MECÁNICA COMPUTACIONAL I

Capítulo 1

Introducción a la Computación

Contenido

- Ejemplos de problemas en Ingeniería Mecánica.
- Modelos numéricos. Cifras significativas. Error absoluto y relativo. Exactitud y precisión. Aproximaciones y tipo de errores.
- Algoritmos: descripción y objetivos.
- Operadores: lógicos, aritméticos y relacionales
- Representación de números en el computador
- Introducción a la programación en MatLab

Solución de problemas: combinación entre técnica e inteligencia

1. Pasos típicos:
   - Existencia de una necesidad ligada a un entorno
2. Planteamiento inicial del problema
3. Análisis inicial
   - Revisar la problemática planteada en su entorno.
   - ¿Qué se quiere?
   - ¿Qué se requiere?

4. Formulación del problema
   - Satisface requerimientos de los involucrados: el de la necesidad y el que lo va a resolver
5. Propuesta de modelo base
   - Modelo conceptual
6. Formulación del modelo matemático
   - Añade nuevas restricciones al modelo
7. Determinar soluciones al modelo matemático
   - Analíticas ó Numéricas

8. Análisis de la solución
   - Verificar consistencia con la situación planteada
   - ¿Cumple las expectativas?
   - ¿Permite hacer predicciones?

9. Aplicación de los resultados o reformulación del problema o modelos

EJEMPLO: Una nueva empresa desea construir paracaídas. Para ello requiere conocer la cantidad de tela por paracaídas y solicita su recomendación como especialista.

1. Necesidad : Determinar la cantidad de tela por paracaídas

   Entorno:

   (a) Lanzamiento en caída libre

   (b) Resistencia máxima de un ser humano a impactos

2. Planteamiento inicial del problema: Determinar la cantidad de tela por paracaídas.

3. Análisis inicial

   – Cantidad de tela dependerá de la forma, tamaño, peso, etc.

   – La velocidad final máxima del sujeto es el parámetro crucial para determinar si el paracaídas funciona y, en consecuencia la cantidad de tela que requiere cada paracaídas

4. Formulación del problema: ¿Qué velocidad máxima, si la hubiere, alcanza un paracaidista de masa m que se lanza desde un avión a una altura h?

5. Propuesta de modelo base
   - Caída libre
   - No hay interacción con el medio externo
6. Formulación del modelo matemático

$$F = ma = m\frac{dv}{dt} = mg$$

7. Determinar soluciones al modelo matemático

$$v = \sqrt{2gh}$$

8. Análisis de la solución
   - No existe una velocidad limite salvo la impuesta por la altura
   - No aparece la cantidad de tela como un parámetro
9. Aplicación de los resultados o reformulación del problema o modelos
   - Debemos incluir la fricción con el aire

Re-formulación del modelo matemático

5. Propuesta de modelo base
   - Caída libre
   - Resistencia del aire es importante
6. Formulación del modelo matemático

$$F = ma = m\frac{dv}{dt} = mg + F_R \qquad F_R = -Cv$$

C es un coeficiente que toma en cuenta los efectos de resistencia del medio ambiente

