Manual de prácticas para el curso de

Química Computacional

Andrés Cedillo

Departamento de Química
Universidad Autónoma Metropolitana
Iztapalapa

Otoño 2004

Número de registro 03-2004-090709521900-01
Universidad Autónoma Metropolitana
www.fqt.izt.uam.mx

Química Computacional (214141)

Contenido

Bibliografía de consulta

- MG Sobell
  Unix System V: A PracticalGuide
  Addison, 3rd ed. (1995)
- ES Roberts
  The Art and Science of C
  Addison (1995)
- CF Gerald and PO Wheatley
  Applied Numerical Analysis
  Addison, 6th ed. (1999)
- KB Rojiani
  Programming in C with Numerical Methods for Engineers
  Prentice (1996)
- JM Ortega and AS Grimshaw
  An Introduction to C++ and Numerical Methods
  Addison, 6th ed. (1999)

Sesión #1

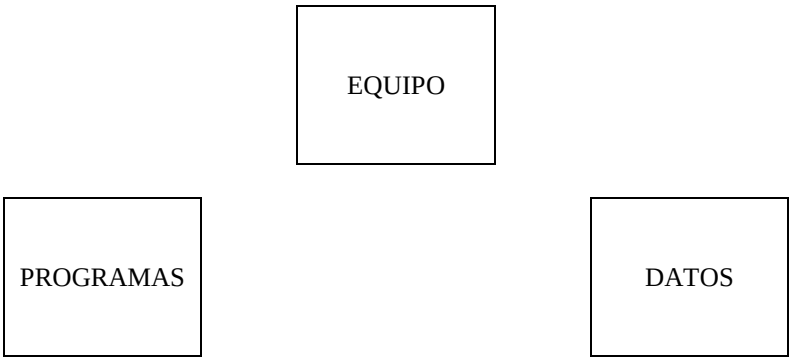
Estructura del curso

Objetivos:

Que el alumno:

- Conozca los fundamentos y el funcionamiento básico del sistema operativo UNIX
- Aprenda a utilizar las estructuras básicas del lenguaje de programación C
- Revise y comprenda el funcionamiento algunos métodos numéricos
- Aplique los conceptos aprendidos para resolver algunos problemas de la química

Elementos básicos del cómputo



Equipo

Procesador central (CPU)

- Control de la computadora
- Proceso de la información

Memoria (RAM)

- Almacena la información que requiere el CPU
  - Programas
  - Datos
- No es permanente

Almacenamiento secundario

- Permanente
  - Discos, cintas magnéticas, etc.

Dispositivos de entrada/salida

- Interacción externa
  - Teclado, monitor, ratón, etc.
  - Dispositivos de comunicación: modem, dispositivo de red, interface para adquisión de datos, etc.

Programa

- Conjunto de instrucciones que realizan un fin

Proceso de la programación

- Definición del problema
- Construcción de un algoritmo
- Escritura de un programa
- Proceso de la información
- Análisis de los resultados

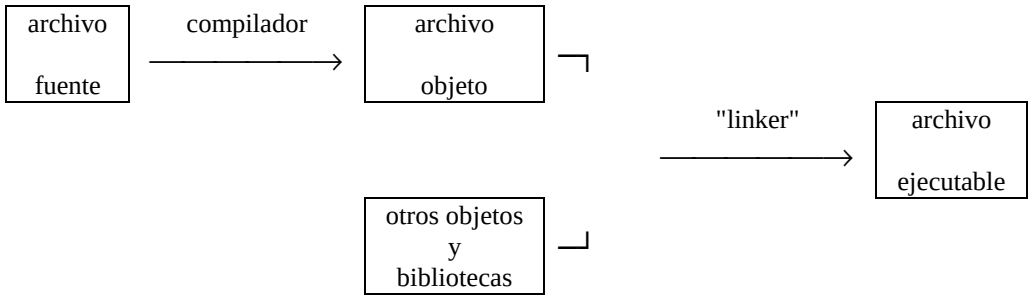
Algoritmo

- Estrategia para resolver un programa
  - Definición clara
  - Proceso realizable
  - Extensión o duración definida (finita)

Compilador

- Programa que transforma un archivo escrito en un lenguaje de programación en otro que puede enteder el CPU (binario)

Proceso de compilación

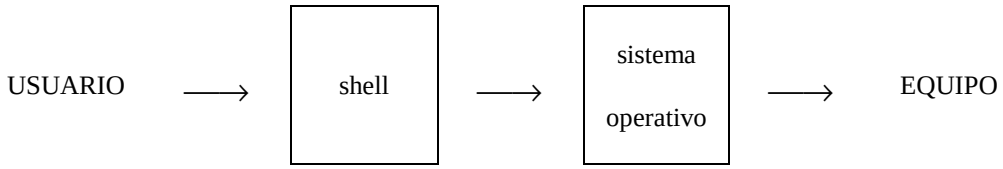


- El compilador detecta errores de sintaxis

Lenguajes de alto nivel

- Lenguaje de programación que pueden entender las personas
  - BASIC
  - FORTRAN
  - C
  - C++
  - Pascal
  - etc.
- En este curso se utilizará el lenguaje C ya que es muy transportable y tiene aplicación en muchos campos
- El archivo fuente está escrito en un lenguaje de programación
  - Estilos de programación
  - Comentarios (documentar)
  - Mantenimiento de "software"
  - Ingeniería de software
    - disciplina de escribir programas que puedan ser entendidos y mantenidos por otros

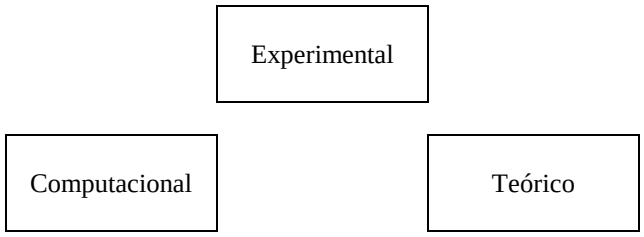
Interacción equipo-usuario



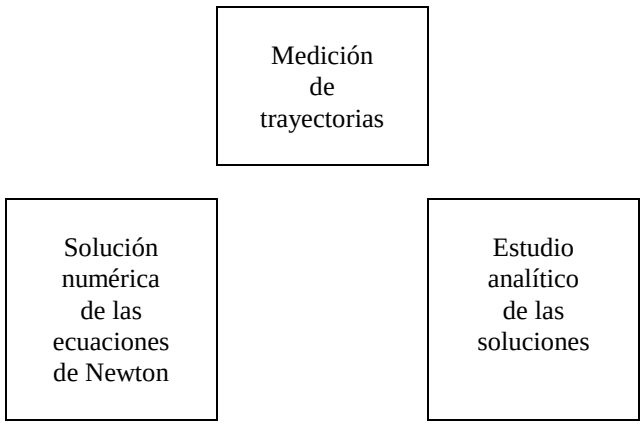
Cómputo científico

JL Zachary
Introduction to scientific programming
Computational problem solving using Mathematica and C
Springer, 1998

Tipos de estudio científico



Ejemplo: Mecánica



Método de solución de un problema computacional

Definir el problema

Expresarlo matemáticamente (modelo)

Construir un algoritmo

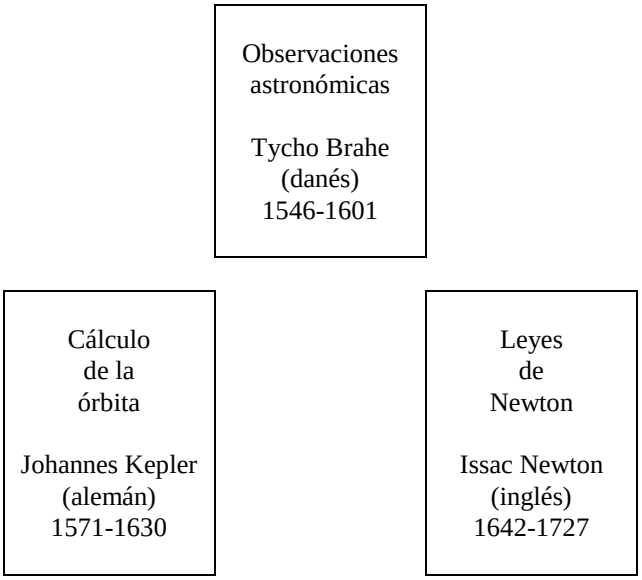
Implementación en la computadora

Análisis de los resultados

- análisis crítico
- confrontación con el problema
- validación de los resultados

Sesión #2

Ejemplo: determinación de la órbita de Marte por Kepler



Definición del problema

- Calcular la órbita de Marte a partir de las observaciones astronómicas de Tycho Brahe

Descripción del modelo

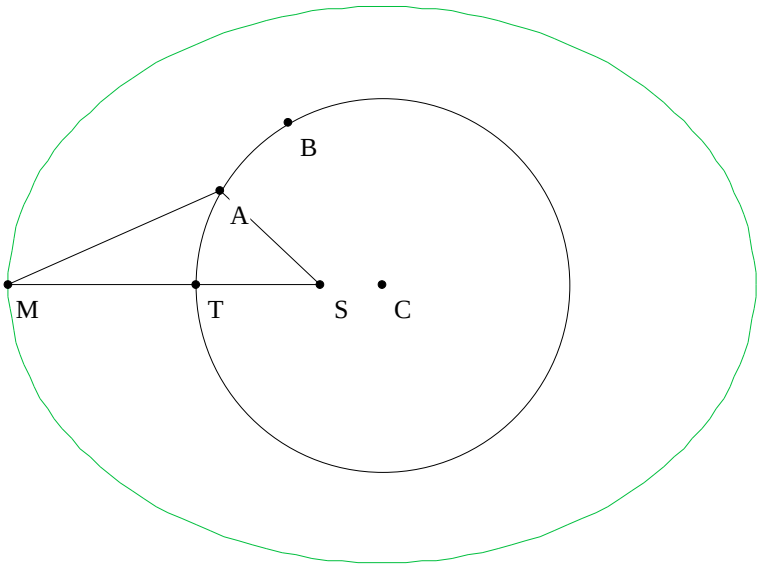


Figura 1. Posición de la Tierra en intervalos de un año marciano.

Datos

- año marciano: 687 días = $(2 \times 365 - 43)$ días
- observaciones astronómicas de Tycho

Hipótesis

- la órbita terrestre es circular
- el Sol (S) no necesariamente se encuentra en el centro del círculo (C)

Modelo

- analizando las posiciones relativas de la Tierra (T, A, B, ...), Marte (M) y el Sol (S), en intervalos de un año marciano, es posible determinar la posición del Sol, respecto al centro, y la órbita de Marte.

Construcción del algoritmo

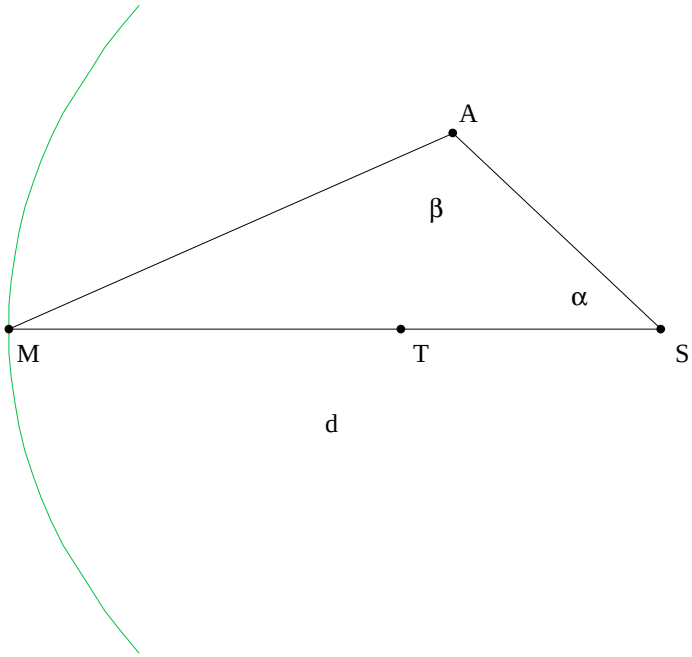


Figura 2. Diagrama para la determinacion de las coordenadas del punto A.

- Se toma la distancia d=MS como unitaria
- Con los ángulos α y β se determina la posición de la Tierra (A)
- Se repite la determinación para otras observaciones, (triángulo MSB, etc.)
- Con las coordenadas de, al menos, tres puntos es posible encontrar el centro de la órbita terrestre
- Las coordenadas de Marte en diferentes fechas permiten determinar su trayectoria
- La forma de la órbita se determina haciendo un ajuste
- La trayectoria permite conocer la velocidad de Marte

Implementación

- Kepler sólo tuvo a la mano tablas trigonométricas y de logarítmos
- Los cálculos los realizó a mano
- Kepler esperaba terminar este trabajo en unos cuantos meses, sin embargo tardó 5 años, en los cuales acumuló 900 páginas de cálculos
- En la actualidad, el trabajo de Kepler lo puede realizar una computadora típica en una noche

Análisis de los resultados

Fuentes de error

- La incertidumbre en las observaciones de Tycho
- La excentricidad de la órbita terrestre
- El número de cifras reportadas en las tablas de funciones
- Errores en los cálculos a mano

Resultados

- La órbita de Marte es elíptica
- La velocidad no es constante

Contribuciones posteriores de Kepler

Con este tipo de trabajo, Kepler formuló sus tres leyes que rigen el movimiento planetario. En 1627 publicó las Tablas Rudolfianas, las cuales predecian la posición de los planetas. Estas tablas fueron 30 veces más precisas que las que estaban disponibles en esa época.

La correspondencia del modelo con las obervaciones experimentales fue la mejor confirmación!

Introducción al UNIX

Características

- multiusuario
- multitareas

Cada usuario tiene

- clave del usuario (login)
- contraseña (password)
- conjunto de permisos
- directorio de entrada (HOME)

Sesiones

- Al iniciar una sesión hay que introducir el login y el password
- Para terminar una sesión hay que teclear el comando exit
- El sistema UNIX distinge mayúsculas de minúsculas

Tipos de archivos

- datos
- ejecutables
- directorios

Sistema de directorios

- organización de los archivos
- discusión detallada en clase (revisar el tutorial de UNIX)
- dirección relativa y absoluta

Ordenes más comunes

Manipulación de directorios

cd	cambiar el directorio de trabajo (sin parámetros regresa a HOME)
mkdir	crear un directorio nuevo
rmdir	remover un directorio

Manipulación de archivos

ls [-lFa]	lista los archivos del directorio actual
cp	copia archivos
mv	mueve archivos (también permite renombrarlos)
rm	borra archivos
cat	permite ver el contenido de un archivo (no conviene usarlo en archivos binarios)
head	muestra las primeras diez líneas de un archivo
tail	muestra las últimas diez líneas de un archivo
grep	permite buscar una secuencia de caracteres en un archivo
sort	ordena un archivo alfabéticamente
diff	muestra las diferencias entre archivos
man	muestra informacion de una orden del UNIX

- El nombre de un archivo puede contener los siguientes caracteres {a-z, A-Z, 0-9, _, .)

Uso de comodines

Los carateres {*, ?} tiene una función especial en el shell. El símbolo * representa cualquier combinación de carateres, así

```
ls nom*
```

listará todos los archivos cuyo nombre inicie con las letras nom, mientras que

```
ls nom?
```

lista todos los archivos cuyo nombre tiene cuatro caracteres y los tres primeros son nom.

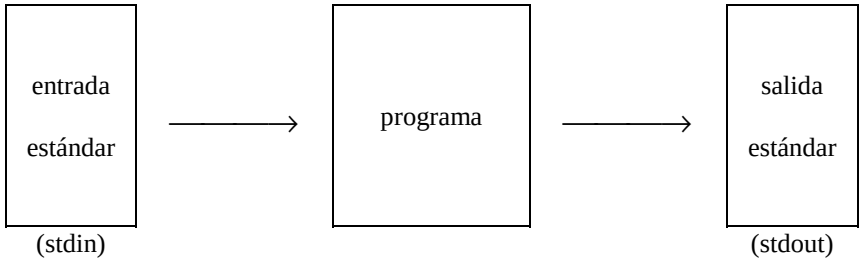
Sesión práctica: Manejo de archivos y directorios en UNIX

- Inicio y fin de una sesión
- Práctica con el sistema de directorios
- Manipulación de archivos

Sesión #3

Redirección en UNIX

Para un programa en UNIX:



En general, la entrada estándar es el teclado y la salida estándar es el monitor.

Redirección de la salida estándar

Suponga que los archivos bibl1 y bibl2 contienen referencias bibliográficas. En un formato tabular, cada renglón tiene al autor en la primera columna y a continuacion tema, editorial y año de publicación.

La orden siguiente muestra a los archivos bibl1 y bibl2 en la pantalla, uno detras del otro:

```
cat bibl1 bibl2
```

pero

```
cat bibl1 bibl2 > bibliografia
```

envía la información al archivo bibliografia (y no a la salida estándar). En el ejemplo siguiente, el archivo autor contiene la misma información que bibliografia, excepto que está ordenada alfabéticamente:

```
sort bibliografia > autor
```

Esta forma de redirección crea un archivo, si éste no existe, o bien lo remplaza, en caso de existir. Hay que tener cuidado ya que al remplazar al archivo, la información que contenía desaparece.

Exsite otra forma de redirección, la cual añade la salida de un programa a la información que ya existe en un archivo:

```
date > documento
cat bibl1 bibl2 >> documento
```

En este caso la primera instrucción envía la fecha a documento y la segunda añade el contenido de bibl1 y bibl2.

Redirección de la entrada estándar

Si un archivo contiene toda la información que un programa tomaría de la entrada estándar para funcionar, entonces la orden siguiente permite ejecutar al programa sin necesidad de teclear los datos:

```
programa < archivo
```

Encadenado (piping)

Cuando la salida estándar de un programa puede usarse como entrada para otro programa, a este proceso se le llama encadenado de procesos. Este se realiza de la manera siguiente:

```
cat bibl1 bibl2 | sort > autor
```

El archivo autor contiene toda la información de los archivos bibl1 y bibl2 ordenada alfabéticamente.

Para obtener todas las referencias de Lenguaje C, ordenadas por autor, se debe teclear:

```
cat bibl? | grep " C  " | sort
```

Editor de textos vi

El programa vi es un editor de pantalla. Para editar un archivo se debe teclear

```
vi archivo
```

El programa trabaja en un ambiente que ocupa toda la pantalla. Para salir del programa presione: `Esc :q!` cuando no quiere guardar los cambios, o bien: `Esc ZZ` para salir y guardar.

Modos del programa vi

Modo de comandos

En este modo inicia el programa. Se pueden borrar lineas o carateres, cambiar un carater por otro, moverse por el texto, etc.

`dd`	borra una línea
`x`	borra un carater
`i`	entra al modo de inserción
`o`	inserta una línea nueva abajo de la posición del cursor y entra al modo de inserción
`a`	permite entrar al modo de inserción cuando el cursor está al final de una línea
`u`	deshace el último cambio (undo)
`ZZ`	salir guardando los cambios

Modo de inserción

Permite ingresar texto. Para entrar al modo de inserción, se debe presionar {i, o, a} en el modo de comandos. Para regresar al modo de comados se debe presionar la tecla `Esc`.

Modo de la línea inferior

Dentro del modo de comandos, al presionar la tecla `:` el cursor aparece en la última línea de la pantalla. En ese momento se puede teclar otras instrucciones, por ejemplo:

`w`	guardar
`q`	salir
`q!`	salir sin guardar
`1`	ir a la primera línea del archivo
`num`	ir la la línea *num*.
`$`	ir a la última línea del archivo
`set showmode`	muestra en que modo se encuentra el programa
`set number`	muestra el número de cada línea del archivo
`set nonumber`	deja de mostrar el número de línea

`set autoindent` alinea el texto que se inserta con la línea anterior (útil para programar)

La orden set fija la configuración del programa mientras está en uso, sin embargo esta configuración se pierde al salir del programa. Para mantener la configuración durante toda la sesión, utilice la siguiente orden en el shell csh:

```
setenv EXINIT "set showmode autoindent"
```

Sesión práctica: Redirección y edición en vi

- Practicar comandos de UNIX
- Redirección
- Usar el editor vi para crear y modificar algunos archivos

Sesión #4

Cifras significativas

El número de cifras significativas de una variable observable depende de la precisión del proceso de medición. Cuando se combinan variables, el número de cifras significativas resultantes puede cambiar.

Suma

Si Δ representa un error absoluto,

$$a = a_1 \pm \Delta_a$$
$$b = b_1 \pm \Delta_b$$
$$c \equiv a + b = a_1 + b_1 \pm \left(\Delta_a + \Delta_b\right) = c_1 + \Delta_c$$
$$\Delta_c = \max\left(\Delta_a, \Delta_b\right)$$

por tanto, el número de decimales no puede aumentar.

Producto

Si δ representa un error relativo ($\delta < 1$),

$$a = a_1 \cdot \left(1 \pm \delta_a\right)$$
$$b = b_1 \cdot \left(1 \pm \delta_b\right)$$
$$c \equiv a \cdot b = \left(a_1 \cdot b_1\right)\left(1 \pm \delta_a\right)\left(1 \pm \delta_b\right) = \left(a_1 \cdot b_1\right)\left[1 \pm \left(\delta_a + \delta_b\right) + \delta_a \delta_b\right] = c_1 \cdot \left(1 \pm \delta_c\right)$$
$$\delta_c = \max\left(\delta_a, \delta_b\right)$$

así, el número de cifras significativas no puede aumentar.

Sin embargo al hacer una diferencias entre cantidades similares, el número de cifras significativas puede disminuir bruscamente.

Al evaluar funciones más complicadas, cada caso debe tratarse individualmente. Por ejemplo:

$$x = x_1 \pm \Delta_x$$
$$f \equiv e^x = e^{x_1} \cdot e^{\pm\Delta_x} = e^{x_1} + e^{x_1}\left(e^{\pm\Delta_x} - 1\right)$$
$$= f_1 \pm f_1 \cdot \delta_f = f_1 \cdot \left(1 \pm \delta_f\right)$$
$$\delta_f = \max\left(e^{\Delta_x} - 1{,}1 - e^{-\Delta_x}\right)$$

Error por redondeo

El número de cifras que se obtiene en una operación matemática siempre es finito y depende de las características de la calculadora.

Por ejemplo, si $\frac{1}{3} \approx 0.3333$

$$\frac{1}{3} + \frac{1}{3} + \frac{1}{3} \approx 0.3333 + 0.3333 + 0.3333 = 0.9999$$

Pérdida de cifras

Al realizar una diferencia entre cantidades de magnitudes similares, hay una disminución en el número de cifras. Por ejemplo, este problema se puede presentar al resolver una ecuación cuadrática $ax^2 + bx + c = 0$. Si a = 1, b = -320 y c = 16, las raíces calculadas con 4 cifras son:

x_1 = 319.9 y x_2 = 0.1

Mientras que las raíces con 6 cifras resultan ser:

x_1 = 319.950 y x_2 = 0.0500078

En la segunda raíz el error por el redondeo es del 100%.

Propagación del error

Otro tipo de problema aparece cuando un error que es pequeño se va acumulando al realizar un número muy grande de operaciones. A este problema se le denomina propagación del error y puede llegar a ser muy grande en algunos casos, como en el ejemplo siguiente.

Considere a la secuencia de integrales $E_n \equiv \int_0^1 x^n e^{x-1} dx$, con $n = 0, 1, ...$

Al integrar por partes se obtiene una relación de recurrencia, $E_n = 1 - nE_{n-1}$, con $E_0 = 1 - e^{-1}$. Además, $E_{n-1} > E_n > 0$.

