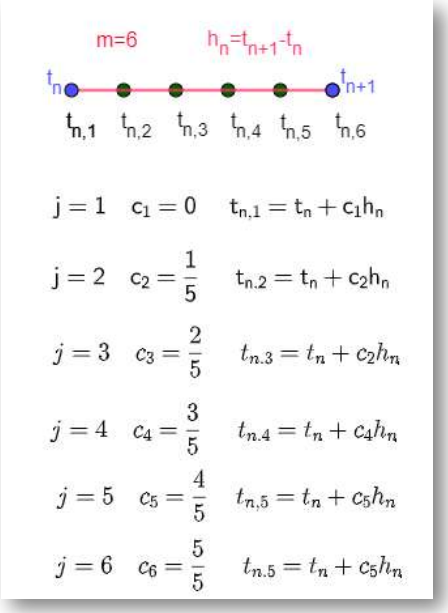


Figura 6.11. Representación de los nodos de la forma de cuadratura con $m = 6$ nodos

En la expresión anterior se observa que se precisa también calcular una aproximación de los valores $y(t_n + c_i h_n)$. Considerando de nuevo la siguiente igualdad

$$y_{n,i} = y\left(t_n + c_i h_n\right) = y\left(t_n\right) + \int_{t_n}^{t_n + c_i h_n} f\left(t, y\left(t\right)\right) dt$$

y utilizando alguna fórmula de cuadratura de m pasos, se puede obtener una aproximación en estos puntos

$$y\left(t_n + c_i h_n\right) \approx y_n + h_n \sum_{j=1}^{m} a_{ij} \underbrace{f(t_n + c_j h_n, y_{n,j})}_{K_{n,j}}$$

En consecuencia, un método de Runge-Kutta queda definido a partir de los valores c_i, los pesos b_i y la matriz a_{ij}.

Se define un método Runge-Kutta de m etapas como el método numérico que dada una aproximación y_n nos da una aproximación a dicha solución en el punto $t_{n+1} = t_n + h_n$ mediante las siguientes fórmulas:

$$t_{n,i} = t_n + c_i h_n \quad , \quad 1 \leq i \leq m$$

$$\begin{cases} y_{n,1} = y_n & , \quad K_{n,1} = f\left(t_{n,1}, y_{n,1}\right) \\ y_{n,i} = y_n + h_n \displaystyle\sum_{j=1}^{m} a_{ij} K_{n,j} & , \quad K_{n,i} = f\left(t_{n,i}, y_{n,i}\right) \\ \quad 2 \leq i \leq m \end{cases}$$

$$y_{n+1} = y_n + h_n \sum_{j=1}^{m} b_j K_{n,j}$$

Cuando los métodos son explícitos, la matriz A es triangular inferior. Los métodos de Runge-Kutta pueden describirse a través del llamado tablero de Butcher [13] donde se pueden disponer los parámetros A, b y c de la forma siguiente:

[13] John Charles Butcher (nacido en 1933 en Auckland, Nueva Zelanda) es un matemático reconocido por sus contribuciones a los métodos numéricos para ecuaciones diferenciales, en particular los métodos de Runge-Kutta y el "Butcher tableau". Estudió en el Auckland University College y la Universidad de Sídney, donde obtuvo su Ph.D. en 1961. En 2013, fue nombrado Oficial de la Orden del Mérito de Nueva Zelanda por sus servicios a las matemáticas.

0	0	0	0	$\ldots$	0	0
c_2	a_{21}	0	0	$\ldots$	0	0
c_3	a_{31}	a_{32}	0	$\ldots$	0	0
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
c_m	a_{m1}	a_{m2}	a_{m3}	$\ldots$	$a_{mm_{-1}}$	0
	b_1	b_2	b_3	$\ldots$	b_{m-1}	b_m

Si se consideran el esquema explícito y el paso de longitud constante h, el método para obtener y_{n+1} a partir de y_n puede escribirse también de la forma siguiente,

$$K_1 = f\left(t_n, y_n\right)$$

$$K_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{m} a_{ij} K_j\right) \quad 2 \leq i \leq m$$

$$y_{n+1} = y_n + h\sum_{j=1}^{m} b_j K_j$$

Video 6.3. Tableros de Butcher

Ejemplo 6.11. Escribe el algoritmo del método Runge-Kutta explícito de tres etapas cuyo tablero tiene la forma

$$\begin{array}{r|lll} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 3/4 & 0 & 3/4 & 0 \\ \hline & 2/9 & 3/9 & 4/9 \end{array}$$

Dado el PVI,

$$y'(t) = t + e^{y(t)}, y(0) = 0$$

aproxima mediante dicho método la solución en el tiempo $t = 0.4$ tomando como paso $h = 0.2$.

El método dado por el tablero se corresponde con el siguiente algoritmo

$$\begin{aligned} K_1 &= f(t_n, y_n) \\ K_2 &= f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_1) \\ K_3 &= f(t_n + \frac{3}{4}h, y_n + \frac{3}{4}hK_2) \end{aligned}$$

$$y_{n+1} = y_n + \frac{h}{9}(2K_1 + 3K_2 + 4K_3)$$

Para aproximar la solución del PVI del enunciado en $t = 0.4$, partiendo de $t_0 = 0$ e $y_0 = 0$ con paso $h = 0.2$, siendo $f(t, y) = t + e^y$, se deben realizar dos pasos.

Con el siguiente código Matlab/Octave se pueden realizar los cálculos. Al precisar dos pasos, se escribe cada paso directamente.

```
%Definición de la función
f=@(t,y) t+exp(y)
%Valores
h=0.2;t0=0;y0=0;
```

```
%Paso 1
K1=f(t0,y0)
K2=f(t0+h/2,y0+h/2*K1)
K3=f(t0+3*h/4,y0+3*h/4*K2)
y1=y0+h/9*(2*K1+3*K2+4*K3)
%Paso 2
t1=t0+h;
K1=f(t1,y1)
K2=f(t1+h/2,y1+h/2*K1)
k3=f(t1+3*h/4,y1+3*h/4*K2)
y2=y1+h/9*(2*K1+3*K2+4*K3)
```

Ejemplo 6.12. Considera el mismo método Runge-Kutta explícito de tres etapas dado en el ejemplo anterior, es decir, el definido por el tablero

$$\begin{array}{r|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 3/4 & 0 & 3/4 & 0 \\ \hline & 2/9 & 3/9 & 4/9 \end{array}$$

Encuentra la solución del PVI,

$$\begin{aligned} y'(t) &= 0.7y - t^2 + 1 \\ y(1) &= 1 \end{aligned}$$

en el intervalo $[1, 2]$ considerando 5 pasos.

```
clear all
%Definición función
f=@(t,y)  0.7*y-t.^2+1;
%Definición intervalo
t0=1;tf=2;%Intervalo
N=5;%Pasos (N+1 puntos)
h=(tf-t0)/N; %Longitud del paso
t(1)=t0;y0=1;y(1)=y0;
```

```
for n=1:N
    k1=f(t(n),y(n));
    k2=f(t(n)+h/2,y(n)+h*k1/2);
    k3=f(t(n)+h*3/4,y(n)+h*k2*3/4);
    y(n+1)=y(n)+(2*k1+3*k2+4*k3)*h/9;
    t(n+1)=t(n)+h;
end
%Se representa la solución obtenida
plot(t,y,'*')
legend('y')
xlabel(['Metodo con tablero y N=',num2str(N),'
pasos'])
hold on
%Se representa la solución dada por Matlab/Octave
%con el comando numérico ode45
[t,sol]=ode45(f,[t0,tf],y0);
plot(t,sol)
legend('y')
hold off
```

La función ode45 en MATLAB se utiliza para resolver numéricamente EDOs. Es un método basado en la fórmula de Runge-Kutta de orden 4 y 5, que se mostrará en el apartado 6.5. La sintaxis básica es: `[t,x] = ode45(odefun, tinter, x0)`. Donde `odefun` es la función que define la EDO, `tinter` es el intervalo de tiempo y `x0` son las condiciones iniciales.

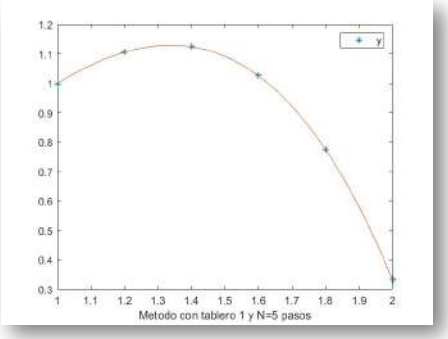


Figura 6.12. Solución del ejemplo

Ejemplo 6.13. Considerando el mismo método explícito de Runge-Kutta de tres etapas dado en el ejemplo anterior, resuelve el siguiente problema de valor inicial:

$$\begin{array}{l} y'''(t) - 4y'(t) = t + 3\cos t + e^{-2t} \quad , \; t \in [0,2] \\ y(0) = 2 \quad , \; y'(0) = -23/40 \; , \; y''(0) = 29/4 \end{array}$$

En este caso la EDO es de orden 3. Llamando $y_1 = y$, $y_2 = y'$, el problema de valor inicial se escribirá

Llamando $y_1 = y, \; y_2 = y', \; y_3 = y''$, se tiene

$$y_1' = y_2 \qquad y_2' = y_3 \qquad y_3' = 4y_2 + t + 3\cos(t) \; + e^{-2t}$$

$$y_1(0) = 2 \qquad y_2(0) = -23/40 \qquad y_3(0) = 29/4$$

En este caso,

$$\begin{pmatrix} y_1'(t) \\ y_2'(t) \\ y_3'(t) \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ 4y_2 + t + 3\cos(t) \; + e^{-2t} \end{pmatrix} \quad , \quad \begin{pmatrix} y_1(0) \\ y_2(0) \\ y_3(0) \end{pmatrix} = \begin{pmatrix} 2 \\ -23/40 \\ 29/40 \end{pmatrix}$$

Trabajando en forma vectorial, la primera componente de y es la solución del PVI, la segunda es la derivada primera y la tercera es la segunda derivada de la solución. El código Matlab/Octave a utilizar es el siguiente.

```
clear all
%Definición función
f=@(t,y)  [y(2);y(3);4*y(2)+t+3*cos(t)+exp(-2*t)];
%Definición intervalo
t0=0;tf=1;%Intervalo
N=5;%Pasos (N+1 puntos)
h=(tf-t0)/N; %Longitud del paso
t(1)=t0;y0=[2;-23/40;29/4];
y(:,1)=y0;
```

```
for n=1:N
    k1=f(t(n),y(:,n));
    k2=f(t(n)+h/2,y(:,n)+h*k1/2);
    k3=f(t(n)+h*3/4,y(:,n)+h*k2*3/4);
    y(:,n+1)=y(:,n)+(2*k1+3*k2+4*k3)*h/9;
    t(n+1)=t(n)+h;
end
plot(t,y,'*')
legend('y_1','y_2','y_3')
xlabel(['Metodo con tablero y N=',num2str(N),' pasos'])
hold on
%Se compara con la solución numérica dada con ode45
[t,sol]=ode45(f,[t0,tf],y0);
plot(t,sol)
legend('y_1','y_2','y_3')
hold off
```

En la siguiente imagen se muestra la solución obtenida con la solución dada por Matlab/Octave.

