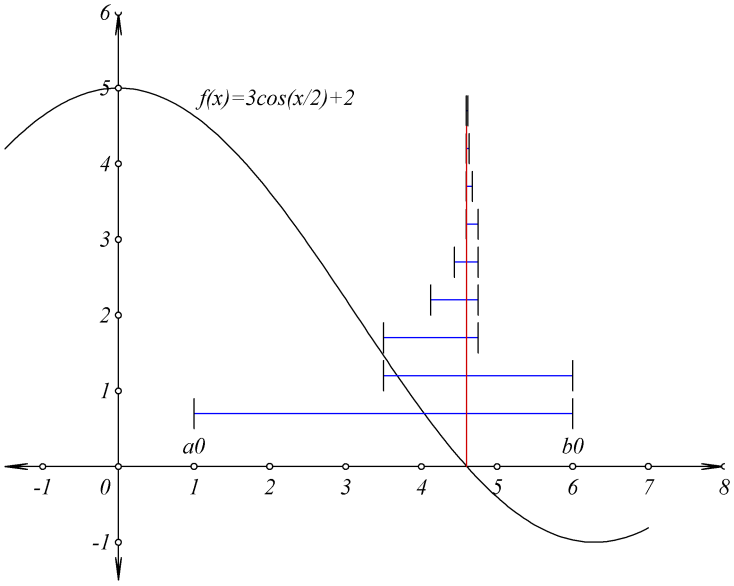
MÉTODOS NUMÉRICOS
Una exploración basada en Scheme

Enrique Comer Barragán



24 de marzo, 2009
↪ Edición Preliminar 0.23 ↩
Instituto Tecnológico de Tijuana

Edición preliminar 0.23
24 de marzo, 2009

Índice general

Prefacio

The purpose of computing is insight, not numbers.

R. W. Hamming. Autor de
Numerical Methods for Scientist and Engineers

El contenido de esta obra se conforma de acuerdo al curso de Métodos Numéricos en el plan de estudios actual de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Tijuana. Se hace énfasis tanto en los aspectos matemáticos conceptuales como algorítmicos. Para la implementación de estos últimos se utiliza el lenguaje de programación Scheme [25].

La edición de este libro se realizó con TeXnicCenter (ambiente para LaTeX/MiKTeX) y software auxiliar como WinGCLC y Asymptote.

Agradezco a la Academia de Sistemas y Computación por la revisión y observaciones para mejorar esta obra, así como a la DGEST por el período sabático 2007-2008, gracias al cual se ha realizado este proyecto.

ECB

comer@cemati.com

Capítulo 1

Teoría de errores

1.1. Importancia de los métodos numéricos

La simulación y el modelado matemático, moverán el siglo XXI, así como el vapor movió el siglo XIX.

William H. Press. Autor de *Numerical Recipes*

Gran parte de la tecnología actual depende de la solución de **modelos matemáticos**, desde la programación empotrada de una calculadora científica y el cálculo estructural de un edificio multinivel con estructuras de acero, hasta el diseño y simulación de aeronaves y vuelos espaciales. La solución de un modelo matemático relativamente sencillo puede obtenerse de manera analítica. Sin embargo, para la gran mayoría de los modelos matemáticos del mundo real, las soluciones analíticas pueden no existir o ser extremadamente complejas, por lo cual se recurre a **métodos numéricos** que aproximen las soluciones dentro de ciertos márgenes de tolerancia.

El análisis de los métodos numéricos nos permite realizar estimaciones tanto de la eficiencia o complejidad de los algoritmos asociados, así como de la confiabilidad de los resultados numéricos obtenidos durante su aplicación.

Entre las disciplinas que hacen uso intensivo de los métodos numéricos podemos mencionar:

- Análisis Estadístico

- Matemáticas financieras
- Análisis de elemento finito
- Análisis estructural
- Química computacional
- Investigación de operaciones
- Electromagnetismo computacional
- Mecánica computacional
- Procesamiento de imágenes
- Procesamiento de señales
- Simulación por computadora
- Computación multi-escala
- Metereología

Debido a la gran variedad de aplicaciones y especialidades atendidas por los métodos numéricos, podemos encontrar en la literatura términos asociados como los siguientes:

- Matemáticas numéricas
- Algoritmos numéricos
- Computación científica
- Análisis numérico
- Matemáticas algorítmicas
- Matemáticas computacionales
- Teoría de la aproximación[1]

[1]Esta disciplina matemática, forma la base teórica de gran parte de los métodos numéricos

1.2. Conceptos básicos

El estudio y aplicación de los métodos numéricos se basa en forma esencial en la representación y manipulación de los números reales en una computadora digital. Ya que la memoria de una computadora es finita, ésta sólo puede representar números racionales. La estrategia para hacer funcional y eficiente la aproximación de los números reales por números de máquina en binario se presenta en el estándar IEEE 754 [13]

Los términos asociados en esta sección incluyen:

- Confiabilidad
- Estabilidad y convergencia
- Propagación del error
- Análisis asintótico
- Notación científica
- Forma decimal normalizada
- Números de máquina decimales con k dígitos
- Números de punto flotante (Estándar IEEE 754)

1.2.1. Cifras significativas

La noción intuitiva de cifras significativas de un número está directamente relacionada con la precisión de los ins-trumentos o procesos que lo generan.

Definición 1.2.1 (Cifras significativas)**.** El número de **cifras significativas** de un número x corresponde al número de cifras en la mantisa de su representación en notación científica.

Ejemplo 1.2.1. Ya que $0.00123 = 1.23 \cdot 10^{-3}$, decimos que 0.00123 tiene tres cifras significativas.

Ejemplo 1.2.2. El número $3210 = 3.210 \cdot 10^3$ posee cuatro cifras significativas.

Note que en el ejemplo anterior, hemos mantenido el 0 de las unidades. Si el origen del número no garantizara el valor de sus unidades, entonces deberíamos escribir directamente $3.21 \cdot 10^3$ lo que indicaría que contamos con sólo tres cifras significativas.

Definición 1.2.2 (Aproximación con t cifras significativas)**.** Sean x_v y x_c los valores verdadero y calculado de una cierta cantidad, con $x_v \neq x_c$. Decimos que x_c aproxima a x_v con t **cifras significativas** si t es el mayor entero no negativo para el cual

$$\left| \frac{x_v - x_c}{x_v} \right| \leq 5 \times 10^{-t} \tag{1.1}$$

Para el caso $x_v = x_c$, x_c aproxima x_v con las cifras significativas propias.

Como se observa en [8, pág. 53] esta definición dada para decimales puede extenderse para cualquier base dada.[2]

Ejemplo 1.2.3. El número 3.1416 aproxima a 3.1415926 en 6 cifras significativas, ya que

$$\frac{|3.1415926 - 3.1416|}{|3.1415926|} = 2.3554932 \times 10^{-6} \leq 5 \times 10^{-6}$$

Como se observa, no es necesario que coincidan los dígitos de las cifras significativas.

Ejercicio 1.2.1. Calcular el número de cifras significativas con que 9.99 aproxima a 10

Ejercicio 1.2.2. Calcular el número de cifras significativas con que 1005 aproxima a 1000

1.2.2. Precisión y exactitud

El estudio e implementación en computadora de los métodos numéricos requiere de un cuidado especial en el manejo de los números, de forma que los

[2]Es importante notar que en la desigualdad (1.1) se utiliza algunas veces $<$ como en el caso de [5, pág. 21]. Se recomienda resolver el ejercicio 1.2.2, para ambas definiciones y reflexionar sobre sus respuestas.

resultados que entregue un programa sean confiables. Para hablar de dicha confiabilidad[3] distinguiremos entre los términos precisión y exactitud.

Aunque en las aplicaciones es necesario conocer la precisión de los datos de entrada (a un algoritmo), que pueden depender directamente de los instrumentos utilizados[4], nos concentraremos aquí en la precisión de la representación de un número en la memoria de la computadora, lo cual está relacionado con la siguiente representación de números reales.

Definición 1.2.3 (números de punto flotante)**.** De-cimos que un número x está representado como *número de punto flotante* si se expresa como

$$x = \pm d_0.d_1d_2\cdots d_{p-1} \times \beta^e \tag{1.2}$$

donde p es el número de dígitos en la mantisa o *significando*, β es la base y e es el exponente entero. Además, cuando $d_0 \neq 0$ decimos que está en su forma normalizada .

Ejemplo 1.2.4. El número 314.159 en representación de punto flotante es 0.314159×10^3 y como punto flotante normalizado 3.14159×10^2.

Ya que las computadoras digitales trabajan (en su mayoría) con el sistema binario, el estándar IEEE 754 especifica tanto la representación como el manejo de los números binarios de punto flotante.[5]

Definición 1.2.4 (números binarios de punto flotante normalizado)**.** Decimos que un número x está representado como *número binario de punto flotante normalizado* si se expresa como

$$x = (-1)^s(1+f)2^e \tag{1.3}$$

donde s es el signo, f es la fracción en binario y e el exponente.

La precisión de un número está directamente relacionada con el número de dígitos en su representación de punto flotante.

[3]La revista *Reliable Computing* se especializa en métodos basados en intervalos (entre otros) que garantizan y acotan todos sus resultados.

[4]*El Pequeño Larousse Ilustrado*, 2002 define *Precisión (de un instrumento de medida)* como la cualidad global de un instrumento que le permite dar indicaciones que coinciden, con mucha aproximación, con el valor verdadero de la magnitud que debe medirse.

[5]Para información adicional sobre este estándar puede consultar el excelente artículo de Goldberg en [10].

Definición 1.2.5 (precisión en formatos IEEE 754)**.** En base a la ecuación (1.3) el estándar IEEE 754 define los siguientes cuatro tipos de formatos[6] (ver tabla (1.1))que determinan la precisión de un número de punto flotante (adaptación de [10, pp. 173-174]):

Tabla 1.1: Precisión en Formatos IEEE 754

parámetro	Formatos			
	sencillo	s. extendido	doble	d. extendido
e_{max}	$+127$	≥ 1023	$+1023$	> 16383
e_{min}	-126	≤ -1022	-1022	≤ -16382
e bits	8	≤ 11	11	≥ 15
f bits	23	31	52	≥ 63
total bits	32	≥ 43	64	≥ 79

Ya que $2^{-23} \approx 1.19 \cdot 10^{-7}$ y $2^{-52} \approx 2.22 \cdot 10^{-16}$, el formato sencillo tiene una fracción con precisión de 7 decimales y el formato doble, una precisión de 15 decimales.

Las operaciones con números reales tanto en MATLAB[7] como en DrScheme[8] se realizan en doble precisión.

Ejemplo 1.2.5. Utilizando la función en Scheme[9], real->ieee-doble, en la figura (1.1), podemos obtener la representación en formato de doble precisión mediante para los números 0.3 y 64

[6]Los formatos extendidos en la tabla 1.1, ofrecen flexibilidad de diseño para los distintos fabricantes de computadoras. Por ejemplo, para el formato doble extendido, las computadoras SPARC y PowerPC ofrecen los siguientes campos (en bits): $e = 15$, $f = 112$. En cambio INTEL ofrece: $e = 15$ y $f = 63$ con un bit extra para manejo explícito del dígito más significativo de la fracción.

[7]MATLAB (MATrix LABoratory) es uno de los sistemas de software numérico más exitosos en la actualidad. Para una introducción sobresaliente, orientada a métodos numéricos, se le invita a consultar el libro de Cleve Moler [18]

[8]Una de las implementaciones más completas del lenguaje de programación Scheme, el cual emplearemos a lo largo del texto para ilustrar e implementar los métodos numéricos. Para una excelente introducción se recomienda el tutorial de Dorai Sitaram [25]

[9]Esta función así como el resto de las utilizadas en este libro, podrán ser obtenidas (eventualmente) visitando http://cemati.com/math/areas/scheme/, o bien el sitio de distribución PLaneT Package Repository en [21]. Se recomienda instalar PLT-Scheme versión 4.1 (o superior) disponible en http://www.plt-scheme.org, y por el momento, sencillamente copiar el archivo slab.ss (que complementa este libro) al folder PLT de su instalación.

```
> (real->ieee-doble 0.3)
("0" "01111111101"
 "0011001100110011001100110011001100110011001100110011")
> (real->ieee-doble 64)
("0" "10000000101"
 "0000000000000000000000000000000000000000000000000000")
>
```

Ya que $(0.3)_{10} = (0.1\overline{0011})_2$ nunca podrá representarse de forma exacta en un sistema binario finito.

Como se observa en el ejemplo 1.2.5 hay números reales (de hecho una cantidad infinita) que no podrán ser representados en forma exacta en un sistema numérico de base fija. Note además que el exponente e' en el formato IEEE 754 obedece (en el caso de doble precisión) a la ecuación $e = e' - 1023$.

Ejercicio 1.2.3. Confirme los resultados del ejemplo 1.2.5, utilizando la ecuacion 1.3.

Ejercicio 1.2.4. Determine las representaciones en binario de punto flotante normalizado (en formato de doble precisión) de los siguientes números en base 10

a) -0.1

b) 1024

c) $1.1920928955078125e - 007$

Problema 1.2.1. Ya que la *recta de los flotantes* no "cubre" la recta real, calcular la distancia que existe entre los siguientes números reales y sus dos números flotantes vecinos[10] bajo el formato de doble precisión del estándar IEEE 754:

a) $2008.09 \cdot 10^{77}$

b) 64.3

c) $-444.7 \cdot 10^{-27}$

[10]Esto sería equivalente a encontrar para un número real x, funciones *piso* $\lfloor x \rfloor$ y *techo* $\lceil x \rceil$ pero en el *edificio* de los números flotantes.

Problema 1.2.2. Generalice el resultado del problema 1.2.1 y descubra una expresión o modelo matemático para determinar la distancia entre números flotantes consecutivos en intervalos de la forma $[2^k, 2^{k+1}]$, donde k es un entero en el rango permitido.

En contraste con el concepto de precisión que se asocia a la representación de un número en particular (p. ej. bajo el formato de doble precisión del estándar IEEE 754), la exactitud de un número hace referencia a una cantidad objetivo, y busca el máximo acercamiento posible a dicho objetivo[11], en el sentido de la definición 1.2.2.

Obtener resultados rigurosos y confiables, con la precisión y exactitud adecuadas para cada aplicación es uno de los propósitos principales del diseño, análisis e implementación de los métodos numéricos.

Conceptos como *convergencia* y acotación del error mediante el *análisis asintótico*, son claves y se considerarán en diferentes partes de nuestro texto.

Código Scheme 1.1: Función real->ieee-doble

```
; real a ieee de 64-bit
; numero real -> lista de strings
; entrada: r
;  salida: (s e' f)
;   donde: s contiene el bit del signo
;        : e' tal que e=e'-1023, representa el exponente
;        : f contiene la fraccion
(define (real->ieee-doble r)
  (ieee8->ieee-doble
   (map fill-byte
        (map (lambda(n) (number->string n 2))
             (bytes->list
              (real->floating-point-bytes r 8 #t))))))
```

1.2.3. Incertidumbre y sesgo

En el análisis de datos numéricos generalmente podemos tener errores que tienen que ver con la falta ya sea de precisión o de exactitud.

[11] *Pequeño Larousse Ilustrado: exactitud*: Cualidad de ser medido, calculado o expresado con todo rigor: *hora exacta*.

Definición 1.2.6 (incertidumbre y sesgo)**.** Llamamos incertidumbre o imprecisión a la falta de precisión, y sesgo o inexactitud, a la falta sistemática de exactitud, ya sea por debajo o bien por arriba de la cantidad exacta.