El cálculo de estas integrales en forma recursiva y por integración numérica se presenta en la Tabla siguiente. La evaluación recursiva se realizó en una calculadora truncando los resultados a cuatro cifras (calc(4)), y mediante un programa en lenguaje C, usando variables tipo float (32 bits) y double (64 bits). La integración númerica se programó también en C.

Tabla. Cálculo de E_n con diferente número de cifras decimales

	E_n			
n	calc(4)	float	double	int num
0	0.6321	0.632121	0.632121	0.632121
1	0.3679	0.367879	0.367879	0.367879
2	0.2642	0.264241	0.264241	0.264241
3	0.2074	0.207277	0.207277	0.207277
4	0.1704	0.170893	0.170893	0.170893
5	0.1480	0.145534	0.145533	0.145533
6	0.1120	0.126796	0.126802	0.126802
7	0.2160	0.112430	0.112384	0.112384
8	-0.7280	0.100563	0.100932	0.100932
9	7.552	0.094933	0.091612	0.091612
10	-74.52	0.050674	0.083877	0.083877
11	820.7	0.442581	0.077352	0.077352

Nota: Los valores de la columna double difieren apreciablemente de los obtenidos por integración numérica para $n > 15$.

En los cálculos recursivos, al inicio, la sucesión es decreciente, pero posteriormente el valor absoluto de los términos crece sin límite, oscilando en signo. Este comportamiento no corresponde con las propiedades de la sucesión y tampoco se observa en los resultados obtenidos por integración numérica.

En general, cuando de comparan cálculos numéricos con mediciones experimentales debe haber una compatibilidad entre las cifras significativas y el error numérico.

Programación en C

Características generales

El lenguaje C es un lenguaje de programación de alto nivel y de aplicaciónes muy variadas. En este curso, su uso principal será el cómputo numérico.

Se recomienda definir el estilo de programación que será usado durante todo el curso.

Es muy conveniente incluir tantos comentarios como sea necesario para explicar lo que hace un programa, así como para aclara algunos puntos clave de los algoritmos utilizados. También en conveniente incluir comentarios que le indique al usuario que partes del programa deben ser modificadas cuando se cambian los parámetros o las condiciones más comunes.

Todo programa contiene una función `main`. Al ejecutar el programa, éste realiza las instrucciones que se encuentran en la función main.

Las instrucciones pueden extenderse por varias líneas y normalmente terminan en un punto y coma (;).

```
/*
 * Archivo: hola.c
 * ---------------
 * Este programa imprime un mensaje de bienvenida.
 * Su unico uso es didactico.
 */

#include <stdio.h>

main()
{
  printf("Hola, bienvenido al curso.\n");
}
```

Variables

Las variables son porciones de la memoria que almacenan información. Toda variable tiene un tipo y un nombre.

El tipo de una variable dependerá de la información que se desee almacenar. Para números enteros están los tipos `int` y `long`, para variables reales se usan los tipos `float` y `double`, para caracteres de texto está el tipo `char`, etc. Los valores máximo y mínimo de estos tipos de variables se deben comentar en clase.

Para utilizar una variable en C es necesario definirla. A este proceso se le denomina declaración:

```
<tipo> <nombre>;
int i, alumnos, unidad;
double distancia, costo, masa, concentracion;
```

Para que una variable tenga un valor se debe realizar una asignación. Para esta acción se usa el operador de asignación, =,

```
int alumnos;
double distancia, concentracion;

alumnos = 2;
distancia = 16.3492;
concentracion = 1.023e-5;
```

Los nombres de las variables pueden contener letras, números y el caracter de subrayado, _. Sin embargo, el primer caracter no puede ser un número. El lenguaje C si distingue entre mayúsculas y minúsculas.

Expresiones

Una expresión es una combinación de variables, operadores y funciones.

Operadores

Los operadores aritméticos son: { + - * / % }, donde % calcula el residuo de la división de dos enteros. Algunos operadores actúan antes que otros, el orden en el siguiente: primero actúa - cuando se usa para cambiar el signo, después { * / % } y al final { + - }. Cuando hay dos o más operadores con la misma prioridad, la evaluación se realiza de izquierda a derecha. Para cambiar el orden de evaluación se deben usar paréntesis. Por ejemplo

```
a + b /c
(a + b) / c
a / b * c
a / (b * c)
```

Conversión automática

Al realizar operaciones entre variables de diferente tipo, todas las cantidades se convierten al tipo de la variable de mayor tamaño. Así, al combinar variables double e int, todas las cantidades se convierten a double. Sin embargo si se realiza una división entre enteros, el resultado siempre es un entero. Por ejemplo, 1 / 2 resulta ser 0. Si esto no es lo que se quiere, se debe calcular 1.0 / 2.0 o 1.0 / 2 o bien realizar una conversión manual: (double) 1 / 2

```
int i, j;
double cociente;
cociente = (double) i /j;
```

De esta forma i se transforma en double y al dividir, el resultado también es de tipo double.

Sesión práctica: Compilación de un programa en C

- Escriba los siguientes programas con vi y compilelos

```
/*
 * Archivo: esfera.c
 * -----------------
 * Este programa calcula el volumen y la superficie
 * de una esfera e imprime los resultados.
 */

#include <stdio.h>
#define pi 3.14159

main()
{
  double radio, volumen, superficie;

  printf("Este programa calcula el volumen y la superficie"
         " de una esfera.\n");
  printf("Radio de la esfera: ");
  scanf("%lf", &radio);

  superficie = 4 * pi * radio * radio;
  volumen = superficie * radio / 3;

  printf("Radio= %f\n  Volumen= %f  Superficie= %f\n",
         radio, volumen, superficie);
}
```

```
/*
 * Archivo: sumaent.c
 * ------------------
 * Este programa calcula la suma de enteros consecutivos.
 */

#include <stdio.h>

main()
{
  int maximo, suma;

  printf("Este programa calcula la suma de enteros consecutivos"
         " del 1 al n.\n");
  printf("Ultimo entero: ");
  scanf("%d", &maximo);

  suma = maximo * (maximo + 1) / 2;

  printf("n= %d  suma= %d\n", maximo, suma);
}
```

- Escriba un programa que lea el costo de un objeto y la tasa de impuesto. Este programa deberá imprimir el costo original, el monto del impuesto y la cantidad total a pagar.

Sesión #5

Sistemas numéricos de punto flotante

- Aproximación al conjunto de los número reales
- Sistema finito

Base decimal

Elementos: $x = \pm m \times 10^{e}$

mantisa: $1 \le m < 10$

exponente: $e_{\min} \le e \le e_{\max}$

Si p es el número máximo de dígitos que puede tener la mantisa,

$$m = a_0 + a_1 10^{-1} + a_2 10^{-2} + \cdots + a_{p-1} 10^{-(p-1)}$$

$$a_0 = 1, 2, 3, \ldots, 9$$

$$a_i = 0, 1, 2, \ldots, 9 \quad (i = 1, 2, \ldots, p-1)$$

entonces p está relacionada con error relativo que se comete al representar a un número real por un elemento del sistema numérico.

Los valores de $e_{\min}$ y $e_{\max}$ determinan a los números de menor y mayor tamaño que se pueden representar en el sistema de punto flotante.

Así, { p, $e_{\min}$, $e_{\max}$ } definen completamente al sistema numérico.

Ejemplo

Sean { $p = 1$, $e_{\min} = -1$, $e_{\max} = 1$ }

$m = a_0 = 1, 2, 3, ..., 9$

$e = -1, 0, 1$

$x \in$ {1E-1, 2E-1, ..., 9E-1, 1, 2, ..., 9, 1E1, 2E1, ..., 9E1, 0, -0.1, ..., -0.9, -1, ..., -9, -10,..., -90}

Errores de tamaño

Números que por su magnitud no pueden ser representados por el sistema numérico.

0.2 x 0.2 = 0.04 < 0.1 "underflow"

50 + 50 = 100 > 90 "overflow"

Errores por redondeo

Representación aproximada de un número real:

1/3 ≈ 1E0 / 3E0 ≈ 3E-1

El error relativo de la aproximación es $\left| \frac{\frac{1}{3} - 3 \cdot 10^{-1}}{\frac{1}{3}} \right| = \left| \frac{1 - 9 \cdot 10^{-1}}{1} \right| = 0.1 = 10\%$

Base binaria

Elementos: $x = \pm m \times 2^{e}$

mantisa: $1 \le m < 2$

exponente: $e_{\min} \le e \le e_{\max}$

Si p es el número máximo de dígitos que puede tener la mantisa,

$$m = a_0 + a_1 2^{-1} + a_2 2^{-2} + \cdots + a_p 2^{-p}$$

$$a_0 = 1$$

$$a_i = 0, 1 \quad (i = 1, 2, \ldots, p)$$

Ejemplos

$$\left(1.011 \times 2^{-10}\right)_2 = \left(1 + 0/2 + 1/4 + 1/8\right) \times 2^{-(1\times 2 + 0 \times 1)} = 1.375 \times 2^{-2} = 0.34375 = 3.4375 \times 10^{-1}$$

$$0.3333 = 1.333 \times 2^{-2} = \left(1.010\,101\,010\,011\,001 \times 2^{-10}\right) \approx 0.33324$$

El programa binary.c genera la representación binaria de un número tipo float.

```
/*
 * Archivo binary.c
 * ----------------
 * Este programa genera la representacion binaria
 * de un numero real, tipo float.
 * La mantisa tiene 24 digitos
 * binarios y un exponente con 6 bits.
 *  x = signo * mantisa * 2 ^ expon
 */

#include <stdio.h>
#include <math.h>
#define PRECISION 24

main()
{
  int    i, expon, expabs,
         mantisa[PRECISION], exponen[PRECISION];
  double x, xabs, residuo;

  printf("Representaci¢n binaria de un numero real.\n");
  printf("Numero real: ");
  scanf("%lf", &x);
  printf("\n");

  xabs = fabs(x);
  expon = 0;

  /* Determinación del exponente
   * ---------------------------
   * Se trabaja con el valor absoluto.
   * El numero xabs se multiplica o divide por 2 hasta que quede
   * en el rango 1 <= xabs < 2
   */

  if (xabs > 1.0)
  {
    while(xabs >= 2)
    {
      expon++;
      xabs /= 2;
    }
  }
```

```
  else
  {
    while(xabs < 1)
    {
      expon--;
      xabs *= 2;
    }
  }

  /* Representacion binaria de la mantisa
   * ------------------------------------
   * La representacion binaria de xabs se almacena en
   * el arreglo mantisa.
   * Dado que la parte entera siempre es uno, el bit del
   * primer decimal esta en mantisa[0]
   */

  for ( i=0 ; i<PRECISION ; i++) mantisa[i] = 0;

  i = 0;
  xabs -= 1;
  residuo = 0.5;

  do
  {
    if (xabs >= residuo)
    {
      mantisa[i] = 1;
      xabs -= residuo;
    }
    i++;
    residuo /= 2;
  }
  while ( (i < PRECISION) && (xabs > 0) );

  /* Representacion binaria de expon
   * -------------------------------
   * La representacion binaria del valor absoluto del entero
   * expon se almacena en el arreglo exponen.
   * Los digitos se almacenan en orden inverso, el primer
   * digito ocupa la ultima posicion en el arreglo.
   */

  expabs = abs(expon);
  for ( i=0 ; i<PRECISION ; i++) exponen[i] = 0;

  i = 0;
  do
  {
    if ( (expabs % 2) == 1 )
      exponen[PRECISION - i - 1] = 1;
    i++;
    expabs /= 2;
  }
  while ( (i < PRECISION) && (expabs > 0) );

  /* Impresion de la representacion binaria
   * --------------------------------------
   * La impresion de la mantisa es directa.
   */

  printf("%e:\n ", x);
  if (x < 0) printf("-");
  printf("1.");
  for ( i=0 ; i<PRECISION ; i++) printf("%1d",mantisa[i]);

  /* Para el exponente, se salta los ceros iniciales. */

  printf(" E ");
  if (expon < 0) printf("-");
  if (expon == 0) printf("0");
  i = 0;
  while ( (exponen[i] == 0) && (i < PRECISION) ) i++;
  for ( ; i<PRECISION ; i++) printf("%1d", exponen[i]);
  printf("\n");
}
```

Este programa produce los siguientes resultados:

```
Representación binaria de un numero real.
-1.000000e+000:     -1.00000000000000000000000 E  0
 5.000000e-001:      1.00000000000000000000000 E -1
 1.000000e-001:      1.10011001100110011001101 E -100
 3.333000e-001:      1.01010101010110010011000 E -10
 3.402823e+038:      1.11111111111111111111110 E  1111111
 1.175494e-038:      1.11111111111111111111111 E -1111111
```

Determinación del radio de la Tierra

Método de Eratóstenes (~ 225 AC)

Información

Al medio día del primer día de verano, en Siena (hoy Assuán), una vara vertical no genera sombra sobre el piso.

En Alejandría, el ángulo que forma el sol con la vertical en aproximadamente 1/50 de círculo.

De acuerdo con las mediciones de esa época, la distancia entre Alejandría y Assuán es aproximadamente 5000 estadios, una unidad cuya longitud es desconocida.

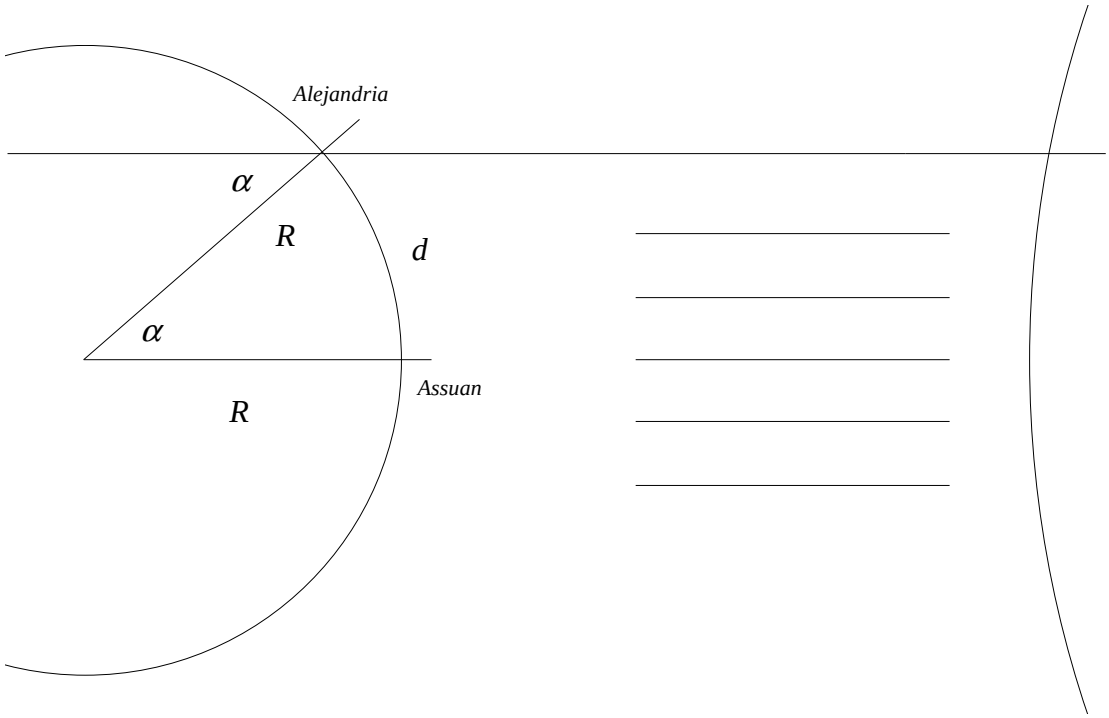


Figura 3. Esquema para la determinación del radio terrestre.

Referencias

C Sagan, *Cosmos*, Random House, ch. 1, New York (1980)

J Jeans, *Historia de la Física*, FCE, pp. 110-111, México (1982)

Alcance visual

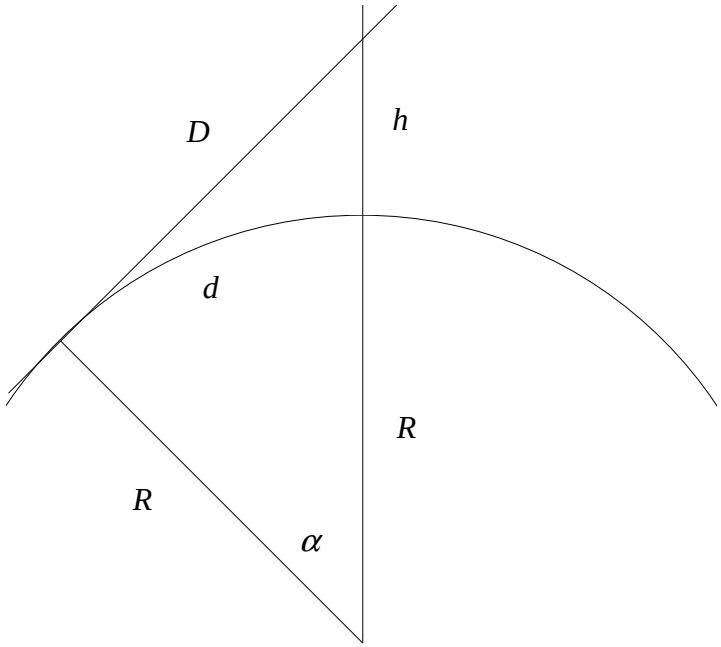


Figura 4. Alcance visual.

Para un observador, situado a una altura *h* sobre la superficie de una esfera, el punto más lejano que puede observar está a una distancia *d* del punto de referencia. Para determinar el alcance visual se requiere conocer el radio de la Tierra. Busque esta constante en la literatura.

Sesión práctica

- Utilice el programa binary.c para examinar la representación binaria de algunos números reales.
- Escriba un programa que determine el radio terrestre, usando el metodo de Eratóstenes.
    - Utilice los datos originales.
    - Recuerde que la función double sin(double x) requiere que el argumento esté en radianes.
    - En qué unidades se obtiene el resultado?
    - De un atlas, la distancia aproximada entre Assuán y Alejandría es 840 km. Cuál es la estimación del radio con este valor?
    - Compare sus resultados con el radio de la Tierra. (No olvide usar sólo las cifras que son significativas.)
    - Discuta las limitaciones del modelo
- Escriba un programa que determine el alcance visual. Verifique su programa con algunos casos límite y calcule el alcance visual con ejemplos representativos.

Sesion #6

Error numérico en las raíces de una ecuación cuadrática

Considere una ecuación cuadrática con raíces reales,

$$ax^2 + bx + c = 0 \quad ,$$

en donde el discriminante es positivo,

$$\Delta = b^2 - 4ac \geq 0 \quad .$$

En este caso las raíces son

$$x_1 = \frac{-b + d}{2a} \quad \text{y} \quad x_2 = \frac{-b - d}{2a} \quad ,$$

en donde

$$d \equiv \sqrt{\Delta} = \sqrt{b^2 - 4ac} \geq 0 \quad .$$

Cuando $b^2 >> |4ac|$, $d \approx |b|$, y, en alguna de las dos raíces, el numerador será muy pequeño. Al realizar una diferencia puede presentarse la pérdida de cifras significativas, por lo que el error numérico será mucho más grande en alguna de las dos raíces.

Método para minimizar el error numérico

Dado que $b = \text{sgn}(b) \cdot |b|$. La fórmula general toma la forma siguiente,

$$x = \text{sgn}(b) \frac{-|b| \pm d}{2a} \quad ,$$

y en el cálculo de una raíz no aparece una diferencia de cantidades,

$$r_1 = -\text{sgn}(b) \frac{|b| + d}{2a} \quad .$$

La ecuación cuadrática puede factorizarse utilizando sus raíces,

$$ax^2 + bx + c = a\left(x^2 + \frac{b}{a}x + \frac{c}{a} \right) = a(x - r_1)(x - r_2) = a\left[x^2 - (r_1 + r_2)x + r_1 r_2 \right] \quad .$$

Por lo que $c = ar_1 r_2$, y

$$r_2 = \frac{c}{ar_1} \quad .$$

Así, en el cálculo de las dos raíces no se evalúan diferencias.

El programa quad.c calcula las raíces reales de una ecuación cuadrática por ambos métodos.

```
/*
 * Archivo: quad.c
 * ---------------
 * Este programa calcula las raíces de un polinomio
 * cuadratico.
 * Esta limitado a polinomios con rices reales.
 * Las raíces se calculan usando la formula general
 * y por el procedimiento que mimimiza el error
 * por redondeo.
 */

#include <stdio.h>
#include <math.h>

main()
{
  float a, b, c, discr,
        x1, x2, r1, r2,
        f1, f2;

  printf("Programa para resolver ecuaciones cuadraticas\n");
  printf("de la forma:    a*x*x + b*x + c == 0\n");

  printf("  a: ");
  scanf ("%f", &a);

  printf("  b: ");
  scanf ("%f", &b);

  printf("  c: ");
  scanf ("%f", &c);

  discr = b * b - 4 * a * c;

  if (discr < 0)
    printf("El discriminante es negativo y las raíces son complejas!\n");
  else
  {
    /* evaluacion de la raíces con la formula general */

    discr = sqrt(discr);
    x1 = 0.5 * (-b + discr) / a;
    x2 = -0.5 * (b + discr) / a;
    f1 = a * x1 * x1 + b * x1 + c;
    f2 = a * x2 * x2 + b * x2 + c;
    printf(" x1= %e \t f(x1)= %e\n", x1, f1);
    printf(" x2= %e \t f(x2)= %e\n", x2, f2);

    /* minimizacion del error numerico */

    r1 = - 0.5 * (fabs(b) + discr) / a;
    if (b < 0) r1 *= -1;
    r2 = c / (a * r1);
    f1 = a * r1 * r1 + b * r1 + c;
    f2 = a * r2 * r2 + b * r2 + c;
    printf("\n r1= %e \t f(x1)= %e\n", r1, f1);
    printf(  " r2= %e \t f(x2)= %e\n", r2, f2);
  }
}
```

El programa anterior utiliza variables tipo float para hacer más evidente el error numérico.

A continuación se muestran algunos ejemplos en donde el error numérico es apreciable. Para cada conjunto de coeficientes, el primer bloque representa las raíces calculadas con la fórmula general, mientras que el segundo utiliza el método presentado en esta sesión.

```
Programa para resolver ecuaciones cuadraticas
de la forma:    a*x*x + b*x + c == 0

  a: 1
  b: 200000
  c: -3
 x1= 0.000000e+000       f(x1)= -3.000000e+000
 x2= -2.000000e+005      f(x2)= -3.000000e+000

 x1= -2.000000e+005      f(x1)= -3.000000e+000
 x2= 1.500000e-005       f(x2)= -7.556137e-008

  a: 1
  b: -16340
  c: 2
 x1= 1.634000e+004       f(x1)= 2.000000e+000
 x2= 4.882813e-004       f(x2)= -5.978516e+000

 x1= 1.634000e+004       f(x1)= 2.000000e+000
 x2= 1.223990e-004       f(x2)= -6.517043e-008

  a: 2
  b: -400000
  c: -1
 x1= 2.000000e+005       f(x1)= -1.000000e+000
 x2= 0.000000e+000       f(x2)= -1.000000e+000

 x1= 2.000000e+005       f(x1)= -1.000000e+000
 x2= -2.500000e-006      f(x2)= -2.524962e-008

  a: 5
  b: 800000
  c: 2
 x1= 0.000000e+000       f(x1)= 2.000000e+000
 x2= -1.600000e+005      f(x2)= 2.000000e+000

 x1= -1.600000e+005      f(x1)= 2.000000e+000
 x2= -2.500000e-006      f(x2)= 5.055550e-008

  a: 1
  b: 2000
  c: -0.25
 x1= 1.220703e-004       f(x1)= -5.859360e-003
 x2= -2.000000e+003      f(x2)= -5.859360e-003

 x1= -2.000000e+003      f(x1)= -5.859360e-003
 x2= 1.250000e-004       f(x2)= -1.604470e-009
```

En todos los casos se puede ver que una de las raíces, la de mayor valor absoluto, es idéntica en ambos métodos, ya que las operaciones realizadas son las mismas. Sin embargo, cuando se evalúa la función en la segunda raíz, se observa que el valor absoluto de la función es mucho menor.