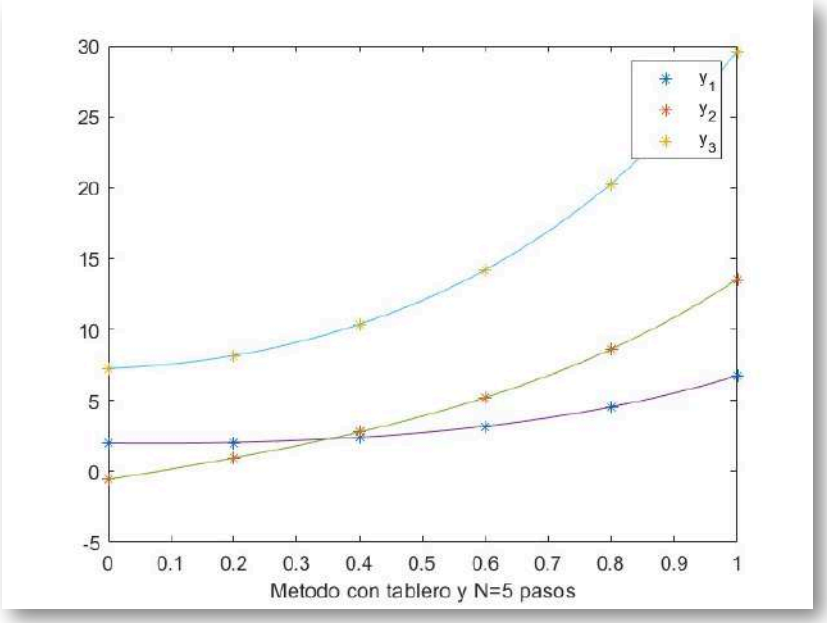


Figura 6.13. Solución del ejemplo

6.4.3.1 Algunos métodos de Runge-Kutta

En este apartado se mostrarán algunos métodos de Runge -Kutta con su tablero de Butcher.

Método de Euler

0	0
	1

Método de Heun

0	0	0
1	1	0
	$1/2$	$1/2$

Método de Kutta de Tres Etapas

0	0	0	0
$1/2$	$1/2$	0	0
1	-1	2	0
	$1/6$	$4/6$	$1/6$

También podría escribirse de la forma siguiente.

$$
\begin{aligned}
K_1 &= f\left(t_n, y_n\right) \\
K_2 &= f\left(t_n + h \cdot \frac{1}{2}, y_n + h \cdot \frac{1}{2} K_1\right) \\
K_3 &= f\left(t_n + h \cdot 1, y_n + h\left[-1K_1 + 2K_2\right]\right) \\
y_{n+1} &= y_n + h\left[\frac{1}{6}K_1 + \frac{4}{6}K_2 + \frac{1}{6}K_3\right]
\end{aligned}
$$

Método de Heun de Tres Etapas

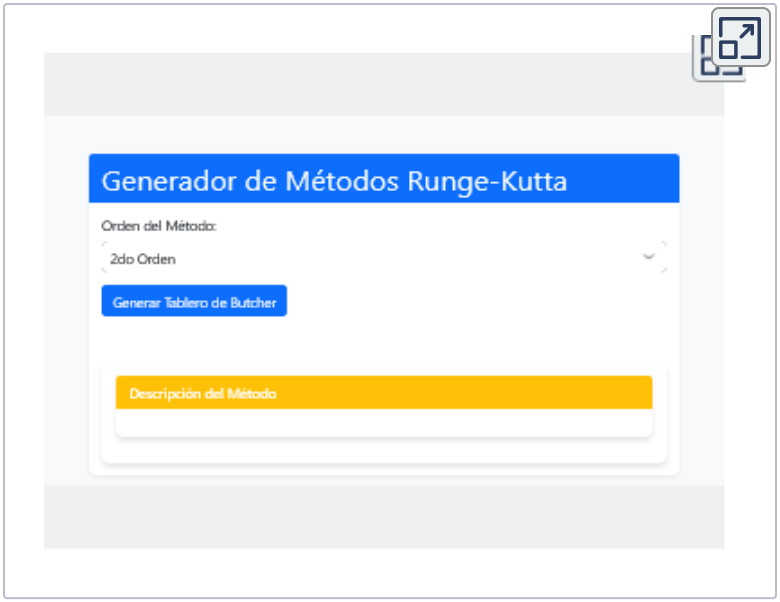
$$\begin{array}{r|lll} 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 \\ 2/3 & 0 & 2/3 & 0 \\ \hline & 1/4 & 0 & 3/4 \end{array}$$

$$\begin{aligned} K_1 &= f\left(t_n, y_n\right) \\ K_2 &= f\left(t_n + h \cdot \frac{1}{3}, y_n + h \cdot \frac{1}{3} K_1\right) \\ K_3 &= f\left(t_n + h \cdot \frac{2}{3}, y_n + h \cdot \frac{2}{3} \cdot K_2\right) \\ y_{n+1} &= y_n + h\left[\frac{1}{4} K_1 + \frac{3}{4} K_3\right] \end{aligned}$$

Método de Runge-Kutta Clasico

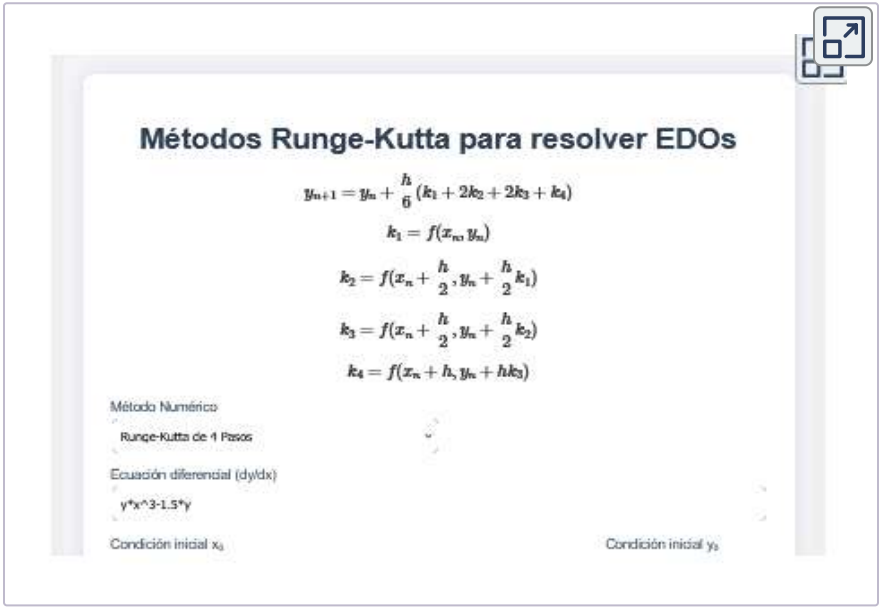
$$\begin{array}{r|llll} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

$$\begin{aligned} K_1 &= f\left(t_n, y_n\right) \\ K_2 &= f\left(t_n + h \cdot \frac{1}{2}, y_n + h \cdot \frac{1}{2} K_1\right) \\ K_3 &= f\left(t_n + h \cdot \frac{1}{2}, y_n + h \cdot \frac{1}{2} \cdot K_2\right) \\ K_4 &= f\left(t_n + h \cdot 1, y_n + h \cdot 1 \cdot K_3\right) \\ y_{n+1} &= y_n + h\left[\frac{1}{6} K_1 + \frac{2}{6} K_2 + \frac{2}{6} K_3 + \frac{1}{6} K_4\right] \end{aligned}$$



Interactivo 6.4. Método de Runge-Kutta asociado a tableros

Con el siguiente interactivo se pueden resolver EDOs de primer orden y comparar el resultado por distintos métodos de Runge-Kutta.



Interactivo 6.5. Comparativa métodos de Runge-Kutta

Se muestra a continuación ejemplos de aplicación de PVI resueltos con Matlab/Octave aplicando distintos métodos de Runge-Kutta.

Ejemplo 6.14. Se considera la ecuación diferencial de segundo orden de un sistema masa-resorte sometido a una fuerza externa $F(t)$

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F\,(t)$$

Si las condidiones iniciales son $x(0) = 1, x'(0) = 0$ y $F(t) = sen(2t)$, $m = 2kg, c = 3$ N-s/m y $k = 5$ N/m, se pide:

- Plantea el problema a resolver en su forma general.
- Determina mediante Euler con $h = 0.01s$ el tiempo necesario para que el bloque pase por $x = 0m$.
- Resuelve mediante el método de Runge-Kutta de orden 4 con $h = 0.01s$ en el intervalo $[0, 2]$.

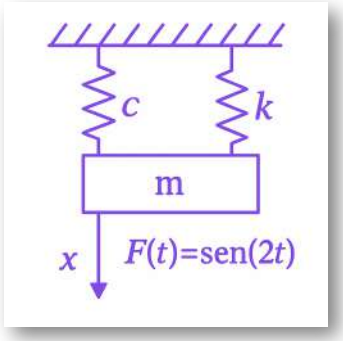


Figura 6.14. Sistema mecánico

Con los datos del problema, el PVI es,

$$\begin{array}{l} 2\frac{d^2x}{dt^2} + 3\frac{dx}{dt} + 5x = sen\,(2t) \\ x\,(0) = 1 \quad x'\,(0) = 0 \end{array}$$

Para escribir la EDO en su forma general, se considera $x_1 = x, x_2 = x'$ y se tendrá

$$x_2' = -\frac{3}{2}x_2 - \frac{5}{2}x_1 + \frac{sen\,(2t)}{2}$$

$$x_1\,(1) = 0 \qquad x_2\,(0) = 0$$

Es decir,

$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} x_2 \\ -\frac{3}{2}x_2 - 5x_1 + sen\,(2t) \end{pmatrix} \quad , \quad \begin{pmatrix} x_1\,(0) \\ x_2\,(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Utilizando el método de Euler con paso $h = 0.01$, se calcula en que instante aproximadamente se obtendrá que $x = 0$.

```
clear all
%Definición función
f=@(t,x)  [x(2);-3/2*x(2)-5/2*x(1)+sin(2*t)/2];
%Definición intervalo
t0=0;
h=0.01; %Longitud del paso
t(1)=t0;x0=[1;0];
x(:,1)=x0;n=1; pos=1;
while pos>0
    t(n+1)=t(n)+h;
    x(:,n+1)=x(:,n)+h*f(t(n),x(:,n));
    pos=x(1,n+1);
    n=n+1;
end
t(n)   %Aproximadamente en 1.85 segundos
```

Para resolver la EDO utilizando el método de Runge-Kutta de orden 4 con $h = 0.01$, se puede considerar el siguiente código Matlab/Octave.

```
t0=0;
h=0.01; %Longitud del paso
t(1)=t0;x0=[1;0];
x(:,1)=x0;N=200;
```