7. Determinar soluciones al modelo matemático

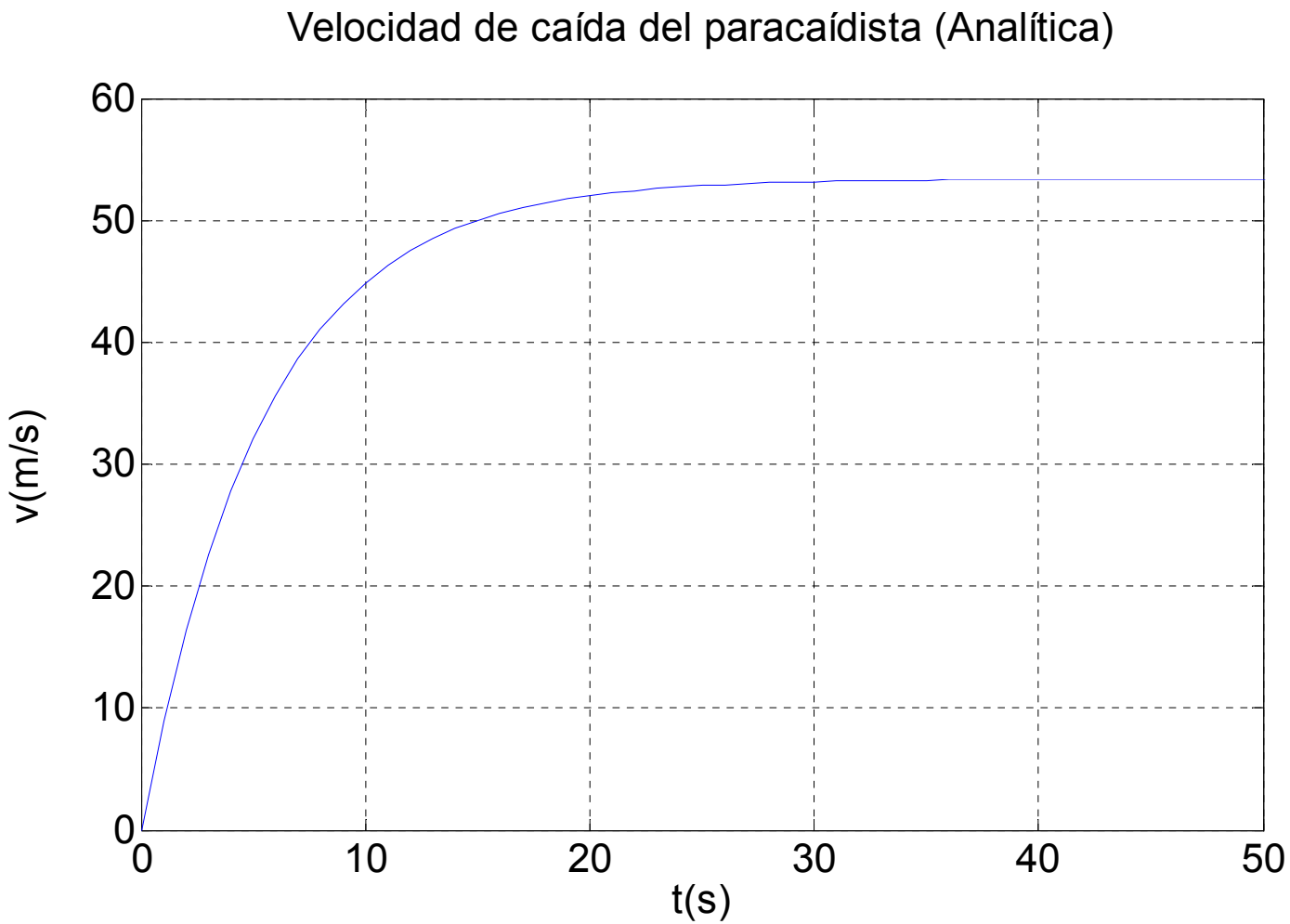
$$v(t) = \frac{gm}{C}\left(1 - e^{-(C/m)t}\right)$$

8. Análisis de la solución
   - Existe una velocidad limite dada por $v = \frac{gm}{C}$
   - Modelo satisface expectativas

9. Aplicación de los resultados o reformulación del problema o modelos
   - Conocidas las condiciones del problema (m, C, g) podemos determinar la velocidad limite.
   - Por ejemplo para m= 68.1 Kg, C= 12.5 Kg/s y g=9.8 m/s^2 obtenemos:

$$\boxed{v = 53.4 m/s}$$

Pudiéramos determinar inclusive la evolución de la velocidad en el tiempo. En ese caso tendríamos:



Buscando la solución de la EDO (MATHEMATICA™)

In[1]:=
$$\text{DSolve}\left[\left\{y'[t] == g - \frac{c}{m}\, y[t],\ y[0] == 0\right\},\ y[t],\ t\right]$$

Out[1]=
$$\left\{\left\{y[t] \to \frac{e^{-\frac{ct}{m}}\left(-1 + e^{\frac{ct}{m}}\right) g\, m}{c}\right\}\right\}$$

In[4]:=
$$\text{Simplify}\left[\frac{e^{-\frac{ct}{m}}\left(-1 + e^{\frac{ct}{m}}\right) g\, m}{c}\right]$$