Proyecto #1: Simulación de una titulación ácido-base

Para la titulación de un volumen V_a de un ácido débil, de concentración C_a, con una base fuerte, de concentración C_b, se tienen las siguientes relaciones estequiométricas,

	HA	+	OH^-	$\longrightarrow$	A^-	+	H_2O	
inicio	n_a							$n_a = C_a V_a$
se añade			n_b					$n_b = C_b V_b$
reacciona	αn_b		αn_b					$0 < \alpha < 1$
equilibrio	$n_a - \alpha n_b$		$(1-\alpha) n_b$		αn_b			

Dado que $V = V_a + V_b$, en el equilibrio se satisface la expresión para la constante de equilibrio:

$$K = \frac{\left[A^-\right]}{[HA]\left[OH^-\right]} = \frac{\alpha \frac{n_b}{V}}{\frac{n_a - \alpha n_b}{V}(1-\alpha)\frac{n_b}{V}} = \frac{\alpha}{1-\alpha}\frac{V}{n_a - \alpha n_b} \quad .$$

Esta expresión se reduce a una ecuación cuadrática para α. La ecuación presenta dos raíces reales y la de menor valor absoluto se encuentra entre 0 y 1.

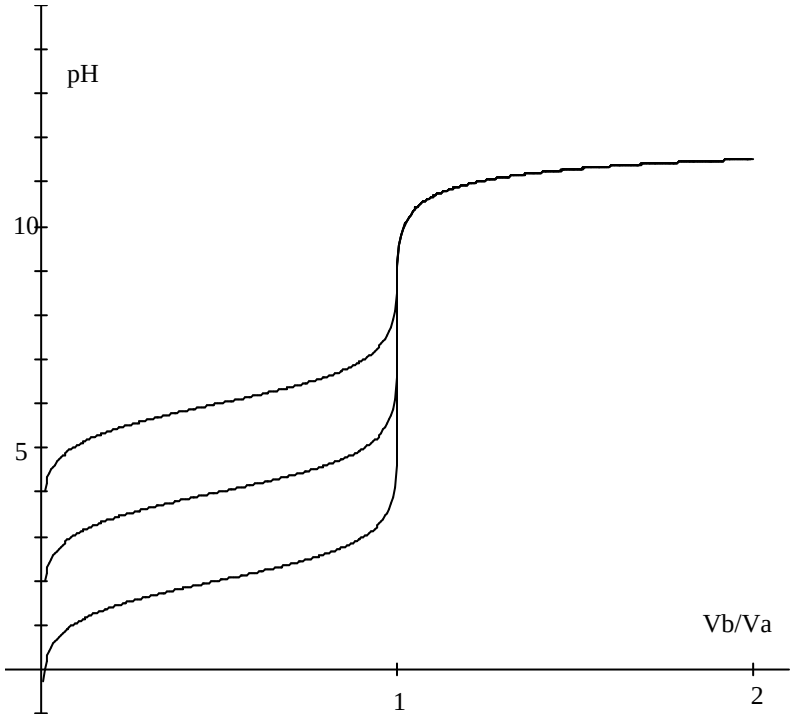


Figura 5. Curvas de titulación para diferentes ácidos débiles. Las gráficas están calculadas con Ca = Cb = 0.01M

Sesión práctica

- Escriba un programa para simular la titulación ácido-base descrita anteriormente.
- Utilice el programa para generar los datos que le permitan graficar las curvas de titulación de acidos débiles con K_a = 1E-2, 1E-4, 1E-6.
- El programa debe leer K_a, C_a, V_a, C_b, V_b e imprimir V_b y el pH de la solución en el equilibrio.

Sesión #7

Condicionales

Operadores

En el lenguaje C se tienen los siguientes operadores para establecer condiciones:

```
>       mayor que
<       menor que
>=      mayor o igual
<=      menor o igual
==      igual que
!=      diferente
```

Así, una condición puede falsa o verdadera. Por ejemplo, si x = 0, entonces,

```
x > 0   falso
x < 0   falsa
x >= 0  verdadero
x <= 0  verdadero
x == 0  verdadero
x != 0  falso
```

Para combinar dos o más condiciones se usan los operadores lógicos:

```
&&      y (AND)
||      o (OR)
```

La expresión `( cond1 && cond2 )` será veradera sólo si ambas condiciones son verdaderas, en cualquier otro caso, la expresión es falsa. Por otra parte, `( cond1 || cond2 )` es falsa sólo si ambas condiciones son falsas, en otro caso, la expresión es veradera.

El operador de negación cambia el valor de una expresión, esto es, a una condición veradera la vuelve falsa y viceversa,

```
!       negación
```

El orden de ejecución de estos operadores, junto con los operadores aritméticos, es el siguiente:

```
{ - ! } , { * / % } , { + - } , { < <= > >= } , { == != } , { && } , { || }
```

Estructura if

La estructura if permite realizar conjuntos de operaciones diferentes dependiendo del valor de una condición. Por ejemplo, el programa quad.c de la sesión #6 calcula las raíces de la ecuación cuadrática cuando el discriminante es positivo, ya que, en caso contrario, el programa no puede evaluar la raíz cuadrada de un número negativo.

Existen tres formas de usar la estructura if y se muestran a continuación:

```
/* estructura if con una sola opción */

if( condicion )
{
    /* este bloque se ejecuta solo cuando condicion es veradera */
}

/* estructura if con dos opciones posibles */

if( condicion )
{
    /* este bloque se ejecuta cuando condicion es veradera */
}
else
{
    /* este bloque se ejecuta cuando condicion es falsa */
```

```
}

/* estructura if con mas de dos opciones */

if( cond1 )
{
    /* este bloque se ejecuta si cond1 es es veradera */
}
else if( cond2 )
{
    /* este bloque se ejecuta si cond1 es falso, pero cond2 es veradera */
}
.
.
.
else
{
    /* este bloque se ejecuta cuando todas las condiciones anteriores son falsas */
}
```

Cuando se tienen varias condiciones es preferible programar estructuras if dentro de otras estructuras if (condicionales anidados). Por ejemplo

```
/* ecuacion cuadratica: a * x * x + b * x + c == 0 */

if( a == 0 )
{
  /* la ecuacion no tiene termino cuadratico
   *
   * si b != 0 la ecuacion es lineal y tiene una raíz real
   * si b == 0 se deben analizar las posibilidades
   *
   * estas posibilidades se pueden programar con una
   * estructura if
   */
}
else
{
  /* si discriminante < 0 se tienen raíces complejas
   * si discriminante > 0 se tienen raíces reales diferentes
   * si discriminante == 0 se tienen dos raíces reales iguales
   *
   * en este caso hay tres opciones, se puede optar por la
   * tercera forma de la estructura if
   */
}
```

Estructura switch ... case

Cuando una expresión puede tomar sólo un conjunto de valores bien definidos (normalmente de tipo int, long o char), la estructura switch es de gran utilidad:

```
/* estructura switch ... case */
switch( expresion )
{
  case valor1:
    /* este bloque se ejecuta cuando expresion == valor1 */
    break;
  case valor2:
    /* este bloque se ejecuta cuando expresion == valor2 */
    break;
  .
  .
  .
  default:
    /* este bloque se ejecuta cuando expresion es diferente de
     * todos los valores que aparecen previamente              */
    break;
}
```

La instrucción break permite salirse de la estructura switch.

Sesión práctica

Escriba un programa que resuelva una ecuación de la forma $a\,x^2 + b\,x + c = 0$.

Este programa debe tomar en cuenta todos los valores posibles de *a*, *b* y *c*, imprimir los mensajes adecuados para cada caso y calcular correctamente la o las soluciones.

Resuelva manualmente los casos siguientes y uselos para verificar que el programa funciona correctamente.

a	*b*	*c*
0	0	0
0	0	1
0	1	0
1	0	0
0	1	1
1	1	0
1	0	1
1	0	-1
1	2	1
1	2	3
1	2	-3

Sesión #8

Ciclos

Un ciclo es una estructura que permite repetir un conjunto de instrucciones. En el lenguaje C hay tres estructuras para ciclos.

Estructura for

La estructura for se utiliza para realizar un número determinado de repeticiones. En esta estructura se requiere de una variable de control (normalmente de tipo int) que se utiliza como un contador.

```
/* sintaxis de la estructura for */

for( valor inicial ; condicion de permanencia ; cambio en la variable de control)
{
  /* este bloque se repite mientras la variable de control cumpla con
   * la condicion de permanencia.
   *
   * al entrar a la estructura for, la variable de control toma el valor inicial.
   *
   * cada vez que se llega al final de la estructura, la variable cambia
   * de acuerdo con el cambio descrito en la ultima parte de la primera linea.
   */
}

/* ejemplo: el ciclo imprime 10 estrellas */

for( i = 0 ; i < 10 ; i++ ) printf("*");

/* ejemplo: el ciclo imprime 4 estrellas */

for( i = 0 ; i < 20 ; i += 5 ) printf("*");

/* ejemplo: el ciclo imprime 5 estrellas */

for( i = 5 ; i > 0 ; i-- ) printf("*");
```

Si la condición es falsa al iniciar la estructura for, el bloque de instrucciones no se ejecutará!

Estructura while

La estructura while repite un bloque de intrucciones mientras que una condición sea verdadera. Esta estructura no requiere de una variable de control.

```
/* sintaxis de la estructura */

while( condicion )
{
  /* este bloque se repite mientras que condicion sea veradera */
}

/* ejemplo: el ciclo imprime 5 estrellas */

j = 0;
while( j < 20 )
{
  printf("*");
  j++;
}
```

Si la condición es falsa al entrar a la estructura while, el bloque de instrucciones no se ejecutará!

Estructura do...while

La estructura do...while permite realizar ciclos en los que la condición se revisa al final.

```
/* sintaxis de la estructura do...while */

do
{
  /* este bloque se ejecuta mientras que condición sea veradera */
} while( condicion );

/* ejemplo: el ciclo imprime 5 estrellas */

j = 0;
do
{
  printf("*");
  j++
} while ( j < 5 );
```

Como la condición se prueba al final, el bloque se ejecutará al menos una vez!

Ejemplos

```
/* evaluacion de un factorial usando las tres estructuras */

int j, factorial, n;
n = 13;

factorial = 1;
for( j = 2 ; j <= n ; j++ ) factorial *= j;

factorial = 1;
j = 2;
while( j <= n)
{
  factorial *= j;
  j++;
}

factorial = 1;
j = 1;
do
{
  j++;
  factorial *= j;
} while( j < n );

/*
 * Archivo: sumain.c
 * -----------------
 * Este programa suma los enteros desde 1 hasta n.
 */

#include <stdio.h>

main()
{
  int i, n, suma;

  printf("Este programa suma los enteros consecutivos desde 1 hasta n.\n");
  printf("n = ");
  scanf ("%d", &n);

  suma = 0;
  if (n > 0)
  {
    for( i = 1 ; i <= n ; i++ ) suma += i;
    printf("n= %d  suma= %d\n", n, suma);
  }
  else
    printf("n = %d\n"
           "n debe ser mayor que cero!\n", n);
}
```

```
/* modificacion al programa de titulacion acido-base */

/* lectura de Ca, Va, Ka, Cb */
do
{
  printf("Volumen de base (en L): ");
  scanf ("%lf", &Vb);
  if( Vb >= 0 )
  {
    /* calculo del pH */
    printf("Para finalizar use un valor negativo.\n");
  }
} while( Vb >= 0 );

/* tabla de potencias de a */

x = 1.0;
for( j = 1 ; j <= n ; j++ )
{
  x *= a;
  printf("a^%d= %e\n", j, x);
}
```

Sesión práctica

- Utilice el programa sumain.c . Incremente *n* y observe que ocurre cuando el valor de la suma excede el valor del entero máximo. Discuta sus observaciones.

```
Este programa suma los enteros consecutivos desde 1 hasta n.
n = 0
n debe ser mayor que cero!
n= 1       suma= 1
n= 4       suma= 10
n= 10      suma= 55
n= 100     suma= 5050
n= 1000    suma= 500500
n= 64000   suma= 2048032000
n= 128000  suma= -397870592
```

- Escriba un programa para calcular la suma de *n* términos de una progresión geométrica, $1 + a + a^2 + \cdots + a^{n-1}$.
  - Para verificar sus resultados, incluya la impresión del valor exacto de la suma parcial.
  - Evalúe la suma con $a = 2$ y $a = 0.5$. Comente la relación de estos resultados con la representación binaria de un número.
- Escriba un programa que calcule la suma de *n* términos de la progresión armónica, $\sum_{j=1}^{n} \frac{1}{j}$.
  - Utilice una variable tipo float para la suma.
  - Las sumas parciales de la progresión armónica crecen indefinidamente, sin embargo, por usar una variable tipo float, llega un momento en que la suma deja de crecer a pesar de que se añaden más términos.
  - Relacione este comportamiento con el error de redondeo y sugiera una opción para minimizar el problema.
- Al usar la función printf para imprimir número reales puede usar %f (imprime 6 decimales), %e (imprime en notación científica) o %g (el programa elige representación la más corta entre %f y %e). Se puede controlar el espacio total a utilizar por la variable y el número de decimales a imprimir: %(espacio total).(decimales)(tipo de impresión) . Por ejemplo %13.5e imprime un real en un espacio de 13 caracteres, con 5 decimales, y en notación científica. Practique estas formas de impresión en sus programas.

Sesión #9

Funciones

En el lenguaje C, una función es un conjunto de instrucciones agrupadas bajo un nombre. Las funciones puden verse como programas dentro de un programa.

En general, los compiladores incluyen un conjunto de bibliotecas estándar que contienen muchas funciones. Por ejemplo:

```
printf("Hola\n")
sqrt( b * b - 4 * a * c )
log ( c_OH )
```

En particular, main() es una función, es la función que se ejecuta al iniciar un programa.

A la acción de ejecutar las instrucciones de una función se le denomina *llamada* de una función.

La mayoría de las funciones requieren de algunos parámetros para realizar su cometido, a los parámetros se les denomina argumentos. Los argumentos de una función se especifican dentro de un paréntesis. Cuando una función no requiere de argumentos, los paréntesis quedan vacíos como en main().

Muchas funciones regresan un valor cuando son llamadas, sin embargo no todas lo hacen. Al programar una función se debe decidir si ésta debe regresar o no un valor. Por ejemplo, si se ejecuta sqrt(x) la función sqrt regresa un número de tipo double. Una función que calcula la distancia que hay desde el origen del sistema de cordenadas hasta un punto del espacio R^3, requiere de tres argumentos reales (x, y, z), y debe regresar un número real.

Declaración de una función

Al igual que las variables, las funciones deben declararse antes de usarse. A la declaración de una función se le llama prototipo de la función. El prototipo indica el tipo de resultado que se regresa al ejecutar una función, el nombre de la función y, entre paréntesis, el tipo de cada argumento:

```
/* prototipo de una función
 * <tipo de resultado> <nombre>( <tipo de cada uno de los argumentos> );
 * por ejemplo:
 */
double sqrt( double );
double distancia( double, double, double );
int maximo( int, int);
void pausa(int);
char getch(void);
```

Una función es de tipo void cuando no regresa ningún valor. Cuando la lista de argumentos es void, la función no requiere de ningún argumento.

Los prototipos de las funciones de un programa deben estar antes de la función main().

Definición de una función

Las funciones se definen fuera de main(). Las instrucciones que forman una función están delimitadas por llaves, al igual que main(). Siempre que sea necesario, se pueden declarar variables dentro de una función y éstas son variables locales, esto es, sólo existen dentro de la función que las declaró.

```
/* definición de una función
 *
 * <tipo> <nombre>( <tipo y nombre local de cada argumento> )
 * {
 *   declaración de variables locales
 *   bloque de instrucciones
 * }
 */
```

```
/* ejemplos */

double distancia( double x, double y, double z)
{
  double r2;

  r2 = x * x + y * y + z * z;
  return sqrt(r2);
}

int maximo( int a, int b)
{
  if( a > b ) return a;
  else return b;
}
```

Note que en la definición de la función no hay un punto y coma en la primera línea, a diferencia del prototipo, el cual termina con punto y coma. Una función es como un subprograma.

Cuando una función debe regresar un valor, la definición de la función debe incluir al menos una instrucción return. Cuando la función encuentra una instrucción return, la función termina, se evalúa la expresión que se encuentra a la derecha de return, y regresa el valor de la expresión calculada.

Llamada de una función

Una función puede ser llamada por main() o por cualquier otra función. La llamada incluye el nombre y los argumentos dentro del paréntesis. Los argumentos pueden ser expresiones.

```
max = maximo( j, 25);
printf("d= %g\n", distancia(x1 - x2, y1 - y2, z1 - z2) );
```

El valor de cada argumento se copia en una variable local, por lo que una función no cambia las variables. Las variables locales sólo existen dentro de la función que las declara, así, aún cuando haya variables con nombres iguales dentro de varias funciones, estas variables son independientes.

Una función incluso puede llamarse a sí misma. A este tipo de funciones se les llama recursivas. Si se define una función recursiva, debe tenerse el cuidado de incluir una condición que finalize el proceso, ya que en el caso contrario el programa se quedaría indefinidamente ejecutando llamadas a la misma función.

Variables globales

Cuando varias funciones requieren accesar a un mismo conjunto de variables, y enviarlas como argumentos es muy complicado, una posibilidad consiste en declarar a estas variables como globales.

Las variables globales pueden ser accesadas y modificadas en cualquier parte del programa. Se recomienda evitar en lo posible el uso de variables globales, ya que la posibilidad de que cualquier función las pueda modificar hace más complicado el proceso de detección de errores y requiere de mucho más cuidado en la programación de las funciones. Por otro lado, cuando una función requiere de variables globales, al transportarla a otro programa requerirá también de las mismas variables globales.

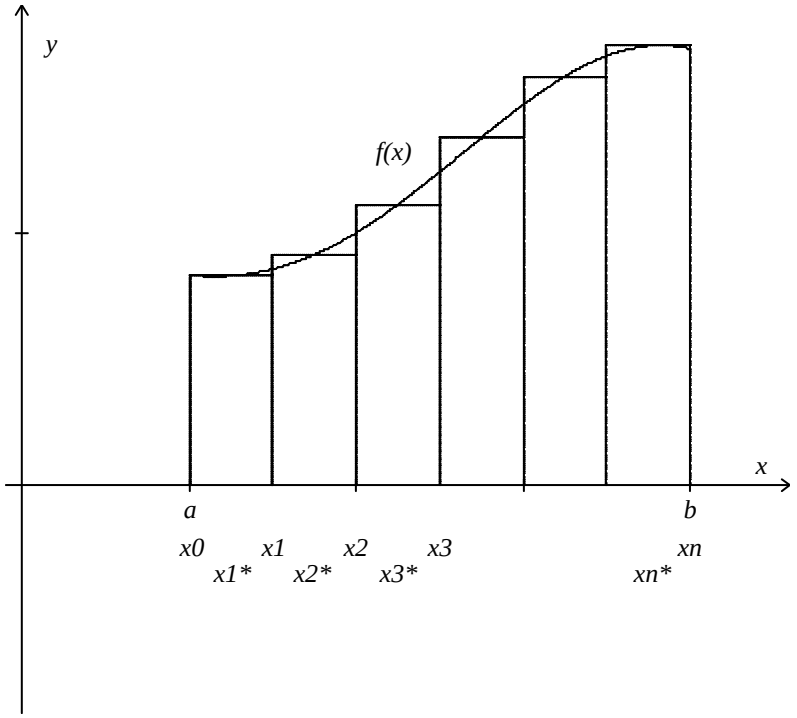
Integración numérica

Para una función matemática $f(x)$ no negativa, la integral definida $\int_a^b f(x)dx$ representa el area bajo la curva $y = f(x)$ y limitada por las rectas $x = a$ y $x = b$.

Existe una infinidad de integrales que no se pueden representar como una combinación finita de funciones elementales, en este caso una opción para evaluar esta integral consiste en obtener una aproximación numérica del área. Otro caso en que es útil la integración numérica ocurre cuando sólo se tiene una tabla de valores de la función, ya sea por que provienen de mediciones, o bien, porque es muy complicado calcular los valores.

Aproximación rectangular

En esta aproximación se hace una partición del intervalo [*a*, *b*] y en cada partición el área se aproxima por la superficie de un rectángulo.



La altura de cada rectángulo corresponde al valor de la función en el punto medio, $f_i^* \equiv f\left(x_i^*\right)$, en donde $x_i^* \equiv \frac{1}{2}\left(x_i + x_{i-1}\right)$. Dado que la partición es uniforme, la base de todos los rectángulos es igual, $h = \dfrac{b-a}{n}$, y la posición de las particiones está dada por $x_i = a + ih$. Así, el area de una partición puede escribirse como $A_i = h \cdot f_i^* = hf\left(a + \left[i - \frac{1}{2}\right]h\right)$ y la integral puede aproximarse por

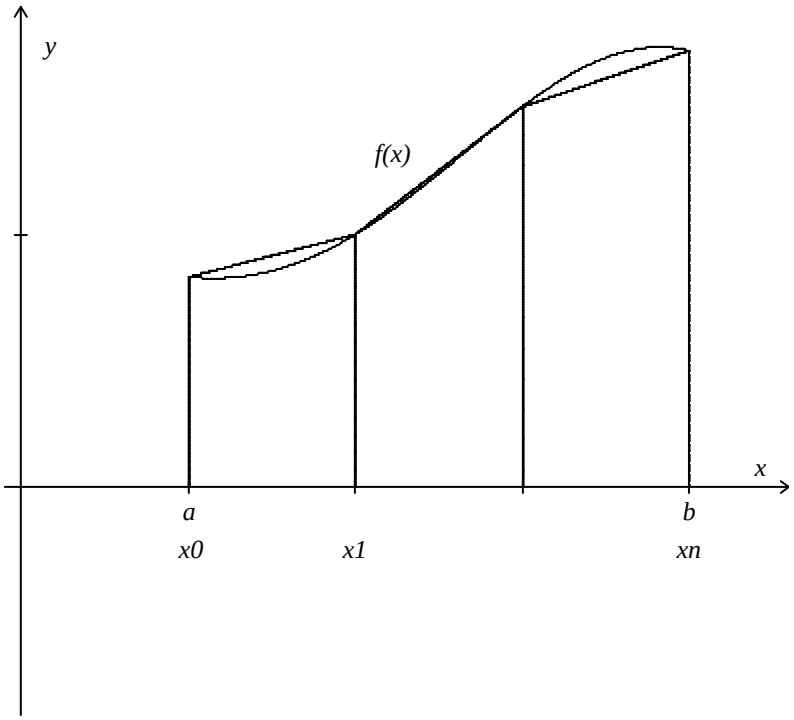
$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} A_i = h\sum_{i=1}^{n} f_i^* = h\sum_{i=1}^{n} f\left(a + \left(i - \tfrac{1}{2}\right)h\right).$$

Aproximación trapezoidal

En la aproximación trapezoidal, se hace una partición similar a la del método rectangular, sólo que el área de cada partición se aproxima por la superficie de un trapecio.