El manejo de la incertidumbre o imprecisión puede realizarse mediante distribuciones de probabilidad, en tanto que el manejo de la inexactitud, mediante rangos o intervalos.

Ejemplo 1.2.6. Supongamos que un profesor debe iniciar siempre sus clases a las 7 : 00 am. Si existe incertidumbre, podría iniciar con una distribución normal con media de 7 : 05 y desviación estándar de 1 minuto, lo cual indica que el 99.7 % de las veces iniciaría en el intervalo $[7 : 02, 7 : 08]$. Por otro lado, si existe (solamente) sesgo, entonces empezaría sistemáticamente (por ejemplo) a las 7 : 07.

Ejemplo 1.2.7. Si especificamos que el valor de una resistencia eléctrica es de $100 \pm 5\,\%\ \Omega$, estamos indicando que su valor real debe estar en el intervalo $[95, 105]$.

Existe actualmente una nueva generación de métodos numéricos basados en intervalos, rectángulos, cajas e hipercajas (para espacios mayores que 3D), cuyo propósito es obtener soluciones numéricas precisas y exactas. Para mayor información puede consultar el texto de Aberth [2] y el curso de Warwick Tucker en [27].

1.3. Tipos de errores

En esta sección veremos los tipos esenciales de errores presentes (en mayor o menor grado) en la aplicación de los métodos numéricos. Además de estos errores, podemos considerar: el error humano, observaciones inexactas propios del equipo experimental y errores de modelado. Ver el texto de Friedman y Kolman [8, pp. 54-55] para ejemplos.

1.3.1. Error absoluto y relativo

Definición 1.3.1 (Error absoluto y relativo)**.** Sea x_v un valor numérico verdadero y sea x_c el valor calculado que aproxima el verdadero, entonces definimos al error $er(x_c)$ y al error relativo $rel(x_c)$, así como sus correspondientes en valores absolutos, como sigue:

$$er(x_c) = x_v - x_c \tag{1.4}$$

$$er_a(x_c) = |x_v - x_c| \tag{1.5}$$

$$rel(x_c) = \frac{x_v - x_c}{x_v} \tag{1.6}$$

$$rel_a(x_c) = \left| \frac{x_v - x_c}{x_v} \right| \tag{1.7}$$

Asumiendo claro, que en las ecuaciones (1.6) y (1.7) se cumple $x_v \neq 0$.

Ejemplo 1.3.1. Calculemos los errores en la definición (1.3.1) para el caso de aproximar el valor de $\sqrt{11}$ con 3.31. Consideraremos que el valor verdadero de $\sqrt{11}$ está redondeado[12] a siete decimales, es decir $x_v = 3.3166248$, entonces:

$$\begin{aligned} er(3.31) &= 3.3166248 - 3.31 \\ &= 0.0066248 \\ er_a(3.31) &= |0.0066248| = 0.0066248 \\ rel(3.31) &= \frac{3.3166248 - 3.31}{3.3166248} \\ &= 0.0019975 \\ rel_a(3.31) &= |0.0019975| = 0.0019975 \end{aligned}$$

Los errores relativos también pueden ser expresados en %. En nuestro caso, $rel(3.31) \approx 0.20\,\%$.

Ejercicio 1.3.1. Calcule los cuatro errores de la definición (1.3.1) para el caso de aproximar el valor verdadero de $\pi/2$ con $11/7$. Para iniciar sus cálculos, favor de utilizar redondeo a siete decimales.

Problema 1.3.1. Extienda la definición (1.3.1) especificando cómo manejar el caso $x_v = 0$.

[12] La operación de *redondeo* aquí es distinta a la que veremos en la sección 1.3.2

1.3.2. Error por redondeo

A diferencia del error "externo" que provocamos cuando redondeamos a cierto número de decimales (p. ej. 9.9 a 10), el error por redondeo que interesa en cálculo numérico es el que ocurre en la computadora o calculadora debido a las limitaciones propias de la representación del número (digamos en el sistema de punto flotante de precisión sencilla o de doble precisión en el estándar IEEE 754)

Definición 1.3.2 (Error por redondeo)**.** Dado un número real x y un sistema de representación de punto flotante que "almacena" x como $fl(x)$, el error por redondeo es

$$er_{fl}(x) = x - fl(x) \tag{1.8}$$

El error por redondeo acumulado $er_{flops}(x, y)$ es el que resulta después de realizar una serie de operaciones aritméticas de punto flotante (o flops[13]), donde cada operación contribuye su "cuota" de error por redondeo. Estas operaciones aritméticas podemos definirlas como:

$$\begin{aligned} x \oplus y &= fl(fl(x) + fl(y)) \\ x \ominus y &= fl(fl(x) - fl(y)) \\ x \otimes y &= fl(fl(x) \times fl(y)) \\ x \oslash y &= fl(fl(x) \,/\, fl(y)) \end{aligned} \tag{1.9}$$

Para calcular $er_{flops}(x, y)$ es necesario realizar en secuencia cada una de las *flops* y en cada etapa (como lo indican las ecuaciones en (1.9)) calcular el error por redondeo resultante.

Ejemplo 1.3.2 (Moler [17])**.** Calculemos los resultados de la siguiente secuencia:

$$\begin{aligned} a &= 4/3 \\ b &= a - 1 \\ c &= b + b + b \\ e &= 1 - c \end{aligned}$$

Una implementación en Scheme resulta en el código y su ejecución indicado en la figura 1.1.

[13]flops: (del inglés) floating point operations. Algunos sistemas (como MATLAB) tienen la capacidad de contar las *flops* realizadas durante una serie de cálculos.

Figura 1.1: Ejemplo de error por redondeo

Código	$\Longrightarrow$	Resultado
(define a (/ 4. 3))		{a=1.3333333333333333}
(define b (- a 1))		{b= 0.3333333333333326}
(define c (+ b b b))		{c= 0.9999999999999998}
(define e (- 1 c))		{e=2.220446049250313e-016}
e		2.220446049250313e-016

Note que la primera linea del código anterior utiliza (define a (/ 4. 3)) para indicar (con el punto) que 4 es un número real, simplemente quitando el punto obtenemos el resultado indicado[14] en la figura 1.2, donde se ha eliminado el error, debido a que en el lenguaje Scheme, las operaciones aritméticas con enteros y racionales son exactas.

Hay dos errores de redondeo especiales: error de underflow (o *bajoflujo*,) cuando $x(\neq 0)$ es más pequeño que el menor (en magnitud) de los números de punto flotante disponibles y el sistema lo representa como cero (o bien genera una *excepción* durante el cálculo).

Por otra parte, se genera un error de overflow (o *sobreflujo*,) cuando x excede el máximo (en magnitud) de los números representables como punto flotante y el sistema lo representa como +inf o -inf según sea el caso.

Ejercicio 1.3.2. Explore y descubra cuáles son los números que generan *underflow* o bien *overflow* bajo diferentes sistemas (p. ej. en MATLAB y PLT Scheme).

Para medir el error por redondeo utilizamos ya sea el *error relativo* o bien las ulps[15]. En general (como lo indica Goldberg en [10]) si el número de punto

[14] Los resultados entre llaves, indican que no hay resultado en pantalla; sólo se actualiza la tabla de asociaciones entre variables y valores.

[15] ulps: *units of the last place*, unidades de la última posición.

Figura 1.2: Ejemplo de eliminación del error por redondeo, al utilizar números racionales

Código	$\Longrightarrow$	Resultado
(define a (/ 4 3))		$\{a = 1\frac{1}{3}\}$
(define b (- a 1))		$\{b = \frac{1}{3}\}$
(define c (+ b b b))		$\{c = 1\}$
(define e (- 1 c))		$\{e = 0\}$
e		0

flotante $d_0.d_1 \cdots d_{p-1} \times \beta^e$ se utiliza para representar x, entonces se tendrá un error en ulps de:

$$er_{fl}(x) = |d_0.d_1 \cdots d_{p-1} - (x/\beta^e)|\, \beta^{p-1} \tag{1.10}$$

Ejercicio 1.3.3. Utilizando la ecuación (1.10) calcule el error por redondeo en *ulps* para el caso de aproximar 0.0123456 con 1.23×10^{-2}.

Proyecto 1.1. Diseñe e implemente un programa en Scheme (o en el lenguaje de su preferencia) que ofrezca la funcionalidad de definir un sistema de punto flotante parametrizado, y permita realizar experimentos y cálculo de errores, en secuencias de operaciones aritméticas.

1.3.3. Error por truncamiento

De forma similar al caso del error por redondeo, hay que distinguir entre el error "externo" que provocamos cuando truncamos un número a cierta cantidad de decimales (p. ej. 9.99 a 9.9) y el error de truncamiento que interesa fundamentalmente en el análisis numérico. Este último tiene que ver con la famosa serie de Taylor[16] y la propiedad de que toda función analítica puede ser aproximada mediante polinomios.

[16]Brook Taylor (1685-1731). Analista, geómetra y filósofo inglés. Publicó libros de perspectiva y diferencias finitas.

Teorema 1.3.1 (Teorema de Taylor)**.** *Sea $f(x)$ una función en los reales definida en el intervalo $[a, b]$ y que satisface*

1. *$f(x) \in C^n[a, b]$*
2. *$f^{n+1}(x)$ existe para $x \in (a, b)$*

Entonces, para cualesquier x_0, x en $[a, b]$,

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + R_n(x) \tag{1.11}$$

donde $R_n(x) = \frac{f^{(n+1)}(c_x)}{(n+1)!}(x - x_0)^{n+1}$ (para c_x entre x_0 y x inclusive) se denomina residuo *y a la sumatoria llamamos* polinomio de Taylor[17] *de orden n, denotado por $P_n(x)$. Cuando fijamos x_0, la ecuación (1.11) se denomina* expansión de Taylor de orden n de $f(x)$ alrededor de x_0*. Además si hacemos $h = x - x_0$ podemos expresar el* orden de crecimiento[18] *del residuo como $R_n(x) = O(h^{n+1})$* ◇

Demostración. (Adaptada de Banach [3] pp. 114-116) Consideremos la ecuación:

$$\begin{aligned} f(x) = f(x_0) + \frac{x - x_0}{1!}f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \cdots \\ + \frac{(x - x_0)^n}{n!}f^{(n)}(x_0) + w(x - x_0)^{n+1} \end{aligned}$$

Si la ecuación anterior ha de cumplirse, encontraremos que w deberá ser igual a $\frac{f^{(n+1)}(\xi)}{(n+1)!}$ (que corresponde al factor correspondiente en $R_n(x)$) para un valor de ξ entre x_0 y x.

Definamos $\varphi(t)$ como sigue:

$$\begin{aligned} \varphi(t) = f(x) - f(t) - \frac{x - t}{1!}f'(t) - \frac{(x - t)^2}{2!}f''(t) - \cdots \\ - \frac{(x - t)^n}{n!}f^{(n)}(t) - w(x - t)^{n+1} \end{aligned}$$

[17] Cuando $x_0 = 0$ el nombre de la serie cambia a serie de MacLaurin, en honor a Colin MacLaurin (1698-1746) matemático y físico escocés.

[18] La notación básica para *orden de crecimiento* es (usando nuestra "variable" h) $O(g(h))$ (pron. "O grande" de $g(h)$) y nos proporciona una estimación del error. Si escribimos $f(h) = O(g(h))$ (para toda h) esto significa que existe una constante C tal que se cumple $|f(h)| \leq C|g(h)|$ (para toda h). Para un excelente tratamiento de este tema y del análisis asintótico se recomienda el libro de Graham, Knuth y Patashnik [11, pp. 443-463].

Observamos que se cumple $\varphi(x_0) = \varphi(x) = 0$. Si ahora derivamos $\varphi(t)$ con respecto a t obtenemos:

$$\begin{aligned}\varphi'(t) = -f'(t) &- \left[\frac{x-t}{1!}f''(t) - f'(t)\right] \\ &- \left[\frac{(x-t)^2}{2!}f^{(3)}(t) - \frac{x-t}{1!}f''(t)\right] - \cdots \\ &- \left[\frac{(x-t)^n}{n!}f^{(n+1)}(t) - \frac{(x-t)^{(n-1)}}{(n-1)!}f^{(n)}(t)\right] \\ &+ w(n+1)(x-t)^n\end{aligned}$$

Simplificando la ecuación anterior (cancelando términos iguales) obtenemos:

$$\varphi'(t) = -\frac{(x-t)^n}{n!}f^{(n+1)}(t) + w(n+1)(x-t)^n$$

Aplicando ahora el teorema de Rolle[19], sabemos que existe una ξ entre x y x_0 tal que $\varphi'(\xi) = 0$, por tanto:

$$w(n+1)(x-t)^n = \frac{(x-t)^n}{n!}f^{(n+1)}(t)$$

de donde obtenemos el resultado deseado:

$$w = \tfrac{f^{(n+1)}(\xi)}{(n+1)!}, \text{ con } \xi \text{ entre } x \text{ y } x_0.$$

□

Nota: La fórmula para el residuo en el teorema anterior garantiza que $|R_n(x)| \leq |\hat{R}_n(x)|$, donde:

$$\hat{R}_n(x) = \frac{M_x}{(n+1)!}(x-x_0)^{n+1} \tag{1.12}$$

donde $M_x = max\{f^{(n+1)}(c_x)\}$ tal que c_x está entre x_0 y x inclusive. De esta forma, $|\hat{R}_n(x)|$ nos proporciona una *cota superior* efectiva para la magnitud del residuo.

[19]Michel Rolle (1652-1719). Matemático francés, especialista en análisis, álgebra y geometría. El teorema de Rolle nos dice que si una curva continua cruza el eje x en dos puntos y tiene una tangente en todos sus puntos intermedios, entonces tiene al menos una tangente paralela al eje x.

Ejemplo 1.3.3. Calculemos $P_3(x)$ alrededor de $x_0 = 0$ para la función e^x, y estimemos $\hat{R}_3(x)$.

$$\begin{aligned} P_3(x) &= \frac{e^0}{0!}x^0 + \frac{e^0}{1!}x^1 + \frac{e^0}{2!}x^2 + \frac{e^0}{3!}x^3 \\ &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 \\ R_3(x) &= \frac{e^{c_x}}{4!}x^4 \\ \hat{R}_3(x) &= \frac{M_x}{24}x^4 \end{aligned}$$

donde, debido a que la función e^x es *creciente*, tenemos:

$$M_x = \begin{cases} 1 & x \leq 0 \\ e^x & x > 0 \end{cases}$$

Ejercicio 1.3.4. Calcule $P_3(x)$, el polinomio de aproximación de Taylor de orden 3, alrededor de $x_0 = 0$, así como el residuo $R_e(x)$, para la función $f(x) = cos(x/2)$,

Definición 1.3.3 (Error por truncamiento)**.** Decimos que existe un error por truncamiento, cuando no podemos calcular un valor verdadero y_v de forma explícita y tenemos que reemplazarlo por un *cálculo aproximado* y_c, y lo denotamos por $er_{tr}(y_v) = y_v - y_c$. Normalmente, la aproximación y_c se obtiene mediante la suma de los primeros términos de una serie infinita convergente.

Ejemplo 1.3.4. Utilizando el resultado del ejemplo (1.3.3), calculemos los errores de truncamiento, cuando:

a) Aproximamos e con $P_3(1)$

b) Aproximamos e^{-1} con $P_3(-1)$

Para el caso (a), tenemos $y_v = e$ y $y_c = P_3(1)$, por lo cual, utilizando siete decimales obtenemos: $y_v = 2.7182818$, y

$$\begin{aligned} y_c &= 1 + 1 + \frac{1}{2}(1)^2 + \frac{1}{6}(1)^3 \\ &= 2 + 1/2 + 1/6 \\ &= 8/3 \end{aligned}$$

Ya que $8/3 = 2.6666667$ (redondeando a siete decimales,) entonces
$er_{tr}(2.7182818) = 2.7182818 - 2.6666667 = 0.0516151$.