Se implementa el tablero para obtener por el método de Runge-Kutta la solución aproximada.

```
for n=1:N %
   k1=f(t(n),x(:,n));
   k2=f(t(n)+h/2,x(:,n)+h*k1/2);
   k3=f(t(n)+h/2,x(:,n)+h*k2/2);
   k4=f(t(n)+h,x(:,n)+h*k3);
   x(:,n+1)=x(:,n)+(k1+2*k2+2*k3+k4)*h/6;
   t(n+1)=t(n)+h;
end
plot(t,x,'*')
legend('sol aprox')
```

Ejemplo 6.15. Resuelve el problema $y' = yt^3 - 1.5y$ con la condición inicial $y(0) = 1$ en el intervalo $[0, 1.8]$ considerando 9 pasos en el método Runge-Kutta clásico (de cuarto orden):

$$\begin{array}{r|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

En este caso la EDO es de orden 1. Se tiene $f(t, y) = yt^3 - 1.5y, y(0) = 1$. Como se pide considerar 9 pasos, se tendrá $h = 1.8/9$.

```
clear all
f=@(t,y)  y*t^3-1.5*y;%Función
t0=0;tf=1.8;%Intervalo
N=5;%Pasos (N+1 puntos)
h=(tf-t0)/N; %Longitud del paso
t(1)=t0;y0=1;y(1)=y0;
```

Se escribe el código Matlab/Octave para resolver el problema definiendo los valores K_i correspondientes al tablero.

```
for n=1:N %
    k1=f(t(n),y(n));
    k2=f(t(n)+h/2,y(n)+h*k1/2);
    k3=f(t(n)+h/2,y(n)+h*k2/2);
    k4=f(t(n)+h,y(n)+h*k3);
    y(n+1)=y(n)+(k1+2*k2+2*k3+k4)*h/6;
    t(n+1)=t(n)+h;
end
plot(t,y,'*')
legend('sol aprox')
```

En este ejemplo, como se puede obtener la solución exacta que es la función $y(t) = e^{\frac{t\left(t^3-6\right)}{4}}$, se representa en una misma gráfica esta curva y la aproximación obtenida.

```
hold on
x=linspace(t0,tf,30);
plot(x,exp((x.^4-6*x)/4),'r')
legend('sol aprox','sol exacta')
hold off
```

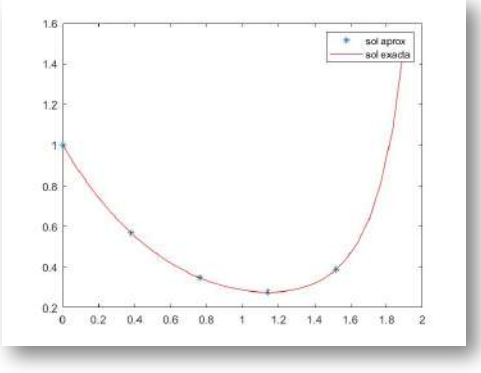


Figura 6.15. Solución exacta y valores aproximados

En el siguiente ejemplo, se programa el método de Runge-Kutta construyendo los valores K_i a partir de la definición de las matrices del tablero de Butcher asociado.

Ejemplo 6.16. Modifica el código del ejercicio anterior para definir los valores K_i a través de las matrices A,B y C que definen el tablero asociado al método de Runge-Kutta de cuarto orden.

```
f=@(t,y)  y*t^3-1.5*y;%Función
t0=0;tf=1.9;%Intervalo
N=5;%Pasos (N+1 puntos)
h=(tf-t0)/N; %Longitud del paso
%Condiciones iniciales
t(1)=t0;y0=1;y(1)=y0;
%Definición tablero
C=[0 1/2 1/2 1];
A=[0 0 0 0;1/2 0 0 0;0 1/2 0 0;0 0 1 0];
B=[1/6 2/6 2/6 1/6];
m=length(B);
for n=1:N %
    %Construcción Ki con datos tablero en cada paso
    kn=[f(t(n),y(n))];
    for j=2:m
       kn = [kn, f(t(n)+h*C(j),y(n)+h*kn*A(j,1:j-
1)')];
    end
    %Aproximación en punto siguiente
    y(n+1)=y(n)+h*kn*B';
    t(n+1)=t(n)+h;
end
plot(t,y,'*')
%Solución exacta
hold on
x=linspace(t0,tf,30);
plot(x,exp((x.^4-6*x)/4),'r')
legend('sol aprox','sol exacta')
hold off
```

En el siguiente ejemplo, se resuelve un sistema de ecuaciones diferenciales utilizando el método de Runge-Kutta de cuarto orden.

Ejemplo 6.17. Considera el problema plano de dos cuerpos puntuales que se mueven atraídos según la ley de la gravitación universal de Newton. En unidades apropiadas, las ecuaciones del movimiento adoptan la forma siguiente:

$$\begin{aligned} y_1' &= y_3 \\ y_2' &= y_4 \\ y_3' &= -\frac{y_1}{\left({y_1}^2 + y_2^2\right)^{3/2}} \\ y_4' &= -\frac{y_2}{\left({y_1}^2 + y_2^2\right)^{3/2}} \end{aligned}$$

donde (y_1, y_2) son las coordenadas de uno de los cuerpos (satélite) en un sistema con origen en el otro cuerpo central. Los valores (y_3, y_4) representan el vector velocidad del cuerpo satélite.

Utiliza el método de Runge-Kutta clásico para encontrar la solución con las condiciones iniciales siguientes

$$y_1(0) = 1,\ y_2(0) = 1,\ \ y_3(0) = 0,\ \ y_4(0) = 1$$

Según la primera ley de Kepler: "los planetas describen´ orbitas elípticas alrededor del sol, que ocupa uno de los focos de esa elipse". En general, las soluciones del problema de dos cuerpos son cónicas (pueden ser elipses, parrábolas o hipérbolas), pero si se toman las condiciones iniciales siguientes:

$$y_1(0) = 1,\ y_2(0) = y_3(0) = 0,\ \ y_4(0) = 1$$

la solución es una circunferencia de centro el origen con radio 1 y con periodo 2π.

Se muestra a continuación el código Matlab/Octave para resolver el problema.

El tablero de Butcher para el método de Runge-Kutta clásico de cuarto orden es

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

```
clear all
%Definición de la función
f=@(t,y) [y(3);y(4);-y(1)/(y(1)^2+y(2)^2)^(3/2);-
y(2)/(y(1)^2+y(2)^2)^(3/2)];
%Datos PVI
t0=0;tf=8*pi;
y(:,1)=[1;1;0;1];h=0.1;N=ceil(tf/h);t(1)=t0;
for n=1:N
    k1=f(t(n),y(:,n));
    k2=f(t(n)+h/2,y(:,n)+h*k1/2);
    k3=f(t(n)+h/2,y(:,n)+h*k2/2);
    k4=f(t(n)+h,y(:,n)+h*k3);
    y(:,n+1)=y(:,n)+(k1+2*k2+2*k3+k4)*h/6;
    t(n+1)=t(n)+h;
end
%Representación de la solución
plot(y(1,:),y(2,:),'r')
axis equal
```

En la siguiente imagen se muestra la trayectoria del satélite una vez resuelto el problema.

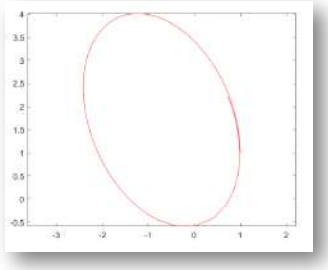


Figura 6.16. Trayectoria del satélite

Para resolver este ejercicio de forma alternativa, se pueden utilizar dos funciones creadas por el profesor Eduardo Casas (ficheros Matlab).

- la función mirkf4c.m para definir los parámetros del método de Runge-kutta clásico: A, b, c que devuelve como salida.
- la función pasorkf.m para definir un paso del método de Runge-Kutta que se considere con la sintáxis:

  `pasorkf(f,tn,hn,yn,a,b,c)`

  Los parámetros de entrada son: `f`, `tn`, `hn`, `yn`, `a`, `b`, `c` que son respectivamente, la función f, $t(n)$, el paso h, $y(n)$ y los parámetros A, b y c del método de Runge-Kutta que se considere.

Utilizando estas funciones, el código Matlab/Octave para encontrar la solución se muestra a continuación.

```
%Definición de la función
f=@(t,y) [y(3);y(4);-y(1)/(y(1)^2+y(2)^2)^(3/2);-
y(2)/(y(1)^2+y(2)^2)^(3/2)];
%Datos PVI
t0=0;tf=8*pi;
y0=[1;1;0;1];
y(1)=y0;t(1)=t0;
N=100;h=tf/N;
%Se definen el tablero
[a,b,c]=mirkf4c;
for iter=1:N
    %Se ejecuta el paso utilizando la función pasokf.m
    y(:,n+1)=pasorkf(f,t(n),h,y(:,n),a,b,c);
    t(n+1)=t(n)+h;
end
%Representación de la solución
plot(y(1,:),y(2,:),'r')
axis equal
```

6.4.4 Error de los métodos de Runge-Kutta

A continuación, se estudiará el error en los métodos de Runge–Kutta. Para ello, se comienza por observar que dichos métodos pueden expresarse de la siguiente forma:

$$y_{n+1} = y_n + h_n \Phi_f \left(t_n, y_n, h_n\right) \qquad 0 \leq n \leq N-1$$

Se introduce el concepto de orden de un método numérico para ecuaciones diferenciales ordinarias para medir la rapidez con la que decrece el error global al reducir el tamaño del paso h. Se presentan los conceptos fundamentales para caracterizar este orden [4], [6].

El error local del método en el paso n es la cantidad

$$\varepsilon_n = y\left(t_n + h_n\right) - \left[y\left(t_n\right) + h_n \Phi_f\left(t_n, y\left(t_n\right), h_n\right)\right]$$

El error global del método en el paso n es la cantidad

$$e_n = y\left(t_n + h_n\right) - \left[y_n + h_n \Phi_f\left(t_n, y_n, h_n\right)\right]$$

Se puede observar que:

1. El error local es que se comete al dar un paso partiendo del valor exacto de la solución en el instante t_n.
2. El error global es el que se comete al dar un paso a partir de la aproximación de $y_n \approx y(t_n)$ acumulando los errores de los pasos previos.

Se dirá que el método es de orden p si $||\varepsilon_n|| = O(h_n^{p+1})$, es decir, si existe $C > 0$ de forma que $||\varepsilon_n|| \leq C h_n^{p+1}$.

Para determinar el orden de un método de Runge-Kutta se emplean las condiciones de orden de Butcher, un conjunto de restricciones que deben satisfacerse sobre los parámetros A, b y c. Tomando el vector de m componentes $e = (1, 1, ..., 1)^T$ siempre se exige $Ae = c$ y luego, dependiendo del orden deseado, se deben exigir condiciones adicionales. Se indican las condiciones a cumplir para los tres primeros órdenes.

- **Orden 1**. $b^T e = 1$
- **Orden 2**. Además de la condición de orden 1, $b^T c = 1/2$
- **Orden 3**. Además de las de orden 1 y 2, se deben cumplir $b^T c^2 = 1/3, b^T Ac = 1/6$

Puede comprobarse que el método de Euler es de orden 1, los de Heun de 2 y 3 etapas poseen, respectivamente, orden 2 y 3. El método de Kutta de 3 etapas es de orden 3 y el de Runge-Kutta clásico de orden 4.