Out[4]=
$$\frac{\left(1 - e^{-\frac{ct}{m}}\right) g\, m}{c}$$

Aplicación de un modelo numérico

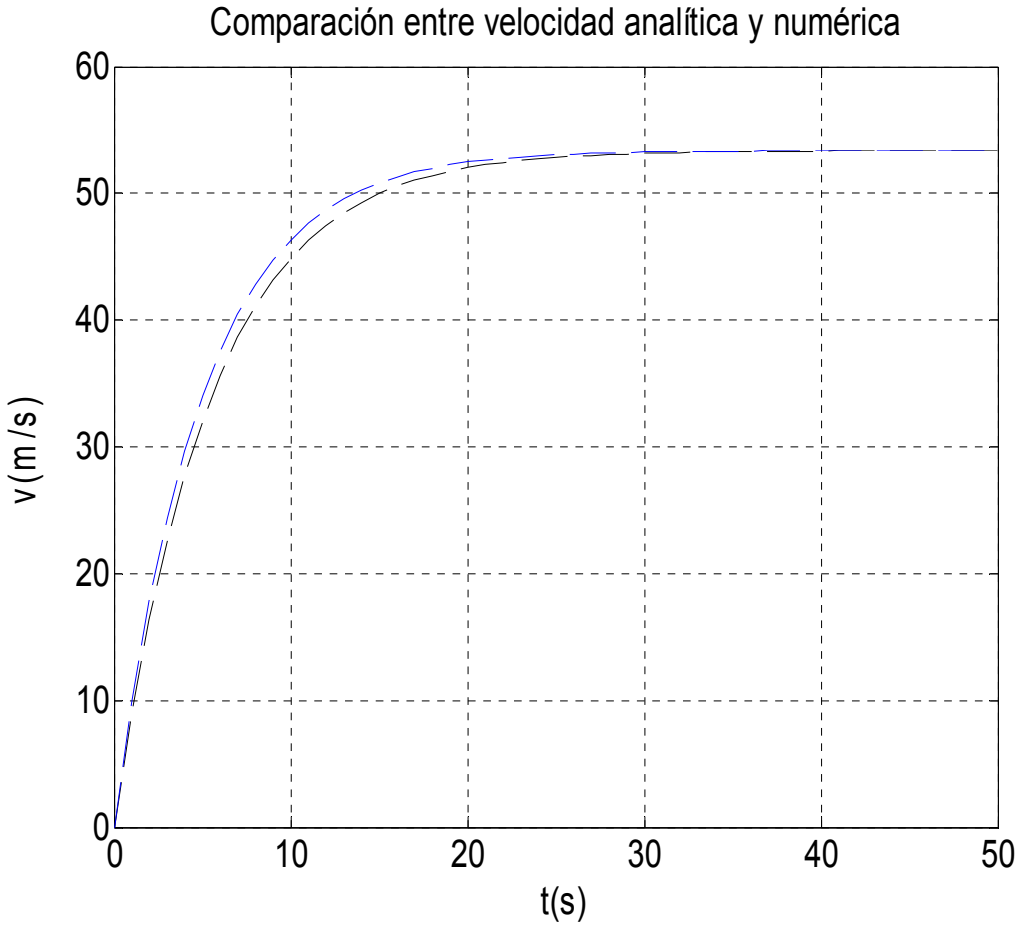
Supongamos que no podemos hallar la solución analítica. Un posible modelo numérico podría ser obtenido a partir de

$$F = ma = m\frac{dv}{dt} = mg - Cv$$

De la definición de derivación obtenemos

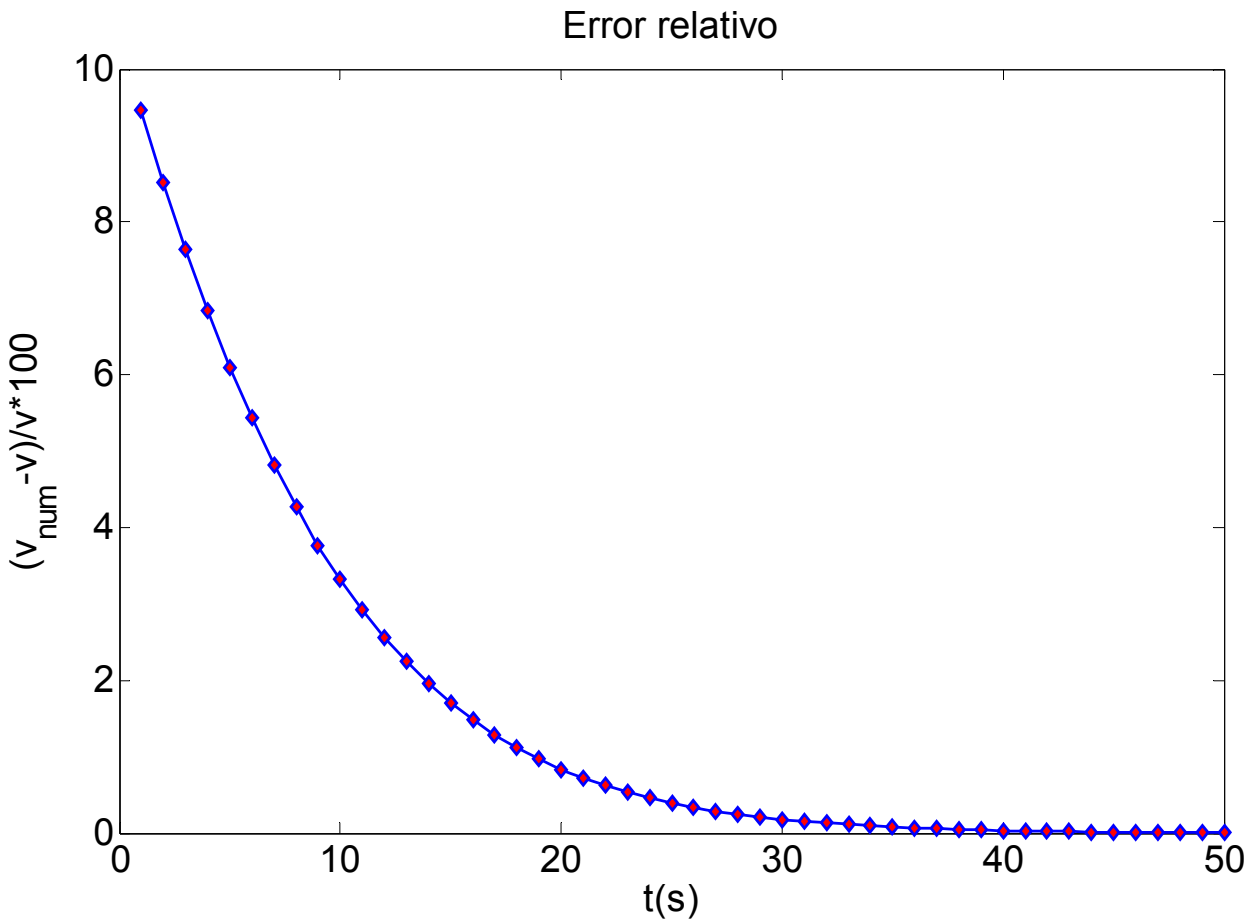
$$\frac{v(t+\Delta t)-v(t)}{\Delta t} \approx g - \frac{C}{m}v(t) \longrightarrow v(t+\Delta t) \approx v(t) + \left[g - \frac{C}{m}v(t) \right] \Delta t$$