Para el caso (b) tenemos $y_v = e^{-1}$ y $y_c = P_3(-1)$, por lo cual, utilizando siete decimales obtenemos: $y_v = 0.3678794$, y

$$\begin{aligned} y_c &= 1 + (-1) + \frac{1}{2}(-1)^2 + \frac{1}{6}(-1)^3 \\ &= 0 + 1/2 - 1/6 \\ &= 1/3 \end{aligned}$$

Ya que $1/3 = 0.3333333$ (redondeando a siete decimales,) entonces
$er_{tr}(0.3678794) = 0.3678794 - 0.3333333 = 0.0345461$.

Ejercicio 1.3.5. Verifique que los errores de truncamiento encontrados en el ejemplo (1.3.4), estan por debajo de los $\hat{R}_3(x)$ correspondientes encontrados en el ejemplo (1.3.3).

1.3.4. Error numérico total

Por lo visto hasta este punto, los métodos numéricos requieren un cuidadoso reconocimiento y estudio de los errores involucrados.

Definición 1.3.4 (Error numérico total)**.** El error numérico total es la suma de los errores por redondeo y los errores por truncamiento.

Los errores por redondeo y por truncamiento guardan una relación muy interesante que señala Chapra y Canale [6, pp. 97-99] y denominan *principio de incertidumbre numérica*, ya que conforme aumenta el incremento h (p. ej. $(x - x_0)$ en la expansión de Taylor) también aumenta el error por truncamiento, y para reducirlo es necesario aumentar el número de términos en la sumatoria, pero al hacerlo aumenta el número de operaciones de punto flotante, lo que aumenta el error por redondeo. Se trata entonces de encontrar un *punto de equilibrio* a partir del cual, se tienen *rendimientos decrecientes*.

1.4. Software de cómputo numérico

El software numérico actual[20] ofrece un panorama muy prometedor, ya que además de la calidad en los programas y la búsqueda de conectividad en-

[20]Que incluye con frecuencia funcionalidad propia de los CAS (*Computer Algebra Systems*).

tre los diferentes sistemas, también se busca estandarizar algunos aspectos de la semántica (ver p. ej. la iniciativa del *Numerical Mathematics Consortium* [20]).

1.4.1. Software de acceso libre

Entre los sistemas de acceso libre orientados al software numérico se podemos incluir:

- Axiom: sistema de álgebra computacional (con MathAction)
- Calc3D: software para cálculo y graficación orientado a geometría y estadística
- EULER: poderoso laboratorio de computación numérica (con Yacas)
- FreeMat: Colección de funciones básicas para matemáticas
- GnuPlot: Excelente graficador de dominio público (no GNU)
- Jacal: Sistema de álgebra computacional basado en Scheme
- Mathscribe: herramientas para mentes científicas
- NonEuclid: software interactivo en Java para geometría hiperbólica
- Octave: excelente sistema afin a MATLAB
- PyLab: una colección de funciones para cálculo numérico y visualización. Basado en Python
- RLab: laboratorio para computación numérica
- Sage: Sistema integrador de gran potencia y versatilidad que incluye otros paquetes matemáticos *Open Source* de alta calidad. http://www.sagemath.org
- Scilab: uno de los paquetes de computación numérica y científica más importantes y exitosos (desarrollado en el instituto francés INRIA)
- Singular: un sistema de álgebra computacional para computación con polinomios

- Surf: software para visualización de geometría algebraica real
- Winplot: un programa sencillo pero muy versátil para graficar funciones matemáticas
- wxMaxima: un paquete clásico para matemáticas numéricas y computación simbólica. Sistema basado en Lisp

1.4.2. Software comercial

Entre los sistemas más relevantes tenemos:

- Derive: Sistema shareware para cómputo numérico y simbólico.
- LabView: Plataforma de cómputo numérico y simulación con énfasis en sistemas electrónicos empotrados, de gran importancia en la industria
- MAPLE: Sistema preferido en ambientes académicos y cuyo núcleo de procesamiento simbólico se incorpora en otros sistemas comerciales
- MathCAD: Editor de documentos que integra valiosas capacidades de cómputo numérico y de visualización
- Mathematica: Sofisticado y muy exitoso sistema de cómputo numérico y simbólico, con grandes capacidades de visualización
- MATLAB: Abreviación de "MATrix LABoratory", este es el sistema estándar en aplicaciones de ingeniería
- Scientific Workplace: Excelente editor científico de gran flexibilidad y que integra MAPLE como su núcleo de computación simbólica

1.4.3. Bibliotecas de funciones

Orientadas principalmente a la eficiencia en la solución de sistemas lineales de dimensión muy grande, estas bibliotecas son esenciales para el desarrollo de aplicaciones en la industria, así como en laboratorios de investigación. Entre las más recomendadas podemos mencionar:

- IMSL
- JavaNumerics

- LAPACK/BLAS
- EISPACK
- LINPACK
- NAG

En las próximas revisiones se presentarán referencias y una breve descripción de estos importantes sistemas de apoyo a la computación numérica.

1.5. Métodos iterativos

A diferencia de los **métodos directos** donde la solución a una ecuación o sistema de ecuaciones se logra siempre "al primer intento" siguiendo paso a paso un procedimiento determinado[21], los **métodos iterativos** obtienen la solución como resultado de una serie de aproximaciones generadas sucesivamente a partir de una "aproximación inicial a la solución".

Definición 1.5.1 (Método iterativo)**.** Llamamos **Método iterativo** (univariable) a un procedimiento que acepta:

a) $f(x)$, la función a iterar[22], tal que $ran(f) \subseteq dom(f)$ y además cumple cierto **criterio de convergencia**[23]

b) $x_0 \in dom(f)$, la aproximación inicial a la solución

c) ϵ, la tolerancia del error

y encuentra, mediante un número finito de iteraciones, una solución aproximada x^* (con error $\leq \epsilon$) , para la ecuación $x = f(x)$. Llamamos **punto fijo** al valor x que cumple $x = f(x)$.

Normalmente, la implementación de un método iterativo incluye como parámetro adicional, el número máximo de iteraciones permitidas, para el

[21] Como ejemplo podemos mencionar el procedimiento para calcular la derivada de una función que veremos en la sección 4.1

[22] Iterar una función significa aquí, *aplicarla repetidamente*, p. ej. $x_3 := f(f(f(x_0)))$ asigna a x_3 el valor de la tercera iteración de f aplicada a x_0

[23] P. ej. la *condición de Lipschitz*, que veremos en la sección 2.3.1

caso en que se utilice una función que no cumpla con el *criterio de convergencia*.

En la figura 1.2 presentamos el código Scheme para una máquina iterativa básica, que utilizando el enfoque de programación orientada a objetos, implementa el método iterativo univariable de la definición 1.5.1 anterior.

Ejemplo 1.5.1. Resolvamos la ecuación $x = cos(x + \frac{1}{4})$ con ayuda de la función crea-mit.

```
> (define mit (crea-mit 'g (lambda(x)
                             (+ (cos x) 1/4))
                           0.01 15))
> (mit 'itera 0.7)
(0 : 0.7)
(1 : 1.0148421872844886)
(2 : 0.7777539838103506)
(3 : 0.9624913205131414)
(4 : 0.8214773395993339)
(5 : 0.9311403125086086)
(6 : 0.8469194965541674)
(7 : 0.9122943326484682)
(8 : 0.8619327515053936)
(9 : 0.900971529122722)
(10 : 0.8708486501259518)
(11 : 0.8941776672648917)
(12 : 0.8761601962888252)
(13 : 0.8901059258316943)
(0.7
 converge
 a
 0.8793297110716177
 en
 14
 iteraciones)
>
```

Código Scheme 1.2: Función crea-mit para crear una máquina iterativa básica

```
;; función crea-mit (rev. 02, 2008-05-27; en mit.scm)
;; propósito:: crear una Máquina ITerativa (MIT) básica
;; atributos:: idf : string          ; id de la función
;;               f : procedimiento   ; función a iterar
;;            epsi : real            ; tolerancia
;;               n : entero positivo; máx de iteraciones
;; métodos:: 'show          -> presenta los atributos
;;           'itera <real> -> itera empezando con <real>
;;           'fun <proced> -> redefine la función f
;;           'idf <string> -> redefine nombre función
;;
(define (crea-mit idf f epsi n)
  (let ((imit idf) (fmit f) (emit epsi) (nmit n))
    (lambda(mensaje . param)
      (case mensaje
        ((itera)
         (display `(0 : ,(car param))) (newline)
         (do ((k 1 (+ k 1))
              (xp (car param) (fmit xp))
              (x (fmit (car param)) (fmit x)))
           ((or (<= (abs (- x xp)) emit)
                (>= k nmit))
            (if (<= k nmit)
                `(,(car param) converge a ,x en
                   ,k iteraciones)
                `(,(car param) no converge en
                   ,k iteraciones)))
           (display `(,k : ,x ,(fmit x) ,(abs (- x xp))))
           (newline)))
        ((fun) (set! fmit (car param)))
        ((eps) (set! emit (car param)))
        ((id) (set! imit (car param)))
        ((show) (display `("MIT" idf=,imit eps=
                                    ,emit n=,nmit))
                (newline))
        (else `(no conozco el mensaje ,mensaje))))))
```

Nota: en la sección 3.1, veremos métodos iterativos aplicados a la solución de sistemas de ecuaciones lineales de la forma $Ax = b$.

Capítulo 2

Métodos de solución de ecuaciones

Presentamos aquí algunos métodos numéricos fundamentales para la solución de **una ecuación no lineal univariable**. Los sistemas de ecuaciones no lineales serán tratados en el siguiente capítulo.

2.1. Métodos basados en intervalos

Cuando para encontrar la solución a una ecuación, digamos $f(x) = 0$ partimos de un intervalo $[a, b]$ dentro del cual sabemos que se encuentra la solución, y paso a paso reducimos dicho intervalo hasta obtener $[a_n, b_n]$ tal que $|b_n - a_n| < \epsilon$, para $\epsilon > 0$ como la tolerancia, decimos que hemos utilizado un método de intervalo o método cerrado.

El método de intervalo clásico es el método de bisección de Bolzano, que veremos en la sección siguiente (2.2).

Como se mencionó en la sección 1.2.2, recientemente se han desarrollado métodos basados en intervalos (en general hiper-cajas) que van "encerrando" la solución en regiones cada vez más pequeñas, logrando con ello lo que actualmente se conoce como computación confiable (en ingl. *reliable computing*). Para mayores detalles sobre estos nuevos métodos se recomienda consultar [27].

2.2. Método de bisección

El método de bisección se basa en el siguiente teorema de Bolzano[1]

Teorema 2.2.1 (Teorema de Bolzano)**.** *Una función $f : \mathbf{R} \to \mathbf{R}$ es cero de al menos un valor de x entre a y b si f es continua en el intervalo cerrado $[a, b]$ y $f(a)$ y $f(b)$ tienen signos opuestos.*

La estrategia de este método, es partir de un intervalo $[a, b]$ que cumple la condición $f(a)f(b) < 0$ y en cada iteración bisectarlo para obtener un nuevo intervalo $[a_1, b_1]$ que también cumple con $f(a_1)f(b_1) < 0$, hasta obtener un intervalo $[a_n, b_n]$ que cumple $f(a_n)f(b_n) < 0$ pero además $|b - a| \leq \epsilon$, para un ϵ correspondiente a la tolerancia del error.

Presentamos a continuación el algoritmo resultante:

Algoritmo 2.1 Método de bisección

Dado: f: función asociada a la ecuación $f(x) = 0$ a resolver, a y b: valores iniciales que cumplen $f(a)f(b) < 0$, ϵ: tolerancia del error.

Entrega: c: la solución aproximada a la ecuación $f(x) = 0$ con un error $\leq \epsilon$.

```
BISECCIÓN(f, a, b, ε)
    repeat
        s ← (a + b)/2
        if f(s) = 0 then                          ▷ caso excepcional
            return s
        else if f(a)f(s) < 0 then
            b ← s
        else
            a ← s
        end if
    until |b − a| ≤ ε
    return s
end
```

Ejemplo 2.2.1. La siguiente interacción con la función **biseccion** (ver código 2.1) nos permite resolver $cos(x) = 0$ con una tolerancia de 10^{-3} mediante el método de bisección, utilizando $a = 0$ y $b = 2$.

[1]Bernhard Bolzano (1781-1848), matemático checoescovaco especializado en análisis, cuyo trabajo ayudó a establecer la importancia de las demostraciones rigurosas [14].

Código Scheme 2.1: Función **biseccion** asociada al algoritmo 2.1

```
;; implementa el método de bisección
;; asume: f(a0)f(b0)<0
(define (biseccion f a0 b0 ep)
  (let ciclo ((a a0) (b b0))
    (let ((s (/ (+ a b) 2)))      ; biseccion de [a,b]
      (cond [(zero? (f s)) s]     ; f(s)=0? => entrega s
            [(< (abs (- b a)) ep); cumple tolerancia?
             s]                   ;   entrega s
            [(< (* (f a) (f s)) 0) ; f(a)f(s)<0?
             (ciclo a s)]         ;   busca en [a,s]
            [else (ciclo s b)])))) ; busca en [s,b]
```

```
> (biseccion cos 0 2. .001)
1.57080078125
```

A continuación presentamos las interacciones resultantes al resolver este problema mediante la función **biseccion-data** (presentada en el código 2.2)

```
> (biseccion-data cos 0 2. .001 6)
(1.57080078125
 -4.454455103366037e-006
 (0 0 2.0)
 (1 1.0 2.0)
 (2 1.5 2.0)
 (3 1.5 1.75)
 (4 1.5 1.625)
 (5 1.5625 1.625)
 (6 1.5625 1.59375)
 (7 1.5625 1.578125)
 (8 1.570312 1.578125)
 (9 1.570312 1.574219)
 (10 1.570312 1.572266)
 (11 1.570312 1.571289))
```

Note que en este caso, la función entrega una lista que contiene respectivamente, la solución s, el error $f(s)$ y los datos de las iteraciones en la forma

$(k_i\ a_i\ b_i)$.

Código Scheme 2.2: Función **biseccion-data** asociada al algoritmo 2.1

```
;; implementa el método de bisección
;; entrega (solucion error data)
;; donde: data = {i a(i) b(i)}
;; asume: f(a0)f(b0)<0
(define (biseccion-data f a0 b0 ep decs)
  (let ((ft (make-formater-round-decs decs))
        (data '()))
    (let ciclo ((a a0) (b b0) (k 0))
      (set! data (cons (list k (ft a) (ft b))
                       data))
      (let ((s (/ (+ a b) 2)))
        (cond [(zero? (f s))          ; caso excepcional
               (cons s (cons 0 (reverse data)))]
              [(< (abs (- b a)) ep) ; cumple tolerancia?
               (cons s (cons (f s)  ; entrega s, f(s)
                     (reverse data)))]; y data en orden
              [(< (* (f a) (f s)) 0)  ; f(a)f(s)<0?
               (ciclo a s (+ k 1))]   ; busca en [a,s]
              [else (ciclo s b        ; busca en [s,b]
                           (+ k 1))])))))
```

Sugerencia 2.1. Si la función $f(x)$ de nuestro problema no es muy conocida, o demasiado laboriosa para bosquejar "a mano", se recomienda utilizar alguno de los paquetes de software matemático vistos en la sección 1.4, y hacer un análisis gráfico de la función, para determinar el número de ceros dentro de cierto intervalo, así como para determinar los valores cercanos a dicho cero(s). Este análisis será especialmente importante para el método de Newton-Raphson que veremos en la sección 2.4.1.