6.5 Métodos de Runge Kutta encajados

Los métodos vistos hasta este momento tienen una dependencia directa con el paso h considerado, por lo que su elección puede condicionar enormemente el funcionamiento del propio método. Se plantea en este apartado cómo determinar cuál es el paso de tiempo h mas indicado analizando los llamados métodos encajados [4], [6].

Estos métodos consisten en una variación del paso h a partir del control de los errores de aproximacion obtenidos a medida que se calculan las diferentes iteraciones del método.

Así, si llegado el momento, el error de aproximación supera la tolerancia establecida para este, el método varía el paso para tener una mejor aproximacion. Por el contrario, si el error es menor que la tolerancia, el método amplía el paso con el fin de acelerar el rendimiento.

Estos métodos emplean dos fórmulas de diferente orden, que comparten evaluaciones intermedias de la función derivada, para calcular dos aproximaciones del siguiente paso: una más precisa y otra menos precisa. La diferencia entre ambas proporciona una estimación del error, que puede usarse para ajustar automáticamente el tamaño del paso en los métodos de paso variable, optimizando el equilibrio entre precisión y número de operaciones a realizar para encontrar la solución.

El procedimiento general consiste en utilizar dos métodos en la integración:

$$y_{n+1} = y_n + h_n \Phi_f \left(t_n, y_n, h_n\right) \qquad orden\ p$$

$$\widehat{y}_{n+1} = y_n + h_n \psi_f \left(t_n, y_n, h_n\right) \qquad orden\ q > p$$

donde las matrices A y c del método de orden p están encajados en $\widehat{A}$ y $\hat{c}$ del método de orden $q > p$.

Al ser el método de orden q más preciso, el valor $\varepsilon_n = \|\widetilde{y}_{n+1} - y_{n+1}\|$ es una estimación del error local cometido por el primer método. No obstante, como $\widehat{y}_{n+1}$ es mejor aproximación de la solución que y_{n+1}, entonces se tomará como aproximación $\widehat{y}_{n+1}$ y la cantidad $\varepsilon_n = \|\widetilde{y}_{n+1} - y_{n+1}\|$ será una cota superior de la integración.

A modo de ejemplo, se verán los métodos de Fehlberg que son una familia de métodos de Runge–Kutta encajados que proporcionan dos aproximaciones de distinto orden. En este libro, se consideran únicamente los métodos de orden 4 y 5. Su esquema más conocido es el RKF45, usado ampliamente en librerías como ode45 de MATLAB.

En estos métodos los parametros A y c del esquema de orden 4 coinciden con las 5 primeras filas de $\widehat{A}$ y $\hat{c}$ del esquema de orden 5. Por tanto, las etapas $K_{n,i}$ para $1 \leq i \leq 5$ son idénticas en ambos metodos, lo que reduce el número de evaluaciones de f.

Runge-Kutta-Felhberg 4-5 #1

$$
\begin{array}{r|cccccc}
0 & & & & & & \\
2/9 & 2/9 & & & & & \\
1/3 & 1/12 & 1/4 & & & & \\
3/4 & 69/128 & -243/128 & 135/64 & & & \\
1 & -17/12 & 27/4 & -27/5 & 16/15 & & \\
\hline
orden\ 4 & 1/9 & 0 & 9/20 & 16/45 & 1/12 &
\end{array}
$$

$$
\begin{array}{r|cccccc}
0 & & & & & & \\
2/9 & 2/9 & & & & & \\
1/3 & 1/12 & 1/4 & & & & \\
3/4 & 69/128 & -243/128 & 135/64 & & & \\
1 & -17/12 & 27/4 & -27/5 & 16/15 & & \\
5/6 & 65/432 & -5/16 & 13/16 & 4/27 & 5/144 & \\
\hline
orden\ 5 & 47/450 & 0 & 12/25 & 32/225 & 1/30 & 6/25
\end{array}
$$

Para pasar de t_n a t_{n+1} y hallar una cota del error local de aproximación de $y(t_{n+1})$, se aplica el método de orden 5 para obtener $\widehat{y}_{n+1}$ y el método de orden 4 para obtener y_{n+1}. La estimación del error será $\|\widehat{y}_{n+1} - y_{n+1}\|$.

Además, si

$$y_{n+1} = y_n + h_n \sum_{j=1}^{m} b_j K_{n,j}$$

$$\widehat{y}_{n+1} = y_n + h_n \sum_{j=1}^{\widehat{m}} \widehat{b}_j K_{n,j} \qquad \widehat{m} > m$$

la estimación del error se puede obtener de la forma siguiente

$$\|\widehat{y}_{n+1} - y_{n+1}\| = h_n \sum_{j=1}^{\widehat{m}} \left(\widehat{b}_j - b_j\right) K_{n,j}$$

En consecuencia, basta conocer los parámetros del esquema de orden q y el vector de estimadores $(\widehat{b}_j - b_j)$ para evaluar el error local en cada paso

$$\varepsilon_n = ||\widehat{y}_{n+1} - y_{n+1}||$$

La información que se precisa para el esquema Runge-Kutta-Felhberg 4-5 #1 se incluye en el siguiente tablero.

0						
$2/9$	$2/9$					
$1/3$	$1/12$	$1/4$				
$3/4$	$69/128$	$-243/128$	$135/64$			
1	$-17/12$	$27/4$	$-27/5$	$16/15$		
$5/6$	$65/432$	$-5/16$	$13/16$	$4/27$	$5/144$	
$orden5$	$47/450$	0	$12/25$	$32/225$	$1/30$	$6/25$
$estimadores$	$1/150$	0	$-3/100$	$16/75$	$1/20$	$-6/25$

La función rkf45a.m desarrollada por Eduardo Casas (ficheros Matlab) devuelve los valores A, b, c, est para el par de métodos encajados de orden 4 y 5 del método de Runge-Kutta-Fehlberg #1.

Runge-Kutta-Felhberg 4-5 #2

El siguiente tablero describe otro método Runge-Kutta-Fehlberg donde se encajan también un esquema de orden 4 y otro de orden 5.

0						
$1/4$	$1/4$					
$3/8$	$3/32$	$9/32$				
$12/13$	$1932/2197$	$-7200/2197$	$7296/2197$			
1	$439/216$	-8	$3680/513$	$-845/4104$		
$1/2$	$-8/27$	2	$-3544/2565$	$1859/4104$	$-11/40$	
$orden5$	$16/135$	0	$6656/12825$	$28561/56430$	$-9/50$	$2/55$
$estimadores$	$-1/360$	0	$128/4275$	$2197/75240$	$-1/50$	$-2/55$

La función rkf45b.m desarrollada por Educardo Casas devuelve los valores A, b, c, est para el par de métodos encajados de orden 4 y 5 del método de Runge-Kutta-Fehlberg #2.

Algoritmo esquemas Runge-Kutta encajados

A partir de dos métodos encajados de órdenes p y q con $p < q$, el algoritmo para cálcular un paso y los estimadores es el siguiente.

$$\begin{cases} t_{n,i} = t_n + c_i h_n \quad , \quad 1 \leq i \leq \widehat{m} \\ \quad K_{n,1} = f\left(t_{n,1}, y_{n,1}\right) \\ \quad K_{n,i} = f\left(t_n + c_i h_n, y_n + h_n \displaystyle\sum_{j=1}^{i-1} a_{ij} K_{n,j}\right) \quad 2 \leq i \leq \widehat{m} \\ \hat{y}_{n+1} = y_n + h_n \displaystyle\sum_{j=1}^{\widehat{m}} b_j K_{n,j} \\ \varepsilon_n = h_n \left\| \displaystyle\sum_{j=1}^{\widehat{m}} est_j K_{n,j} \right\| \end{cases}$$

La función rkf.m, desarrollada por Eduardo Casas (ficheros Matlab), implementa un paso con la estimación del error. La sintáxis es la siguiente

```
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
```

Los parámetros de entrada son:

- f: nombre de la función que define la ecuación diferencial
- tn: el instante del que se parte
- hn: el paso que se va a dar
- yn: la aproximación de la solución conocida en el instante t_n
- a,b,c,est: los parámetros del algoritmo A,b,c y los estimadores

Los parámetros de salida son

- yn1: la aproximación de $y(t_{n+1})$ con $t_{n+1} = t_n + h_n$
- en: una estimación del error $\varepsilon_n = ||y(t_{n+1}) - y_{n+1}||$

A partir de la estimación del error, el método encajado ajusta dinámicamente el tamaño del paso basándose en un umbral de tolerancia. En cada iteración se calculan dos soluciones de distinto orden y, al comparar su diferencia (el estimador de error local) con un umbral que combina tolerancias absoluta y relativa, se decide si aceptar o rechazar el paso. Si el error queda dentro del umbral, se avanza; si no, se descarta la aproximación y se reduce el paso. Por último, se actualiza h para asegurar que el siguiente paso cumpla la precisión requerida.

La tolerancia adaptativa deseada se define a partir del error absoluto ε_a y el error relativo ε_r. Dado que el error de q es menor que el de p, se considera para la tolerancia el del método de orden mayor

$$TOL = \varepsilon_a + \varepsilon_r ||\hat{y}_{n+1}||$$

A continuación se muestra el control adaptativo del paso, que consiste en estimar el error local en cada iteración y ajustar automáticamente el siguiente paso para mantenerlo dentro de la tolerancia establecida.

1. **Inicio razonable.** Se elige un primer h_0 que no sea ni demasiado grande ni demasiado pequeño:

   $$h_0 = \max(\varepsilon_a + \varepsilon_r||y_0||, h_{\min})$$

   donde $h_{\min}$ puede ser, por ejemplo, la longitud del intervalo total dividido por 10^6.

2. **¿Se acepta el paso?** Tras calcular la solución de orden mayor $\hat{y}_{n+1}$ y la de orden menor y_{n+1}, se define el error estimado:

   $$\varepsilon_n = \|\hat{y}_{n+1} - y_{n+1}\|$$

   y se compara con la tolerancia

   $$TOL = \varepsilon_a + \varepsilon_r||\hat{y}_{n+1}||$$

   Si $\varepsilon_n \leq TOL,$ se acepta la aproximación.

3. **Ajuste del tamaño de paso.** Si el paso ha fallado, es decir, si $\varepsilon_n > TOL$ (o para optimizar), se debe reducir h_n. Se calcula para ello un factor de corrección:

   $$r_n = 0.9\left(\frac{TOL}{\varepsilon_n}\right)^{\frac{1}{p+1}}$$

   Además, como medida de seguridad, se reducirá siempre el paso al menos a la mitad. Teniendo en cuenta los dos criterios, se tomará

$$h_n = \min\{r_n, r_{\min}\}\, h_n$$

Se considera como $r_{\min}$ un valor entre 0.1 y 0.5, se toma $r_{\min} = 0.5$

4. **Previsión del paso siguiente**. Se calcula r_n de la forma indicada en el punto 3 y se toma

$$h_{n+1} = \begin{cases} h_n & si \;\; r_n \in [1,\; 1.1] \\ \min\{r_n, r_{\max}\}\, h_n & \text{en otro caso} \end{cases}$$

Si ha habido un fallo del paso en la determinación de una de las dos últimas aproximaciones de la solución, entonces se toma $r_{\max} = 1$, en caso contrario se fija $r_{\max} = 5$.