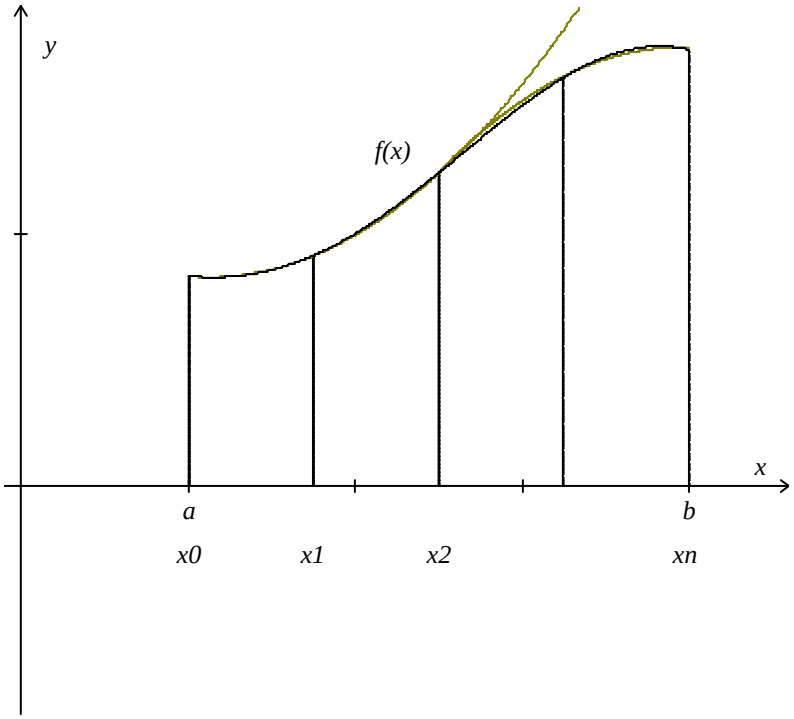
El área de un trapecio está dada por $A_i = \frac{1}{2}h \cdot \left[f_i + f_{i-1}\right]$, en donde $f_i \equiv f\left(x_i\right)$ y $x_i = a + ih$. Por lo que la aproximación trapezoidal a la integral toma la forma

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} A_i = \tfrac{1}{2}h\sum_{i=1}^{n}\left(f_i + f_{i-1}\right) = \frac{h}{2}\left(f_0 + 2\sum_{i=1}^{n-1} f_i + f_n\right).$$



Aproximación paraboidal (regla de Simpson)

En las aproximaciones anteriores, dentro de cada partición el integrado fue remplazado por funciones constante y lineal, esto es, polinomios de grado cero y uno. En un siguiente nivel, se usará un polinomio cuadrático. Para determinar una parábola se necesitan tres puntos, así que por cada dos particiones se usa una función parabólica. Por esta razón, el número de particiones debe ser par.



Por cada tres puntos consecutivos de la partición, $\{(x_{i-2}, f_{i-2}),(x_{i-1}, f_{i-1}),(x_i, f_i)\}$, pasa un polinomio cuadrático, $P_2(x) = Ax^2 + Bx + C$. Una vez que A, B y C han sido determinadas, el área bajo la parábola se determina por integración,

$$A_i = \int_{x_{i-2}}^{x_i} (Ax^2 + Bx + C)dx = \left(A\frac{x^3}{3} + B\frac{x^2}{2} + Cx \right)\Bigg|_{x_{i-2}}^{x_i} = \frac{h}{3}(f_{i-2} + 4f_{i-1} + f_i).$$

Así, la regla de Simpson toma la forma

$$\int_a^b f(x)dx \approx \frac{h}{3}(f_0 + 4f_1 + f_2 + f_2 + 4f_3 + f_4 + \cdots + f_{n-2} + 4f_{n-1} + f_n)$$
$$= \frac{h}{3}[f_0 + f_n + 4(f_1 + f_3 + \cdots + f_{n-1}) + 2(f_2 + f_4 + \cdots + f_{n-2})],$$

en donde *n* debe ser par.

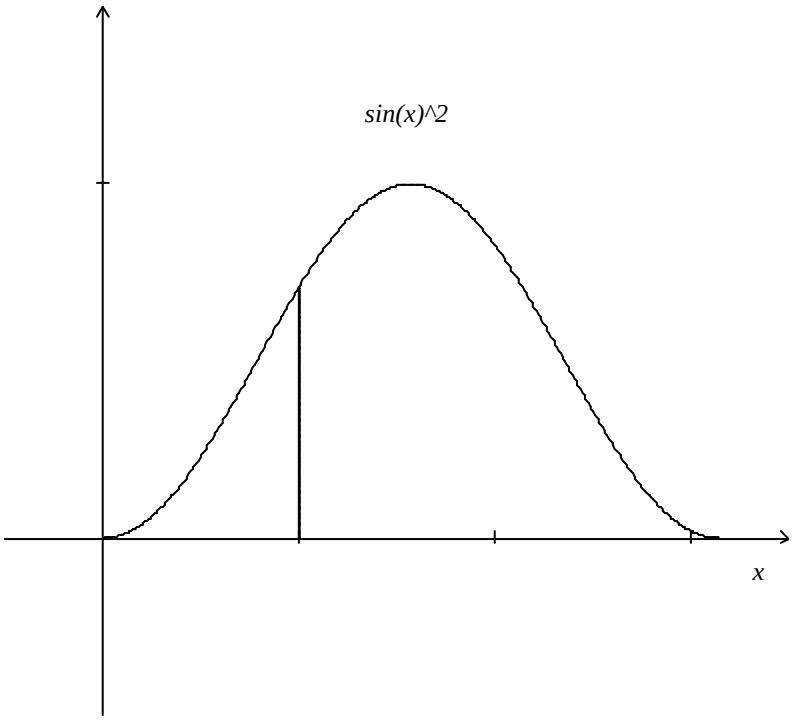
Sesión práctica

- Escriba un programa que aproxime la integral de una función con el método de rectángulos. Utilice una función para realizar el cálculo de la integral y otra función para el integrando.

```
/*
 * Archivo: int_rec.c
 * ------------------
 * Programa para integrar la funcion integrando(x)
 * usando rectangulos (punto medio).
 * Este programa usa funciones.
 */

#include <stdio.h>
#include <math.h>

double integrando( double);
double rectangular( double, double, int);

main()
{
  double a, b, integral;
  int intervalos;

  printf("Integracion numerica. Aproximacion rectangular.\n");
  printf("Limite inferior: ");
  scanf ("%lf", &a);

  printf("Limite superior: ");
  scanf ("%lf", &b);

  printf("Numero de intervalos: ");
  scanf ("%d", &intervalos);

  integral = rectangular( a, b, intervalos);
  printf("n= %-5d  integral= %16.8e\n", intervalos, integral);
}
```

  - Calcule la integral aproximada con 1, 2, 4, 8, 16, 32, 64, ... intervalos, hasta que el valor de la aproximación ya no cambie.

  - Para probar su programa use la función $f(x) = \sin^2(x)$, integrando en los intervalos [0, 1] y [0,π]. Compare los resultados numéricos con el valor exacto de la integral, $\int_a^b \sin^2(x)dx = \frac{1}{2}(b-a) - \frac{1}{4}(\sin 2b - \sin 2a)$.



- Repita el mismo procedimiento para: $\int_0^1 \exp\left(-x^2\right)dx$, $\int_{-1}^1 \exp\left(-x^2\right)dx$, $\int_0^1 \sqrt{1-x^2}\,dx$, $\int_{-1}^1 \sqrt{1-x^2}\,dx$, $\int_{-\pi}^{\pi} \sin x dx$, $\int_0^{\pi} \cos x dx$.

- Añada al programa anterior dos funciones. Una que aproxime la integral usando trapecios y otra que utilice la regla de Simpson. Calcule nuevamente las integrales del problema anterior y compare la eficiencia de cada método.

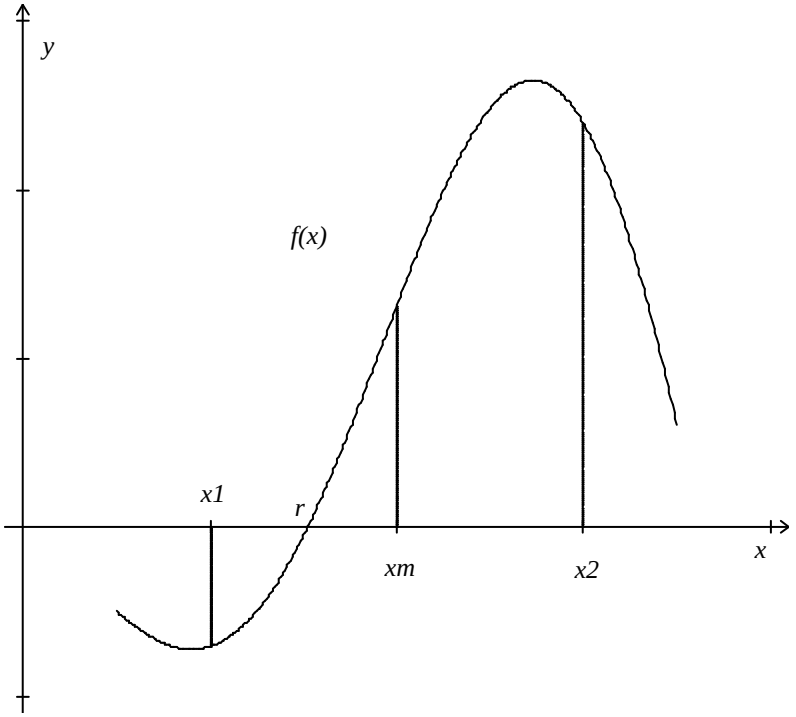
Sesión #10

Resolución de ecuaciones no lineales

Si $f(x)$ es una función no lineal, a los valores $x = r$ que satisfacen la ecuación $f(r) = 0$, se les denomina raíces (o ceros) de la función f.

Método de bisección

Si f es una función continua en el intervalo $[a, b]$ y $f(a)\, f(b) < 0$, entonces existe, al menos, un punto $r \in [a, b]$ tal que $f(r) = 0$.



Dado un intervalo que contiene a la raíz de la función, el método de bisección consiste en ir reduciendo el tamaño de dicho intervalo, partiendolo a la mitad, hasta que la longitud del intervalo sea menor que el máximo error permitido.

Si la raíz se encuentra en el intervalo $\left[x_1, x_2\right]$, con el punto medio, $x_m = \frac{1}{2}\left(x_1 + x_2\right)$, se forman dos intervalos de la mitad del tamaño del original, $\left[x_1, x_m\right]$ y $\left[x_m, x_2\right]$. Sólo resta determinar en cual de ellos se encuentra la raíz, para ello basta con analizar el signo de la función en los extremos.

Considere el siguiente ejemplo. Sea $f(x) = x^2 + x - 3$, dado que $f(1) = -1$ y $f(2) = 3$, entonces existe al menos una raíz en el intervalo [1, 2]. Sea 0.001 el error maximo que se permite en la raíz. El método de la bisección se aplica de la manera siguiente: $x_m = 1.5$ y $f_m \equiv f\left(x_m\right) = 0.75$. En el intervalo [1, 1.5] la función cambia de signo, por lo tanto la raíz se encuentra en [1, 1.5]. Los resultados de las estapas siguientes se presentan en forma tabular.

etapa	x1	f1	x2	f2	xm	fm	|x2 - x1|
1	1.00000	-1.00000	2.00000	3.00000	1.50000	+0.75000	1.00000
2	1.00000	-1.00000	1.50000	0.75000	1.25000	-0.18750	0.50000
3	1.25000	-0.18750	1.50000	0.75000	1.37500	+0.26560	0.25000
4	1.25000	-0.18750	1.37500	0.26560	1.31250	+0.03516	0.12500
5	1.25000	-0.18750	1.31250	0.03516	1.28125	-0.07715	0.06250
6	1.28125	-0.07715	1.31250	0.03516	1.29688	-0.02124	0.03125
7	1.29688	-0.02124	1.31250	0.03516	1.30469	+0.00690	0.01563
8	1.29688	-0.02124	1.30469	0.00690	1.30079	-0.00717	0.00781
9	1.30079	-0.00717	1.30469	0.00690	1.30274	-0.00013	0.00390
10	1.30274	-0.00013	1.30469	0.00690	1.30372	+0.00341	0.00195

En este caso, la raíz se encuentra en el intervalo [1.30274, 1.30372]. Dado que la función es un polinomio cuadrático, la raíz puede obtenerse usando la fórmula general, o bien el procedimiento descrito en la sesión #6. Así $r \approx 1.30278$.

El algoritmo del método de la bisección se puede escribir en la forma siguiente,

```
x1 = a;
x2 = b;
f1 = f(x1);
f2 = f(x2);
while( fabs(x2 - x1) > error )
{
  xm = 0.5 * (x1 + x2);
  fm = f(xm);
  if( f1 * fm < 0 )
  {
    x2 = xm;
    f2 = fm;
  }
  else
  {
    x1 = xm;
    f1 = fm;
  }
}
```

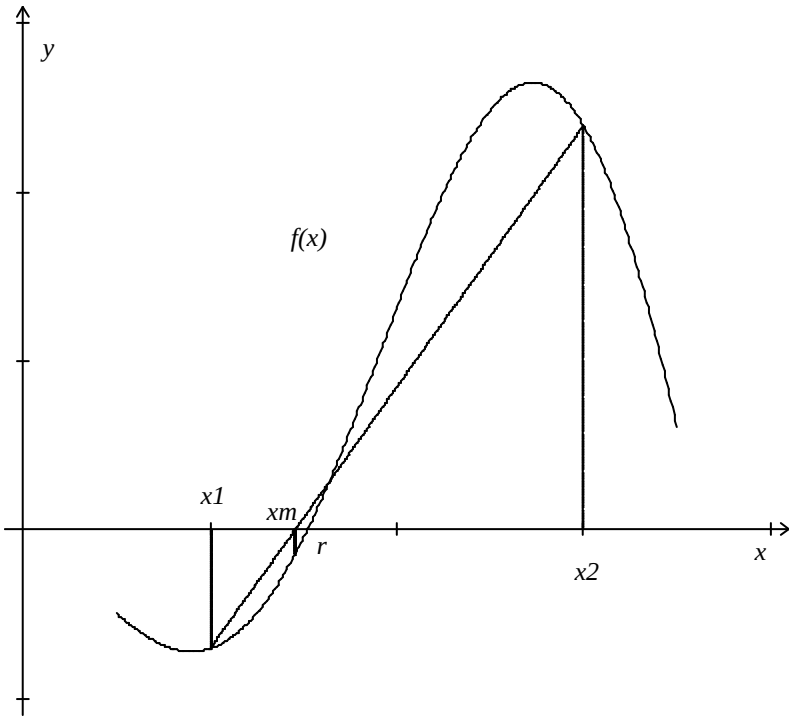
Este método siempre encuentra una raíz, aunque puede requerir de muchas etapas para alcanzar la precisión deseada. Por ejemplo, sea l_0 la longitud del primer intervalo, después de n etapas la longitud del intervalo que contiene a la raíz es $l_0\, 2^{-n}$. Si ε es el error máximo, entonces $l_0\, 2^{-n} < \varepsilon$, y el número de iteraciones será $n > \log(l_0 / \varepsilon) / \log 2$. En el ejemplo anterior, $l_0 = 1$, $\varepsilon = 10^{-3}$, por lo que $n > 9.96$; para una precisión mayor, por ejemplo $\varepsilon = 10^{-10}$, se requiere $n > 32$.

A continuación se presenta un método de requiere de menos etapas para alcanzar la misma precisión, sin embargo en el método nuevo no hay garantía de encontrar la raíz.

Método de la secante

El método de la secante requiere de dos puntos iniciales que estén cerca de la raíz. Con estos puntos (x_1 y x_2) el método construye una recta secante y estima la raíz (x_m). Dado que la función no necesariamente coincide con la secante, x_m no es una raíz de la ecuación, por lo que se descarta x_1 y se repite el procedimiento con x_2 y x_m.

En general, el método de la secante converge mucho más rápido que el de la bisección. Note que no es un requisito que la función tenga signos diferentes en los puntos de inicio.



Por ejemplo, para la misma función del ejemplo anterior, se obtiene los resultados siguientes,

```
j    x1       x2       xm        fm        |x2 - x1|
1  0.00000  1.00000  1.50000  7.500e-001  1.000e+000
2  1.00000  1.50000  1.28571 -6.122e-002  5.000e-001
3  1.50000  1.28571  1.30189 -3.204e-003  2.143e-001
4  1.28571  1.30189  1.30278  1.524e-005  1.617e-002
```

Así, después de cuatro iteraciones se obtiene una raíz, $r \approx 1.30278$, con un error máximo de 0.001. Observe que en este caso la raíz no se encuentra entre los puntos iniciales. Para los mismos puntos que se usaron en el método de la bisección, se obtiene la misma raíz después de cuatro iteraciones.

```
j    x1       x2       xm        fm        |x2 - x1|
1  1.00000  2.00000  1.25000 -1.875e-001  1.000e+000
2  2.00000  1.25000  1.29412 -3.114e-002  7.500e-001
3  1.25000  1.29412  1.30290  4.649e-004  4.412e-002
4  1.29412  1.30290  1.30278 -1.119e-006  8.787e-003
```

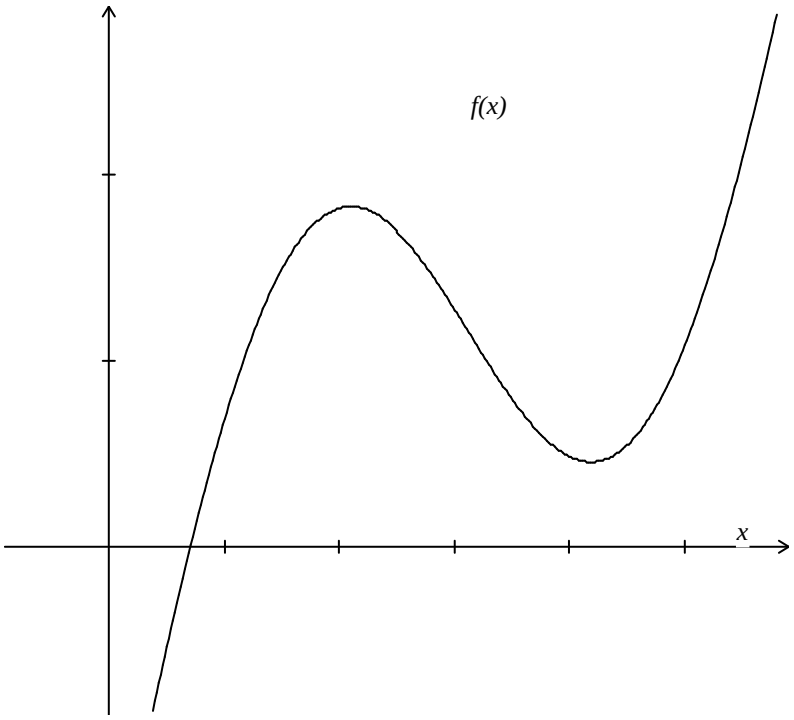
El método de la secante se puede representar por este algoritmo,

```
f1 = f(x1);
f2 = f(x2);
while( fabs(x2 - x1) > error )
{
  pendiente = (f2 - f1) / (x2 - x1);
  xm = x1 - f1 / pendiente;
  fm = f(xm);
  x1 = x2;
  f1 = f2;
  x2 = xm;
  f2 = fm;
}
```

Observe que cuando la pendiente de la secante es cero, el método no puede continuar y se deben cambiar los datos iniciales.

Este método puede fallar aún cuando los puntos iniciales no estén lejos de la raíz. Para funciones como la que se muestra en la figura siguiente el método presenta problemas. Si los puntos iniciales se encuentran después del máximo local, entonces el método de la secante se queda iterando alrededor del mínimo. Por esta razón es importante definir un número máximo de iteraciones e incluir un contador. Si el

contador excede el máximo de iteraciones se debe imprimir un mensaje para que el usuario se dé cuenta del problema.



Para la función $f(x) = x + 2\sin(x) - 2$, con *a* = 2.5 y *b* = 3, el método de la secante se queda iterando alrededor del mínimo que se encuentra cerca de *x* = 4. Cuando *a* = 3 y *b* = 2.5, el programa encuentra una raíz en *x* ≈ 0.7046, después de 24 iteraciones

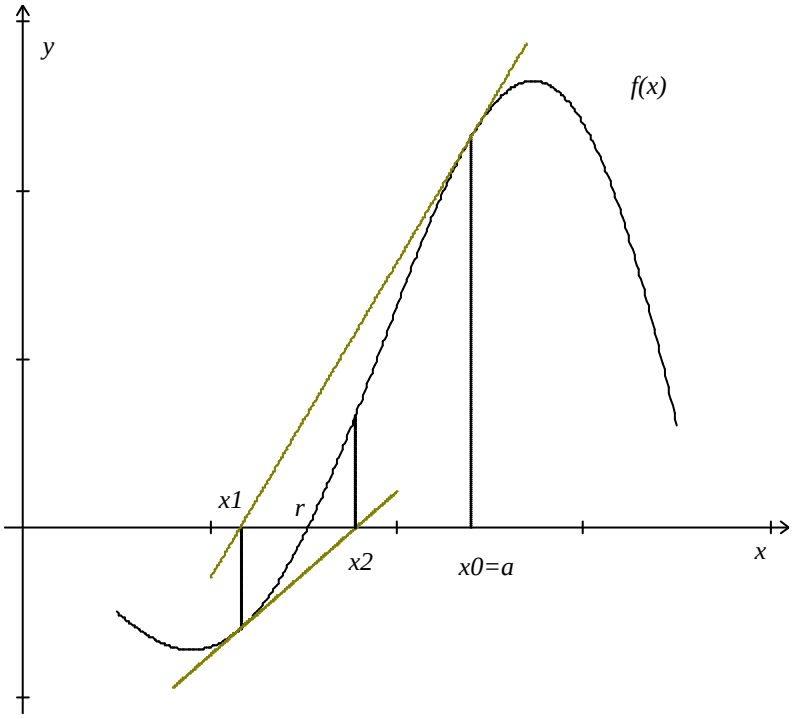
Sesión práctica

- Escriba los programas correspondientes a los dos métodos presentados.
- Pruebe sus programas con la función usada en los ejemplos.
- Verifique con ambos métodos que las funciones siguientes tienen las raíces indicadas
  - $g(x) = 3x + \sin x - e^x$ { 0.360422, 1.89003}
  - $h(x) = e^x - 4x^2$ { -0.407777, 0.714806, 4.30658 }
- La función $k(x) = -\left(1 + x^2\right)$ no tiene raíces reales. Cómo se comportan sus programas?
- Encuentre todas las raíces de $f(x) = x^3 - 2x + 1$.

Sesión #11

Método de Newton

El método de Newton puede verse como una variación del método de la secante. En lugar de utilizar una recta secante de la ecuación no lineal se usa la recta tangente. Para conocer la tangente a una curva es necesario poder calcular la derivada de la función, y este requisito no siempre puede ser cumplido.



Cuando es posible calcular la derivada en el punto x_i, la ecuación de la recta tangente a la curva $y = f(x)$ toma la forma,

$$r(x) = f_i + (x - x_i)m_i \ ,$$

en donde $f_i = f(x_i)$ y $m_i = f'(x_i)$. La aproximación a la raíz de $f(x)$ es el punto en el cual la recta $r(x)$ intersecta al eje horizontal, $r(x_{i+1}) = 0$. Así,

$$x_{i+1} = x_i - \frac{f_i}{m_i} = x_i - \frac{f(x_i)}{f'(x_i)} .$$

Al igual que en el método de la secante, el método de Newton no es aplicable cuando la pendiente de la recta es cero. También es conveniente incluir un límite para el número de iteraciones.