Ejemplo 2.2.2. Resolvamos ahora la ecuación $3cos(\frac{x}{2}) + 2 = 0$ con una tolerancia de 10^{-2}, llamando a la función **biseccion-data** para $a = 1$ y $b = 6$.

```
>(biseccion-data (lambda(x)(+ (* 3 (cos (/ x 2))) 2))
                 1 6. 0.01 6)
```

```
(4.5986328125
 0.002701681335155026
 (0  1  6.0)
 (1  3.5  6.0)
 (2  3.5  4.75)
 (3  4.125  4.75)
 (4  4.4375  4.75)
 (5  4.59375  4.75)
 (6  4.59375  4.671875)
 (7  4.59375  4.632812)
 (8  4.59375  4.613281)
 (9  4.59375  4.603516))
```

La figura 2.2 nos permite visualizar el proceso para este ejemplo.

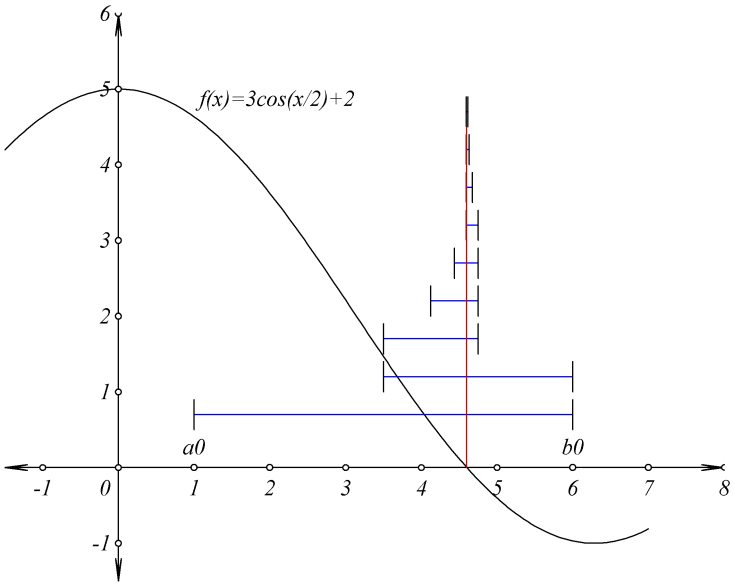


Figura 2.1: Bisecciones para el ejemplo 2.2.2

Ejercicio 2.2.1. Encuentre todos los ceros de las siguientes funciones en el intervalo indicado y para la precisión indicada.

a). $f(x) = \sqrt{\frac{2}{\pi x}} cos(x - \frac{\pi}{4})$ en $[1, 7]$, con $\epsilon = 0.001$

b). $f(x) = xsenh(x - 2) - 100$, en $[-4, 8]$, con $\epsilon = 10^{-7}$

Problema 2.2.1. ¿Cuál es el número máximo posible de iteraciones del método de bisección, dado un intervalo $[a, b]$ y un valor para ϵ?

Problema 2.2.2. Una vez resuelto el problema 2.2.1, encuentre una función $f(x)$ así como su intervalo $[a, b]$ y ϵ asociado, de manera que se encuentre el cero justo en el número máximo de iteraciones.

Sugerencia 2.2. Explore algunas otras implementaciones del método de bisección. Se recomienda especialmente revisar ejemplos en MATLAB o si-milares.

2.3. Método de aproximaciones sucesivas

Este método, conocido también como método de punto fijo o método iterativo estándar es uno de los métodos recomendados cuando queremos resolver una ecuación de la forma $x = f(x)$ y la función $f(x)$ cumple ciertas condiciones como lo veremos en las siguientes dos secciones.

2.3.1. Condición de Lipschitz

Definición 2.3.1 (Condición de Lipschitz)**.** Una función $f(x)$ definida en el intervalo $[a, b]$ se dice que satisface una condición de Lipschitz, si existe una constante $L > 0$ tal que

$$|f(x_1) - f(x_2)| \leq L\,|x_1 - x_2| \tag{2.1}$$

para cualquier par de números $x_1, x_2 \in [a, b]$.

Observamos que cualquier función $f(x)$ donde la expresión $\frac{|f(x_1)-f(x_2)|}{|x_1-x_2|}$ se pueda simplificar a $\frac{K}{g(x_1,x_2)}$, donde K es una constante y el valor de $g(x_1, x_2)$ se pueda hacer arbitrariamente pequeño para $x_1, x_2 \in [a, b]$, no puede satisfacer una condición de Lipschitz.

Si $f(x) \in C^1[a,b]$ (es decir, si $f'(x)$ existe y es continua) entonces $f(x)$ satisface también una condición de Lipschitz. Esto se cumple ya que, por el *teorema del valor medio para derivadas* existe una c entre x_1 y x_2 tal que:

$$f'(c) = \frac{f(x_1) - f(x_2)}{x_1 - x_2} \tag{2.2}$$

y entonces, $|f(x_1) - f(x_2)| = |f'(c)(x_1 - x_2)| \leq L\,|x_1 - x_2|$, para cualesquiera $x_1, x_2 \in [a,b]$.

2.3.2. Iteración y convergencia

El método de **aproximaciones sucesivas**, **método iterativo** y también conocido como **método de punto fijo**, es uno de los métodos más sencillos e ilustra (a diferencia del método de bisección) el caso cuando no se tiene garantía de obtener la solución. Por tal motivo, el tema central aquí es el concepto de convergencia de una sucesión de aproximaciones.

Definición 2.3.2 (Velocidad de convergencia)**.** Sea $\{x_n\}_{n=1}^{\infty}$ una sucesión de aproximaciones que convergen a s, de forma que $\lim_{n\to\infty} x_n = s$. Si la sucesión de errores $\{\epsilon_n\}_{n=1}^{\infty}$ (donde $\epsilon_n = x_n - s$) satisface

$$\lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^{\alpha}} = K, \quad K > 0 \tag{2.3}$$

para algunos números fijos α, K, entonces α es el orden de convergencia para $\{x_n\}$, y K es la constante asintótica o factor de convergencia.

En base a la definición anterior, destacamos los casos para cuando $\alpha = 1$, y $\alpha = 2$ que corresponden a convergencia lineal, y convergencia cuadrática respectivamente.

El algoritmo 2.2 describe el método.

Algoritmo 2.2 Método de punto fijo

Dado: f: función asociada a la ecuación $f(x) = x$ a resolver, x_0: valor inicial de la solución, ϵ: tolerancia del error, m: número máximo de iteraciones permitidas.

Entrega: s: la solución aproximada a la ecuación $f(x) = x$ con un error $\leq \epsilon$, o bien el mensaje: "sin convergencia en m iteraciones" si la solución no converge en las iteraciones y la precisión deseada.

```
PUNTO-FIJO(f, x_0, ε, m)
    if f(x_0) = x_0 then                          ▷ caso excepcional
        return x_0
    else
        n ← 0
        repeat
            x_{n+1} ← f(x_n)
            n ← n + 1
        until |x_n − x_{n−1}| ≤ ε ó n > m
        if |x_n − x_{n−1}| ≤ ε then
            return x_n
        else
            return "sin convergencia en m iteraciones"
        end if
    end if
end
```

Presentamos en el listado 2.3, una implementación básica del método.

Código Scheme 2.3: Función **punto-fijo** asociada al algoritmo 2.2

```
;; implementa el método de punto fijo
(define (punto-fijo f x0 eps m)
  (if (= (- (f x0) x0) 0)                ; f(x0)=x0?
      x0                                 ; entrega x0
      (let ciclo ((k 1) (x (f x0)))      ; x1=f(x0)
        (cond
          ((<= (abs (- x (f x))) eps)    ; |f(x)-x|<eps
           (f x))                        ; entrega f(x)
          ((> k m)                       ; k>m?
           (string-append
            "sin convergencia en " (number->string m)
            " iteraciones"))             ; mensaje
          (else                          ; eoc
           (ciclo (+ k 1) (f x)))))))    ; x(k+1)=f[x(k)]
```

Una variante de la función punto-fijo que nos da información adicional se presenta en el listado 2.4.

Código Scheme 2.4: Función **punto-fijo-data** asociada al algoritmo 2.2

```
;; implementa el método de punto fijo
;; entrega ((solucion|mensaje) error data)
;; donde: data = {i f[x(i)]}
(define (punto-fijo-data f x0 eps m decs)
  (let ((ft (make-formater-round-decs decs))
        (data (list (list 0 x0))))
    (if (= (- (f x0) x0) 0)
        (cons x0 (cons 0 data))
        (let ciclo ((k 1) (x (f x0)))
          (set! data (cons (list k (ft x))
                           data))
          (cond ((<= (abs (- x (f x))) eps)
                 (cons (f x)
                       (cons (- (f x) x)
                             (reverse data))))
                ((>= k m)
                 (cons
                  (string-append
                   "sin convergencia en "
                   (number->string m)
                   " iteraciones")
                  (cons (- (f x) x)
                        (reverse data))))
                (else
                 (ciclo (+ k 1) (f x)))))))))
```

2.4. Métodos basados en interpolación

2.4.1. Método de Newton-Raphson

Entre los métodos más populares en matemáticas computacionales tenemos al creado por Isaac Newton[2] y Joseph Raphson[3].

El atractivo de éste método es su rapidez cuadrática de convergencia, como veremos más adelante. Una desventaja es que para la aplicación del método se requiere tener tanto la función $f(x)$ que quiere resolverse, así como su derivada.

Dada una función $f(x)$ definida en un intervalo $[a, b]$, tal que $f(x) \in C^2[a, b]$ y existe una $s \in [a, b]$ tal que $f(s) = 0$ entonces si escojemos un valor inicial de x cercano a s, podemos utilizar un polinomio interpolador de Taylor para aproximar la función $f(x)$ por:

$$f(s) = f(x_0) + \frac{f'(x_0)}{1!}(s - x_0) + \frac{f''(c)}{2!}(s - x_0)^2$$

Haciendo $f(s) = 0$ y asumiendo que el término de segundo grado es muy pequeño, se tiene:

$$0 \approx f(x_0) + \frac{f'(x_0)}{1!}(s - x_0)$$

Ahora podemos despejar s para obtener:

$$s \approx x_0 - \frac{f(x_0)}{f(x_0)}$$

Si lo expresamos en términos del método iterativo univariable, la función a iterar es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.4}$$

El pseudocódigo para el método de Newton-Raphson estándar se presenta en el algoritmo 2.3

[2]Sir Isaac Newton (1642-1727), reconocido matemático, físico y astrónomo inglés, su obra maestra *Philosophae Naturalis Principia Mathematica* es considerada como uno de los libros científicos más importantes de todos los tiempos. Destaca junto con (Gottfried Wilhelm von) Leibniz (1646-1716), por su invención independiente del Cálculo.

[3]Joseph Raphson (1648-1715), matemático inglés

Algoritmo 2.3 Método de Newton-Raphson estándar

Dado: f: función asociada a la ecuación $f(x) = 0$ a resolver, f': derivada de $f(x)$, x_0: aproximación inicial a la solución, ϵ: tolerancia del error, y m: número máximo de iteraciones

Entrega: s: la solución aproximada a la ecuación $f(x) = 0$, o bien el mensaje "sin solución aceptable en m iteraciones", o bien "$f'(x_n) = 0$ en la iteración n"

```
NEWTON-RAPHSON(f, f', x_0, ε, m)
    n ← 0
    repeat
        if f'(x_n) = 0 then return "f'(x_n) = 0 en la iteración n"
        else
            x_{n+1} = x_n − f(x_n)/f'(x_n)
            n ← n + 1
        end if
    until |x_n − x_{n−1}| ≤ ε ó n > m
    if |x_n − x_{n−1}| ≤ ε then return x_n
    else return "sin solución aceptable en m iteraciones"
    end if
end
```

Presentamos a continuación una implementación en Scheme:

Código Scheme 2.5: Función para el método de Newton-Raphson

```
(define (newton-raphson f df x0 eps m)
  (display `(x(,0)= ,x0)) (newline)
  (let ciclo ((xp x0) (dfp (df x0)) (k 1))
    (if (= dfp 0)
        `(lo siento: la derivada de f en ,x0
             es 0 en la ,k iteracion)
        (let ((x (- xp (/ (f xp) dfp))))
          (cond
            ((<= (abs (- x xp)) eps)
             (display `(x(,k)= ,x delta= ,(abs (- x xp))))
             (newline)
             `(solucion ,x en ,k iteraciones))
            ((> k m) `(sin solucion en
                           ,(- k 1) iteraciones))
            (else (display `(x(,k)= ,x delta=
                               ,(abs (- x xp))))
                  (newline)
                  (ciclo x (df x) (+ k 1)))))))))
```

Ejercicio 2.4.1. Aplique el método de Newton-Raphson para encontrar una raiz de la ecuación $x^3 + x - 2 = 0$. Utilice $x_0 = 2$ y $\epsilon = 10^{-5}$. Compruebe la solución encontrada.

2.4.2. Método de la secante

Este método se basa en la utilización de dos puntos x_0 y x_1 como aproxi-maciones iniciales a la solución de la ecuación $f(x) = 0$, y calcula el tercer punto x_2, resolviendo la siguiente ecuación de la secante, para $y = 0$ y $x_2 = x$:

$$y - f(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1) \tag{2.5}$$

para obtener:

$$x_2 = x_1 - \frac{(x_1 - x_0)f(x_1)}{f(x_1) - f(x_0)}$$

si renombramos ahora x_2, x_1 y x_0 por x_{n+1}, x_n y x_{n-1} respectivamente, obtenemos la ecuación recursiva que caracteriza al método de la secante[4]:

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1})f(x_n)}{f(x_n) - f(x_{n-1})} \tag{2.6}$$

Algoritmo 2.4 Método de la secante

Dado: f: función asociada a la ecuación $f(x) = 0$ a resolver, x_0 y x_1: dos aproximaciones iniciales a la solución, ϵ: tolerancia del error, y m: número máximo de iteraciones

Entrega: s: la solución aproximada a la ecuación $f(x) = 0$, o bien el mensaje "sin solución aceptable en m iteraciones"

```
METODO-SECANTE(f, x_0, x_1, ε, m)
    n ← 1
    repeat
        if f(x_n) = f(x_{n-1}) then
            return "Sin solución: secante horizontal"
        else
            x_{n+1} ← x_n − (x_n − x_{n-1})f(x_n)/[f(x_n) − f(x_{n-1})]
            n ← n + 1
        end if
    until |x_n − x_{n-1}| ≤ ε ó n > m
    if |x_n − x_{n-1}| ≤ ε then
        return x_n
    else
        return "sin solución en m iteraciones"
    end if
end
```

2.4.3. Método de Aitken

Este método, también conocido como método de aceleración de Aitken o Método Δ^2 de Aitken, permite acelerar la convergencia de una sucesión convergente.

[4]Note que esta ecuación es semejante a la utilizada por el método de Newton-Raphson, considerando la aproximación $f'(x_{n-1}) \approx \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}$.