5. **Iterar.** Se avanza y se repite el proceso: si el error vuelve a ser demasiado grande, se reduce h; si sobra margen, se aumenta.

A continuación se ilustra el proceso con varios ejemplos usando el método Runge–Kutta–Felhberg 4-5 #1.

Ejemplo 6.18. Resuelve el siguiente problema con $\varepsilon_a = 10^{-12}, \varepsilon_r = 10^{-10}$

$$\begin{aligned} &y''(t) + \frac{y(t)}{\left(y^2(t)+z^2(t)\right)^{3/2}} = 0 \qquad t \in [0, 20] \\ &z''(t) + \frac{z(t)}{\left(y^2(t)+z^2(t)\right)^{3/2}} = 0 \\ &y(0) = 0.5, \;\; y'(0) = 0 \\ &z(0) = 0, \;\; z'(0) = \sqrt{3} \end{aligned}$$

Se considera $y_1(t) = y(t)$, $y_2(t) = z(t)$, $y_3(t) = y'(t)$, $y_4(t) = z'(t)$ y se incluye en el código Matlab/Octave los datos del problema.

```
format long
%Problema a resolver
f=@(t,y) [y(3);y(4);-y(1)./(y(1).^2+y(2).^2).^1.5;-
y(3)./(y(1).^2+y(2).^2).^1.5];
t0=0;tf=20;y0=[0.5;0;0;sqrt(3)];
%Error absoluto y relativo
ea=10^-12;er=10^-10;
%Métodos encajados
[a b c est]=rkf45a;
```

Se define el paso inicial.

```
hmin=(tf-t0)/10^6;p1=0.2; % p1=1/(1+4) p=4 en RKF 4-5
hn=max(hmin,(ea+er*norm(y0))^p1);
```

Se comienza a iterar y se comprueba si la estimación del error es inferior a la tolerancia. Si se cumple, se avanza estableciendo el nuevo paso.

```
%Cálculo del paso con estimación error
tn=t0;yn=y0;t=[t0];y=[y0];
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
%Cálculo de la tolerancia
TOL=ea+er*norm(yn1);
if en<TOL
        %Se acepta el paso
        tn=tn+hn;yn=yn1;
        t=[t tn];
        y=[y yn];
        %Se establece el nuevo paso
        rn=0.9*(TOL/en)^p1;
        if rn<1|rn>1.1
            hn=min(rn,5)*hn;
        end
else
        disp('Fallo del paso ')
        [en TOL]
end
```

Este bloque de código se itera hasta que se detecte un rechazo de paso, en cuyo caso se reduce su tamaño, o hasta haberse completado el intervalo de integración. Se modifica el código para detectar si se ha llegado al final del intervalo de integración.

```
fallo=0;
while (tn<=tf)&(fallo==0)
    %Cálculo del paso con estimación error
    [yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
    %Cálculo de la tolerancia
    TOL=ea+er*norm(yn1);
    if en<TOL
            %Se acepta el paso
            tn=tn+hn;yn=yn1;
            %Si se ha llegado al final del intervalo
            %Se deja de iterar
            if tn>=tf
                 disp('FIN')
                 break
            end
            %Se acepta la aproximación
            t=[t tn];
            y=[y yn];
            %Se establece el nuevo paso
            rn=0.9*(TOL/en)^p1;
            if rn<1|rn>1.1
                hn=min(rn,5)*hn;
            end
    else
            disp('Fallo del paso ')
            fallo=1;
            [en TOL]
    end
end
```

En este ejemplo, al ejecutar el código anterior, se ha aceptado siempre el paso y no es necesario hacer reducción de h en ningún momento.

Teniendo en cuenta el ejemplo 6.17, este PVI modeliza el movimiento de dos cuerpos que se mueven atraidos según la ley de la gravitación universal de Newton. Las coordenadas $(y_1(t), y_2(t)) = (y(t), z(t))$ son las de uno de los cuerpos (satélite) con origen el otro cuerpo central. Para representar la solución, que en este caso es una elipse, se debe considerar la primera y segunda componente del vector y.

```
%Representación de la solución aproximada del
movimiento
plot(y(1,:),y(2,:),'*')
axis equal
legend('(y(t),z(t))')
```

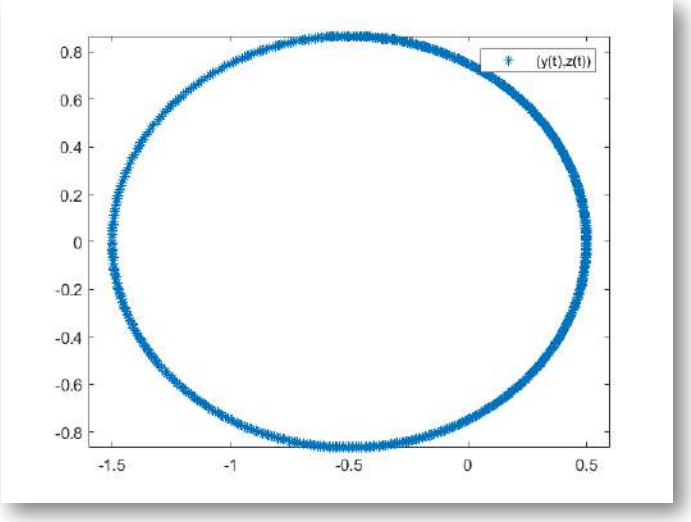


Figura 6.17. Movimiento del satélite

El siguiente ejemplo ilustra el procedimiento a seguir cuando resulta necesario reducir el tamaño de paso en algún momento.

Ejemplo 6.19. Resuelve el siguiente problema con $\varepsilon_a = 10^{-12}, \varepsilon_r = 10^{-10}$

$$\begin{aligned} y_1'(t) &= 1 + y_1(t)^2 y_2(t) - 4y_1(t) \qquad t \in [0,1] \\ y_2'(t) &= 3y_1(t) - y_1(t)^2 y_2(t) \\ y_1(0) &= \tfrac{3}{2}, \;\; y_2(0) = 3 \end{aligned}$$

Se define en primer lugar el PVI.

```
f=@(t,y) [1+y(1).^2*y(2)-4*y(1);3*y(1)-y(1)^2.*y(2)];
t0=0;tf=20;y0=[3/2;3];
```

Se consideran los errores absolutos y relativos y el esquema RKF 4-5.

```
%Error absoluto y relativo
ea=10^-12;er=10^-10;
%Métodos encajados
[a b c est]=rkf45a;
```

Se considera el paso inicial.

```
hmin=(tf-t0)/10^6;p1=0.2; % p1=1/(1+4) p=4 en RKF 4-5
hn=max(hmin,(ea+er*norm(y0))^p1);
```

Se itera con el mismo código que en el ejemplo anterior, hasta que se produzca un fallo en el paso o se llegue al final del intervalo de integración.

```
tn=t0;yn=y0;t=[t0];y=[y0];
fallo=0;
while (tn<=tf)&(fallo==0)
    %Cálculo del paso con estimación error
    [yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
    %Cálculo de la tolerancia
    TOL=ea+er*norm(yn1);
```

```
    if en<TOL
            %Se acepta el paso
            tn=tn+hn;yn=yn1;
            %Si se ha llegado al final del intervalo
            %Se deja de iterar
            if tn>=tf
                disp('FIN')
                break
            end
            %Se acepta la aproximación
            t=[t tn];  y=[y yn];
            %Se establece el nuevo paso
            rn=0.9*(TOL/en)^p1;
            if rn<1|rn>1.1
                hn=min(rn,5)*hn;
            end
    else
            disp('Fallo del paso ')
            fallo=1;
            [en TOL]
    end
end
```

Tras varias iteraciones, se produce fallo en el paso, por lo que hay que reducirlo.

```
%Se reduce el paso
rn=0.9*(TOL/en)^0.2;
hn=min(rn,0.5)*hn;
%Se calcula con el nuevo paso la aproximación
%y le estimación del error
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]
```

Se comprueba que `en` es menor que `TOL` por lo que se acepta la aproximación y por prevención se reducen también los siguientes dos pasos.

```
%Se acepta la aproximación
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%====   Reducción 1
%Se reduce el paso para la siguiente aproximación
rn=0.9*(TOL/en)^0.2;hn=min(rn,1)*hn;
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]  %Se comprueba que en>TOL
%Dado que en<TOL se acepta la aproximación
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%====   Reducción 2
%Se reduce el paso para la siguiente aproximación
rn=0.9*(TOL/en)^0.2;hn=min(rn,1)*hn;
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]  %Se comprueba que en<TOL
```

Como el error es menor que la tolerancia, se acepta el paso y se vuelve a iterar hasta terminar o hasta que se produzca otro fallo.

```
%Se acepta yn1 y tn calculados anteriormente
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%Se itera hasta que se termine o se produzca fallo
rn=0.9*(TOL/en)^0.2;hn=min(rn,5)*hn;
fallo=0;
while (tn<=tf)&(fallo==0)
    %Cálculo del paso con estimación error
    [yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
    %Cálculo de la tolerancia
    TOL=ea+er*norm(yn1);
    if en<TOL
        tn=tn+hn;yn=yn1;
        %Se acepta la aproximación
        t=[t tn];  y=[y yn];
        %Se deja de iterar, si es final del intervalo
        if tn>=tf
            disp('FIN')
            break
        end
```

```
        %Se establece el nuevo paso
        rn=0.9*(TOL/en)^p1;
        if rn<1|rn>1.1
            hn=min(rn,5)*hn;
        end
    else
        disp('Fallo del paso ')
        fallo=1; [en TOL]
    end
end
```

Vuelve a producirse un fallo, por lo que de nuevo se debe hacer una reducción de h y dos pasos más con reducción por prevención.

```
rn=0.9*(TOL/en)^0.2; %Se reduce el paso
hn=min(rn,0.5)*hn;
%Calculo nuevo paso y estimación del error
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL] %Se comprueba que en<TOL
%Se acepta la aproximación
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%      ====   Reducción 1
%Se reduce el paso para la siguiente aproximación
rn=0.9*(TOL/en)^0.2;hn=min(rn,1)*hn;
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]  %Se comprueba que n<TOL
%Dado que en<TOL se acepta la aproximación
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%      ====   Reducción 2
%Se reduce el paso para la siguiente aproximación
rn=0.9*(TOL/en)^0.2;hn=min(rn,1)*hn;
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]   %Se comprueba que en<TOL
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
```

Comprobado que `en<TOL`, se vuelve a iterar hasta terminar o hasta que haya que hacer una nueva reducciónd e h.

```
%Se establece el nuevo paso
rn=0.9*(TOL/en)^p1;
if rn<1|rn>1.1
    hn=min(rn,5)*hn;
end
%Se itera
fallo=0;
while (tn<=tf)&(fallo==0)
    %Cálculo del paso con estimación error
    [yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
    %Cálculo de la tolerancia
    TOL=ea+er*norm(yn1);
    if en<TOL
            %Se acepta el paso
            tn=tn+hn;yn=yn1;
            t=[t tn];   y=[y yn];
            if tn>=tf
                disp('FIN')
                break
            end
            %Se establece el nuevo paso
            rn=0.9*(TOL/en)^p1;
            if rn<1|rn>1.1
                hn=min(rn,5)*hn;
            end
    else
            disp('Fallo del paso ')
            fallo=1; [en TOL]
    end
end
```

Como no se vuelve a producir ningún fallo en el paso, se llega al final del intervalo. El número de pasos realizados es 61, basta ver el número de componentes del vector t escribiendo el código Matlab/Octave `length(t)-1`.