Al aplicar el método de Newton a la función $f(x) = x^2 + x - 3$, con un error máximo en la raíz de 0.001, se obtiene los resultados siguientes,

```
j   x(j-1)    f(j-1)       xj    |xj - x(j-1)|
1  1.00000 -1.000e+000  1.33333  3.333e-001
2  1.33333  1.111e-001  1.30303  3.030e-002
3  1.30303  9.183e-004  1.30278  2.546e-004
```

```
j   x(j-1)    f(j-1)       xj    |xj - x(j-1)|
1 -1.00000 -3.000e+000 -4.00000  3.000e+000
2 -4.00000  9.000e+000 -2.71429  1.286e+000
3 -2.71429  1.653e+000 -2.34101  3.733e-001
4 -2.34101  1.393e-001 -2.30317  3.784e-002
5 -2.30317  1.432e-003 -2.30278  3.971e-004
```

Al iniciar en $x = 1$, se obtiene la raíz $r \approx 1.30278$, mientras que partiendo de $x = -1$ se llega a $r \approx -2.30278$. En el segundo caso la primera estimación genera un salto grande, sin embargo el método se estabiliza rápidamente.

El método de Newton es un poco más rápido que el de la secante, aunque requiere de evaluar dos funciones por etapa (la ecuación y la derivada). En este sentido, ambos realizan un número similar de operaciones.

Sistemas de ecuaciones no lineales

Si se tiene un conjunto de n ecuaciones no lineales con n incógnitas, $f_j(x_0, x_1, ..., x_{n-1}) = f_j(\mathbf{x}) = 0$, para $j = 0, 1, ..., n-1$, este problema se puede escribir en forma matricial, $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = 0$. Sea $\boldsymbol{x}_i$ la aproximación inicial a la solución, la siguiente aproximación puede escribirse en la forma $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{h}$, en donde el vector $\boldsymbol{h}$ debe satisfacer la ecuación $\boldsymbol{f}(\mathbf{x}_i + \boldsymbol{h}) = 0$. A partir de un desarrollo en series de Taylor se puede obtener una estimación para $\boldsymbol{h}$,

$$\boldsymbol{f}(\boldsymbol{x}_i + \boldsymbol{h}) \approx \boldsymbol{f}(\boldsymbol{x}_i) + \begin{pmatrix} \nabla f_0(\boldsymbol{x}_i)\cdot\boldsymbol{h} \\ \nabla f_1(\boldsymbol{x}_i)\cdot\boldsymbol{h} \\ \vdots \\ \nabla f_{n-1}(\boldsymbol{x}_i)\cdot\boldsymbol{h} \end{pmatrix} = \boldsymbol{f}(\boldsymbol{x}_i) + \begin{pmatrix} \sum_{k=0}^{n-1} \frac{\partial f_0}{\partial x_k} h_k \\ \sum_{k=0}^{n-1} \frac{\partial f_1}{\partial x_k} h_k \\ \vdots \\ \sum_{k=0}^{n-1} \frac{\partial f_{n-1}}{\partial x_k} h_k \end{pmatrix} = \boldsymbol{f}(\boldsymbol{x}_i) + \mathbf{D}(\boldsymbol{x}_i)\boldsymbol{h} = 0\,,$$

por lo que el vector $\boldsymbol{h}$ es solución del problema matricial $\mathbf{D}\,\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{x}_i)$ y los elementos de la matriz $\mathbf{D}$ son las derivadas de las funciones f_j, $D_{jk} = (\partial f_j / \partial x_k)$. Observe que la ecuación matricial es prácticamente igual a la que se obtuvo para una ecuación no lineal. Una vez que se ha resuelto para $\boldsymbol{h}$, la siguiente aproximación a la solución es $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{h}$, y este procedimiento se repite hasta alcanzar el criterio de convergencia. Más adelante se tratarán métodos para resolver problemas matriciales.

Sesión práctica

- Programe el método de Newton
- Pruebe su programa con el ejemplo anterior
- Utilice el método de Newton con los ejercicios de la sesión previa

Sesión #12

Proyecto #2: Equilibrio químico (solubilidad)

Para un conjunto de concentraciones iniciales, la determinación de las concentraciones al equilibrio normalmente lleva a resolver una función polinomial. Este problema numérico se puede resolver con cualquiera de los métodos descritos en las secciones anteriores, sin embargo, el método de Newton se puede implementar fácilmente, ya que la derivada también será un polinomio.

Considere el problema de la solubilidad de un sólido iónico, en presencia de un ión común. Para fines demostrativos, se calculará la solubilidad de algunos sulfuros metálicos, con iones sulfuro en el medio. Para un problema más general, el procedimiento que se describe a continuación se puede generalizar fácilmente.

La estequiometría del proceso puede representarse en la forma siguiente,

	$M_mS_n\,(s)$ $\longrightarrow$	$m\,M^{a+}\,(ac)$ +	$n\,S^{2-}\,(ac)$	$0 = ma - 2n$
inicio			C_0	$C_0 \geq 0$
se disuelve s		$m\,s$	$n\,s$	$s > 0$
equilibrio		$m\,s$	$n\,s + C_0$	

En este caso s representa el número de moles del sólido que se disuelven por unidad de volumen, y queda determinado por la constante de equilibrio,

$$K_s = \left[M^{a+}\right]^m \left[S^{2-}\right]^n = (ms)^m \left(ns + C_0\right)^n .$$

Entonces la ecuación a resolver es un polinomio de grado $(m + n)$ en s,

$$f(s) = \left[M^{a+}\right]^m \left[S^{2-}\right]^n - K_s = (ms)^m \left(ns + C_0\right)^n - K_s .$$

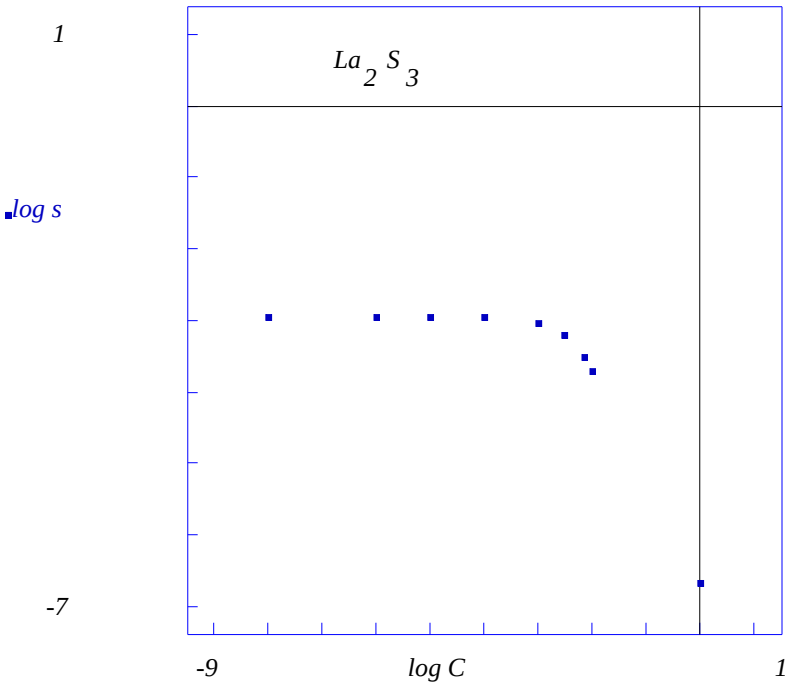
Para usar el método de Newton se requiere de una aproximación inicial. Esta puede obtenerse fácilmente si se asume que $s << C_0$, por lo que el polinomio se transforma una ecuación que se resuelve algebráicamente.

Cuando $C_0 = 0$, la solubilidad se puede calcular de forma directa, por lo que este caso no requiere de un método numérico.

Sesión práctica

- Escriba un programa para resolver el problema de la solubilidad
- El programa debe verificar que los datos están en el rango correcto. En particular, revise que las características del sólido corresponden con las de una especie neutra.
- Programe las ecuaciones para la aproximación inicial y la solubilidad en ausencia de sulfuros en funciones separadas y haga llamadas a éstas cuando sea necesario.
- Para este problema, la aproximación inicial propuesta permite obtener siempre una solución en el rango correcto? Qué otra opción se puede tomar?

- Aplique el programa para los sulfuros siguientes: Ag_2S, CdS, FeS, CuS, La_2S_3. Utilice las concentraciones $C_0 = 0M$, $10^{-2}M$, $10^{-4}M$, $10^{-6}M$. Busque las constantes de equilibrio en la literatura para que su programa utilice los valores correctos.



- Calcule previamente la solubilidad en ausencia de sulfuro, $C_0 = 0$, y verifique el resultado de su programa. Para $C_0 > 0$, se debe observar el efecto de ión común. Grafique sus resultados en escala logarítmica y compare el comportamiento de cada especie. Todos los compuestas presentan la misma tendencia?

Sesión #13

Arreglos

Un arreglo es una colección ordenada de variables de un mismo tipo.

Declaración

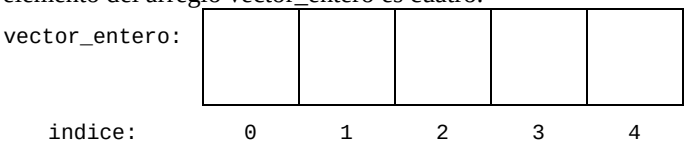
Para usar un arreglo, éste debe ser declarado. La declaración de un arreglo debe incluir el tipo de variables que contiene, el nombre y el tamaño (número de elementos que contiene).

```
/* declaracion de un arreglo
 * <tipo> <nombre>[<tamano>];
 */
int vector_entero[5];
```

Así, el arreglo vector_entero tiene cinco elementos de tipo entero.

Uso de los elementos de un arreglo

Los elementos de un arreglo se diferencían entre sí por su posición dentro del conjunto. **El primer elemento siempre es el elemento con índice igual a cero**, el segundo tiene el índice uno, y así sucesivamente hasta llegar al último elemento, el cual tiene índice igual a (tamaño - 1). Por lo que el índice del último elemento del arreglo vector_entero es cuatro.



Para referirse a un elemento de un arreglo se debe especificar el nombre del arreglo y, entre corchetes, el índice:

```
/* referencia a un elemento
 * <nombre>[<indice>]
 */
l = vector_entero[4];
m = vector_entero[0];
k = vector_entero[i - j + 3];
```

Como se observa en el ejemplo, el índice puede ser una expresión que da como resultado un entero.

```
/* ejemplo  */
int j, promedio, calificacion[22];
...
promedio = 0;
for( j = 0; j < 22; j++) promedio += calificacion[j];
promedio /= 22;
```

Si se usa un índice que es mayor que el último elemento, **no se genera un error**, y en este caso se accesa un espacio contiguo en la memoria del sistema. Este hecho generar resultados impredecibles y se pueden generar efectos colaterales.

Uso de arreglos en funciones

Una función puede usar arreglos como argumentos.

En el prototipo, para indicar que un argumento es un arreglo, se incluyen corchetes vacíos después del tipo de variable.

En la definición de la función, para indicar que uno de los argumentos es un arreglo, se incluyen corchetes vacíos después del nombre de la variable. Recuerde que el nombre de los argumentos que aparece en la definición es un nombre local.

Por último, al llamar a una función, basta con incluir el nombre del arreglo. No se usan corchetes ni el tamaño.

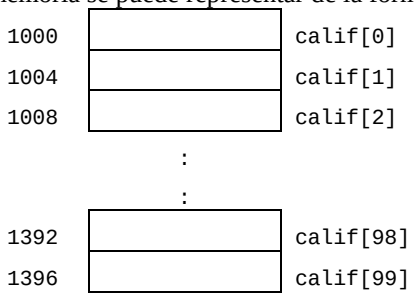
Analice el ejemplo siguiente,

```
/* uso de arreglos en funciones */
...
#define N_ELEM 100
int promedio(int[], int);
...
main()
{
  int n_eval, calif[N_ELEM];
  ...
  n_eval = 22;
  ...
  printf("Promedio= %d\n", promedio(calif, n_eval) );
}

int promedio( int arreglo[], int num)
{
  int j, suma;
  suma = 0;
  for( j = 0; j < num; j++)
    suma += arreglo[j];
  return suma / num;
}
```

En la función main() se declara un arreglo de 100 enteros, llamado calif. Así, el programa le reserva a calif el espacio necesario para 100 variables enteras. La definición de la función promedio() no requiere conocer el tamaño del arreglo que va a recibir, sin embargo, la variable num indica cuantos elementos del arreglo se deben utilizar. Como se mencionó anteriormente, num no debe ir más allá del último elemento.

En el ejemplo, se usan 22 elementos del arreglo calif, aunque el programa ha reservado espacio para 100. No es recomendable reservar espacio que exceda en gran medida las necesidades reales.

Los arreglos se almacenan en la memoria en forma contigua. Suponga que la dirección de memoria del primer elemento del arreglo calif es 1000. Si el compilador usa cuatro bytes para representar a un entero, el uso de memoria se puede representar de la forma siguiente:



El nombre del arreglo realmente contiene la dirección de memoria del primer elemento y, cuando se trata de accesar un elemento en particular, el programa determinar la dirección de memoria de dicho elemento sumado el número de bytes que corresponden a los elementos previos. Por ese motivo, las funciones no requieren conocer el tamaño del arreglo. Esta misma situación provoca que se accesen regiones de memoria ajenas al arreglo cuando el índice excede el tamaño.

En la llamada de la función promedio(), el argumento calif envía la dirección de memoria del primer elemento del arreglo.

Al usar un arreglo como argumento de una función, se está enviando una dirección de memoria a la función. Si la función modifica un elemento de un vector, lo hace directamente en la memoria, por esta razón, las funciones si pueden modificar el contenido de los arreglos. Con las variables no ocurre lo mismo,

ya que al llamar a la funcion se hace una copia de los valores en variables locales y la dirección de memoria de los argumentos nunca llega a la función.

Operaciones con vectores

Un arreglo se puede usar para representar a un vector. Sólo se debe tener en mente que al primer elemento le corresponde el índice cero.

El ejemplo siguiente calcula el producto punto entre dos vectores.

```
/* Archivo: prodint.c
 * ------------------
 * Este programa calcula el producto interno o
 * producto punto entre dos vectores.
 * La dimension maxima es N_ELEM.
 */
#include <stdio.h>
#define N_ELEM 10

double prodint(double[], double[], int);
void vec_leer(double[], int);

main()
{
  int n;
  double vector1[N_ELEM], vector2[N_ELEM];

  printf("\nCalculo del producto interno entre dos vectores.\n");
  printf("Numero de elementos de los vectores ( < %d): ", N_ELEM);
  scanf ("%d", &n);

  printf("\nVector A\n");
  vec_leer(vector1, n);
  printf("\nVector B\n");
  vec_leer(vector2, n);

  printf("\nA.B= %f\n", prodint(vector1, vector2, n) );
}

/* Rutina para calcular el producto interno */

double prodint(double v1[], double v2[], int renglon)
{
  double suma;
  int r;

  suma = 0;
  for( r = 0; r < renglon; r++)
    suma += v1[r] * v2[r];

  return suma;
}

/* Rutina para leer los elementos de un vector
 *
 * Si desea imprimir un mensaje que identifique al
 * vector, debe hacerlo antes de llamar a esta
 * rutina.
 */

void vec_leer(double v[], int renglon)
{
  int j;
  double x;

  for( j = 0; j < renglon; j++)
  {
    printf("v[%2d]: ", j);
    scanf ("%lf", &x);
    v[j] = x;
  }
}
```

Observe que la función vec_leer modifica a un arreglo, ya que le asigna valores a sus elementos.

Sesión práctica

- Escriba un programa que lea dos vectores tridimensionales y que calcule la suma, el producto punto y el producto cruz entre ellos.
- Incluya en su programa una función que imprima los elementos de un vector. Llamela vec_print.

Sesión #14

Arreglos multidimensionales

Para representar a una matriz se requiere de un arreglo con dos índices. En en lenguaje C se pueden tener tantos índices como uno quiera. En este caso, los índices también empiezan en cero.

Matriz de n x m: $\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,m-1} \\ a_{10} & a_{11} & \cdots & a_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,m-1} \end{bmatrix}$

```
/* declaracion de una matriz */
double matriz[n][m];
double mat_idn[3][3] =
  { {1, 0, 0},
    {0, 1, 0},
    {0, 0, 1}};
```

Un objeto de tres índices es el tensor de Levi-Civitta,

$$\varepsilon_{jkl} \equiv \begin{cases} +1 & (j,k,l) \in \{(0,1,2),(1,2,0),(2,0,1)\} \\ -1 & (j,k,l) \in \{(0,2,1),(1,0,2),(2,1,0)\} \\ 0 & \text{en otro caso} \end{cases}.$$

Este objeto se puede utilizar para calcular el producto cruz entre dos vectores.

```
int ejkl(int j, int k, int l)
{
  switch(j)
  {
    case 0:
      if( k == 1 && l == 2 ) return 1;
      else if( k == 2 && l == 1 ) return -1;
      else return 0;
    case 1:
      if( k == 2 && l == 0 ) return 1;
      else if( k == 0 && l == 2 ) return -1;
      else return 0;
    case 2:
      if( k == 0 && l == 1 ) return 1;
      else if( k == 1 && l == 0 ) return -1;
      else return 0;
  }
}

void prodcruz(double v1[3], double v2[3], double cruz[3])
{
  double suma;
  int j, k, l;
  int LeviCivitta[3][3][3];
  for( j = 0; j < 3; j++)
    for( k = 0; k < 3; k++)
      for( l = 0; l < 3; l++)
        LeviCivitta[j][k][l] = ejkl(j, k, l);
  for( j = 0; j < 3; j++)
  {
    suma = 0;
    for( k = 0; k < 3; k++)
      for( l = 0; l < 3; l++)
        suma  += LeviCivitta[j][k][l] * v1[k] * v2[l];
    cruz[j] = suma;
  }
}
```

Uso de arreglos multidimensionales en funciones

Cuando se usan arreglos multidimensionales como argumentos de funciones es necesario incluir el tamaño de cada dimensión en el prototipo y en la definición de la función. Sólo el tamaño de la primera dimensión es opcional.

Al llamar a la función solamente se debe incluir el nombre del arreglo, ya que este contiene la dirección de memoria.

A continuación se muestra un programa que calcula la suma de dos matrices.

```
/*
 * Archivo: sumamat.c
 * ------------------
 * Suma dos matrices de dimension n x m.
 * La dimension maxima es REN x COL.
 */

#include <stdio.h>
#define REN 10
#define COL 10

void mat_sumar(double[REN][COL], double[REN][COL],
               double[REN][COL], int, int);
void mat_leer(double[REN][COL], int, int);
void mat_escribir(double[REN][COL], int, int);

main()
{
  int nren, ncol;
  double matriz1[REN][COL],
         matriz2[REN][COL],
         matsuma[REN][COL];

  printf("Suma de matrices.\n");
  printf("Dimension maxima: %d x %d\n\n", REN, COL);

  printf("Renglones(<= %d): ", REN);
  scanf ("%d", &nren);
  printf("Columnas(<= %d): ", COL);
  scanf ("%d", &ncol);

  if( nren > 0 && nren <= REN && ncol > 0 && ncol <= COL)
  {
    printf("Matriz A.\n");
    mat_leer(matriz1, nren, ncol);
    printf("Matriz B.\n");
    mat_leer(matriz2, nren, ncol);

    mat_sumar(matriz1, matriz2, matsuma, nren, ncol);

    printf("Resultados finales:\n");
    printf("-------------------\n");
    printf("\nMatriz A.\n");
    mat_escribir(matriz1, nren, ncol);
    printf("\nMatriz B.\n");
    mat_escribir(matriz2, nren, ncol);
    printf("\nMatriz A + B.\n");
    mat_escribir(matsuma, nren, ncol);
  }
  else
  {
    printf("El taman~o de la matriz esta fuera de los limites!\n");
    printf("Dimension maxima: %d x %d\n", REN, COL);
  }
}

/* Rutina para sumar dos matrices
 * de la misma dimension.
 *
 * El resultado queda en la matriz suma.
 */
```

```
void mat_sumar(double m1[REN][COL], double m2[REN][COL],
               double suma[REN][COL], int renglon, int columna)
{
  int i, j;

  for( i = 0; i < renglon; i++)
    for( j = 0; j < columna; j++)
      suma[i][j] = m1[i][j] + m2[i][j];
}

/* Rutina para leer los elementos de una matriz
 *
 * Si desea imprimir un mensaje que identifique a
 * la matriz, debe hacerlo antes de llamar a esta
 * rutina.
 */

void mat_leer(double m[REN][COL], int renglon, int columna)
{
  int i, j;
  double x;

  for( i = 0; i < renglon; i++)
  {
    for( j = 0; j < columna; j++)
    {
      printf("m[%2d][%2d]: ", i, j);
      scanf ("%lf", &x);
      m[i][j] = x;
    }
    printf("\n");
  }
}

/* Rutina para imprimir los elementos de una matriz
 *
 * Si desea imprimir un mensaje que identifique a
 * la matriz, debe hacerlo antes de llamar a esta
 * rutina.
 */

void mat_escribir(double m[REN][COL], int renglon, int columna)
{
  int i, j;
  double x;

  for( i = 0; i < renglon; i++)
  {
    for( j = 0; j < columna; j++)
      printf("%7g ", m[i][j]);
    printf("\n");
  }
}
```

Algebra de matrices

$$C = A \pm B \quad c_{jk} = a_{jk} \pm b_{jk}$$
$$C = AB \quad c_{jk} = \sum_{l} a_{jl} b_{lk}$$
$$C = \alpha A \quad c_{jk} = \alpha \cdot a_{jk}$$

Sesión práctica

Escriba un programa que multiplique dos matrices

Sesión #15

Sistemas de ecuaciones lineales

Un sistema de ecuaciones lineales puede escribirse en la forma

$$\begin{aligned} a_{00}x_0 + a_{01}x_1 + \cdots + a_{0,m-1}x_{m-1} &= b_0 \\ a_{10}x_0 + a_{11}x_1 + \cdots + a_{1,m-1}x_{m-1} &= b_1 \\ \vdots & \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots + a_{n-1,m-1}x_{m-1} &= b_{n-1} \end{aligned},$$

o bien, en forma matricial

$$\mathbf{A}\vec{x} = \vec{b},$$

en donde $\mathbf{A}$ es una matriz de *n* renglones x *m* columnas, x y b son vectores columna de *n* renglones.

Para un sistema de *n* ecuaciones con *n* incógnitas la matriz $\mathbf{A}$ es una matriz cuadrada de *n* x *n*. Este sistema tiene solución única siempre que el determinante de la matriz sea diferente de cero. En esta sesión sólo se considera este caso.

Eliminación gaussiana

El método de la eliminación gaussiana tiene por objetivo reducir un sistema de ecuaciones a un sistema triangular.

Para transformar el sistema de ecuaciones se utilizan las siguientes operaciones:

- intercambiar renglones
- multiplicar un renglón por una constante
- añadir a un renglón el múltiplo de otro

A estas operaciones se les denomina operaciones elementales sobre los renglones. Este tipo de operaciones sólo combinan a las ecuaciones del sistema, pero no cambian su solución.

A continuación se muestra como funciona este método con un ejemplo.