Conocida la velocidad en el instante inicial (t=0) podemos calcularla en el instante t=Δt. Luego, con esta nueva velocidad, el cálculo se repite de manera iterativa para obtener la velocidad en cada instante Δt



–Las predicciones de velocidad terminal coinciden

–Sin embargo durante los instantes iniciales se producen discrepancias entre los valores teóricos y numéricos



Estudiar la naturaleza de estos métodos y la manera en las cuales sus descripciones reflejan de manera fidedigna la realidad es el objeto de los métodos numéricos.

¿Cómo se construyeron los gráficos mostrados en las láminas anteriores?

MATLAB®

“MATLAB® es un lenguaje de alto rendimiento para computación técnico-científica.

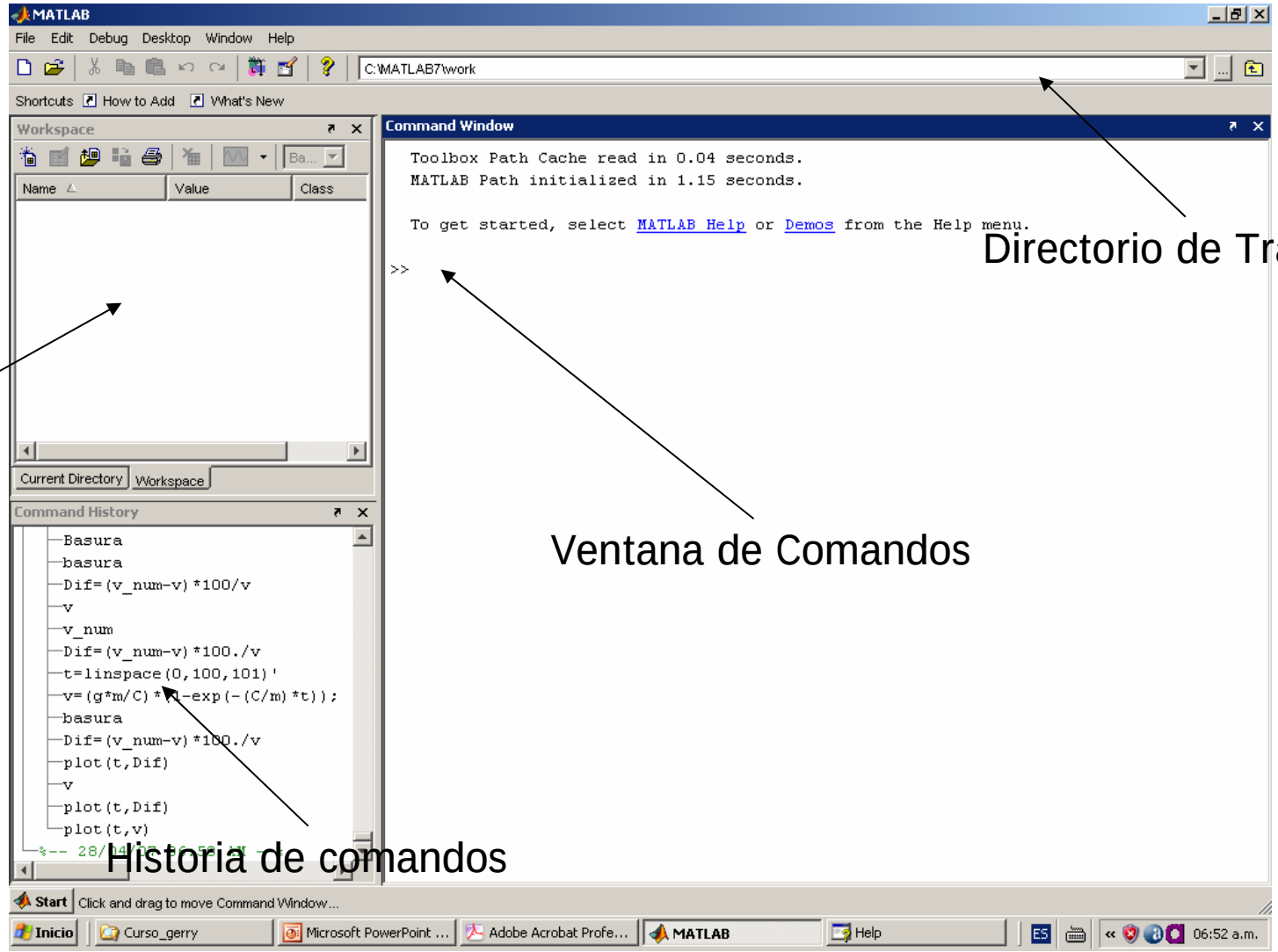
Integra computación, visualización y programación en un ambiente amigable.