En la siguiente figura se muestra la solución del PVI en el intervalo $[0, 1]$.

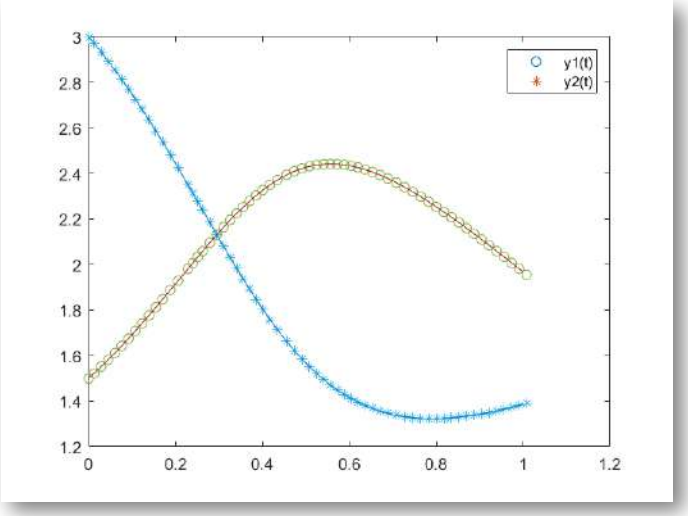


Figura 6.18. Representación de la solución del ejemplo

Ejemplo 6.20. Resuelve el siguiente problema con $\varepsilon_a = 10^{-12}, \varepsilon_r = 10^{-10}$

$$y'''(t) + e^t y''(t) - 10ty'(t) + y/t) = 1 \qquad t \in [0, 8.9]$$
$$y(0) = 1,\ y'(0) = -2,\quad y''(0) = 0$$

Considerando $y_1 = y, y_2 = y', y_3 = y''$, el PVI se podrá escribir de la forma

$$\begin{pmatrix} y_1' \\ y_2' \\ y_3' \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ 1 - e^t y_3 + 10ty_2 - y_1 \end{pmatrix} \qquad \begin{pmatrix} y_1(0) \\ y_2(0) \\ y_3(0) \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}$$

En Matlab/Octave se definirá la función de la EDO y las condiciones iniciales de la forma siguiente.

```
f=@(t,y) [y(2);y(3);1-exp(t)*y(3)+10*t*y(2)-y(1)];
t0=0;tf=8.9;y0=[1;-2;0];
```

Se consideran los errores absolutos y relativos y el esquema RKF 4-5.

```
%Error absoluto y relativo
ea=10^-12;er=10^-10;
%Métodos encajados
[a b c est]=rkf45a;
```

Se considera el paso inicial.

```
hmin=(tf-t0)/10^6;p1=0.2; % p1=1/(1+4) p=4 en RKF 4-5
hn=max(hmin,(ea+er*norm(y0))^p1);
```

Se itera con el mismo código visto en ejemplos anteriores, hasta que se produzca un fallo en el paso o se llegue al final del intervalo de integración.

```
tn=t0;yn=y0;t=[t0];y=[y0];
fallo=0;
while (tn<=tf)&(fallo==0)
    %Cálculo del paso con estimación error
    [yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
    %Cálculo de la tolerancia
    TOL=ea+er*norm(yn1);
    if en<TOL
            %Se acepta el paso
            tn=tn+hn;yn=yn1;
            %Si se ha llegado al final del intervalo
            %Se deja de iterar
            if tn>=tf
                disp('FIN')
                break
            end
            %Se acepta la aproximación
            t=[t tn];  y=[y yn];
            %Se establece el nuevo paso
            rn=0.9*(TOL/en)^p1;
            if rn<1|rn>1.1
                hn=min(rn,5)*hn;
            end
```

```
    else
            disp('Fallo del paso ')
            fallo=1;  [en TOL]
    end
end
```

Tras varias iteraciones, se produce fallo en el paso, por lo que hay que reducirlo.

```
%Se reduce el paso
rn=0.9*(TOL/en)^0.2;
hn=min(rn,0.5)*hn;
%Se calcula con el nuevo paso la aproximación
%y le estimación del error
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);  [en TOL]
```

Se comprueba que `en` es menor que `TOL` por lo que se acepta la aproximación y por prevención se reducen también los siguientes dos pasos.

```
%Se acepta la aproximación
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%   ====   Reducción 1
%Se reduce el paso para la siguiente aproximación
rn=0.9*(TOL/en)^0.2;hn=min(rn,1)*hn;
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]  %Se comprueba que en>TOL
%Dado que en<TOL se acepta la aproximación
tn=tn+hn;yn=yn1;t=[t tn];y=[y yn];
%   ====   Reducción 2
%Se reduce el paso para la siguiente aproximación
rn=0.9*(TOL/en)^0.2;hn=min(rn,1)*hn;
[yn1,en]=rkf(f,tn,hn,yn,a,b,c,est);
TOL=ea+er*norm(yn1);
[en TOL]  %Se comprueba que en<TOL
```

Como el error es menor que la tolerancia, se acepta el paso y se vuelve a iterar no produciéndose ningún fallo y llegando al final del intervalo. En la gráfica siguiente se muestra la solución del problema.

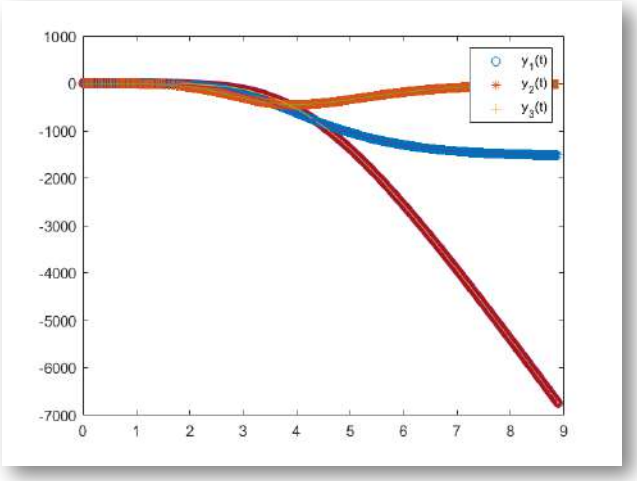


Figura 6.19. Representación de la solución del ejemplo

6.6 Problemas de contorno. Método de tiro

En este apartado se estudiará el problema de contorno siguiente:

$$y''(t) = f(t, y(t), y'(t)) \qquad t \in [t_o, t_f]$$
$$y(t_o) = y_o \quad , \quad y(t_f) = y_f$$

Al carecer de la condición $y'(t_0)$, no es posible integrar directamente con Runge-Kutta. En su lugar, se formula y resuelve el siguiente problema de valor inicial:

$$(P) \quad \begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = f(t, y_1(t), y_2(t)) \qquad t \in [t_o, t_f] \\ y_1(t_o) = y_o \quad , \quad y_2(t_o) = x \end{cases}$$

buscando el valor de x de forma que la solución $y_x = (y_{x,1}, y_{x_2})^T$ cumpla que $y_{x,1}$ para el valor de $t = t_f$ sea y_f [6].

Para resolver este problema con Matlab/Octave se puede utilizar la función oderkf.m desarrollada por Eduardo Casas (ficheros Matlab que resuelve el problema de valor inicial (P) para un valor de x dado. Esta función tiene la siguiente sintáxis:

```
[t,y]=oderkf(f,t0,y0,tf,ea,er)
```

donde:

- $\varepsilon_a = 10^{-14}, \varepsilon_r = 10^{-12}$.
- f es la función diferencial.
- t es el vector fila de instantes t_n donde se ha calculado la aproximación de la solución de (P).
- y es el vector solución que tiene dos filas $y_{x,1}$ para y_1 e $y_{x,2}$ para y_2.

Una vez resuelto el problema de valor inicial P con pendiente inicial x, el siguiente paso es encontrar el valor de x que haga que la solución satisfaca la condición en el extremo, es decir, $y_1(t_f) = y_f$. Para ello se define la función $g(x)$ buscando el cero de $g(x) = y_1(t_f) - y_f$ por el método de la secante.

```
function [gx,tx,yx]=g(x)
    f=...;   $función que define la EDO
    to=...;tf=...;   %Intervalo
    y0=...;     % Valor inicial de y1 en to
    yo=[y0;x];
    ea=...;     er=...;     %Error absoluto y relativo
    %Resolución del problema de valor inicial P
    [tx,yx]=oderkf(f,to,yo,tf,ea,er);
    %La condición es y1(tf)=yf;
    gx=yx(1,end)-yf;
end
```

Se ilustra este procedimiento en el siguiente ejemplo.

Ejemplo 6.21. Calcula la solución del siguiente problema de contorno considerando $\varepsilon_a = 10^{-14}$ y $\varepsilon_r = 10^{-12}$.

$$t^3 y'' = (y - t\,y')^2$$

$$y(1) = 0, \;\; y(2) = 2 \qquad t \in [1, 2]$$

Paso 1. Se considera el problema de valor inicial a resolver en $[to, tf] = [1, 2]$ considerando $y_1 = y, y_2 = y'$

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ \frac{(y_1 - t y_2)^3}{t^3} \end{pmatrix} \qquad \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ x \end{pmatrix}$$

La solución vectorial del problema $y_x(t)$ tiene como primera componente la función solución y la segunda su derivada

$$y_x(t) = (y(t), y'(t))$$

Paso 2. Se define la función $g(x) = y_{x1}(t_f) - y_f$ siendo $t_f = 2$ e $y_f = y(2) = 2$. A continuación se aplica el método de la secante para encontrar el valor de x que hace $g(x) = 0$.

```
function [gx,tx,yx]=g(x)
    f=@(t,y) [y(2);1/t^3*(y(1)-t*y(2))^2];
    to=1;
    tf=2;
    yo=[0;x];
    ea=10^-14;
    er=10^-12;
    %Se resuelve el problema de valor inicial
    [tx yx]=oderkf(f,to,yo,tf,ea,er);
    gx=yx(1,end)-2; %Valor final igual a tf
end
```

Paso 3. Se aplica el método de la secante a la función $g(x)$ para hallar x de forma que $g(x) = 0$.

```
format long
%se busca cambio de signo
[g(1) g(1.5)]
%Se aplica el método de la secante
x0=1;x1=1.5;tol=10^-12;
gx0 = g(x0);
gx1 = g(x1);
em =eps/2;
maxiter=1000;
for i = 1:maxiter
   mk = (gx1-gx0)/(x1-x0);
   if abs(mk)>em*abs(gx1)
      xk = x1 - gx1/mk;
   else
      disp('Error secante')
      break
   end
   if abs(xk - x1) < tol
       disp('Solución raíz')
       [x1 g(x1)]
       break;
   end
   x0=x1;gx0=gx1;
   x1=xk;gx1=g(xk);
end
%Solución: 1.264241117657183
```