Considere el siguiente sistema de tres ecuaciones con tres incógnitas,

$$\begin{pmatrix} 4 & -2 & 1 \\ -3 & -1 & 4 \\ 1 & -1 & 3 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 15 \\ 8 \\ 13 \end{pmatrix}.$$

Cuando se realizan operaciones elementales, los cambios también afectan al vector *b*. Por esta razón, cuando se utiliza este método manualmente, se trabaja simultáneamente con la matriz **A** y con el vector *b*. Este proceso es más sencillo si se usa el concepto de la "matriz aumentada", [**A**|*b*], la cual es una matriz con una columna adicional y contiene a los elementos del vector *b*. En este ejemplo la matriz aumentada es la siguiente,

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ -3 & -1 & 4 & 8 \\ 1 & -1 & 3 & 13 \end{array}\right].$$

En un sistema triangular, los elementos bajo la diagonal son cero. Para transformar el sistema de ecuaciones en uno triangular aplicaremos algunas operaciones elementales:

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ 0 & -10 & 19 & 77 \\ 0 & -2 & 11 & 37 \end{array}\right] \begin{array}{l} \\ 3R_1 + 4R_2 \\ -R_1 + 4R_3 \end{array}.$$

Para obtener ceros bajo el primer elemento de la diagonal se realizaron las operaciones que se muestan a la derecha de la matriz. Por ejemplo, en el segundo renglón, éste se multiplicó por 4 y al resultado se le añadió el primer rengón multiplicado por 3. Por este procedimiento se eliminan todos los elementos bajo la diagonal:

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ 0 & -10 & 19 & 77 \\ 0 & 0 & 36 & 108 \end{array}\right] \begin{array}{l} \\ \\ -R_2 + 5R_3 \end{array}.$$

El sistema triangular resultante se resuelve de abajo hacia arriba, sustituyendo las soluciones ontenidas en las etapas previas:

$$\begin{aligned} x_2 &= \frac{108}{36} = 3 \\ x_1 &= \frac{77 - 19x_2}{-10} = -2 \\ x_0 &= \frac{15 - x_2 + 2x_1}{4} = 2 \end{aligned}.$$

Así, el vector de soluciones del sistema de ecuaciones es

$$\vec{x} = \begin{bmatrix} 2 \\ -2 \\ 3 \end{bmatrix}.$$

En este ejemplo se puede observar que los coeficientes crecen rápidamente, por lo que es más recomendable trabajar con números fraccionarios.

El mismo ejemplo se resuelve con números fraccionarios:

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ -3 & -1 & 4 & 8 \\ 1 & -1 & 3 & 13 \end{array}\right] \qquad \left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ 0 & -2.5 & 4.75 & 19.25 \\ 0 & -0.5 & 2.75 & 9.25 \end{array}\right] \begin{array}{l} \\ -\left(\frac{-3}{4}\right)R_1 + R_2 \\ -\left(\frac{1}{4}\right)R_1 + R_3 \end{array},$$

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ 0 & -2.5 & 4.75 & 19.25 \\ 0 & 0 & 1.8 & 5.4 \end{array}\right] \begin{array}{l} \\ \\ -\left(\frac{-0.5}{-2.5}\right)R_2 + R_3 = -0.2R_2 + R_3 \end{array}.$$

En este caso, el determinante de la matriz transformada coincide con el determinante de la matriz original, pero para una matriz triangular, el determinante es simplemente el producto de todos los elementos de la diagonal.

Si **U** es la matriz triangular y **L** se forma con el negativo de los coeficientes de las transformaciones,

$$\mathbf{L} \equiv \begin{bmatrix} 1 & 0 & 0 \\ -0.75 & 1 & 0 \\ 0.25 & 0.2 & 1 \end{bmatrix} , \qquad \mathbf{U} \equiv \begin{bmatrix} 4 & -2 & 1 \\ 0 & -2.5 & 4.75 \\ 0 & 0 & 1.8 \end{bmatrix} ,$$

entonces **L U** = **A**. A este procedimiento se le llama la descomposición LU de la matriz **A**. Para fines prácticos, los elementos de **L** que están bajo la diagonal se pueden almacenar en la matriz **U**, ya que **U** tiene ceros bajo la diagonal.

En resumen, la eliminación gaussiana permite:

- resolver el sistema de ecuaciones
- calcular el determinante de la matriz **A**
- obtener la descomposición LU

Algoritmos

Para escribir un algoritmo, considere un caso genérico,

$$\left(\begin{array}{cccc|c} a_{00} & a_{01} & \cdots & a_{0,n-1} & b_0 \\ a_{10} & a_{11} & \cdots & a_{1,n-1} & b_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} & b_{n-1} \end{array}\right),$$

$$\left(\begin{array}{cccc|c} a_{00} & a_{01} & \cdots & a_{0,n-1} & b_0 \\ 0 & a'_{11} & \cdots & a'_{1,n-1} & b'_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n-1,1} & \cdots & a'_{n-1,n-1} & b'_{n-1} \end{array}\right) \qquad \begin{array}{ll} -l_{10}R_0 + R_1 & l_{10} = \dfrac{a_{10}}{a_{00}} \\ \vdots & \vdots \\ -l_{n-1,0}R_0 + R_{n-1} & l_{n-1,0} = \dfrac{a_{n-1,0}}{a_{00}} \end{array} ,$$

en donde, para $d = 0$,

$$a'_{jk} = a_{jk} - l_{jd}a_{dk} \qquad , \quad (d < j\,, k < n) \quad , \qquad l_{jd} = \frac{a_{jd}}{a_{dd}} \quad ,$$

$$b'_j = b_j - l_{jd}b_d \qquad , \quad (d < j < n).$$

Este procedimiento se repite para cada elemento de la diagonal. La matriz **L** se forma con los coeficientes l_{jd},

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ l_{10} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ l_{n-2,0} & l_{n-2,1} & \cdots & 1 & 0 \\ l_{n-1,0} & l_{n-1,1} & \cdots & l_{n-1,n-2} & 1 \end{pmatrix}.$$

Con las expresiones anteriores se puede construir el algoritmo de la eliminación gaussiana.

```
/* Eliminacion gaussiana */
for( d = 0; d < nren - 1; d++)
  for( r = d + 1; r < nren; r++)
  {
    coef = a[r][d] / a[d][d];
    for( c = d + 1; c < ncol; c++)
      a[r][c] -= coef * a[d][c];
    b[r] -= coef * b[d];
    a[r][d] = 0;
  }
```

La matriz triangular se resuelve de abajo hacia arriba.

```
/* Sustitucion en reversa */
x[nren - 1] = b[nren - 1] / a[nren - 1][nren - 1];
for( r = nren - 2; r > -1; r--)
{
  suma = b[r];
  for( c = nren -1; c > r; c--)
    suma -= a[r][c] * x[c];
  x[r] = suma / a[r][r];
}
```

Sesión práctica

Construya dos funciones una para la eliminación y otra para la sustitución.

Escriba un programa para resolver un sistema de ecuaciones.

Pruebe su programa con el ejemplo descrito.

Resuelva el sistema siguiente:

$$\begin{pmatrix} 6 & 1 & -6 & -5 \\ 4 & -3 & 0 & 1 \\ 2 & 2 & 3 & 2 \\ 0 & 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ -7 \\ -2 \\ 0 \end{pmatrix}$$

Sesión #16

Otros métodos para los sistemas de ecuaciones lineales

Cuando se tienen varios sistemas de ecuaciones que involucran a la misma matriz **A**, se puede extender el método de eliminación o recurrir a la descomposición LU. Estos sistemas tiene la forma,

$$\mathbf{A}\vec{x}_i = \vec{b}_i \quad , \quad (\, i = 0, 1, ..., p\text{ -}1\,) \quad .$$

Si se resolviera cada uno por separado, siempre se harían las mismas operaciones sobre **A** y b_i. La matriz triangular sería siempre la misma, pero los vectores transformados no, ya que cada uno es diferente.

Extensión de la eliminación gaussiana

Para no repetir las operaciones sobre **A**, la primera opción consiste en incluir a todos los vectores b_i en la matriz aumentada,

$$\left[\mathbf{A} \;\middle|\; \vec{b}_0 \;\middle|\; \vec{b}_1 \;\middle|\; \cdots \;\middle|\; \vec{b}_{p-1}\right] = \left[\mathbf{A} \mid \mathbf{B}\right],$$

en donde la matriz **B** tiene *n* renglones x *p* columnas, y las columnas son los vectores b_i. Con los vectores x_i se forma la matriz **X**, y el problema de resolver varios sistemas con la misma matriz se transforma en el problema matricial

$$\mathbf{A}\,\mathbf{X} = \mathbf{B}\,,$$

en donde las matrices **X** y **B** tienen de las mismas dimensiones.

Para realizar este proceso se deben adaptar el método de eliminación y el de sustitución en reversa para que realicen sus operaciones sobre las matrices **B** y **X**, y no sobre los vectores *b* y *x*. A continuación se resaltan las partes de los algoritmos que cambian.

```
/* Eliminacion gaussiana usando matrices*/
for( d = 0; d < nren - 1; d++)
  for( r = d + 1; r < nren; r++)
  {
    coef = a[r][d] / a[d][d];
    for( c = d + 1; c < ncol; c++)
      a[r][c] -= coef * a[d][c];
    for( cb = 0; cb < ncolb; cb++)
      b[r][cb] -= coef * b[d][cb];
    a[r][d] = 0;
  }

/* Sustitucion en reversa usando matrices*/
for( cb = 0; cb < ncolb; cb++)
{
  x[nren - 1][cb] = b[nren - 1][cb] / a[nren - 1][nren - 1];
  for( r = nren - 2; r > -1; r--)
  {
    suma = b[r][cb];
    for( c = nren -1; c > r; c--)
      suma -= a[r][c] * x[c][cb];
    x[r][cb] = suma / a[r][r];
  }
}
```

Observe que los cambios consisten en usar arreglos bidimensionales y en repetir los procedimientos para todas las columnas de las matrices nuevas.

Uso de la descomposición LU

La segunda opción consisten en realizar primero la descomposición LU usando el método de eliminación gaussiana, pero sólo sobre la matriz **A**. La transformación de las columnas de la matriz **B** sólo requiere de los coeficientes de la matriz **L**, y éstos se encuentran almancenados bajo la diagonal de la matriz **A**, una vez que se le ha aplicado el algortimo de descomposición. Ya que **B** ha sido transformada, se aplica la sustitución en reversa.

```
/* Solucion de problema matricial
 * usando la descomposicion LU.
 *
 * en este caso:
 *   ncolA = nrenA
 *   nrenB = nrenA
 */
lu( matrizA, nrenA, ncolA);
transformaB( matrizB, nrenB, ncolB, matrizA, nrenA, ncolA);
sustitucion_mat( matrizA, matrizB, matrizX, nrenA, ncolA, ncolB);
```

Matriz inversa

La inversa de una matriz se puede calcular fácilmente con estos métodos. Si para el problema matricial, la matriz **B** es una matriz identidad de *n* x *n*,

$$\mathbf{A}\,\mathbf{X} = \mathbf{I} \ ,$$

entonces, por la unicidad de la matriz inversa, $\mathbf{X} = \mathbf{A}^{-1}$.

Eliminacíon gaussiana con pivote parcial

Si se desea resolver el sistema de ecuaciones

$$\begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 2 & 3 & 2 \\ 4 & -3 & 0 & 1 \\ 6 & 1 & -6 & -5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -2 \\ -7 \\ 6 \end{pmatrix}$$

con el método de eliminación gaussiana, el programa tratará de dividir entre cero para calcular los coeficientes l_{j0}, y el método no se puede completar.

Para resolver este problema hay que cambiar el orden de las ecuaciones. Esto equivale a intercambiar renglones en la matriz aumentada, es decir en **A** y en *b*. El método de pivote parcial consiste en buscar, en la columna en la que se está aplicando la eliminación, el elemento con valor absoluto más grande. A este elemento se le denomina pivote. La busqueda sólo se realiza entre los elementos de la diagonal y los que están abajo de ésta . Ya que se ha localizado el pivote, se intercambian los renglones.

En el ejemplo, para realizar la eliminación bajo a_{00}, el pivote será el máximo del conjunto { $|a_{00}|$, $|a_{10}|$, ... , $|a_{n-1,0}|$ }, en este caso el 6 del último renglón, por tanto, se intercambian los renglones primero y sexto.

Ahora ya se puede aplicar la eliminación bajo a_{00}. Este procedimiento se repite para a_{11}, ..., $a_{n-2,n-2}$. Cuando queda el último renglón, este sólo tiene un elemento, el $a_{n-1,n-1}$.

Al finalizar la eliminación se tiene una matriz triangular, la solución se obtiene aplicando la sustitución en reversa.

```
/* Eliminacion gaussiana con pivote parcial */
...
for( d = 0; d < nren - 1; d++)
{
  piv = pivote( a, nren, ncol, d);
  if( piv != d )
    ec_inter(a, nren, ncol, b, nren, ncolb, d, piv);
  /* continua la eliminacion */
  ...
}

/* Posicion del pivote */
int pivote( double m[REN][COL], int nren, int ncol, int d)
{
  ...
  piv = fabs(m[d][d]);
  ipiv = d;
  for( r = d + 1; r < nren; r++)
  {
    x = fabs(m[r][d]);
    if( x > piv )
    {
      piv = x;
      ipiv = r;
    }
  }
  return ipiv;
}

/* Intercambio de ecuaciones */
void ec_inter(double mA[REN][COL], int rA, int cA,
              double mB[REN][COL], int rB, int cB,
              int renj, int renk)
{
  mat_inter_ren(mA, rA, cA, renj, renk);
  mat_inter_ren(mB, rB, cB, renj, renk);
}

/* Intercambio de dos renglones (j, k) */
void mat_inter_ren(double m[REN][COL], int nren, int ncol, int j, int k)
{
  ...
  for( c = 0 ; c < ncol ; c++ )
  {
    x = m[j][c];
    m[j][c] = m[k][c];
    m[k][c] = x;
  }
}
```

Sesión práctica

- Modifique el método de eliminación para incluir matrices de términos independientes
- Escriba un programa para implementar la eliminación con pivote parcial
- Pruebe sus programas con los ejercicios de la sesión anterior
- Resuelva el sistema de ecuaciones que se presenta en esta sesión con el método de eliminación con pivote parcial
- Para la matriz de 3 x 3 de la sesión anterior, obtenga su inversa. Verifique que el producto de ambas coincide con la matriz identidad.

Sesión #17

Proyecto #3: Balanceo de reacciones

Método algebráico

Para balancear una reacción química se debe resolver un sistema homogéneo de ecuaciones lineales. Los métodos descritos en las sesiones previas pueden adaptarse para resolver este tipo de problemas.

La reacción siguiente se utilizar para reducir minerales de hierro,

$$a\,\mathrm{Fe_2O_3}(s) + b\,\mathrm{CO}(g) \longrightarrow c\,\mathrm{CO_2}(g) + d\,\mathrm{Fe}(s)\,.$$

En el método algebráico, para cada uno de los elementos presentes en la reacción, se escribe una igualdad entre el número de átomos presentes en cada lado de la reacción:

$$\begin{array}{lll} \text{Fe:} & 2a = d & 2a + 0b + 0c - 1d = 0 \\ \text{O:} & 3a + b = 2c & 3a + 1b - 2c + 0d = 0\,. \\ \text{C:} & b = c & 0a + 1b - 1c + 0d = 0 \end{array}$$

En este caso se obtiene un sistema homogéneo de tres ecuaciones con cuatro incógnitas, por lo que el sistema tiene, al menos, un grado de libertad.

Para resolver el sistema de ecuaciones se utiliza el método de eliminación gaussiana con pivote parcial. **Observe que este método se aplicará a una matriz que no es cuadrada.** Para empezar, se escribe el sistema de ecuaciones en notación matricial,

$$\mathbf{A}\vec{x} = \begin{pmatrix} 3 & 1 & -2 & 0 \\ 2 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \vec{0}\,,$$

en donde ya se ha realizado el primer intercambio de columnas para llevar al primer pivote a la diagonal. A continuación se muestran los pasos que sigue el método de eliminación ,

$$\begin{pmatrix} 3 & 1 & -2 & 0 \\ 2 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \longrightarrow -\tfrac{2}{3} \begin{pmatrix} 3 & 1 & -2 & 0 \\ 0 & -\tfrac{2}{3} & \tfrac{4}{3} & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 3 & 1 & -2 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -\tfrac{2}{3} & \tfrac{4}{3} & -1 \end{pmatrix} \longrightarrow \tfrac{2}{3} \begin{pmatrix} 3 & 1 & -2 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & \tfrac{2}{3} & -1 \end{pmatrix}.$$

El sistema de ecuaciones resultante puede escribirse en la forma

$$\begin{aligned} 3a + b - 2c &= 0 \\ b - c &= 0\,, \\ \tfrac{2}{3}c &= d \end{aligned}$$

en donde d es arbitraria. Resolviendo por sustitución en reversa se obtiene

$$\vec{x} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} \frac{1}{2}d \\ \frac{3}{2}d \\ \frac{3}{2}d \\ d \end{pmatrix} = d\begin{pmatrix} \frac{1}{2} \\ \frac{3}{2} \\ \frac{3}{2} \\ 1 \end{pmatrix} \quad , \qquad \vec{x}\big|_{d=2} = \begin{pmatrix} 1 \\ 3 \\ 3 \\ 2 \end{pmatrix} \quad .$$

En general, se acostumbra eligir aquel valor de d que genera el conjunto de coeficientes enteros de valor más pequeño, por lo que la reacción toma la forma,

$$\mathrm{Fe_2O_3}(s) + 3\mathrm{CO}(g) \longrightarrow 3\mathrm{CO_2}(g) + 2\,\mathrm{Fe}(s) \, .$$

Matriz de composición

Existe una forma alternativa para generar la matriz del sistema de ecuaciones. Sea *n* el número total de elementos que aparecen en la reacción (en este caso, *n* = 3) y *m* el número total de especies (*m* = 4). La matriz de composición es una matriz de *n* x *m*, en donde el elemento a_{ij} representa el número de atómos del elemento *i* que hay en la especie *j*:

	0	1	2	3
	Fe_2O_3	CO	CO_2	Fe
O	3	1	2	0
Fe	2	0	0	1
C	0	1	1	0

.

Observe que esta matriz es muy parecidad a la matriz del sistema de ecuaciones homogéneo que se construyó previamente. Las dos últimas columnas de la matriz de composición (que corresponden a las especies que son productos de la reacción) tienen signo contrario al que aparece en la matriz **A**, por lo que en la solución del sistema de ecuaciones habrá elementos positivos (reactivos) y negativos (productos). Al aplicar el método de eliminación se obtiene la matriz siguiente,

$$\begin{pmatrix} 3 & 1 & 2 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 3 & 1 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & -\frac{2}{3} & 1 \end{pmatrix},$$

que corresponde al sistema de ecuaciones

$$\begin{aligned} 3\nu_0 + \nu_1 + 2\nu_2 &= 0 \\ \nu_1 + \nu_2 &= 0 \\ -\tfrac{2}{3}\nu_2 &= -\nu_3 \end{aligned} \quad ,$$

en donde la variable ν_3 es arbitraria. Si se elige $\nu_3 = -1$, entonces la matriz aumentada de este sistema de ecuaciones es idéntica a la matriz que se obtiene por el método de eliminación,

$$\left[\begin{array}{ccc|c} 3 & 1 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & -\frac{2}{3} & 1 \end{array}\right].$$

Por lo tanto, la sustitución en reversa se aplica directamente a la matriz que resulta de la eliminación, **tomando en cuenta que, en lugar de tener un vector *b* de términos independientes, se usa la última columna de la matriz**. Al resolver el sistema se obtiene $\nu_2 = -3/2$, $\nu_1 = 3/2$ y $\nu_0 = 1/2$, y el vector de soluciones contiene elementos uno y otro signo,

$$\vec{x} \equiv \begin{pmatrix} \nu_0 \\ \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 3/2 \\ -3/2 \\ -1 \end{pmatrix},$$

los positivos son los reactivos, mientras que los negativos corresponden a los productos. Se puede observar que este vector es similar al obtenido previamente, por lo que la reacción balanceada es la misma.

Cuando hay especies cargadas se debe añadir la ecuación de conservación de carga. Esto se realiza agregando un renglón que incluya la carga de cada especie.

Por ejemplo, para la reacción iónica $C(s) + ClO_2(g) + OH^-(ac) \rightarrow ClO_2^-(ac) + CO_3^{2-}(ac) + H_2O(l)$, la matriz de composición es

	ClO_2^-	CO_3^{2-}	H_2O	C	ClO_2	OH^-
Cl	1	0	0	0	1	0
O	2	3	1	0	2	1
C	0	1	0	1	0	0
H	0	0	2	0	0	1
q	−1	−2	0	0	0	−1

Algoritmo

1. Numere las especies que aparecen en la reacción.

   Sea m el número de especies, X_j representa la j-ésima especie, q_j su carga y $j = 0, 1, ..., m - 1$.

   En el ejemplo, $m = 4$, y, para fines demostrativos, la numeración empieza por los productos,

j	X_j	q_j
0	CO_2	0
1	Fe	0
2	Fe_2O_3	0
3	CO	0

2. Construya la matriz de composición (n x m).

	CO_2	Fe	Fe_2O_3	CO
C	1	0	0	1
O	2	0	3	1
Fe	0	1	2	0

3. Aplique el método de eliminación a la matriz de composición.

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 2 & 0 & 3 & 1 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & -3/2 & 1/2 \end{pmatrix}$$

4. Obtenga la solución aplicando la sustitución en reversa, considerando que $\nu_{m-1} = -1$.

$$\begin{pmatrix} \nu_0 \\ \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2/3 \\ -1/3 \\ -1 \end{pmatrix}$$

5. Utilice un múltiplo del vector solución para trabajar con coeficientes enteros y escriba la ecuación balanceada. Tome en cuenta que las especies con coeficiente positivo aparecen de un lado de la reacción, mientras que las que tienen coeficiente negativo deben estar del otro lado.

$$\mathrm{Fe_2O_3}(s) + 3\,\mathrm{CO}(g) \longrightarrow 3\,\mathrm{CO_2}(g) + 2\,\mathrm{Fe}(s)$$

Casos especiales

Al tratar este problema pueden aparecer algunos casos especiales.

1) Después de aplicar el método de eliminación aparece una línea con ceros en las primeras *m* columnas y un valor diferente de cero, *a*, en la última columna,

$$\left(\begin{array}{ccc|c} & \vdots & & \vdots \\ 0 & \cdots & 0 & a \end{array}\right).$$

La ecuación asociada con esta línea es $0 = a\,\nu_{m-1}$. Así, $\nu_{m-1} = 0$. Dado que todos los coeficientes son proporcionales a ν_{m-1}, entonces todos son cero y esto indica que no existe ninguna reacción balanceada entre estas especies. En este caso se debe revisar si falta alguna especie, o bien si hubo algún error en la matriz de composición.

2) La matriz de composición tiene *n* renglones x (*n* + *k*) columnas, con $k > 1$ (hay *k* grados de libertad). En este caso las primeras *n* columnas (de la columna 0 a la columna n -1) generan la matriz triangular (matriz **A**), y las últimas *k* columnas (de la columna *n* a la columna $n + k$ -1) juegan el papel de una matriz de términos independientes (matriz **B**). El problema se resuelve como un problema matricial, [**A**|**B**] y aparecen *k* vectores solución.

Usando el método descrito previamente, para cada columna de **B** se obtiene un vector solución. Dado que los coeficientes ν para las últimas *k* columnas son arbitrarios, la solución que se obtiene al usar la columna *j* (para $n <= j <= n + k - 1$) corresponde a $\nu_j = -1$, con los restantes ν iguales a cero. Con cada vector se obtiene una reacción balanceada.

En este proceso, las reacciones obtenidas dependen de la numeración. Por ejemplo, si se toman los compuestos CH_4, O_2, H_2O, CO_2, CO, se obtienen las reacciones:

$$CH_4 + 2\,O_2 \rightarrow 2\,H_2O + CO_2 \quad , \qquad CH_4 + {}^3/_2\,O_2 \rightarrow 2\,H_2O + CO\,.$$

Mientras que si se toman en el orden CH_4, O_2, CO_2, H_2O, CO, las reacciones son:

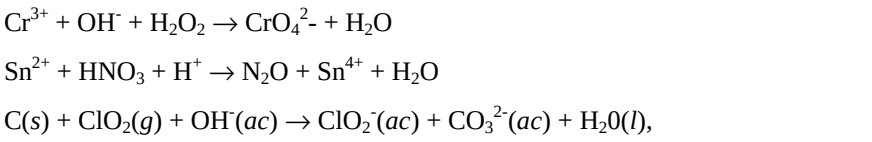
$$CH_4 + 2\,O_2 \rightarrow 2\,H_2O + CO_2 \quad , \qquad 2\,CO + O_2 \rightarrow 2\,CO_2\,.$$

Observe que si se restan las dos reacciones del primer caso se obtiene conversion de CO a CO_2, por lo que ambos casos son algebráicamente equivalentes.