Algoritmo 2.5 Método de Aitken

Dado: f: función asociada a la sucesión convergente $\{x_n\}_{n=0}^{\infty}$ con $x_{n+1} = f(x_n)$, x_0: aproximacion inicial a la solución, ϵ: tolerancia del error, y m: número máximo de iteraciones

Entrega: x^*: la solución aproximada a la ecuación $f(x) = 0$, o bien el mensaje "sin solución aceptable en m iteraciones"

```
    METODO-AITKEN(f, x_0, ε, m)
2:      x_1 ← f(x_0)
        x_2 ← f(x_1)
4:      n ← 0
        repeat
6:          x_{n+3} ← f(x_{n+2})
            x'_n ← x_n − (x_{n+1} − x_n)^2/(x_{n+2} − 2x_{n+1} + x_n)
8:          x'_{n+1} ← x_{n+1} − (x_{n+2} − x_{n+1})^2/(x_{n+3} − 2x_{n+2} + x_{n+1})
            n ← n + 1
10:     until |x'_n − x'_{n−1}| ≤ ε ó n > m
        if |x'_n − x'_{n−1}| ≤ ε then
12:         return x'_n
        else
14:         return "sin solución en m iteraciones"
        end if
16: end
```

2.5. Método de Bairstow

En esta sección veremos un método para resolver el caso cuando $f(x) = p(x)$, es decir $f(x)$ es un polinomio en una variable. Así, nos interesará resolver la ecuación $p(z) = 0$. Note que hemos cambiado el símbolo de la variable de x a z para indicar que trabajaremos en el conjunto de los número complejos $\mathbf{C}$.

Denotaremos nuestro polinomio de grado $n > 2$ por $p_n(z)$ de la forma:

$$p_n(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 \tag{2.7}$$

En general, este método consiste en encontrar iterativamente factores cuadráticos de la forma $(Az^2 + Bz + C)$ para $p_n(z)$, reduciendo cada vez en dos el grado del polinomio, es decir, en cada iteración encontramos un

polinomio $p_{n-2}(z)$ de grado $n-2$. Como las raíces complejas se presentan en pares conjugados, tenemos que:

$$[z-(a+ib)][z-(a-ib)] = z^2 + 2az + (a^2+b^2) = z^2 + Bz + C$$

En lo sucesivo usaremos $B = -r$ y $C = -s$, en conformidad con la notación propia del método. Así buscaremos un $p_{n-2}(z)$ que cumpla:

$$p_n(z) = (z^2 - rz - s)p_{n-2}(z) \tag{2.8}$$

Repetiremos el proceso anterior hasta llegar al caso cuando el grado k del polinomio reducido $p_k(z)$, cumpla con $k \leq 2$ que se resuelve de manera directa.

Para encontrar los valores de r y s que cumplan con la ecuación 2.8, empezaremos con valores iniciales r_0 y s_0. Si realizamos la división de $p_n(z)$ entre el factor cuadrático $z^2 - r_0 z - s_0$, obtenemos el polinomio:

$$q_{n-2}(z) = b_n z^{n-2} + b_{n-1} z^{n-3} + \cdots + b_3 z + b_2$$

de manera que:

$$p_n(z) = (z^2 - r_0 z - s_0)q_{n-2}(z) + R$$

donde en residuo está determinado por:

$$R = b_1(z - r_0) + b_0 \tag{2.9}$$

y los coeficientes $\{b_i\}_0^n$ se calculan recursivamente con las fórmulas siguientes, considerando $k = 0$:

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + r_k b_n \\ b_i &= a_i + r_k b_{i+1} + s_k b_{i+2}, \text{ para } i = n-2, n-3, \ldots, 0 \end{aligned} \tag{2.10}$$

Si resulta que $R = 0$, hemos encontrado el factor cuadrático deseado con $r = r_0$ y $s = s_0$, en caso contrario, utilizamos el **método de Newton-Raphson** para sistemas de ecuaciones (ver sección 3.3.1) hasta encontrar los valores r_k y s_k que logren que $R = 0$, para algún $k > 0$. En cuyo caso $r = r_k$, $s = s_k$ y como $R = 0$, entonces $p_{n-2}(z) = q_{n-2}(z)$ es el siguiente polinomio a iterar.

Para lograr $R = 0$ en la ecuación 2.9, y como b_0 y b_1 dependen de r_k y s_k, buscaremos los cambios Δr y Δs que logren hacer cero el siguiente sistema de ecuaciones[5]:

$$b_0(r_k + \Delta r, s_k + \Delta s) = b_0 + \frac{\partial b_0}{\partial r_k}\Delta r + \frac{\partial b_0}{\partial s_k}\Delta s \tag{2.11}$$

$$b_1(r_k + \Delta r, s_k + \Delta s) = b_1 + \frac{\partial b_1}{\partial r_k}\Delta r + \frac{\partial b_1}{\partial s_k}\Delta s \tag{2.12}$$

Por tanto, se debe cumplir:

$$\begin{aligned} \frac{\partial b_0}{\partial r_k}\Delta r + \frac{\partial b_0}{\partial s_k}\Delta s &= -b_0 \\ \frac{\partial b_1}{\partial r_k}\Delta r + \frac{\partial b_1}{\partial s_k}\Delta s &= -b_1 \end{aligned} \tag{2.13}$$

El trabajo de Bairstow demostró que podemos obtener las derivadas parciales de la ecuación 2.13 a partir de los coeficientes b_i con el siguiente sistema recursivo:

$$\begin{aligned} c_n &= b_n \\ c_{n-1} &= b_{n-1} + r_k c_n \\ c_i &= b_i + r_k c_{i+1} + s_k c_{i+2}, \text{ para } i = n-2, n-3, \ldots, 0 \end{aligned} \tag{2.14}$$

donde las primeras tres c_i corresponden a las derivadas parciales buscadas, es decir: $\frac{\partial b_0}{\partial r_k} = c_1$, $\frac{\partial b_0}{\partial s_k} = \frac{\partial b_1}{\partial r_k} = c_2$ y $\frac{\partial b_1}{\partial s_k} = c_3$. De esta forma, resolviendo el sistema:

$$\begin{aligned} c_1\Delta r + c_2\Delta s &= -b_0 \\ c_2\Delta r + c_3\Delta s &= -b_1 \end{aligned} \tag{2.15}$$

estaremos mejorando las aproximaciones de r_k y s_k con los ajustes:

$$\begin{aligned} r_{k+1} &= r_k + \Delta r \\ s_{k+1} &= s_k + \Delta s \end{aligned} \tag{2.16}$$

hasta que sus errores relativos estén dentro de la tolerancia permitida.

El algoritmo siguiente y su auxiliar, describen el método de Bairstow:

[5]Hemos simplificado aquí la notación, ya que estríctamente, utilizaríamos Δr_k y Δs_k.

Algoritmo 2.6 Método de Bairstow (requiere: algoritmo 2.7)

Dado: p: polinomio en una variable asociada la ecuación $p(z) = 0$ y representado por $(a_n, a_{n-1}, \ldots, a_0)$; r_0 y s_0: aproximaciones iniciales a los factores cuadráticos, ϵ_{rel}: error relativo en porcentaje para los factores cuadráticos, m: número máximo de iteraciones

Entrega: *raices*: conjunto de las soluciones a $p(z) = 0$, ó bien el mensaje "sin solución aceptable en m iteraciones"

```
    METODO-BAIRSTOW(p, r_0, s_0, ε_rel, m)
2:      raices ← φ
        n ← grado del polinomio p(z)
4:      while n > 2 do
            (q, r, s, mje) ← FACTOR-BAIRSTOW(p, r_0, s_0, ε_rel, m)
6:          if mje ≠ " " then
                return mje                          ▷ error en cálculo de factor
8:          else
                raices ← raices ∪ ceros del factor cuadrático z^2 − rz − s
10:             p ← q
                n ← grado del polinomio p
12:         end if
        end while
14:     if n = 2 then
            raices ← raices ∪ ceros del polinomio p
16:     else raices ← raices ∪ {−s/r}
        end if
18: end
```

Algoritmo 2.7 Factor cuadrático para Bairstow (algoritmo 2.6)

Dado: p: polinomio en una variable, representado por $(a_n, a_{n-1}, \ldots, a_0)$; r_0 y s_0: aproximaciones iniciales a los factores cuadráticos, ϵ_{rel}: error relativo en porcentaje para los factores cuadráticos, m: número máximo de iteraciones

Entrega: $(q(z), r, s, mje)$: valores que cumplen la ecuación $p(z) = (z^2 - rz - s)q(z)$ (donde, q está representado por sus coeficientes en orden descendente) y mje="", o bien el mje="sin solución aceptable en m iteraciones"

```
    FACTOR-BAIRSTOW(f, r_0, s_0, ε_rel, m)
 2:     k ← 0
        repeat
 4:         Calcula las b_i utilizando la ecuación 2.10
            Calcula las c_i utilizando la ecuación 2.14
 6:         Determina Δr_k y Δs_k resolviendo el sistema 2.15
            Calcula r_k y s_k aplicando las ecuaciones 2.16
 8:         k ← k + 1
        until (k > m)
10:     if k > m then
            return "Sin solución en m iteraciones"
12:     else
            q(z) está representado por las {b_i}
14:         return (q, r, s, "")
        end if
16: end
```

2.6. Aplicaciones

Aplicación 2.1 (Modelos poblacionales)**.**

Aplicación 2.2 (Determinación de valores característicos)**.**

Capítulo 3

Métodos de solución de sistemas de ecuaciones

3.1. Métodos iterativos para sistemas lineales

Cuando el número de variables de los sistemas a resolver es elevado (por ejemplo, arriba de 100) los métodos directos resultan imprácticos y recurrimos a los métodos iterativos, en esta sección presentaremos dos métodos clásicos fundamentales.

3.1.1. Método de Jacobi

Dado un sistema lineal:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad 1 \leq i \leq n \tag{3.1}$$

podemos despejar la variable x_i para obtener una expresión que relacione $x_i^{(k+1)}$ con $x_i^{(k)}$:

$$x_i^{(k+1)} = -\sum_{\substack{j=1 \\ j\neq i}}^{n} \left(\frac{a_{ij}}{a_{ii}} x_j^{(k)} \right) + \frac{b_i}{a_{ii}}, \quad 1 \leq i \leq n, \quad a_{ii} \neq 0 \tag{3.2}$$

El método de Jacobi consiste básicamente en aplicar la ecuación 3.2 para cada una de las variables, empezando con los valores iniciales correspondientes.

La forma matricial se deriva de forma similar. Partimos de la ecuación $\mathbf{Ax} = \mathbf{b}$ y el hecho de que podemos descomponer la matriz $\mathbf{A}$ como

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$$

, es decir en sus matrices diagonal principal, triangular inferior y triangular superior, respectivamente. Por tanto:

$$\mathbf{Ax} = (\mathbf{D} + \mathbf{L} + \mathbf{U})\mathbf{x} = \mathbf{b} \tag{3.3}$$

y entonces podemos despejar el término $\mathbf{Dx}$ para obtener:

$$\mathbf{Dx} = -(\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{b} \tag{3.4}$$

y como $a_{ii} \neq 0$ para $1 \leq i \leq n$, podemos despejar $\mathbf{x}$ en $\mathbf{Dx}$ para obtener:

$$\mathbf{x} = -\mathbf{D^{-1}}(\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{D^{-1}}\mathbf{b} \tag{3.5}$$

Utilizando este ecuación, obtenemos la ecuación para un paso del método de Jacobi, en forma matricial:

$$\mathbf{x}^{(k+1)} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b} \tag{3.6}$$

donde $\mathbf{x}^{(k)}$ representa el valor de las variables en la k-ésima iteración, empezando con el vector $\mathbf{x}^0$. El proceso se detendrá cuando la distancia euclideana $\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| < \epsilon$, para una determinada tolerancia del error ϵ, donde, claro:

$$\|\mathbf{X} - \mathbf{Y}\| = \left[\sum_{j=1}^{n}(x_i - y_i)^2\right]^{1/2} \tag{3.7}$$

3.1.2. Método de Gauss-Seidel

A diferencia del método de Jacobi, el Método de Gauss-Seidel, utiliza los valores previos de $x_j^{(k)}$, $1 \leq j \leq i-1$ para calcular el valor de $x_i^{(k+1)}$. Esto nos da la siguiente ecuación para el método de Gauss-Seidel:

$$x_i^{(k+1)} = -\frac{1}{a_{ii}}\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} - b_i\right) \tag{3.8}$$

El método de Gauss-Seidel consiste básicamente en aplicar la ecuación 3.8 para cada una de las variables, empezando con los valores iniciales correspondientes.

La forma matricial se deriva de forma similar al método de Jacobi. De la ecuación 3.3, despejamos ahora el término $(\mathbf{D}+\mathbf{L})\mathbf{x}$ para obtener:

$$(\mathbf{D}+\mathbf{L})\mathbf{x}^{(k+1)} = -\mathbf{U}\mathbf{x}^{(k)}+\mathbf{b} \tag{3.9}$$

de donde despejando, encontramos la ecuación de una iteración del método de Gauss-Seidel:

$$\mathbf{x}^{(k+1)} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(k)}+(\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} \tag{3.10}$$

Ya que asumimos que $a_{ii} \neq 0$, para $1 \leq i \leq n$, sabemos que $\mathbf{D}+\mathbf{L}$ es no singular y $\mathbf{x}^{(k)}$ está bien definida para toda iteración k.

3.2. Sistemas de ecuaciones no lineales

En la sección anterior hemos visto, los métodos iterativos básicos para resolver sistemas lineales. Ahora nos toca revisar métodos fundamentales para resolver sistemas de ecuaciones no lineales. Los métodos presentados son extensiones de los vistos en el capítulo 2.

3.2.1. Método iterativo secuencial

Presentamos aquí una generalización del método visto en la sección 2.3.2, donde presentamos el método de punto fijo (algoritmo 2.2.

Se trata aquí de resolver el sistema de ecuaciones:

$$x_i = g_i(x_1, x_2, \ldots, x_n), \quad \text{para, } i = 1, 2, \ldots, n \tag{3.11}$$

Escrito en notación vectorial, nuestro sistema es simplemente: $\mathbf{g}(\mathbf{x}) = \mathbf{x}$.

Para aplicar con éxito el método de punto fijo para sistemas iterando la ecuación:

$$\mathbf{x}_{k+1} = \mathbf{g}(\mathbf{x}_k) \tag{3.12}$$

y garantizar su convergencia a una solución única, requerimos que la función $\mathbf{g}(\mathbf{x})$, cumpla con las condiciones del siguiente teorema:

Teorema 3.2.1 (Convergencia del método de punto fijo para sistemas (adaptado de [8] pp.131-132)). *Sean $g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})$, funciones reales definidas sobre la hipercaja*

$$R := \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_n)^T |\ a_i \leq x_i \leq b_i, \quad i = 1, \ldots, n \right\}$$

y que satisfacen las siguientes condiciones:

1. $g_i(\mathbf{x}) \in C[R],\ i = 1, \ldots, n$

2. *Para cualquier* $\mathbf{x} \in R:\quad \mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})]^T \ \in R$

3. *Existe una constante de Lipschitz* $L < 1$ *para la cual se cumple la condición:*

$$\|\mathbf{g}(\mathbf{x}_1) - \mathbf{g}(\mathbf{x}_2)\| \leq L \|\mathbf{x}_1 - \mathbf{x}_2\|, \quad \mathbf{x}_1, \mathbf{x}_2 \in R \tag{3.13}$$

Entonces el sistema definido por la ecuación 3.11, tiene una solución única $\mathbf{s} \in R$ *y para cualquier* $\mathbf{x}_0 \in R$*, la sucesión* $\{\mathbf{x}_n\}$ *al aplicar la ecuación 3.12, converge a dicha solución.*

Si la función $\mathbf{g}(\mathbf{x})$ posee además primeras derivadas, una manera de verificar si el método de punto fijo para sistemas aplica, es determinando una cota superior para la constante de Lipschitz (ver [8] p. 132) utilizando la ecuación:

$$L = \max_{\mathbf{x} \in R} \left[\sum_{i,j=1}^{n} \left(\frac{\partial g_i}{\partial x_j} \right)^2 \right]^{1/2} \tag{3.14}$$

El algoritmo 3.1 describe los pasos de este método:

Algoritmo 3.1 Método de punto fijo para sistemas

Dado: $\mathbf{g}$: función asociada a la ecuación $\mathbf{x} = \mathbf{g}(\mathbf{x})$ a resolver, $\mathbf{x}_0$: valor inicial de la solución, ϵ: tolerancia del error, m: número máximo de iteraciones permitidas.