Paso 4. Encontrado x, la solución del problema se obtiene con la función g que devuelve en el segundo parámetro de salida el vector t y en el tercero el vector y.

```
x=1.264241117657183;
%tx es el vector t, yx es la solución del problema
[gx,tx,yx]=g(x);
```

Se dibuja la solución del problema de contorno.

```
%Se dibuja la solución
plot(tx,yx(1,:),'o')
```

Se compara con la solución general, que en este caso es conocida. La solución es $y(t) = t\log\left(\frac{t}{C_1 t+1}\right) + C_2 t$ con $C_1 = \frac{2-e}{2e-2}$ y $C_2 = \log(C_1+1)$.

```
%Solución real:
tsol=1:0.05:2;
C1=(2-exp(1))/(2*exp(1)-2);C2=log(C1+1);
ysol=tsol.*log(tsol./(C1*tsol+1))+C2*tsol;
hold on
plot(tsol,ysol)
hold off
legend('Sol Met. Tiro', 'Solución exacta')
```

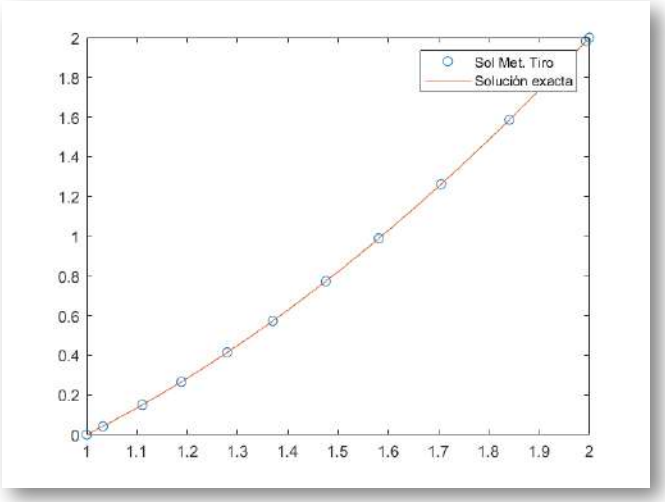


Figura 6.20. Representación de la solución del ejemplo

Ejemplo 6.22. Calcula la solución del siguiente problema de contorno considerando $\varepsilon_a = 10^{-14}$ y $\varepsilon_r = 10^{-12}$.

$$\begin{array}{l} y''(t) = ty^3(t) - 1 - sen(t) \qquad t \in \left[0, \frac{\pi}{2}\right] \\ y(0) = 0,\ y\left(\frac{\pi}{2}\right) = 1 \end{array}$$

Paso 1. Se considera el problema de valor inicial a resolver en $[to, tf] = [1, 2]$ considerando $y_1 = y, y_2 = y'$

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ ty^3(t) - 1 - sen(t) \end{pmatrix} \qquad \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ x \end{pmatrix}$$

La solución vectorial del problema $y_x(t)$ tiene como primera componente la función solución y la segunda su derivada

$$y_x(t) = (y(t), y'(t))$$

Paso 2. Se define la función $g(x) = y_{x1}(t_f) - y_f$ siendo $t_f = \pi/2$ e $y_f = y(\pi/2) = 1$. A continuación se aplica el método de la secante para encontrar el valor de x que hace $g(x) = 0$.

```
function [gx,tx,yx]=ga(x)
    f=@(t,y) [y(2);t*y(1)^3-1-sin(t)];
    to=0;
    tf=pi/2;
    yo=[0;x];
    ea=10^-14;
    er=10^-12;
    %Se resuelve el problema de valor inicial
    [tx yx]=oderkf(f,to,yo,tf,ea,er);
    gx=yx(1,end)-1; %Valor final igual a tf
end
```

Paso 3. Se aplica el método de la secante a la función $ga(x)$ para hallar x de forma que $ga(x) = 0$.

```
%se busca cambio de signo
[ga(1) ga(2)]
%Se aplica el método de la secante
x0=1;x1=2;tol=10^-12;
gx0 = ga(x0);
gx1 = ga(x1);
em =eps/2;
maxiter=1000;
for i = 1:maxiter
   mk = (gx1-gx0)/(x1-x0);
   if abs(mk)>em*abs(gx1)
      xk = x1 - gx1/mk;
   else
      disp('Error secante')
      break
   end
   if abs(xk - x1) < tol
       disp('Solución raíz')
      [x1 ga(x1)]
      break;
   end
   x0=x1;gx0=gx1;
   x1=xk;gx1=ga(xk);
end
%Valor x=1.809633743684593
```

Paso 4. Encontrado x, la solución del problema se obtiene con la función ga que devuelve en el segundo parámetro de salida el vector t y en el tercero el vector y.

```
x=1.809633743684593;
%tx es el vector t, yx es la solución del problema
[gx,tx,yx]=ga(x);
```

Se dibuja la solución del problema de contorno.

```
%Se dibuja la solución
plot(tx,yx(1,:),'o')
```

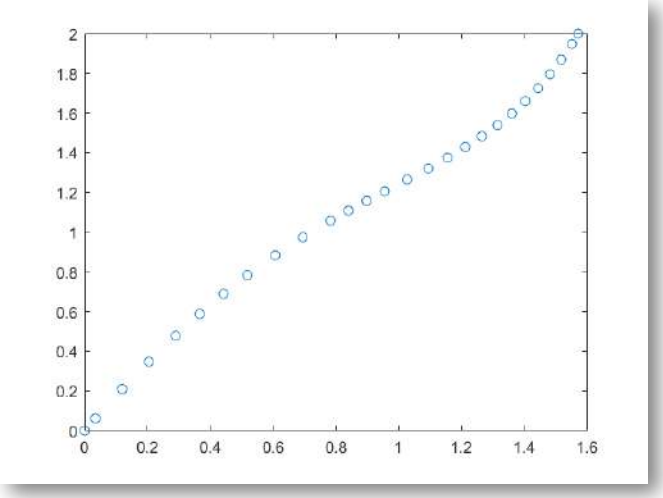


Figura 6.21. Representación de la solución del ejemplo

El método puede adaptarse fácilmente al siguiente ejemplo.

Ejemplo 6.23. Resuelve el siguiente problema:

$$
\begin{aligned}
&y''(t) = sen(y(t)) + 0.01\, t\, y(t) + 0.1\, z(t)\\
&z'(t) = sen(y(t) + z(t)) + w(t)\\
&w'(t) = \cos(y(t) + z(t) + w(t))\\
&y(0) = 0 = z(0), \quad w(0) = 1 \qquad t \in [0, \pi]\\
&y(\pi) + z(\pi) + w(\pi) = 0
\end{aligned}
$$

Se considera

$$y_1 = y,\ \ y_2 = y',\quad y_3 = z,\quad y_4 = w$$

pudiendo escribirse el problema de la forma siguiente

$$\begin{aligned}
y_1' &= y_2 \\
y_2' &= sen\,(y_1) + 0.01\ t\ y_1 + 0.1\ y_3 \\
y_3'\,(t) &= sen\,(y_1 + y_3) + y_4 \\
y_4'\,(t) &= \cos\,(y_1 + y_3 + y_4) \\
y_1\,(0) &= 0 = y_3\,(0)\,,\quad y_4\,(0) = 1\ ,\quad y_2\,(0) = x
\end{aligned}$$

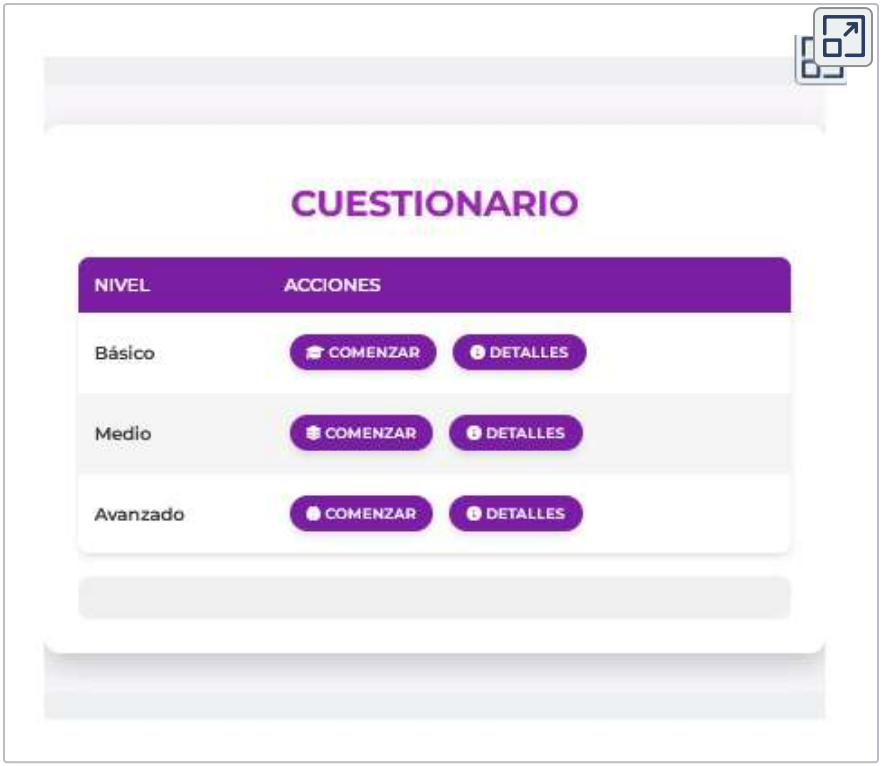
El procedimiento buscará x para que la solución del PVI, yx, verifique

$$y_{x,1}\,(\pi) + y_{x,3}\,(\pi) + y_{x,4}\,(\pi) = 0$$

La función a la que aplicar el método de la secante será la siguiente.

```
function [gx,tx,yx]=ge(x)
    f=@(t,y) [y(2);sin(y(1))+0.01*t*y(1)+0.1*y(3);
    sin(y(1)+y(3))+y(4);cos(y(1)+y(3)+y(4))]
    to=0;
    tf=pi;
    yo=[0;x;0;1];
    ea=10^-14;
    er=10^-12;
    %Se resuelve el problema de valor inicial
    [tx yx]=oderkf(f,to,yo,tf,ea,er);
    gx=yx(1,end)+yx(3,end)+yx(4,end); %Condición en
el extremo
end
```

6.7 Autoevaluación



Interactivo 6.6. Actividad de autoevaluación

6.8 Ejercicios

1 Resuelve el problema de valor inicial:

$$\frac{dy}{dt} = -0.7y + t, \quad y(1) = 1, \quad t \in [1, 2].$$

Utiliza el método de Euler con 10 pasos y representa la solución con MATLAB.