3) Después de la eliminación aparece una o más líneas con todos sus elementos iguales a cero. Estas líneas no se toman en cuenta ya que corresponden a ecuaciones del tipo 0 = 0. Como el número de renglones es menor, el número de columnas que forman la matriz triangular disminuye y, por tanto, el número de columnas de la matriz **B** aumenta (hay más grados de libertad). Así, el problema se reduce al caso anterior.

Sesión práctica

- Modifique las rutinas de eliminación y sustitución para construir un programa que permita balancer reacciones químicas.
- Balancee las reaciones



$$Cr^{3+} + OH^- + H_2O_2 \rightarrow CrO_4^{2}- + H_2O$$

$$Sn^{2+} + HNO_3 + H^+ \rightarrow N_2O + Sn^{4+} + H_2O$$

$$C(s) + ClO_2(g) + OH^-(ac) \rightarrow ClO_2^-(ac) + CO_3^{2-}(ac) + H_20(l),$$

Sesión #18

Apuntadores

En el lenguaje C, los apuntadores son objetos que almacenan la dirección de memoria de alguna variable.

Declaración

Al igual que otros objetos, los apuntadores deben declararse antes de usarlos. Dado que un apuntador almancena la dirección de memoria en donde reside una variable, en la declaración se debe especificar el tipo de variable a la cual *apunta* el apuntador. Para indicar que el objeto que se está declarando es un apuntador, un asterisco debe estar al inicio del nombre, pero solamente en la declaración.

```
/* declaracion de apuntadores */
int  *intptr;
char  *chptr;
double *dprt;
int  *p1, p2;
```

En el ejemplo anterior, intptr y p1 son apuntadores de variables enteras, chptr es un apuntador de variables char y dptr es un apuntador de variables tipo double. Como se observa en la última línea, se pueden declarar conjuntamente variables y apuntadores, siempre y cuando sean del mismo tipo.

Operaciones básicas

Para obtener la dirección de memoria de una variable se usa el operador & (dirección de). De esta forma, mediante una asignación, un apuntador puede contener la dirección de memoria de una variable.

```
/* asignacion de una direccion de memoria */
int xx, yy, zz, *p1, *p2;
xx = 128;
yy = 4;
p1 = &xx;
p2 = &yy;
```

En este caso, la variable xx contiene el valor 128, mientras que p1 contiene la dirección de memoria de la variable xx. Se acostumbra decir que p1 *apunta a la variable* xx. De la misma forma p2 apunta a yy.

```
/* otra asignacion */
p2 = p1;
```

Después de esta instrucción, el contenido de p2 es igual al de p1, por lo que ahora p2 también apunta a xx.

Para referirse al objeto al que apunta un apuntador se usa el operador *.

```
/* uso de la información del apuntador */
zz = *p1 / yy;
p2 = &yy;
*p2 = -1;
*p1 = *p2;
```

En este caso, como p1 apunta a xx, zz toma al valor 32. Nuevamente p2 apunta a yy. En la tercera instrucción, se guarda -1 en la variable apuntada por p2, por lo que yy contiene el valor -1. En la última línea, se copia, en la variable apuntada por p1, el valor de la variable apuntada por p2; esto es, el contenido de yy se copia en xx.

Uso de apuntadores en una función

Excepto por los arreglos, una función no puede modificar el contenido de una variable que se usa como argumento. Para que una función pueda cambiar el contenido de una variable es necesario enviarle la

dirección de memoria. En este caso, la función puede escribir directamente en la memoria y modificar el contenido.

Es por este mecanismo que una función si modifica el contenido de los arreglos. El nombre de un arreglo contiene la dirección de memoria del primer elemento, sin embargo no contiene información acerca del tamaño del arreglo, así que es necesario enviar también información acerca del número de elementos.

Un ejemplo de una función que modifica a un argumento es la función scanf,

```
scanf("%d", &z);
```

Esta instrucción lee un valor del teclado y lo guarda en la variable z. Por lo tanto modifica el contenido. Observe que el segundo argumento es una dirección de memoria y la función escribe directamente en la memoria.

Cuando una función requiere de un apuntador como argumento, en el prototipo y en la definición se usa un asterisco, al igual que al declarar un apuntador. Al llamar a la función, el argumento que corresponde al apuntador debe ser una dirección de memoria, por lo que se usa el operador &.

```
/* uso de apuntadores en una funcion */
...
void inter_int(int *x, int *y);
...
main()
{
  int a, b;
  ...
  inter_int(&a, &b);
  ...
}
...
/* esta funcion intercambia el contenido de dos enteros */
void inter_int(int *x, int *y)
{
  int tmp;
  tmp =  *x;
  *x  =  *y;
  *y  = tmp;
}
```

Observe que dentro de la función se usa el operador que permite accesar el contenido de la variable apuntada.

En general se recomienda modificar sólo aquellas variables que son estrictamente necesarias y, cuando sea posible, es preferible usar funciones que regresan un valor.

Sesión práctica

En la sesión #7 se escribió un programa para resolver una ecuación cuadrática, tomando en cuenta todos los valores posibles de los tres coeficientes de la ecuación.

Hay casos en que la ecuación no se satisface, la ecuación puede reducirse a una ecuación lineal y tiene una sola raíz, puede tener raíces complejas, etc.

Una función que resuelva una ecuación cuadrática y que tome en cuenta todas las posibilidades requiere del uso de apuntadores, ya que es necesario conocer el tipo de solución y las raíces. Es decir, se requieren hasta tres cantidades: una variable de control que indique el tipo de problema y hasta dos soluciones.

La variable de control puede ser un entero y conviene que éste sea el valor que es regresado por la función. Las variables que contengan el valor de las raíces serán argumentos de la función. Por ejemplo esta función puede tener el prototipo siguiente:

```
int cuadratica(double a, double b, double c, double *r1, double *r2);
```

- Escriba una función que resuelva una ecuación cuadrática. Dependiendo de la situación, la variable de control podrá tomar los valores siguientes:

-1	la ecuación es una identidad y hay un número infinito de soluciones
0	la ecuación es inconsistente y no existe ninguna solución
1	la ecuación es lineal y tiene una sola raíz
2	hay dos raíces reales y diferentes
3	hay una raíz doble
4	las raíces son complejas, la parte real está en r1 y la parte imaginaria en r2

- Inserte la función en un programa y, dentro de main(), utilice la estructura switch con la variable de control para que en cada caso se imprima el mensaje correcto.

Sesión #19

Ajuste de una curva

Considere el siguiente problema. A través de un modelo se sabe que el comportamiento de un conjunto de datos debe estar representado por una relación matemática y sólo hace falta determinar los parámetros de la ecuación.

Por ejemplo, cuando preparan las soluciones para una curva patrón, la absorbancia (A) debe ser proporcional a la concentración de la especie que absorbe la radiación (C). En este caso, los datos debe cumplir con una relación de la forma $A = mC + b$, en donde A y C son datos que provienen de las mediciones y m y b son los parámetros de la ecuación, en este caso la pendiente y la ordenada al origen de la recta. Al analizar la presión atmosférica, como función de la altura sobre el nivel del mar, el modelo más simple predice un decrecimiento exponencial en la presión (ley barométrica). La ecuación del modelo puede tener muchos parámetros, como algunas ecuaciones de estado, y no tiene que ser un modelo lineal.

Se le denomina ajuste de una curva al proceso por el cual se determinan los párametros de una ecuación modelo. En general, el método más utilizado es de mínimos cuadrados.

Método de mínimos cuadrados

Suponga que las variables x y y satisfacen la ecuación $y = f(x; a_0, a_1, ..., a_{n-1})$, en donde $a_0, a_1, ..., a_{n-1}$ son los parámetros de la ecuación. Por ejemplo, en el caso de la curva patrón, los parámetros son $b = a_0$ y $m = a_1$, mientras que y representa la absorbancia y x es la concentración.

Para un conjunto de N datos, (x_i, y_i), el error cuadrático se define como las suma de las desviaciones al cuadrado,

$$S = \sum_{i=o}^{N-1} \left[y_i - f\left(x_i; a_0, a_1, \ldots\right) \right]^2 .$$

En este método, el valor de los parámetros es aquel que minimiza el error cuadrático. Por lo tanto, para cada parámetro se tiene una ecuación de la forma

$$0 = \frac{\partial S}{\partial a_k} = 2\sum_i \frac{\partial f\left(x_i; a_0, a_1, \ldots, a_{n-1}\right)}{\partial a_k} \left[y_i - f\left(x_i; a_0, a_1, \ldots, a_{n-1}\right) \right], \quad (k = 0,1,\ldots, n-1) .$$

La solución del sistema de n ecuaciones con n incógnitas permite obtener los parámetros del ajuste. Es importante hacer notar que dependiendo de la forma de la función $f(x; a_0, a_1, ...)$ las ecuaciones pueden ser lineales o no lineales. Para cada caso se necesita un método numérico diferente.

Ya que se ha calculado el valor de los parámetros, la ecuación que relaciona a y con x queda totalmente determinada. Una medida del error en el ajuste es la desviación cuadrática promedio, la cual es simplemente $\sigma^2 = S / N$. Para un buen ajuste al modelo, esta cantidad debe ser pequeña.

Ajuste lineal

Cuando la ecuación del modelo es una recta, se tienen dos parámetros, $f(x; a_0, a_1) = a_0 + a_1 x$, por lo que las ecuaciones que minimizan el error cuadrático son

$$0 = \frac{\partial S}{\partial a_0} = 2\sum_i \left[y_i - a_0 - a_1 x_i \right] = 2\left(\Sigma y_i - a_0 N - a_1 \Sigma x_i \right)$$
$$0 = \frac{\partial S}{\partial a_1} = 2\sum_i x_i \left[y_i - a_0 - a_1 x_i \right] = 2\left(\Sigma x_i y_i - a_0 \Sigma x_i - a_1 \Sigma x_i^2 \right)$$

En este caso se obtiene un sistema de ecuaciones lineales,

$$\begin{pmatrix} N & \Sigma x_i \\ \Sigma x_i & \Sigma x_i^2 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \Sigma y_i \\ \Sigma x_i y_i \end{pmatrix},$$

con la solución siguiente,

$$\begin{aligned} a_1 &= \frac{N\,\Sigma x_i y_i - \Sigma x_i\,\Sigma y_i}{N\,\Sigma x_i^2 - \left(\Sigma x_i\right)^2} \\ a_0 &= \frac{\Sigma y_i - a_1\,\Sigma x_i}{N} \end{aligned}.$$

Estas ecuaciones permiten calcular los parámetros de un ajuste lineal por mínimos cuadrados.

Cuando estas ecuaciones se aplican al problema de determinar la ecuación de una curva patrón, la pendiente a_1 está relacionada con el coeficiente de absorción y la ordenada al origen a_0 debe ser pequeña. Con ambos parámetros se obtiene una ecuación para la recta, $y = a_0 + a_1x$, y con ella se pueden determinar las concentraciones de la muestras, ya que la absorbancia es conocida.

Modelo de una recta que pasa por el origen

Cuando la recta debe pasar por el origen, la pendiente es el único parámetro y $f(x; a_1) = a_1x$, por lo que a_1 está dado por la ecuación

$$a_1 = \frac{\Sigma x_i y_i}{\Sigma x_i^2}.$$

Combinación lineal de funciones

La ecuación del modelo puede ser una combinación lineal de funciones,

$$f\left(x; a_0, a_1, \ldots, a_{n-1}\right) = a_0 g_0(x) + a_1 g_1(x) + \cdots + a_{n-1} g_{n-1}(x) = \sum_{p=0}^{n-1} a_p g_p(x),$$

por ejemplo,

$$f(x; a, b, c) = a + bx + c\log(x),$$

en donde $g_0(x) = 1$, $g_1(x) = x$, y $g_2(x) = \log(x)$.

Para este modelo se obtiene el sistema de ecuaciones lineales

$$\mathbf{A}\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} \sum_i g_0(x_i) y_i \\ \sum_i g_1(x_i) y_i \\ \vdots \\ \sum_i g_{n-1}(x_i) y_i \end{pmatrix},$$

en donde la matriz del problema está dada por

$$\mathbf{A} = \begin{pmatrix} \sum_i g_0(x_i)g_0(x_i) & \sum_i g_0(x_i)g_1(x_i) & \cdots & \sum_i g_0(x_i)g_{n-1}(x_i) \\ \sum_i g_1(x_i)g_0(x_i) & \sum_i g_1(x_i)g_1(x_i) & \cdots & \sum_i g_1(x_i)g_{n-1}(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_i g_{n-1}(x_i)g_0(x_i) & \sum_i g_{n-1}(x_i)g_1(x_i) & \cdots & \sum_i g_{n-1}(x_i)g_{n-1}(x_i) \end{pmatrix}.$$

Dado que, tanto las funciones g_i, como los puntos (x_i, y_i) son conocidos, todos los elementos de la matriz **A** pueden calcularse y el sistema de ecuaciones se puede resolver con algún método numérico, como puede ser la eliminación gaussiana.

Para el ejemplo propuesto, el sistema de ecuaciones toma la forma

$$\begin{pmatrix} N & \sum_i x_i & \sum_i \log(x_i) \\ \sum_i x_i & \sum_i x_i^2 & \sum_i x_i \log(x_i) \\ \sum_i \log(x_i) & \sum_i x_i \log(x_i) & \sum_i \log^2(x_i) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \sum_i y_i \\ \sum_i y_i x_i \\ \sum_i y_i \log(x_i) \end{pmatrix}.$$

En general, la matrix **A** se puede obtener como un producto matricial, $\mathbf{A} = \mathbf{G}\,\mathbf{G}^{\mathrm{T}}$, en donde

$$\mathbf{G} = \begin{pmatrix} g_0(x_0) & g_0(x_1) & \cdots & g_0(x_{N-1}) \\ g_1(x_0) & g_1(x_1) & \cdots & g_1(x_{N-1}) \\ \vdots & \vdots & \ddots & \vdots \\ g_{n-1}(x_0) & g_{n-1}(x_1) & \cdots & g_{n-1}(x_{N-1}) \end{pmatrix}.$$

Por este método las sumas de la matriz **A** son calculadas en la multiplicación matricial, y no explícitamente,

$$A_{ij} = \sum_{k=0}^{N-1} G_{ik} G_{kj}^T = \sum_{k=0}^{N-1} G_{ik} G_{jk} = \sum_{k=0}^{N-1} g_i(x_k) g_j(x_k).$$

De la misma forma, el vector de términos independientes es un producto matricial, $\mathbf{b} = \mathbf{G}\,\mathbf{y}$, en donde

$$\mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix}.$$

Ajuste polinomial

Para un modelo polinomial, $y = a_0 + a_1x + a_2x^2 ... + a_{n-1}x^{n-1}$, por lo que $g_i(x) = x^i$. Para este caso, la matriz **G** tiene sólo potencias de x,

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{n-1} & x_1^{n-1} & \cdots & x_{N-1}^{n-1} \end{pmatrix}.$$

Con esta matriz se construyen **A** y **b** y se puede proceder a resolver el sistema de ecuaciones.

Modelos no lineales

Cuando los parámetros aparecen en forma no lineal se obtiene un sistema de ecuaciones no lineales. Este problema se puede resolver usando una generalización del método de Newton. Por ejemplo, la función

$$f(x;a,b,c) = a + b\exp(cx)$$

tiene un parámetro no lineal y los parámetros son solución del sistema de ecuaciones no lineales

$$\begin{pmatrix} aN + b\Sigma e^{cx_i} - \Sigma y_i \\ a\Sigma e^{cx_i} + b\Sigma e^{2cx_i} - \Sigma y_i e^{cx_i} \\ a\Sigma x_i e^{cx_i} + b\Sigma x_i e^{2cx_i} - \Sigma y_i x_i e^{cx_i} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Algunas funciones pueden linealizarse mediante una transformación, por ejemplo, la ecuación

$$Y = 1 + ab^X$$

se transforma en una ecuación lineal al aplicar la función logaritmo,

$$y = a_0 + a_1 x\,, \quad y = \log(Y-1)\,, \quad x = X\,, \quad a_0 = \log(a)\,, \quad a_1 = \log(b)\,.$$

Cuando la ecuación no puede linealizarse, como la función del primer ejemplo, se obtendrá un sistema de ecuaciones no lineal.

Sesión práctica

Escriba un programa que lea los datos de una curva patrón y que calcule los parámetros del ajuste. El programa también deberá calcular las concentraciones de un conjunto de muestras con absorbancia conocida.

Sesión #20

Ecuaciones diferenciales de primer orden

Una ecuación diferencial de primer orden es una ecuación que relaciona a la función $y(x)$ con su derivada, $y'(x)$. Dada la condición, $y(x_0) = y_0$, el objetivo es encontrar a la función que satisface la ecuación

$$y' = f\big(x, y(x)\big),$$

y que cumple con la condición inicial. Al integrar la ecuación anterior se obtiene una relación integral para la función $y(x)$,

$$y(x) = y(x_0) + \int_{x_0}^{x} f\big(x, y(x)\big)dx\,.$$

Como $y(x)$ es desconocida, los métodos numéricos para resolver ecuaciones diferenciales se basan en diferentes aproximaciones de esta última ecuación.

Método de Euler

El método de Euler consiste en suponer que, en la región de integración, el integrando se mantiene constante e igual a su valor inicial. Así,

$$y(x_0 + h) \approx y(x_0) + \int_{x_0}^{x_0+h} f\big(x_0, y(x_0)\big)dx = y(x_0) + hf(x_0, y_0)\,.$$

Para estimar el error de esta aproximación es necesario desarrollar el integrando en series de Taylor, alrededor $x = x_0$,

$$\begin{aligned} f\big(x_0 + h, y(x_0 + h)\big) &= f(x_0, y_0) + h\left[\frac{\partial f}{\partial x} + \left(\frac{\partial f}{\partial y}y'\right)\right]_{x_0} + \cdots, \\ &= f(x_0, y_0) + O(h) \end{aligned}$$

por lo que al integrar se tiene que

$$y(x_0 + h) = y(x_0) + hf(x_0, y_0) + O\big(h^2\big)\,.$$

Esto es, al evaluar la función en $x_0 + h$ con el método de Euler, el error es del orden de h^2, así que para valores muy pequeños de h, se espera que el error sea mucho más pequeño que los dos primeros términos.

Dado que en general se resuelve para la función $y(x)$ en un intervalo $[a, b]$, se elige un valor de h y se aplica la aproximación tantas veces como sea necesario para alcanzar el punto final del intervalo. Por lo tanto, si $x_0 = a$, entonces $b = x_0 + n\,h = a + n\,h$, o bien, para una partición uniforme de n elementos del intervalo $[a, b]$, el valor de h está dado por

$$h = \frac{b - a}{n}\,.$$

Al aplicar n veces la aproximación, el error acumulado resulta ser $n\,O(h^2)$, pero como n y h tiene una relación inversa, el error global del método es lineal en h, $O(h)$. Este comportamiento ocasiona que, para alcanzar la precisión deseada, se requieran de valores muy pequeños de h y por tanto de valores muy grandes para n.

El método de Euler se puede implementar muy fácilmente y una propuesta se presenta a continuación.

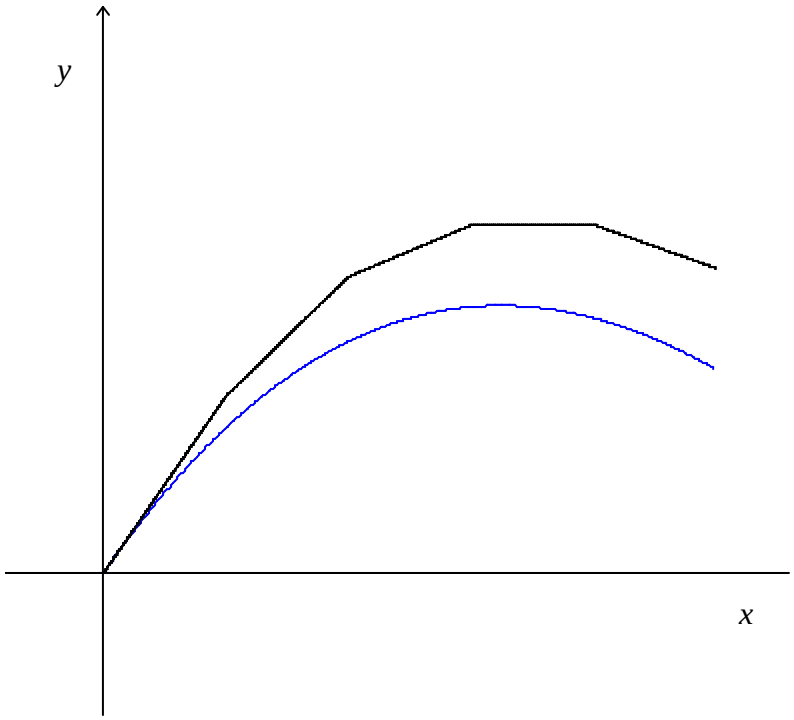
```
/* Solucion numerica de una ecuacion diferencial usando el metodo de Euler */
#include <stdio.h>
#include <math.h>
#define REN 100

double f(double x, double y);
double yeuler(double xo, double yo, double h);
void solucion(double x[REN], double y[REN], int n, double x0, double y0, double h);
void sol_escribir(double x[REN], double y[REN], int n);

main()
{
  int n;
  double a, b, h, fa;
  double x[REN], y[REN];

  printf("\nSolucion de una ecuacion diferencial con el metodo de Euler.\n\n");
  printf("x inicial: ");
  scanf ("%lf", &a);
  printf("x final: ");
  scanf ("%lf", &b);
  printf("y inicial: ");
  scanf ("%lf", &fa);
  printf("numero de intervalos: ");
  scanf ("%d", &n);

  h = (b - a) / n;
  solucion(x, y, n + 1, a, fa, h);
  sol_escribir(x, y, n + 1);
}

/* Rutina que construye la solucion numerica */

void solucion(double vx[REN], double vy[REN], int n,
              double x0, double y0, double h)
{
  int i;
  double x, y;

  x = x0;
  y = y0;
  vx[0] = x;
  vy[0] = y;

  for( i = 1; i < n; i++)
  {
    y = yeuler(x, y, h);
    x = x0 + h * i;
    vx[i] = x;
    vy[i] = y;
  }
}

/* Metodo de Euler */

double yeuler(double xo, double yo, double h)
{
  return yo + h * f(xo, yo);
}
```

Observe que no está incluida la función $f(x, y)$.

El método de Euler también corresponde a un desarrollo en series de Taylor de la función $y(x)$, alrededor de $x = x_0$,

$$y(x_0 + h) = y(x_0) + hy'(x_0) + O(h^2),$$

por lo que la función y se aproxima por su recta tangente.



Método de Euler

Método de Euler mejorado

Una mejor aproximación se obtiene usando el teorema del valor medio en la ecuación integral que permite evaluar a $y(x_0 + h)$,

$$y(x_0 + h) = y(x_0) + hf(\xi, y(\xi)) \approx y(x_0) + hf\left(x_0 + \frac{h}{2}, y\left(x_0 + \frac{h}{2}\right)\right),$$

en donde $\xi \in [x_0, x_0 + h]$. Observe que esta expresión es similar a la aproximación rectangular en la integración numérica.