Entrega: $\mathbf{x}^*$: la solución aproximada a la ecuación $\mathbf{x} = \mathbf{g}(\mathbf{x})$ con un error $\leq \epsilon$, o bien el mensaje: "sin convergencia en m iteraciones" si la solución no converge en las iteraciones y la precisión deseada.

PUNTO-FIJO-SISTEMAS($\mathbf{g}, \mathbf{x}_0, \epsilon, m$)

```
    if g(x_0) = x_0 then                                  ▷ caso excepcional
        return x_0
    else
        n ← 0
        repeat
            x_{n+1} ← g(x_n)
            n ← n + 1
        until ‖x_n − x_{n−1}‖ ≤ ε ó n > m
        if ‖x_n − x_{n−1}‖ ≤ ε then
            return x_n
        else
            return "sin convergencia en m iteraciones"
        end if
    end if
end
```

3.3. Iteración y convergencia en sistemas de ecuaciones

3.3.1. Método de Newton-Raphson para sistemas

El método de Newton-Raphson visto en la sección 2.4.1 también se generaliza a sistemas de ecuaciones, básicamente se requiere extender el concepto de derivada de una función, al **jacobiano** de una función vectorial. Es decir, dado el sistema:

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad \text{para, } i = 1, 2, \ldots, n \tag{3.15}$$

calculamos el jacobiano de $\mathbf{J_f}(\mathbf{x})$ como:

$$\mathbf{J_f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix} \tag{3.16}$$

Podemos expresar ahora el método de Newton-Raphson para sistemas como la aplicación iterada de la ecuación vectorial:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J_f}(\mathbf{x_k})]^{-1}\mathbf{f}(\mathbf{x}_k) \tag{3.17}$$

Algoritmo 3.2 Método de Newton-Raphson para sistemas

Dado: $\mathbf{f}$: función asociada a la ecuación $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ a resolver, $\mathbf{x_0}$: aproximación inicial a la solución, ϵ: tolerancia del error, y m: número máximo de iteraciones

Entrega: $\mathbf{s}$: la solución aproximada a la ecuación $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, o bien el mensaje "sin solución aceptable en m iteraciones", o bien "$\mathbf{J_f}(\mathbf{x}) = \mathbf{0}$ en la iteración n"

```
NEWTON-RAPHSON(f, x_0, ε, m)
    n ← 0
    repeat
        if J_f(x_n) = 0 then return "J_f(x_n) = 0 en la iteración n"
        else
            x_{n+1} = x_n − [J_f(x_n)]^{-1} f(x_n)
            n ← n + 1
        end if
    until ||x_n − x_{n−1}|| ≤ ε ó n > m
    if ||x_n − x_{n−1}|| ≤ ε then
        return x_n
    else
        return "sin convergencia en m iteraciones"
    end if
end
```

3.4. Aplicaciones

Aplicación 3.1 (Manipulador robótico de dos eslabones)**.** Considere un manipulador robótico de dos eslabones con parámetros $L_1, L_2, \theta_1, \theta_2$ que determinan un punto (x, y) en el plano. Resolver los siguiente problemas:

1. (Cinemática directa) Dados $L_1, L_2, \theta_1, \theta_2$, encontrar (x, y)
2. (Cinemática inversa) Dados L_1, L_2, x, y encontrar θ_1, θ_2
3. (Planeación de la trayectoria) Dado un tiempo $t = T$, determinar la trayectoria de la "mano" del robot para moverse de (x_0, y_0) a (x_1, y_1)

Aplicación 3.2 (Modelo económico de Leontief)**.**

Aplicación 3.3 (Cadenas de Markov)**.**

Aplicación 3.4 (Sistemas polinomiales multivariables)**.**

Capítulo 4

Diferenciación e integración numérica

4.1. Diferenciación numérica

De nuestros cursos de *Cálculo*, recordamos el uso de límites para calcular la derivada de una función diferenciable:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Lo que nos sugiere que podemos aproximar $f'(x)$ con la ecuación:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{4.1}$$

utilizando valores pequeños de h. Para determinar el error de dicha aproximación, podemos auxiliarnos con el Teorema de Taylor (1.3.1). Si aproximamos $f(x+h)$ por su polinomio de Taylor de primer orden alrededor de x, tenemos:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2 \tag{4.2}$$

donde ξ es un valor entre x y $x+h$. Despejando $f'(x)$ obtenemos la aproximación dada por la fórmula 4.1, la cual sabemos ahora (por la ecuación anterior) que tiene un error del orden $O(h)$ ya que $\frac{f''(\xi)}{2}h^2 = O(h)$.

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \tag{4.3}$$

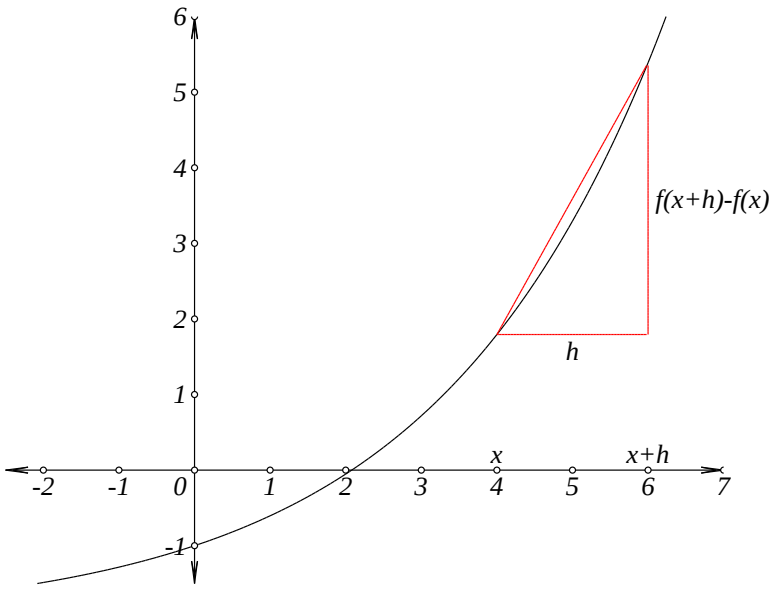


Figura 4.1: Estimación básica de una derivada

Ejemplo 4.1.1. La siguiente tabla (4.1) presenta la derivada de $cot(x)$ en $x = 2$, obtenida aplicando la ecuación 4.1 para diferentes valores de h. Además incluimos el error, ya que en este caso sabemos que $cot'(x) = \frac{-1}{cos^2(x)}$ y por tanto $cot'(2) = -1.2094504$ a siete decimales. Para fines comparativos, hemos agregado en las últimas dos columnas los valores correspon-dientes al utilizar la **fórmula de diferencias centradas** de orden $O(h^4)$ que presentaremos más adelante.

Tabla 4.1: Ejemplo comparativo para la derivada de $cot(x)$

h	$\approx cot'(2)$	error $O(h)$	$\approx cot'(2)$	error $O(h^4)$
0.1	-1.2719025	0.062452083929	-1.2092561	-0.000194298269
0.05	-1.238831	0.029380549229	-1.2094386	-1.1801389e-005
0.01	-1.2150517	0.005601277957	-1.2094504	-1.8713e-008
0.005	-1.2122345	0.002784043309	-1.2094504	-1.169e-009
0.001	-1.2100046	0.000554171084	-1.2094504	-2e-012

Ejemplo 4.1.2. Presentamos en la tabla (4.2) la derivada de e^x en $x = 1$, obtenida aplicando la ecuación 4.1 para diferentes valores de h. Como en el ejemplo anterior, incluimos el error, ya que en este caso sabemos que $(e^x)' = e^x$ y por tanto $(e^x)'(1) = 2.7182818$ a siete decimales. Para fines comparativos, hemos agregado en las últimas dos columnas los valores correspondientes al utilizar la **fórmula de diferencias centradas** de orden $O(h^4)$ como en el ejemplo previo.

Ejercicio 4.1.1. Calcular las primeras tres columnas de la tabla 4.1 para la derivada de la función $tan(x)$ evaluada en $x = \pi/5$.

Problema 4.1.1. Investigue la razón por la cual, en la última columna de la tabla 4.2, se incrementa el error conforme se reduce el valor de h.

Problema 4.1.2. Investigue la existencia de una fórmula para $f''(x)$ con un error del orden $O(h)$.

Con el propósito de encontrar mejores fórmulas para derivación de funciones, aplicaremos en la siguiente sección, el método de sustituir la función $f(x)$ por un polinomio interpolante. Para esto utilizaremos los **polinomios**

Tabla 4.2: Ejemplo comparativo para la derivada de $f(x) = e^x$

h	$\approx e'(2)$	error $O(h)$	$\approx e'(2)$	error $O(h^4)$
0.1	2.858842	-0.140560126415	2.7182728	9.071733e-006
0.01	2.7319187	-0.013636827328	2.7182818	9.06e-010
0.001	2.7196414	-0.001359594074	2.7182818	0.0
0.0001	2.7184177	-0.000135918624	2.7182818	-1e-012
1e-005	2.7182954	-1.3591498e-005	2.7182818	-2.2e-011
1e-006	2.7182832	-1.358972e-006	2.7182818	2e-010
1e-007	2.718282	-1.39947e-007	2.7182818	-1.169e-009

de Lagrange, ya que nos permiten también calcular la derivada en el caso de no tener una fórmula matemática, sino sólo un número finito de puntos (x_i, y_i). para $i = 1, 2, ..., n$.

4.1.1. Fórmulas de diferencia progresiva y regresiva

Básicamente, la idea para obtener estas fórmulas, es sustituir la función a derivar $f(x)$ por su polinomio interpolante de Lagrange $p_n(x)$ obtenido a partir de $(n + 1)$ puntos $x_i,\ n_0 \leq i \leq n - n_0$, para después aproximar $f'(x)$ con $p'_n(x)$. Considerando puntos equidistantes separados una distancia h, para expresar las fórmulas de $f'(x_0)$, utilizaremos la notación siguiente:

$$f_i = f(x_i),\ \text{donde } x_i = x_0 + ih,\ i = n_0, \ldots, n - n_0 \tag{4.4}$$

Tenemos entonces los siguientes tres casos de fórmulas de diferencias, en base a los valores de n_0:

1. Diferencias regresivas, cuando $n - n_0 = 0$
2. Diferencias progresivas, cuando $n_0 = 0$
3. Diferencias centradas, cuando $n_0 = -n/2$

4.1.2. Fórmula de tres puntos

Utilizando $n = 2$ y $n_0 = -1$ obtenemos la fórmula de diferencias centradas para $f'(x_0)$, que aparece en la tabla 4.3 (adaptada de [16])

Tabla 4.3: Fórmulas de diferencias centradas de orden $O(h^2)$ para derivadas

$$\begin{aligned} f'(x_0) &= \frac{f_1 - f_{-1}}{2h} \\ f''(x_0) &= \frac{f_1 - 2f_0 + f_{-1}}{h^2} \end{aligned} \tag{4.5}$$

Tabla 4.4: Fórmulas de diferencias centradas de orden $O(h^4)$ para derivadas

$$\begin{aligned} f'(x_0) &= \frac{-f_2 + 8f_1 - 8f_{-1} + f_{-2}}{12h} \\ f''(x_0) &= \frac{-f_2 + 16f_1 - 30f_0 + 16f_{-1} - f_{-2}}{12h^2} \end{aligned} \tag{4.6}$$

Problema 4.1.3. Verificar experimentalmente las siguientes fórmulas de orden $O(h^2)$ para la tercera y cuarta derivadas:

$$\begin{aligned} f^{(3)}(x_0) &= \frac{f_2 - f_1 + 2f_{-1} - f_{-2}}{2h^3} \\ f^{(4)}(x_0) &= \frac{f_2 - 4f_1 + 6f_0 - 4f_{-1} - f_{-2}}{h^4} \end{aligned}$$

y justificarlas paso a paso mediante la aplicación de los polinomios de Lagrange.

4.1.3. Fórmula de cinco puntos

Utilizando $n = 4$ y $n_0 = -2$ obtenemos la fórmula de diferencias centradas para $f'(x_0)$, que aparece en la tabla 4.4(adaptada de [16])

Una implementación en Scheme para el cálculo de $f'(x_0)$ anterior se presenta en el listado 4.1:

Código Scheme 4.1: Función para calcular $f'(x_0)$ según ecuación 4.4

```
;; derivada num'erica con h
;; [por diferencias centradas O(h^4)]
(define (deriva-con-h f x h)
  (/ (+ (- (f (+ x h h))) (* 8 (f (+ x h)))
        (* -8 (f (- x h))) (f (- x h h)))
     (* 12 h)))
```

Problema 4.1.4. Verificar experimentalmente las siguientes fórmulas de orden $O(h^4)$ para la tercera y cuarta derivadas:

$$f^{(3)}(x_0) = \frac{-f_3 + 8f_2 - 13f_1 + 13f_{-1} - 8f_{-2} + f_{-3}}{8h^3}$$
$$f^{(4)}(x_0) = \frac{-f_3 + 12f_2 - 39f_1 + 56f_0 - 39f_{-1} + 12f_{-2} - f_{-3}}{6h^4}$$

y justificarlas paso a paso mediante la aplicación de los polinomios de Lagrange.

4.2. Integración numérica

En las siguientes dos secciones veremos dos de las llamadas fórmulas de cuadratura de Newton-Cotes, las cuales aproximan una integral definida $\int_a^b f(x)dx$ subdividiendo el intervalo $[a, b]$ en subintervalos igualmente espaciados.

4.2.1. Método del trapecio

Dado el problema de aproximar la integral $\int_a^b f(x)dx$, el método del trapecio divide el intervalo $[a, b]$ en n subintervalos $[x_i, x_{i+1}]$, donde cada x_i está definida por:

$$x_i = a + ih, \text{para, } h = \frac{b-a}{n}, \ i = 0, 1, \ldots, n \tag{4.7}$$

y se integra cada subitervalo aproximando el área bajo la función por el área del trapecio cuya base es h y su altura promedio es $[f(x_i) + f(x_{i+1})]/2$. Por

lo tanto la aproximación de nuestra integral es:

$$\begin{aligned}\int f(x)dx &\approx \frac{1}{2}\sum_{i=1}^{n}[f(x_{i-1})+f(x_i)]h \\ &\approx \frac{h}{2}\sum_{i=1}^{n}[f(x_{i-1})+f(x_i)] \\ &\approx \frac{h}{2}[f(a)+f(b)]+h\sum_{i=1}^{n-1}f(x_i)\end{aligned} \tag{4.8}$$

de donde resulta nuestra fórmula principal de la regla compuesta del trapecio:

$$\int f(x)dx = \frac{h}{2}\left[f(a)+f(b)+2\sum_{i=1}^{n-1}f(x_i)\right]+R_n \tag{4.9}$$

El error R_n del método del trapecio, se deriva mediante la integración del polinomio interpolador lineal de Lagrange para $f(x)$ y asume la existencia de $f''(x)$:

$$R_n = -\frac{h^2(b-a)}{12}f''(c) \tag{4.10}$$

para alguna c entre a y b.

4.2.2. Métodos de Simpson

Existen varias reglas o métodos de Simpson para integración, a continuación presentamos la llamada regla compuesta de Simpson.