Solución

2 Implementa el método de Runge-Kutta explícito de tres etapas (tablero de Butcher) para resolver:

$$\frac{dy}{dt} = t + e^t, \quad y(0) = 0, \quad t = 0.4, \quad h = 0.1.$$

Solución

3 Utilizando el método de Runge-Kutta clásico de cuarto orden, integra el problema:

$$\frac{dy}{dt} = -1.5yt, \quad y(0) = 1, \quad t \in [0, 1.9],$$

considerando 5 pasos.

Solución

4 Considera las oscilaciones amortiguadas descritas por la ecuación:
$x'' + 2\gamma x' + \omega_0^2 x = 0, \quad x(0) = 0,\ x'(0) = 10, \quad \omega_0 = 2,\ \gamma = 0.5, \quad t \in [0, 3].$

Resolver utilizando MATLAB y representar las oscilaciones.

Solución

5 Resuelve el siguiente sistema que describe la competencia entre dos especies por un mismo recurso:

$$y_1' = y_1(4 - 0.0003y_1 - 0.0004y_2),$$
$$y_2' = y_2(2 - 0.0002y_1 - 0.0001y_2),$$

con $y_1(0) = y_2(0) = 10000$ y $t \in [0, 4]$, utilizando el método de Runge-Kutta de cuarto orden.

Solución

6 Resuelve el siguiente problema de valor inicial utilizando el método de Runge-Kutta clásico con un paso $h = 0.4$ y luego con Matlab utilizando el método Runge-Kutta-Fehlberg con ode45 considerando como tolerancia absoluta y relativa respectivamente:10^{-8} y 10^{-6}:

$$y'' = -0.65\,y' + 0.1\,z - 0.5\,\cos(t),$$

$$z'' = -\sin(t) + 0.1\,z + w,$$

$$w'' = \cos(t) + 0.1\,w, \quad t \in [0,\, 10],$$

con condiciones iniciales

$$y(0) = 1,\ y'(0) = 0, \quad z(0) = 0,\ z'(0) = 1, \quad w(0) = 0,\ w'(0) = 0.$$

Solución

7 Resuelve, con el método de Runge-kutta clásico utilizando 20 pasos, el siguiente problema de valor inicial

$$y''' - y'' - 5y' - 3e^x = 0, \quad y(0) = 1,\, y'(0) = 0,\, y''(0) = -1, \quad t \in [0, 2],$$

Compara el resultado resolviendo también con Matlab y el comando ode45 usando el método de Runge-Kutta-Fehlberg con tolerancia $\varepsilon_a = 10^{-12}$, $\varepsilon_r = 10^{-10}$.

Solución

8 Resuelve el siguiente sistema de ecuaciones:

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= -\sin(y_1), \end{aligned} \qquad y_1(0) = 1,\, y_2(0) = 1$$

utilizando el método de Runge-Kutta clásico con $h = 0.1$ y $t \in [0, 2\pi]$ y condiciones iniciales, .

Solución

9 Resuelve:

$$\frac{d^2y}{dt^2} + \frac{dy}{dt} + 3y = 2e^{-t}, \quad y(0) = 0,\, y'(0) = 1, \quad t \in [0, 10]$$

utilizando el método de Euler con un paso $h = 0.1$ y comparar los resultados con Runge-Kutta clásico (RK4).

Solución

RESUMEN

En este capítulo se trata la integración numérica de ecuaciones diferenciales, una herramienta fundamental en la resolución de problemas matemáticos y científicos que no tienen solución analítica. Este capítulo presenta diversas técnicas para resolver problemas de valor inicial mediante métodos numéricos y analiza sus características, errores y aplicabilidad.

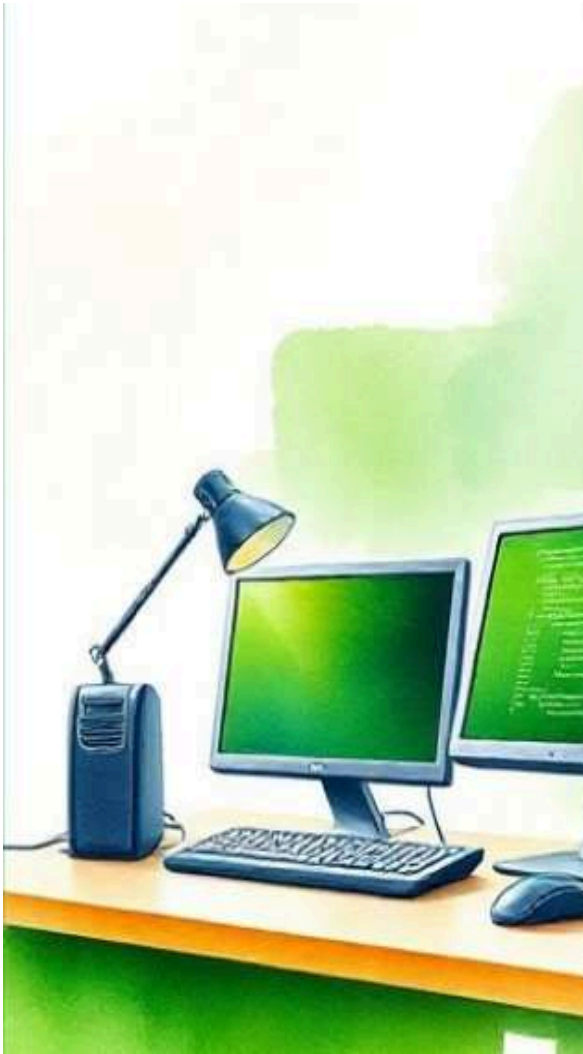
Los contenidos principales del capítulo son los siguientes.

1. **Formulación del problema de valor inicial**
   Se introduce el concepto de ecuaciones diferenciales ordinarias (EDO) y su resolución a través de condiciones iniciales dadas, planteando el problema como una base para aplicar métodos numéricos.

2. **Existencia y unicidad de soluciones**
   Explicación de las condiciones bajo las cuales las soluciones de las EDO están garantizadas y son únicas, un punto clave para entender la validez de los métodos numéricos.

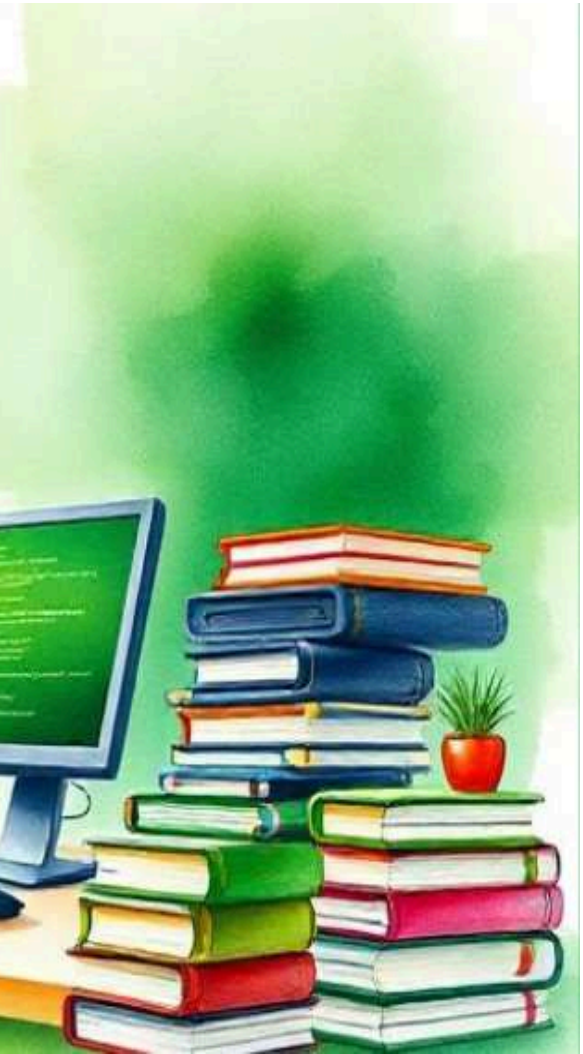
3. Método de Euler

   Este es el método numérico más básico para resolver EDO. Se describe su implementación, ventajas y desventajas, además de discutir el error global acumulado.



4. **Método de Heun**
   Una mejora del método de Euler que utiliza una corrección iterativa para aumentar la precisión. También conocido como método del trapecio, es un ejemplo de método de orden 2.

5. **Métodos de Runge-Kutta**
   Se analizan los métodos de mayor precisión basados en evaluaciones de la pendiente intermedia, destacando el método de cuarto orden como estándar debido a su equilibrio entre precisión y esfuerzo computacional.



6. **Error en los métodos de Runge-Kutta**
   Se detalla cómo se evalúan y minimizan los errores locales y globales en los métodos Runge-Kutta, incluyendo estrategias para ajustar el tamaño del paso.

7. **Métodos de Runge-Kutta encajados**
   Introducción a los métodos adaptativos que ajustan dinámicamente el tamaño del paso en función de una estimación del error, optimizando la eficiencia y precisión del cálculo.

8. **Problemas de contorno y método de tiro**
   Este apartado aborda problemas de contorno, en los cuales las condiciones iniciales y finales están especificadas. Se introduce el método de tiro como técnica para aproximar soluciones.

Bibliografía

[1] Arnold, D. N. (2000). The Patriot Missile Failure. Institute for Mathematics and its Applications. Recuperado de http://www.ima.umn.edu/~arnold/disasters/patriot.html

[2] Arnold, D. N. (2000). The Explosion of the Ariane 5 Rocket. Institute for Mathematics and its Applications. Recuperado de http://www.ima.umn.edu/~arnold/disasters/ariane.html

[3] Arnold, D. N. (2000). The sinking of the Sleipner A offshore platform. Institute for Mathematics and its Applications. Recuperado de http://www.ima.umn.edu/~arnold/disasters/sleipner.html

[4] Aubanell, A., Benseny, A., & Delshams, A. (1993). Útiles básicos de cálculo numérico. Barcelona: Editorial Labor, S.A.

[5] Burden, R. L., & Faires, J. D. (2015). Numerical analysis (10ª ed.). Cengage Learning.

[6] Casas Rentería, Eduardo (2024). Métodos Matemáticos para la Ingeniería. Cálculo Numérico. Recuperado de: https://personales.unican.es/casase/MMI/pdfs/Apuntes.pdf

[7] Faires, J. D., & Burden, R. L. (2012). Numerical methods, 4th. Cengage Learning.

[8] Infante del Río, J. A., & Rey Cabezas, J. M. (2022). Métodos numéricos: teoría, problemas y prácticas con MATLAB. Ediciones Pirámide.

[9] Isaacson, E. & Keller, H.B. (1994). Analysis of Numerical Methods. Dover Publications, Inc, New York.

[10] Mora Flores, W. (2022). Introducción a los métodos numéricos. Implementaciones en el lenguaje R. Recuperado de Matemáticas interactivas

[11] Rivera, J. G., Álvarez, E. E., Galo, J. R., & Tabares, H. A. (2017). Métodos numéricos interactivo. Primera edición. Recuperado de Métodos Numéricos. Interactivo

[12] SIAM News. (1996, octubre). Vol. 29, Number 8.

[13] Süli, E. & Mayers, D. (2003). An Introduction to Numerical Analysis. Cambridge University Press.


DESCARTES
matemáticas interactivas