Usos típicos incluyen aplicaciones matemáticas, implementación de algoritmos, manejo de datos y visualización científica y manejo de datos.

Permite resolver muchos problemas de ingeniería, especialmente aquellos en cuya formulación pueden emplearse matrices y vectores en una fracción del tiempo que tomaría escribir un programa en un lenguaje no interactivo como C o Fortran.”

(Tomado del manual de MATLAB)

Cuando inicia una sección en MATLAB:



WorkSpace

Directorio de Trabajo

Ventana de Comandos

Historia de comandos

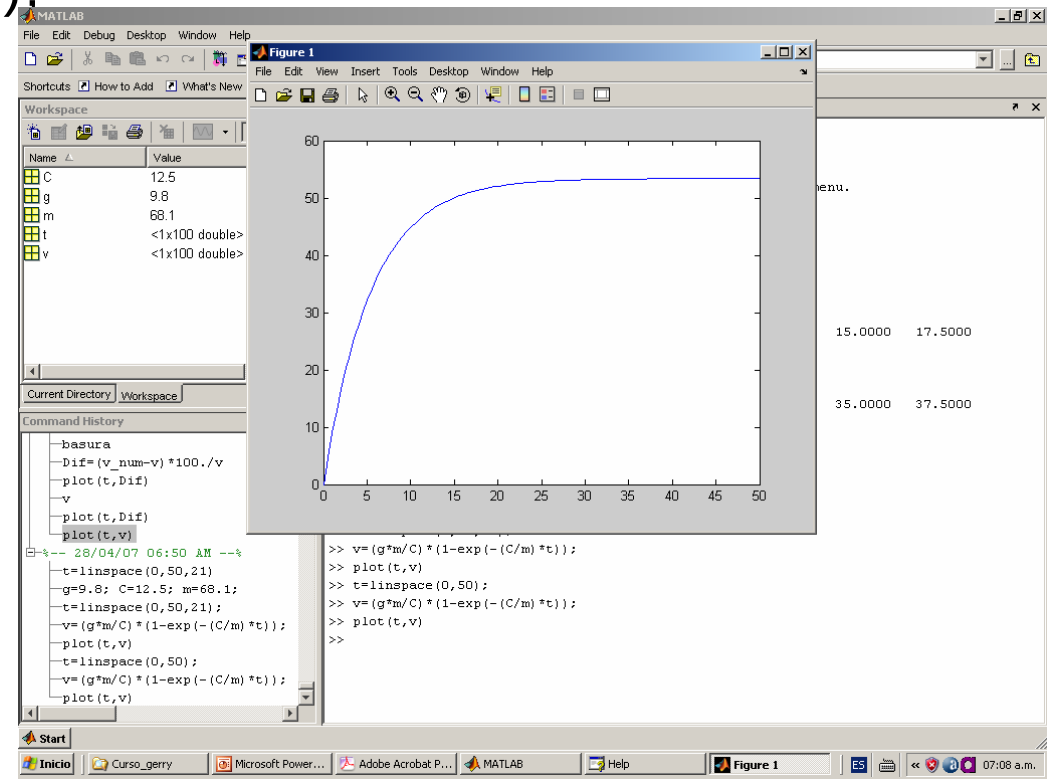
Las gráficas anteriores fueron generadas utilizando los comandos

```
>> t=linspace(0,50);

>> v=(g*m/C)*(1-exp(-(C/m)*t));

>> plot(t,v)
```

Para obtener →



Algunos aspectos:

–Variables

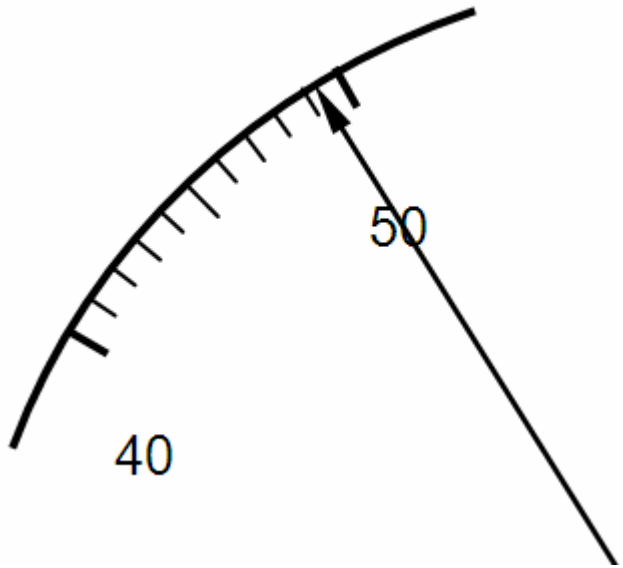
–Uso de punto y coma

–Manejo de matrices y vectores

–Optimizando aspecto del gráfico

Cifras significativas

- El concepto de cifra significativa o dígito ha sido desarrollado para designar formalmente la confiabilidad de un valor numérico.
- Los dígitos significativos de un número son aquellos que pueden ser empleados con confianza.
- Ello corresponde a dígitos ciertos más un dígito estimado. Por ejemplo, en el manómetro de la figura, los dígitos ciertos serían 49, mientras que para el caso de tres cifras se pudiese escribir 49.5.



Los ceros no son siempre cifras significativas. Esto depende de su localización respecto al punto decimal.

Los valores

0.00001845, 0.0001845, 0.001845

tienen todos cuatro cifras significativas.

Similarmente, cuando se tienen ceros al final de algunos números, no esta claro el significado de los mismos. Por ejemplo:

453.00

puede tener 3, 4 ó 5 cifras significativas, dependiendo de cuantos de esos ceros se conocen con confianza.

Esta incertidumbre puede ser resuelta al emplear notación científica. Los valores:

$$4.53 \cdot 10^2$$

$$4.530 \cdot 10^2$$

$$4.5300 \cdot 10^2$$

designan que el número se conoce con tres, cuatro y cinco cifras significativas, respectivamente.

Este concepto de cifras significativas tiene dos implicaciones importantes para el estudio de los métodos numéricos [Chapra]:

- Los métodos numéricos producen resultados aproximados. Por esto se debe desarrollar un criterio para especificar la confiabilidad de los resultados. Una forma puede ser en términos de cifras significativas.
- Aunque números tales como π, e y $\sqrt{7}$ representan cantidades matemáticas específicas, no pueden ser representados exactamente por una cantidad limitada de dígitos.

Error absoluto: si p^* es una aproximación al valor verdadero p, el módulo de la diferencia entre las dos cantidades se conoce como error absoluto. Su valor está definido por:

$$\varepsilon_a = \left| p - p^* \right| \ ,$$

que proporciona una medida de la desviación de p respecto a p^*.

El inconveniente de esta cantidad es que el valor resultante no facilita la apreciación del error, a menos que se conozca la magnitud de las cantidades p y p^* involucradas.

Error relativo: si p^* es una aproximación al valor verdadero p, la relación entre el error absoluto y el módulo del valor verdadero p se conoce como error relativo. Su valor está definido por:

$$\varepsilon_r = \left| \frac{p - p^*}{p} \right|, \; p \neq 0 \;\; ,$$

Esta medida nos permite apreciar mejor cuán cerca se encuentra realmente el valor aproximado del valor real.

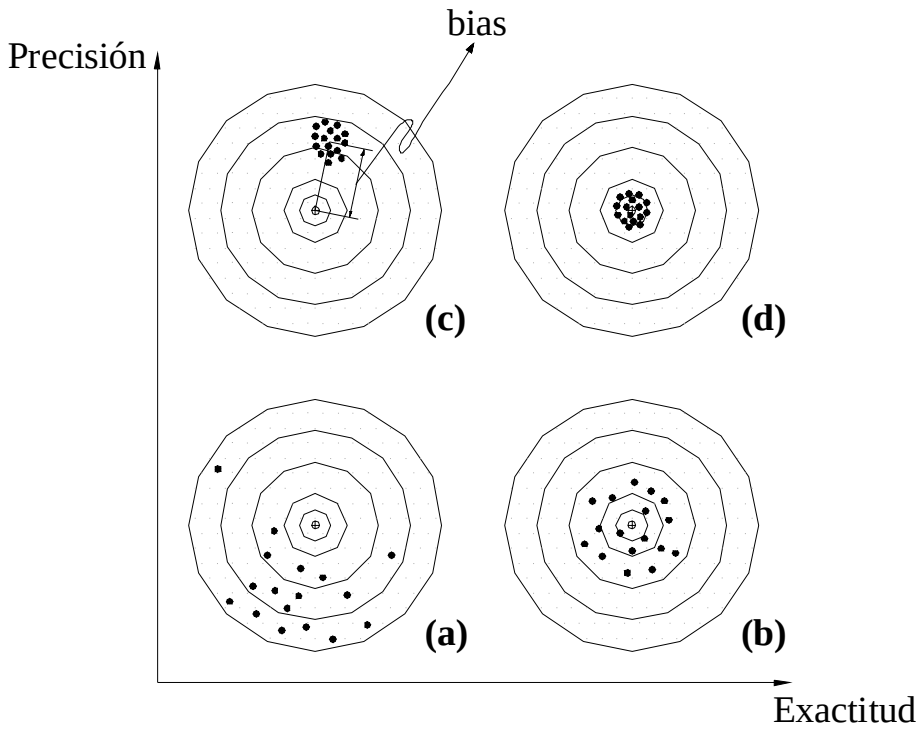
Sin embargo, para valores cercanos a cero, esta representación del error puede llevar a falsas interpretaciones.