A diferencia del método trapezoidal, que utiliza un polinomio interpolador lineal, el método de Simpson sustituye $f(x)$ por un interpolador de Lagrange de segundo grado $p_2(x)$, logrando con ello una reducción en el error por truncamiento.

En este caso integramos cada tres puntos (i.e. cada dos subintervalos) obteniendo:

$$\int_{x_{i-1}}^{x_{i+1}} p_2(x)dx = \frac{h}{3}[f(x_{i-1})+4f(x_i)+f(x_{i+1})] \tag{4.11}$$

por lo que la fórmula resultante es:

$$\int_a^b f(x)dx = \frac{b-a}{3n}[f(x)+f(b)+4A+2B]+O(h^4) \tag{4.12}$$

donde

n es un número par

$h = (b - a)/n$

$A = \sum_{i=1}^{n/2} f[a + (2i - 1)h]$

$B = \sum_{i=1}^{(n/2)-1} f(a + 2ih)$

- $O(h^4)$ representa el orden del error

Presentamos en el listado 4.2 una implementación en Scheme de la regla compuesta de Simpson:

Código Scheme 4.2: Función para la regla compuesta de Simpson

```
; aproxima una integral definida
; utilizando la regla compuesta de Simpson
(define (simpson f a b h)
  (let* ((m (ceiling (/ (- b a)
                        (* 2 h))))
         (n (* 2 m)))
    (* (/ (- b a)
          (* 3 n))
       (+ (f a)
          (f b)
          (* 4 (sumatoria
                (lambda(i)
                  (f (+ a (* 2 i h)
                        (- h))))
                1 (/ n 2)))
          (* 2 (sumatoria
                (lambda(i)
                  (f (+ a (* 2 i h))))
                1 (- (/ n 2)
                     1)))))))
```

Ejemplo 4.2.1. La siguiente interacción presenta cómo calcular numéricamente la integral $\int_0^1 4sen(3x)dx$ para $h = 10^{-3}$.

```
> (simpson (lambda(x)
             (* 4 (sin (* 3 x))))
           0 1 0.001)
2.6533233288017897
```

4.2.3. Integración de Romberg

Los dos métodos de integración previos, requieren como parámetro, ya sea h o n para realizar los cálculos, en el método de Romberg, se utiliza ϵ (la tolerancia del error) y el procedimiento va subdividiendo el intervalo inicial $[a, b]$ recursivamente, en tantos subintervalos como sea necesario. Esto

se logra combinando el método del trapecio, con el método de extrapolación de Richardson.

Definimos primeramente la sucesión de particiones $P_1, P_2, \ldots, P_k$ en base a las siguientes ecuaciones recursivas:

$$\begin{aligned} & n_1 = 1 \\ & n_k = 2n_{k-1}, \quad k \geq 2 \\ & h_1 = b - a \\ & h_k = \frac{h_{k-1}}{2}, \quad k \geq 2 \\ & P_k : x_i = a + (i-1)h_k, \quad 0 \leq i \leq n_k \end{aligned}$$

La aplicación del método del trapecio a la sucesión de particiones, nos lleva a las siguientes ecuaciones de Romberg:

$$\begin{aligned} R_{1,1} &= \frac{h}{2}[f(a) + f(b)] \\ R_{k,1} &= \frac{1}{2}\left\{ R_{k-1,1} + h_{k-1} \sum_{i=1}^{n_{k-1}} f\left[a + (i - \frac{1}{2})h_{k-1}\right] \right\} \end{aligned} \tag{4.13}$$

Aplicando ahora el método de extrapolación de Richardson, obtenemos:

$$\int_a^b f(x)dx = \frac{4R_{k,1} - R_{k-1,1}}{3} + O(h_k^4), \quad k \geq 2 \tag{4.14}$$

Para acelerar la convergencia asociada a la ecuación 4.14, aplicamos la extrapolación de Richardson a la sucesión $\{R_{k,1}\}$, para obtener:

$$R_{k,i} = \frac{4^{i-1}R_{k,i-1} - R_{k-1,i-1}}{4^{i-1} - 1}, \quad k \geq 2, \quad 2 \geq i \geq k \tag{4.15}$$

Con lo cual podemos expresar el método de Romberg, como:

$$\int_a^b f(x)dx = R_{k,i} + O(h_k^{2i}), \quad k \geq 2, \quad 2 \geq i \geq k \tag{4.16}$$

Una manera de definir el algoritmo de Romberg, es utilizar la sucesión de valores $R_{i,i}$ generados por la ecuación 4.16 y detener el proceso en la iteración i que cumpla $|R_{i,i} - R_{i-1,i-1}| < \epsilon$ para un ϵ dado.

Código Scheme 4.3: Función para el método de Romberg

```
(define (romberg f a b eps)
  (define (hk k) ; h del k-esimo subintervalo
    (/ (- b a) (expt 2 (- k 1))))
  (define (romb1 k) ; caso R(k,1)
    (if (= k 1)
        (* (/ (hk 1) 2)
           (+ (f a) (f b)))
        (* 1/2 (+ (romb1 (- k 1))
                  (* (hk (- k 1))
                     (sumatoria
                      (lambda(i)
                        (f (+ a
                              (* (- (* 2 i) 1)
                                 (hk k)))))
                      1 (expt 2 (- k 2))))))))
  (define (romb i j) ; caso R(i,j)
    (if (= j 1)
        (romb1 i)
        (+ (romb i (- j 1))
           (/ (- (romb i (- j 1))
                 (romb (- i 1) (- j 1)))
              (- (expt 4 (- j 1)) 1)))))
  ; ciclo principal
  (do ((k 1 (+ k 1))
       (R (romb1 1) (romb (+ k 1) (+ k 1)))
       (Rs (romb 2 2) (romb (+ k 2) (+ k 2))))
    ((< (abs (- Rs R)) eps) Rs)
    (display `(r(,k : ,k) ,(romb k k))) ; opcional
    (newline)                           ; opcional
    ))
```

Ejemplo 4.2.2. Evaluemos la siguiente integral que aproxima la constante de Euler:

$$\gamma = \int_0^1 \left(\frac{1}{1-t} + \frac{1}{\ln t} \right) dt$$

Ya que en $t = 0$ y en $t = 1$ la función a integrar está indeterminada, utilizaremos para la aproximación los límites $a = 10^{-6}$ y $b = 1 - 10^{-6}$. Consideremos

una tolerancia de 10^{-4}. La siguiente interacción en Scheme nos muestra la solución por el método de Romberg.

```
> (romberg (lambda(t)
                 (+ (/ 1 (- 1 t)) (/ 1 (log t))))
               10e-6 (- 1 10e-6) 0.0001)
(r (1 : 1) 0.7065618370052935)
(r (2 : 2) 0.6070498210096675)
(r (3 : 3) 0.5880919525304922)
(r (4 : 4) 0.5814810778664038)
(r (5 : 5) 0.5789083665486172)
(r (6 : 6) 0.5778819093890034)
(r (7 : 7) 0.5774707636230789)
0.5772417545073438
>
```

El valor de esta famosa constante a 10 decimales es 0.5772156649

Ejercicio 4.2.1. Evalúe por el método de Romberg la siguiente integral considerando $\epsilon = 10^{-6}$

$$\int_0^1 \frac{\ln(1+x)}{1+x^2} dx$$

Calcule el error relativo de su solución, sabiendo que el valor exacto es $\frac{\pi \ln 2}{8}$

4.2.4. Método de cuadratura gaussiana

Antes de resolver la integración de el intervalo general $[a, b]$, resolveremos el problema más sencillo:

$$\int_{-1}^{1} f(x)dx \tag{4.17}$$

esto es necesario, debido a que utilizaremos funciones ortogonales en el intervalo $[-1, 1]$, en particular utilizaremos los polinomios de Legendre de grado n, que denotaremos $P_n(x)$ para aproximar la integral buscada, es decir:

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} P_n(x)dx \tag{4.18}$$

En lugar de utilizar puntos equidistantes para particionar el intervalo de integración $[-1, 1]$ (como fue el caso en los métodos anteriores) ahora

consideramos los nodos x_i y los pesos w_i como parámetros para optimizar la aproximación, es decir:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i), \quad -1 \leq x_i \leq 1, \quad 1 \leq i \leq n. \tag{4.19}$$

Los valores de los nodos x_i corresponden a los ceros de los polinomios de Legendre $P_n(x)$ y los pesos w_i se pueden calcular con la ecuación:

$$w_i = \int_{-1}^{1} \prod_{\substack{i=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} dx \tag{4.20}$$

Ya que los valores de x_i y w_i normalmente se utilizan para n relativamente pequeña, estos se presentan en forma tabular, como en la tabla 4.5 (ver por ejemplo [8] p. 270):

Tabla 4.5: Nodos y pesos para cuadratura gaussiana

n		x_i	w_i
1		0	2
2	$\pm$	0.5773502692	1
3		0.7745966692	0.5555555556
		0	0.8888888889
4	$\pm$	0.8611363116	0.3478548451
	$\pm$	0.3399810436	0.6521451549
5	$\pm$	0.9061798459	0.2369268851
	$\pm$	0.5384693101	0.4786286705
		0	0.5688888889

Para poder resolver el problema en el intervalo general $[a, b]$, es necesario ajustar los valores de x_i y w_i a x_i' y w_i' respectivamente, mediante un cambio de variable, es decir

$$\int_{a}^{b} f(x)dx \approx \sum_{i=1}^{n} w_i' f(x_i'), \quad a \leq x_i \leq b, \quad 1 \leq i \leq n. \tag{4.21}$$

donde

$$x_i' = \frac{a+b}{2} + \frac{b-a}{2} x_i, \quad w_i' = \frac{b-a}{2} w_i \tag{4.22}$$

de donde obtenemos nuestra fórmula principal para el método de cuadratura gaussiana:[1]

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f(\frac{a+b}{2} + \frac{b-a}{2}x_i) \tag{4.23}$$

para $a \leq x_i \leq b, \quad 1 \leq i \leq n$

Proyecto 4.1. Construya un programa en Scheme o en algún otro lenguaje, para obtener los datos de la tabla 4.5 correspondiente a un entero $n > 1$

4.3. Integración múltiple

Para evaluar integrales múltiples en regiones rectangulares, podemos aplicar los métodos de las secciones previas de una manera similar a la composición de funciones. Por ejemplo para el caso bidimensional tenemos:

$$\iint_R f(x,y)dA = \int_a^b \left(\int_c^d f(x,y)dy \right) dx \tag{4.24}$$

donde $R = \{(x,y) | a \leq x \leq b, \ c \leq y \leq d\}$, para ciertas valores $a, b, c, d \in \mathbf{R}$

Para este caso, un método de solución es aplicar primero la regla compuesta de Simpson (ecuación 4.12) para aproximar

$$\int_c^d f(x,y)dy \tag{4.25}$$

donde asumimos x_i como constante.

Subdividiendo $[c,d]$ en un número par m de subintervalos, con $k = (d-c)/m$, entonces $y_j = c + jk$ para $0 \leq j \leq m$, y podemos escribir:

$$\int_c^d f(x,y)dy \approx \frac{k}{3}\left[f(x,y_0) + 4A_x + 2B_x + f(x,y_m)\right] \tag{4.26}$$

donde

$A_x = \sum_{j=1}^{m/2} f(x, y_{2j-1})$

[1]Este método se conoce también como *método de Gauss-Legendre*, el cual no debe confundirse con el llamado *algoritmo de Gauss-Legendre*, utilizado para aproximar el valor de π.

$B_x = \sum_{j=1}^{(m/2)-1} f(x, y_{2j})$

Podemos ahora escribir nuestra integral doble como:

$$\int_a^b \int_c^d f(x,y)dydx \approx \frac{k}{3}\Big[\int_a^b f(x,y_0)dx + 4\int_a^b A_x dx + 2\int_a^b B_x dx + \int_a^b f(x,y_m)dx\Big] \tag{4.27}$$

Ahora aplicamos de nuevo la regla compuesta de Simpson, a cada una de las integrales del lado derecho, donde cada una de ellas tiene la forma siguiente y se aplican para y_j con $0 \leq j \leq m$

$$\int_a^b f(x,y_j)dx \approx \frac{h}{3}\left[f(x_0,y_j) + 4A + 2B + f(x_n,y_j)\right] \tag{4.28}$$

donde

$A = \sum_{i=1}^{n/2} f)(x_{2i-1}, y_j)$

$B = \sum_{i=1}^{(n/2)-1} f(x_{2i}, y_j)$

donde $x_i = a + ih$ para $0 \leq i \leq n$.

4.4. Aplicaciones

Aplicación 4.1 (Generación de variables aleatorias)**.**

Aplicación 4.2 (Generación de tablas de distribuciones de probabilidad)**.**

Aplicación 4.3 (Cinemática de un robot de dos eslabones)**.** Dada una función polinomial que describe la trayectoria de la "mano" de un robot de dos eslabones, calcular su velocidad y aceleración (continuación de la aplicación 3.1).

Aplicación 4.4 (Centros de gravedad)**.**

Capítulo 5

Solución de ecuaciones diferenciales

[Un teorema fundamental del análisis numérico:] consistencia + estabilidad = convergencia

Germund Dahlquist[28]

5.1. Métodos de un paso

Definición 5.1.1 (Método de un paso)**.** Llamamos método de un paso para resolver el problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0, \quad x_0 \leq x \leq b \tag{5.1}$$

a un procedimiento de integración dado por

$$y_{n+1} = ay_n + h[bf(x_n, y_n)] \tag{5.2}$$

donde $h = (b - x_0)/N$, y $n = 0, 1, \ldots, N - 1$

5.1.1. Método de Euler y formas mejorada

El método de Euler utiliza $a = 1, b = 1$ en la ecuación 5.2, por lo cual emplea:

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{5.3}$$

Este se obtiene de considerar una aproximación lineal a la para la solución exacta $Y(x)$ del problema de valor inicial (ecuación 5.1. Es decir, consideramos que $Y(x_0) = y_0$ y:

$$Y'(x) = f[x, Y(x)], \quad x_0 \leq x \leq b \tag{5.4}$$

La aproximación lineal de $Y'(x)$ se representa como:

$$Y'(x) \approx \frac{Y(x+h) - Y(x)}{h} \tag{5.5}$$

de donde, despejando $Y(x+h)$ obtenemos:

$$\begin{aligned} Y(x+h) &\approx Y(x) + hY'(x) \\ &\approx Y(x) + hf[x, y(x)] \end{aligned}$$

Denotando $Y(x+h)$ como y_{n+1} y $f[x, y(x)]$ por $f(x_n, y_n)$ obtenemos la fórmula del método de Euler (ecuación 5.3)

El método de Euler es uno de los más básicos, por lo que es sólo un punto de partida en nuestro estudio. Veamos dos métodos[1] que de manera directa mejoran el método de Euler:

- Método de Heun. Mejora el cálculo de la pendiente, utilizando dos puntos (inicial y final de cada intervalo) para luego obtener un promedio, es decir:[2]

$$\begin{aligned} y^0_{i+1} &= y_i + hf(x_i, y_i) \\ y_{i+1} &= y_i + h\frac{f(x_i, y_i) + f(x_{i+1}, y^0_{i+1})}{2} \end{aligned} \tag{5.6}$$

- Método del punto medio. En este caso se mejora la estimación de la pendiente, utilizando el punto medio entre x_i y x_{i+1}. Por tanto, la fórmula de iteración puede escribirse como:

$$y_{i+1} = y_i + hf(x_{i+1/2}, y_{i+1/2}) \tag{5.7}$$

[1] De hecho, estos métodos se consideran también entre los métodos de Runge-Kutta, que veremos en la sección 5.1.2 (ver [5] pp. 272-276)

[2] Existen varias formas de representar estas ecuaciones.