Debido a que función $y(x)$ no es conocida, su valor en $x = x_0 + ½\ h$ se extrapola usando el método de Euler tradicional,

$$y(x_0 + \tfrac{1}{2}h) \approx y(x_0) + \tfrac{1}{2}hf(x_0, y_0).$$

Este procedimiento se puede describir en la forma siguiente,

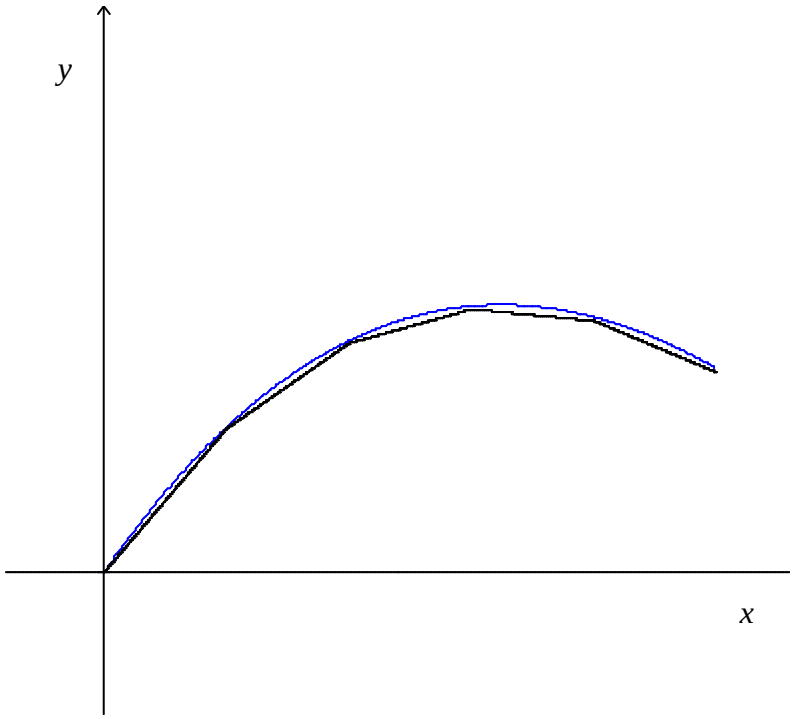
$$y(x_0 + h) \approx y(x_0) + k_2$$

$$k_2 = hf(x_0 + \tfrac{1}{2}h, y_0 + \tfrac{1}{2}k_1).$$

$$k_1 = hf(x_0, y_0)$$

El método resultante presenta un error global cuadrático en *h*, $O(h^2)$, y esta es la razón por la cual se le denomina mejorado. Cada vez que se aplica este método, la función $f(x, y)$ debe evaluarse dos veces, a diferencia del método de Euler que sólo requiere de una evualuación, por lo que el número de operaciones realizadas también se duplica.

```
/* Metodo de Euler modificado */

double yeulerm(double xo, double yo, double h)
{
  double k1, k2;

  k1 = h * f(xo, yo);
  k2 = h * f(xo + 0.5 * h, yo + 0.5 * k1);
  return yo + k2;
}
```



Método de Euler mejorado

Método de Runge-Kutta de cuarto orden

Uno de los métodos más utilizados es el método de Runge-Kutta de cuarto orden. Como su nombre lo indica, el error global es de cuarto orden, $O(h^4)$. Este método requiere de cuatro evaluaciones de la función $f(x, y)$, como se muestra a continuación,

$$y(x_0 + h) \approx y(x_0) + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = hf(x_0, y_0)$$

$$k_2 = hf(x_0 + \tfrac{1}{2}h, y_0 + \tfrac{1}{2}k_1)$$

$$k_3 = hf(x_0 + \tfrac{1}{2}h, y_0 + \tfrac{1}{2}k_2)$$

$$k_4 = hf(x_0 + h, y_0 + k_3)$$

```
/* Metodo de Runge Kutta de 4o orden */

double yrk4(double xo, double yo, double h)
{
  double k1, k2, k3, k4;
```

```
  k1 = h * f(xo, yo);
  k2 = h * f(xo + 0.5 * h, yo + 0.5 * k1);
  k3 = h * f(xo + 0.5 * h, yo + 0.5 * k2);
  k4 = h * f(xo + h, yo + k3);
  return yo + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
}
```

Existen un procedimiento general para obtener aproximaciones de tipo Runge-Kutta con un error global de orden *m*. Los métodos de Euler y Euler mejorado pueden obtenerse por este procedimiento, y son casos particulares para $m = 1, 2$. Tanto la descripción del procedimiento como tabulaciones de los coeficientes involucrados para $m = 1, ..., 8$ pueden consultarse en:

G Engeln-Müllges and F Uhlig
Numerical Algorithms with C
Springer (1996)

Comparación de los métodos presentados

La ecuación diferencial lineal de primer orden

$$y' = f(x, y) = -2x - y,$$

con la condición inicial $y(0) = -1$, puede integrarse directamente y su solución es

$$y(x) = -3e^{-x} - 2x + 2.$$

A continuación se muestran los resultados que se obtienen al aplicar cada uno de los métodos descritos anteriormente. Se utilizan diferentes particiones homogéneas de los intervalos [0, 0.5] y [0, 1], únicamente se muestran los valores de la función $y(x)$ en el extremo final de cada intervalo.

```
METODO    n     EULER     EULER MOD   RUNGE KUT   EXACTO
-------------------------------------------------------
x = 0.5                                           -.819592
          1   -0.500000   -0.875000   -0.820312
          2   -0.687500   -0.831055   -0.819628
          4   -0.758545   -0.822196   -0.819594
          8   -0.790158   -0.820213   -0.819592
         16   -0.805131   -0.819744   -0.819592
         32   -0.812423   -0.819629   -0.819592
         64   -0.816023   -0.819601   -0.819592
-------------------------------------------------------
x = 1.0                                           -1.10364
          1    0.00000    -1.50000    -1.12500
          2   -0.75000    -1.17188    -1.10451
          4   -.949219    -1.11759    -1.10368
          8   -1.03083    -1.10680    -1.10364
         16   -1.06822    -1.10439    -1.10364
         32   -1.08617    -1.10382    -1.10364
         64   -1.09496    -1.10368    -1.10364
-------------------------------------------------------
```

Los resultados muestran que los métodos de mayor orden requieren de un número menor de etapas para satisfacer un mismo criterio de error. Al hacer una comparación se debe tomar en cuenta que los métodos de mayor orden también requieren de un mayor número de evaluaciones de la función $f(x, y)$, y por tanto de más operaciones. Aún considerando esta observación, los resultados obtenidos permiten concluir que el método de Runge-Kutta presenta un mejor desempeño.

Sesion práctica

- Escriba un programa para cada uno de los métodos presentados.
- Verifique las tendencias de la tabla de resultados.
- Resuelva numéricamente el problema de valor inicial $y' = x/y$, con $y(0) = 1$, en el intervalo [0, 1]. Compare los resultados de sus programas con la solución exacta.

Sesión #21

Sistemas de ecuaciones diferenciales de primer orden

Considere el sistema de *N* ecuaciones diferenciales de primer orden

$$\begin{aligned}
\frac{dy_0}{dx} &= f_0\left(x, y_0, \ldots, y_{N-1}\right) \\
\frac{dy_1}{dx} &= f_1\left(x, y_0, \ldots, y_{N-1}\right) \\
&\vdots \\
\frac{dy_{N-1}}{dx} &= f_{N-1}\left(x, y_0, \ldots, y_{N-1}\right)
\end{aligned},$$

en donde y_0, ..., y_{N-1} son funciones de x. Este sistema puede escribirse en notación vectorial,

$$\mathbf{y}' = \frac{d\mathbf{y}}{dx} = \mathbf{f}\left(x, \mathbf{y}\right),$$

en donde $\mathbf{y}$ y $\mathbf{f}$ son vectores definidos en la forma siguiente,

$$\mathbf{y} = \begin{pmatrix} y_0(x) \\ y_1(x) \\ \vdots \\ y_{N-1}(x) \end{pmatrix} \quad , \quad \mathbf{f} = \begin{pmatrix} f_0\left(x, \mathbf{y}\right) \\ f_1\left(x, \mathbf{y}\right) \\ \vdots \\ f_{N-1}\left(x, \mathbf{y}\right) \end{pmatrix}.$$

Los métodos descritos en la sección anterior pueden generalizarse fácilmente para los sistemas de ecuaciones diferenciales.

El método de Runge-Kutta para sistemas de ecuaciones consiste en las mismas expresiones presentadas anteriormente, pero tomando en cuenta que ahora $\mathbf{y}$ y $\mathbf{f}$ son vectores,

$$\begin{aligned}
\mathbf{y}(x+h) &\approx \mathbf{y}(x) + \tfrac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \\
\mathbf{k}_1 &= h\mathbf{f}\left(x, \mathbf{y}(x)\right) \\
\mathbf{k}_2 &= h\mathbf{f}\left(x + \tfrac{1}{2}h, \mathbf{y}(x) + \tfrac{1}{2}\mathbf{k}_1\right) \\
\mathbf{k}_3 &= h\mathbf{f}\left(x + \tfrac{1}{2}h, \mathbf{y}(x) + \tfrac{1}{2}\mathbf{k}_2\right) \\
\mathbf{k}_4 &= h\mathbf{f}\left(x + h, \mathbf{y}(x) + \mathbf{k}_3\right)
\end{aligned} \quad .$$

Observe que en este caso $\mathbf{k}_1$, $\mathbf{k}_2$, $\mathbf{k}_3$ y $\mathbf{k}_4$ también son vectores de *N* renglones.

A continuación se muestran algunas partes de una implementación de este método.

```
/*
 * Archivo: rk4_vec.c
 * ------------------
 * Solucion numerica de un sistema de ecuaciones diferenciales
 * de primer orden: y' = f(x, y) usando el metodo de Runge Kutta
 */

#include <stdio.h>
```

```
#include <math.h>

#define DIM 2
#define npasos 1024

void f_vec(double x, double vy[DIM], double vf[DIM], int d);
void solucion(double vx[npasos], double vy[DIM][npasos], int d, int n,
              double x0, double vyinicial[DIM], double h);
void rk4_vec(double xn, double yn[DIM], double vynew[DIM], int d, double h);
void sol_escribir(double x[npasos], double y[DIM][npasos], int d, int n);
void vec_leer(double v[DIM], int n);

main()
{
  int i, d, n;
  double h, a, b;
  double x[npasos], y[DIM][npasos], yini[DIM];

  ...

  d = DIM;
  h = (b - a) / n;

  solucion(x, y, d, n + 1, a, yini, h);
  sol_escribir(x, y, d, n + 1);
}

/*
 * Evaluacion de la funcion vectorial f(x, y), en donde y es un vector.
 *
 * En este problema: f[0] = y[1], f[1] = -y[0]
 * que equivale a la ecuacion diferencial de 2o orden: y'' = -y
 * El uso de la estructura switch evita que queden elementos sin definir
 * al modificar el programa.
 */

void f_vec(double x, double vy[DIM], double vf[DIM], int d)
{
  int i;

  for( i = 0; i < d; i++)
    switch(i)
    {
      case 0:
        vf[i] = vy[1];
        break;
      case 1:
        vf[i] = -vy[0];
        break;
      default:
        vf[i] = 0;
    }
}

/* Rutina que construye la solucion numerica */

void solucion(double vx[npasos], double vy[DIM][npasos], int d, int n,
              double x0, double vyinicial[DIM], double h)
{
  int i, j;
  double x, y[DIM], ynew[DIM];

  x = x0;
  for( i = 0; i < d; i++) y[i] = vyinicial[i];
  vx[0] = x;
  for( i = 0; i < d; i++) vy[i][0] = y[i];

  for( j = 1; j < n; j++)
  {
    rk4_vec(x, y, ynew, d, h);
    x = x0 + h * j;
    vx[j] = x;
    for( i = 0; i < d; i++) y[i] = ynew[i];
    for( i = 0; i < d; i++) vy[i][j] = y[i];
  }
}
```

Es importante mencionar que las funciones no pueden regresar un vector como resultado de una llamada, por lo que el vector de resultados debe ser un argumento. La mayoría de las operaciones vectoriales pueden realizarse con una estructura for de una sola línea. Se ha usado esta forma, aún cuando el programa sea más largo, para mostrar la similitud con los programas de la sección anterior.

Ecuaciones diferenciales de orden superior

Considere a la ecuación diferencial de orden *N*,

$$\frac{d^N z}{dx^N} = g\left(x, z, z', \ldots, z^{(N-1)}\right).$$

Si se definen las funciones

$$\begin{aligned} y_0(x) &= z(x) \\ y_1(x) &= z'(x) = \frac{d}{dx} z \\ &\vdots \\ y_{N-1}(x) &= z^{(N-1)}(x) = \left(\frac{d}{dx}\right)^{N-1} z \end{aligned} \quad , \quad \begin{aligned} f_0\left(x, y_0, \ldots, y_{N-1}\right) &= y_1 \\ f_1\left(x, y_0, \ldots, y_{N-1}\right) &= y_2 \\ &\vdots \\ f_{N-1}\left(x, y_0, \ldots, y_{N-1}\right) &= g\left(x, y_0, \ldots, y_{N-1}\right) \end{aligned} \quad ,$$

entonces la ecuación diferencial de orden *N* se puede representar por un sistema de *N* ecuaciones diferenciales de primer orden,

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}).$$

Así, las ecuaciones diferenciales de orden superior generalmente se resuelven como un sistema de ecuaciones.

Por ejemplo, la ecuación diferencial

$$\psi'' = -k^2 \psi,$$

en donde *k* es una constante positiva, tiene la representación vectorial

$$\begin{pmatrix} y_0' \\ y_1' \end{pmatrix} = \begin{pmatrix} y_1 \\ -k^2 y_0 \end{pmatrix}.$$

En este caso las soluciones son funciones trigonométricas.

Proyecto #4: Partícula encerrada en presencia de un campo de fuerzas

La descripción de un sistema microscópico está dada por la función de onda del sistema, $\psi(x)$. En general, para sistemas estacionarios, ésta se obtiene resolviendo la ecuación de Schroedinger independiente del tiempo,

$$\hat{H}\psi_n = E_n \psi_n,$$

en donde *H* es un operador diferencial de segundo orden y E_n es el valor propio asociado con ψ_n. Dado que se tiene una ecuación de valores propios, el valor propio, E_n, se obtiene junto con la solución del problema, y casi siempre está relacionado con las condiciones de frontera.

Considere un sistema unidimensional, en donde se tiene una partícula restringida a moverse solamente en el intervalo [0, 2*L*]. Para este sistema la función de onda satisface la ecuación diferencial

$$\frac{-\hbar^2}{2m}\psi'' + V(x)\psi = E\psi ,$$

en donde $V(x)$ es la función de la energía potencial, que depende de las fuerzas que actúan sobre la partícula. Si se hace el cambio de escala $r = x / 2L$, la ecuación puede escribirse en la forma

$$Y'' = \left[v(r) - \varepsilon\right]Y = K(r,\varepsilon)Y ,$$

en donde $Y(r) = \psi(x)$, $v(r) = V(x) / E_0$, $\varepsilon = E / E_0$, y

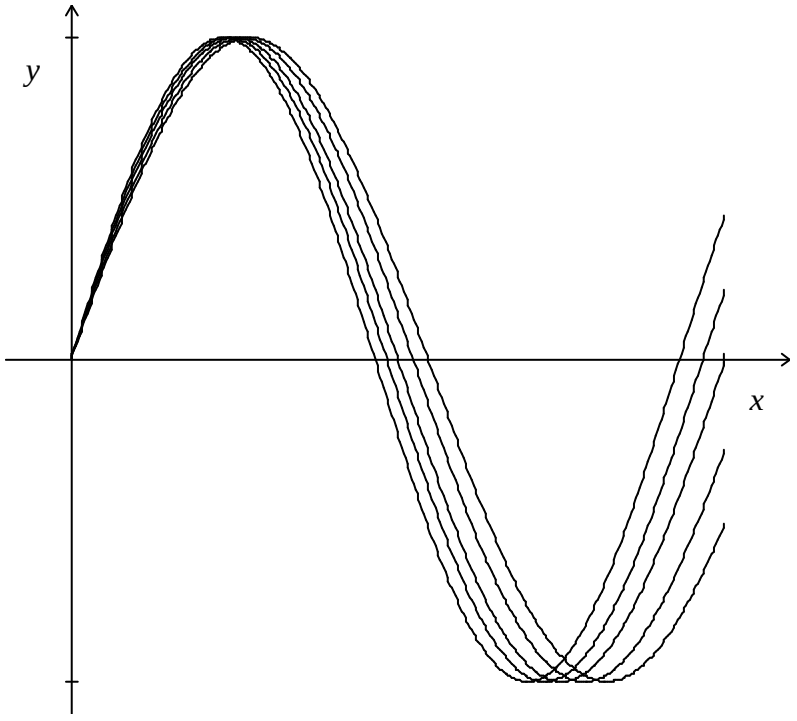
$$E_0 = \frac{\hbar^2}{8mL^2} .$$

Además, la variable adimensional r queda restingida al intervalo [0, 1].

Para una partícula encerrada, la función de onda debe ser cero en las paredes, por lo que $Y(0) = Y(1) = 0$. Esta condición determina a los valores propios, ya que, de todas la soluciones de la ecución diferencial, sólo para algunos valores de ε existen soluciones que cumplen las condiciones de frontera. Por ejemplo, en ausencia de fuerzas, $v(r) = 0$, se tiene la ecuación diferencial

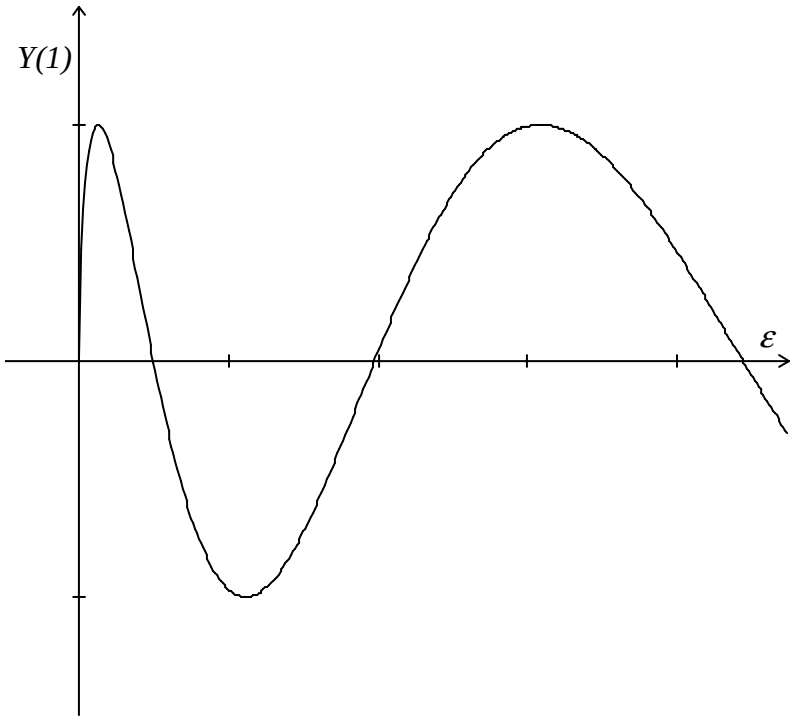
$$Y'' = -\varepsilon Y ,$$

con $\varepsilon > 0$. En la figura siguiente se muestran algunas soluciones para diferentes valores de ε.



Soluciones de la ecuación $Y'' = -\varepsilon Y$ **con** $Y(0) = 0$**, para diferentes valores de** ε**.**

Un método para encontrar los valores propios consiste en resolver la ecuación diferencial para diferentes valores de ε, y buscar aquellos valores de ε para los que $Y(1) = 0$. Para este propósito se puede usar un método como el de la bisección o el de la secante. La dependencia de $Y(1)$ con ε se presenta en una gráfica y puede observarse que sólo para algunos valores de ε se cumple la condición de frontera.



Sesión práctica

- Escriba un programa que permita resolver un sistema de ecuaciones diferenciales.
- Modifique su programa para resolver el problema de una partícula encerrada en presencia de un campo de fuerzas.
- En ausencia de fuerzas, los valores propios son conocidos en forma exacta. Obtenga los dos valores propios de menor energía.
- Encuentre el estado basal y el primer estado excitado para una partícula encerrada en presencia de los potenciales $v(r) = 10\,(r - ½)$, $v(r) = 10\,\sin(\pi r)$ y $v(r) = -10\,\sin(\pi r)$.

```
/* Para el problema de los valores propios es recomendable
 * usar macros para las cantidades que se mantiene fijas
 * como en el siguiente ejemplo. */

#define a 0.0
#define b 1.0
#define DIM 2
#define npasos 128

double E;

void f_vec(double x, double vy[DIM], double vf[DIM]);
double K(double x);
double V(double x);
void solucion(double vx[npasos + 1], double vy[DIM][npasos + 1], int N,
              double x0, double vyinicial[DIM], double h);
void rk4_vec(double xn, double yn[DIM], double vynew[DIM], double h);
void sol_escribir(double x[npasos + 1], double y[DIM][npasos + 1], int N);
```

Comentarios finales

Este curso tiene como finalidad mostrar los elementos básicos un lenguaje de programación y aplicar estos elementos para programar algunos métodos numéricos, todo esto dentro del ambiente UNIX. El curso puede considerarse como una introducción al cómputo científico.

Actualmente, los lenguajes de programación más utilizados para realizar cómputo científico son el lenguaje C y el lenguaje FORTRAN. En estas notas se optó por usar el primero. Ambos lenguajes tiene muchas similitudes y aprender con uno de ellos facilita enormemente el estudio posterior del otro. Por lo cual, la elección uno de ellos no limita los alcances del curso. En estas notas sólo se presentan los elementos básicos del lenguaje C y su dominio requiere de profundizar en muchos aspectos que se encuentran más allá de los objetivos del curso, sin embargo el material cubierto permite escribir programas que realicen cómputo numérico.

Los método numéricos presentados en el curso son una pequeña muestra de los métodos más comunes y de mayor utilidad en las ciencias. Estas notas no pretenden ser una revisión exaustiva ni un curso formal de análisis numérico y el autor recomienda a los estudiantes que revisen la lista de referencias para profundizar en el tema y ampliar sus conocimientos en los aspectos que no se cubren en esta obra.

Finalmente, el uso del sistema operativo UNIX durante este curso permite que el alumno se enfrente al sistema operativo que usan la mayoría de las computadoras dedicadas al cómputo científico en el mundo.

Como complemento del curso, y para que los alumnos puedan practicar en su casa, se recomienda utilizar el compilador público gcc (GNU C Compiler) que es de distribución gratuita y que es desarrollado por la Free Software Foundation. La información sobre este programa se puede obtener del sitio Web www.gnu.org, en donde se encuentran ligas hacia los sitios que contienen los programas que se pueden ejecutar en MS-DOS, Windows y otras plataformas. Del mismo sitio se puede obtener información sobre algunas colecciones de programas que simulan el ambiente UNIX en una computadora personal, entre éstas están: bash, file utilities, text utilities, grep, vim, etc. También, existe la posibiladad de simular un ambiente UNIX en una ventana de Windows. Esta se encuentra disponible en la página www.cygwin.com e incluye el shell bash, el compilador gcc y las herramientas para el manejo de texto, archivos y directorios. Esta aplicación requiere de mucho espacio en disco, pero tiene la ventaja de que las actualizaciones se realizan facilmente. Adicionalmente, existen ambientes de programación disponibles en interner, basados totalmente en Windows o en MS-DOS, que son igualmente útiles para programar, pero que no permiten practicar las ordenes de sistema UNIX que se usan en este curso.