Exactitud y Precisión

La exactitud se refiere a la cercanía de un valor calculado o medido respecto al valor real.

La precisión se refiere a la proximidad que presentan valores calculados o medidos respecto a otros.

La precisión hace referencia al número de cifras significativas representando la dispersión en cálculos o medidas repetitivas de un valor particular.



Exactitud y Precisión

En la mayoría de aplicaciones que surgen en la práctica los problemas son tan complejos que no pueden ser resueltos por técnicas analíticas o exactas. Así, casi nunca existirá un patrón o referencia para juzgar a la solución numérica.

Por ende, nunca debe hablarse en métodos numéricos de la exactitud de un esquema de solución o algoritmo, en contraste debe hablarse en términos de precisión, enfatizando con ello que el esquema numérico converge a una solución.

Así que en análisis numérico el escenario asumido es (c), debiéndose determinar obligatoriamente el eventual bias presente de manera de garantizar que este también tiende a cero a medida que el algoritmo converge a una eventual solución.

Errores por truncamiento: errores asociados a la utilización de una cantidad finita de términos en el cálculo en la aplicación de un método numérico. Por ejemplo, al utilizar tres términos de una serie de Taylor para aproximar un valor f(x):

$$f(x) = f(x_0) + f'(x_0)\cdot(x - x_0) + \frac{f''(x_0)\cdot(x - x_0)^2}{2!} + O\left((x - x_0)^3\right)$$

se tiene que se comete un error por el hecho de truncar el desarrollo. El término $O((x-x_0)^3)$ indica que se está cometiendo un error debido al truncamiento de la serie de Taylor del orden de $(x-x_0)^3$. Por supuesto, al utilizar más términos puede disminuir el orden de magnitud del error cometido.

Errores por redondeo: todos los dispositivos de cálculo representan números con alguna imprecisión.

Los computadores digitales siempre almacenan las cantidades numéricas con una cantidad finita de cifras significativas, de forma que los valores verdaderos no pueden ser representados exactamente.

A esto se llama un *"error de redondeo"*, ya que la fracción decimal esta redondeada.

Un algoritmo es una secuencia de pasos lógicos requeridos para realizar una tarea específica, tal como resolver un problema matemático.

Para cumplir con este objetivo, un buen algoritmo debe tener los siguientes atributos:

- Cada paso debe ser determinístico, esto es, cada enunciado debe ser independiente de las demás partes del proceso.
- El proceso siempre debe terminar después de una cantidad finita de pasos.
- El algoritmo debe ser lo suficientemente general como para poder manejar la mayoría de las situaciones que pueda presentar en el problema.

Ejemplo 1: Algoritmo para la solución de un problema simple de sumar dos números:

Paso 1: Inicio del cálculo

Paso 2: Leer el valor del primer término A.

Paso 3: Leer el valor del segundo término B.

Paso 4: Sumar A y B y almacenar la respuesta en C.

Paso 5: Mostrar la solución C.

Paso 6: Fin del calculo.

Para resolver de forma eficiente y adecuada problemas más complejos, es indispensable estar familiarizado con el problema a resolver, así como con sus posibles soluciones.

Es conveniente tener a la mano toda la información disponible acerca de tales soluciones, de forma que el algoritmo pueda tener los atributos mencionados anteriormente.

Este aspecto se presenta en el siguiente ejemplo.

Ejemplo 2: Hallar la solución a una ecuación de segundo grado de la forma:

$$a \cdot x^2 + b \cdot x + c = 0$$

La ecuación tiene dos soluciones, cuya naturaleza depende de una cantidad llamada discriminante

$$d = b^2 - 4a \cdot c$$

Si d > 0, se tienen soluciones reales y distintas dadas por:

$$x_1 = \frac{-b + \sqrt{d}}{2 \cdot a} \qquad x_2 = \frac{-b - \sqrt{d}}{2 \cdot a}$$

Para d = 0, se tienen soluciones reales e iguales,

$$x_{1,2} = -\frac{b}{2 \cdot a}$$

Para d < 0, se tienen soluciones complejas conjugadas, cuyas partes real e imaginaria se obtienen con