Código Scheme 5.1: Función **euler** para aproximar la solución a $y' = f(x, y)$

```
;; m'etodo de Euler para y'=f(x,y) con y(a)=ya
(define (euler f a b ya n)
  (let ((h (/ (- b a) n)))
    (do ((k 0 (+ k 1))
         (x a (+ x h))
         (y ya
            (+ y (* h (f x y)))) ; un paso
         (data '()
               (cons (list k (ft7 x) (ft7 y))
                     data)))
      ((> k n) (reverse data)))))
```

Ejemplo 5.1.1. Apliquemos el método de Euler para determinar la solución numérica de la ecuación

$$y' = y^2 x e^{x-y} \tag{5.8}$$

para $0 \leq x \leq 1$ y condición inicial $y(0) = 1$. Utilice $n = 10$. Las interacciones con la función euler son las siguientes:

```
>(euler (lambda(x y)
                (* (expt y 2) x (exp (- x y))))
              0 1. 1 10)
((0 0 1)
 (1 0.1 1)
 (2 0.2 1.0040657)
 (3 0.3 1.0130887)
 (4 0.4 1.02818)
 (5 0.5 1.0507423)
 (6 0.6 1.082568)
 (7 0.7 1.1259675)
 (8 0.8 1.183931)
 (9 0.9 1.2603152)
 (10 1.0 1.3600205))
 >
```

5.1.2. Métodos de Runge-Kutta

Los métodos de Runge-Kutta extienden la fórmula general de los métodos de un paso (ecuación 5.2), utilizando el siguiente esquema (ver por ejemplo [6] pp. 734-735)

$$y_{i+1} = y_i + h\phi(x_i, y_i, h) \tag{5.9}$$

donde $\phi(x_i, y_i, h)$ se denomina función incremento y representa una aproximación de la pendiente del intervalo $[x_i, x_{i+1}]$. Dicha aproximación se calcula como una combinación lineal de las pendientes en puntos específicos del intervalo.

Método de Runge-Kutta de cuarto orden

Este es uno de los métodos más utilizados, ya que resuelve el problema de valor inicial con un error del orden $O(h^4)$

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5.10}$$

donde

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1)$$

$$k_3 = f(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_2)$$

$$k_4 = f(x_i + h, y_i + hk_3)$$

Podemos implementar el método de Runge-Kutta de orden 4, mediante el código indicado en 5.2:

Código Scheme 5.2: Función **runge-kutta-4** para aproximar la solución a $y' = f(x, y)$

```
;; M'etodo de Runge-Kutta de orden 4 para y'=f(x,y)
(define (runge-kutta-4 f x0 xn y0 h)
  (let* ((m (/ h 2))
         (paso-rk4
          (lambda(x y)
            (let* ((k1 (f x y))
                   (k2 (f (+ x m) (+ y (* m k1))))
                   (k3 (f (+ x m) (+ y (* m k2))))
                   (k4 (f (+ x h) (+ y (* h k3)))))
              (+ y (* (/ h 6)
                      (+ k1 (* 2 k2) (* 2 k3) k4)))))))
    (let ciclo ((x x0) (y y0) (data (list (list x0 y0))))
      (cond ((>= (+ x h) xn) (reverse data))
            (else
             (ciclo (+ x h)
                    (paso-rk4 x y)
                    (cons (list (ft7 (+ x h))
                                (ft7 (paso-rk4 x y)))
                          data)))))))
```

Problema 5.1.1. Aplique el método de Euler y el método de Runge-Kutta para resolver la ecuación $y' = 2x + cos(y)$ con $y(0) = 0$. Experimente y compare con diferentes parámetros y grafique sus resultados de manera similar a la figura 5.1.

5.2. Métodos de pasos múltiples

Definición 5.2.1 (Método de pasos múltiples, adaptado de [8])**.** Llamamos método de pasos múltiples o método multipaso para resolver el problema de valor inicial (ver ecuación 5.1) a un procedimiento de integración dado por

$$y_{n+1} = hcf(x_{n+1}, y_{n+1}) + \sum_{i=0}^{m-1} [a_i y_{n-i} + hb_i f(x_{n-i}, y_{n-i})] \tag{5.11}$$

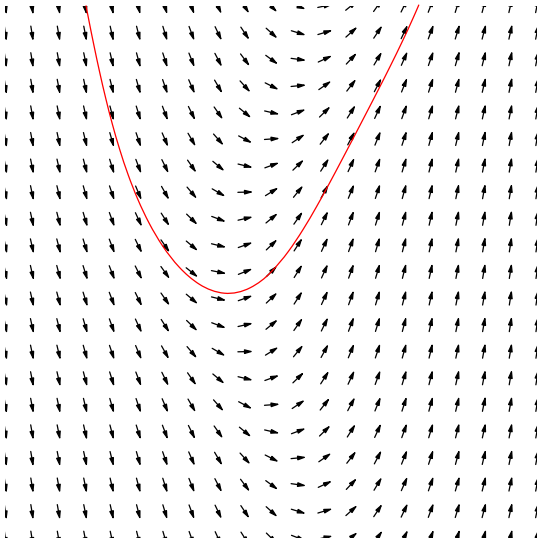


Figura 5.1: Campo direccional para $y' = 2x + cos(y)$ y trayectoria para $y(0) = 0$, en $[-3, -3] \times [3, 3]$

donde $m > 1$, $h = (b - x_0)$ y $n = m - 1, m, \ldots, N - 1$. Es un método de m pasos siempre que $a_{m-1}^2 + b_{m-1}^2 > 0$. Si $c = 0$ el método es explícito o abierto, en caso contrario, el método es implícito o cerrado.

5.2.1. Método de Adams-Bashforth

Este es un método abierto ya que y_{n+1} se determina explícitamente.
El problema de valor inicial a resolver es:

$$y' = f(x, y), \text{ para } y(x_0) = y_0, \ x_0 \leq x < b \tag{5.12}$$

Considerando que estamos en la iteración n y que hemos obtenido la solución exacta $Y_n = y(x_n)$, podemos integrar la ecuación anterior desde x_n a x_{n+1}:

$$\int_{x_n}^{x_{n+1}} y' dx = \int_{x_n}^{x_{n+1}} f[x, Y(x)]dx \tag{5.13}$$

de donde encontramos:

$$Y_{n+1} = Y_n + \int_{x_n}^{x_{n+1}} f[x, Y(x)]dx \tag{5.14}$$

Para estimar la integral de la ecuación anterior, utilizaremos un polinomio interpolador de Lagrange $p_n(x)$ determinado por los puntos

$$(x_0, F_0), \ldots, (x_n, F_n)$$

donde $F_i = F_(x_i)$, $0 \leq i \leq n$, cumple con $F_i = Y'(x_i) = f[x_i, Y(x_i)]$.

Desarrollando las diferencias divididas en términos de F_n obtenemos la fórmula de diferencias regresivas de Newton siguiente:

$$p_n(x) = \sum_{k=0}^{n} (-1)^k \binom{-s}{k} (\nabla^k F_n) \tag{5.15}$$

La fórmula general para el método de Adams-Bashforth de orden m:

$$y_{n+1} = y_n + h\left[y'_n + \frac{1}{2}(\nabla y'_n) + \frac{5}{12}(\nabla^2 y'_n) + \cdots + c_{m-1}(\nabla^{m-1} y'_n)\right] \tag{5.16}$$

donde $c_{m-1} = (-1)^{m-1} \int_1^0 \binom{-s}{m-1} ds$

Proyecto 5.1. Construir un programa de computadora para calcular el valor de la integral $\int_1^0 \binom{-s}{k} ds$, y aplíquelo para determinar las fórmulas para el método de Adams-Bashforth de orden 6 y 7.

Dos dos métodos de Adams-Bashforth más utilizados, son los de orden 3 y 4, los cuales corresponden a las fórmulas siguientes:

- Método Adams-Bashforth de orden 3:

$$y_{n+1} = y_n + \frac{h}{12}[23f(x_n, y_n) - 16f(x_{n-1}, y_{n-1}) + 5f(x_{n-2}, y_{n-2})] \tag{5.17}$$

  con error por truncamiento del orden $O(h^4)$

- Método Adams-Bashforth de orden 4:

$$\begin{aligned} y_{n+1} = y_n + \frac{h}{24}[&55f(x_n, y_n) - 59f(x_{n-1}, y_{n-1}) \\ &+ 37f(x_{n-2}, y_{n-2}) - 9f(x_{n-3}, y_{n-3})] \end{aligned} \tag{5.18}$$

  con error por truncamiento del orden $O(h^5)$

5.2.2. Método de Adams-Moulton

Este es un método cerrado, ya que y_{n+1} se determina implícitamente. La técnica para obtener las fórmulas es similar al caso de Adams-Bashforth.

Presentamos las fórmulas asociadas a los métodos de Adams-Moulton orden 3 y 4:

- Método Adams-Moulton de orden 3:

$$\begin{aligned} y_{n+1} = y_n + \frac{h}{24}[9f(x_{n+1}, y_{n+1}) + 19f(x_n, y_n) \\ - 5f(x_{n-1}, y_{n-1}) + 2f(x_{n-2}, y_{n-2})] \end{aligned} \tag{5.19}$$

  con error por truncamiento del orden $O(h^5)$

- Método Adams-Moulton de orden 4:

$$\begin{aligned} y_{n+1} = y_n + \frac{h}{720}[251f(x_{n+1}, y_{n+1}) + 646f(x_n, y_n) \\ - 264f(x_{n-1}, y_{n-1}) + 106f(x_{n-2}, y_{n-2}) \\ - 19f(x_{n-3}, y_{n-3})] \end{aligned} \tag{5.20}$$

  con error por truncamiento del orden $O(h^6)$

5.3. Sistemas de ecuaciones diferenciales ordinarias

Los métodos descritos en las secciones previas para resolver el problema de valor inicial (ecuación 5.1) se generalizan de forma directa para resolver el problema de valor inicial generalizado, para un sistema de ecuaciones diferenciales de primer orden.

Definición 5.3.1 (Problema de valor inicial generalizado)**.**

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \quad y_i(x_0) = y_{i,0}, \quad x_0 \leq x \leq b \tag{5.21}$$

Para demostrar la existencia y unicidad de la solución al problema anterior, es necesario que las funciones f_i cumplan con la condición de Lipschitz para funciones de varias variables (ver [5] pp. 313-314)

Consideremos el caso de un sistema de dos ecuaciones, a resolver por el método de Euler:

$$\begin{aligned} y_{1,n+1} &= y_{1,n} + hf_1(x_n, y_{1,n}, y_{2,n}), \quad n \geq 0 \\ y_{2,n+1} &= y_{2,n} + hf_2(x_n, y_{2,n}, y_{2,n}), \quad n \geq 0 \end{aligned} \tag{5.22}$$

El proceso de realiza de forma simultánea, con las condiciones iniciales $y_1(x_0) = y_{1,0}$ y $y_2(x_0) = y_{2,0}$.

De manera similar podemos desarrollar las ecuaciones para el caso del método de Runge-Kutta de orden 4.

Proyecto 5.2 (Convergencia vs errores)**.** Considere las diferencias en énfasis entre los órdenes en la velocidad de convergencia de los métodos, y el orden de los errores por truncamiento, ambos con respecto al valor de h. *Sugerencia*: Consulte el excelente ensayo de Lloyd N. Trefethen [26, p. 11].

5.4. Aplicaciones

Aplicación 5.1 (Redes de circuitos eléctricos)**.**

Aplicación 5.2 (Simulación de un péndulo)**.**

Aplicación 5.3 (Ecuaciones de Lorenz: un modelo de convección)**.**

Bibliografía

[1] H. Abelson, G. J. Sussman, con J. Sussman. *Structure and Interpretation of Computer Programs, 2/e*. MIT Press, 1996. Disponible electrónicamente en http://mitpress.mit.edu/sicp/

[2] X. Aberth. *Introduction to Precise Numerical Methods*. Academic Press, 2008

[3] S. Banach. *Cálculo diferencial e integral*. Colección de Textos Politécnicos. Serie Matemáticas. Editorial Limusa, 1996 (original en polaco, 1929)

[4] L. Blum, F. Cucker, M. Shub, S. Smale. *Complexity and Real Computation*. Springer-Verlag, New York, 1998

[5] R. L. Burden, J. D. Faires. *Análisis numérico*. 7ma. edición. Thomson Learning. 2002

[6] S. C. Chapra, R. P. Canale. *Métodos numéricos para ingenieros: con programas de aplicación, 4/e* McGraw-Hill Interamericana, 2003

[7] B. P. Demidovich, I. A. Maron. *Computational Mathematics*. MIR Publishers, Moscú, 1973

[8] M. Friedman, A. Kandel. *Fundamentals of Computer Numerical Analysis*. CRC Press, Inc. 1994

[9] N. Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, 1999

[10] D. Goldberg. *What every computer scientist should know about floating point arithmetic*. Computing Reviews, ACM, march 1991

[11] R. L. Graham, D. E. Knuth, O. Patashnik. *Concrete Mathematics: A foundation for computer science, 2/e*. Addison-Wesley Publ. Co., 1994

[12] R. W. Hamming. *Numerical Methods for Scientists and Engineers, 2/e*. Dover, 1986

[13] IEEE Computer Society. *IEEE Standard for Binary Floating Point Arithmetic*. IEEE Std 754-1985

[14] James y James. *Mathematics Dictionary, 5/e*. Van Nostrand Reinhold, N.Y. 1992

[15] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 2/e*. Addison-Wesley Publ. Co., 1981

[16] John H. Mathews, Kurtis D. Fink. *Métodos Numéricos con MATLAB, 3/e*. Prentice-Hall, 2000

[17] C. Moler. *Floating points:IEEE Standard unifies arithmetic model*. Cleve's Corner. Disponible en http://www.mathworks.com/company/newsletters/news_notes/pdf/Fall96Cleve.pdf

[18] C. Moler. *Numerical Computing with MATLAB*. Disponible en: http://www.mathworks.com/moler/index.html

[19] Numerical Analysis Digest http://www.netlib.org/na-net

[20] Numerical Mathematics Consortium http://www.nmconsortium.org

[21] PLaneT Package Repository. Disponible en: planet.plt-scheme.org.

[22] A. Ralston, P. Rabinowitz. *A First Course in Numerical Mathematics, 2/e*. Dover, 2001

[23] A. Stanoyevitch. *Introduction to MATLAB with Numerical Preliminaries*. John Wiley & Sons, Inc., 2005

[24] Sun Microsystems *Numerical computation guide*. Disponible en: http://docs.sun.com/app/docs/doc/802-5692/. Consultado 2008.02.22

[25] D. Sitaram. *Teach yourself Scheme in fixnum days*. 1998-2001. Documento disponible en:
http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme.html

[26] Lloyd N. Trefethen. *Numerical Analysis*. A ser incluido en: W. T. Gowers, ed., Princeton Companion to Mathematics, Princeton U. Press. 2008. Disponible en: http://www.comlab.ox.ac.uk/people/nick.trefethen/publication/PDF/2006_117.pdf. Consultado 2008.08.07

[27] W. Tucker. *Auto-validating numerical methods*. Curso de verano, 2004. Disponible en: http://www.math.uu.se/∼warwick/summer04/info.html

[28] Gerhard Wanner *Germund Dahlquist's Classical Papers on Stability Theory*. Disponible en: http://www.unige.ch/∼wanner/DQsem.pdf

Índice de Algoritmos

Índice de código Scheme

Índice de figuras

Índice de tablas

Índice alfabético