$$x_r = -\frac{b}{2 \cdot a} \qquad x_i = \frac{\sqrt{-d}}{2 \cdot a}$$

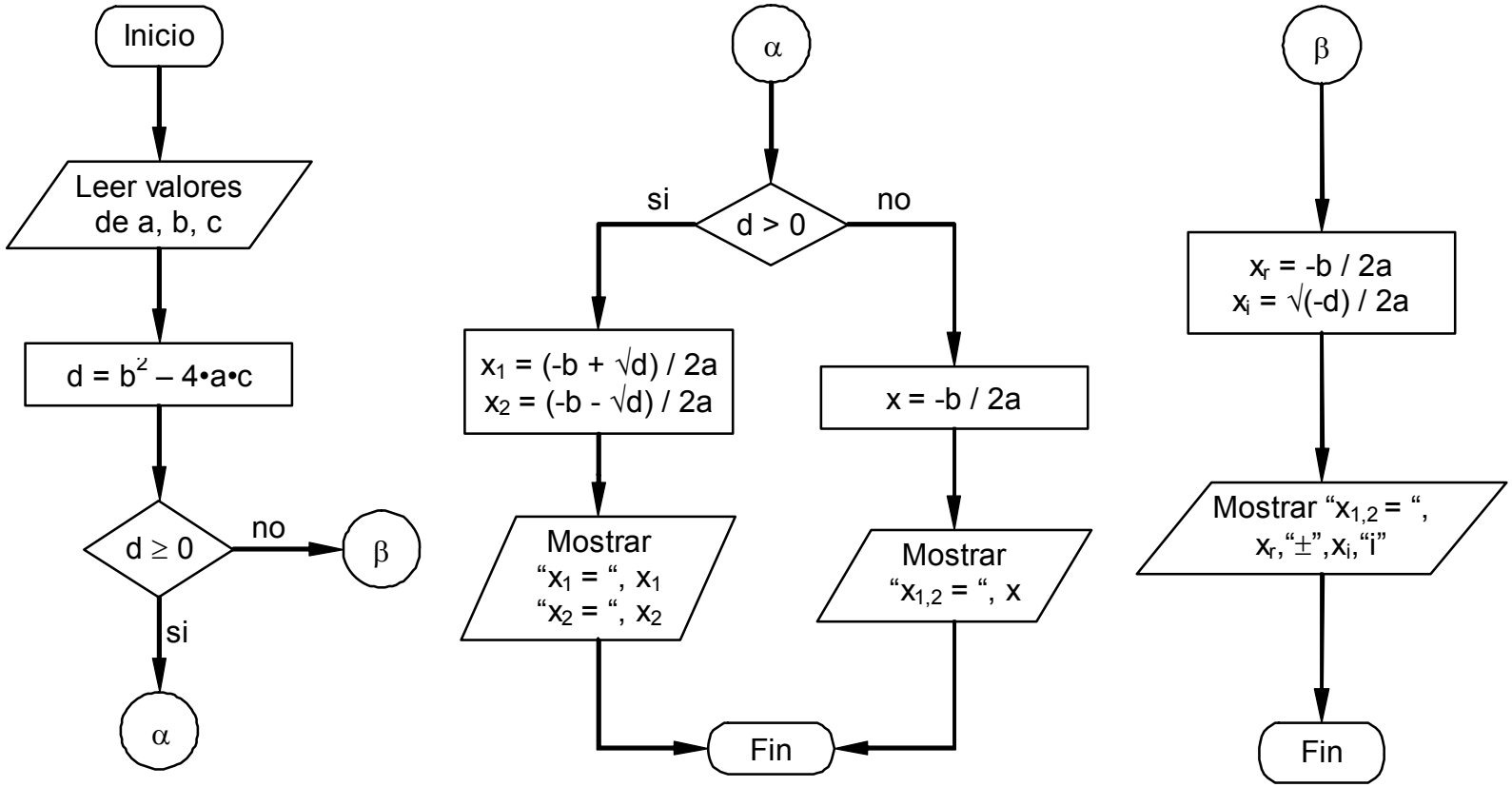
y las soluciones se presentan como:

$$x_1 = x_r + x_i i \qquad x_1 = x_r - x_i i$$

Con toda esta información disponible, es posible escribir un buen algoritmo. La lámina siguiente muestra un algoritmo basado en esta información

Ejemplo

Algoritmo para resolver una ecuación de segundo grado:



Ejemplo MATLAB

```
%  Programa SolEcSegGr
%  SolEcSegGr halla las raices de una ecuación de segundo grado
%  a x^2 + b x + c = 0

clear all;
disp([' Solucion de una ecuacion de segundo grado ']);
disp([' Introduccion de los coeficientes A,B y C ']);
a = input('a = ');
b = input('b = ');
c = input('c = ');

% Calculo del discriminante

d = b^2 - 4*a*c;
if d>=0
    x1=(-b+sqrt(d))/(2*a);
    x2=(-b-sqrt(d))/(2*a);
    disp([' x1 =  ' num2str(x1)]);
    disp([' x2 =  ' num2str(x2)]);
else
    xr=-b/(2*a);
    xi=sqrt(abs(d))/(2*a);
    disp([' x1 =  ' num2str(xr)  ' + i ' num2str(xi)]);
    disp([' x2 =  ' num2str(xr)  ' - i ' num2str(xi)]);
end
```

Operadores aritméticos, relacionales y lógicos

- Los algoritmos asociados a los métodos numéricos siempre requieren que se realicen acciones y comparaciones entre valores y eventos que ocurren dentro de los mismos.
- Estas acciones y comparaciones son realizadas por los operadores (símbolos que, en general, indican alguna relación entre dos o más objetos denominados operandos).
- Los operadores se clasifican en tres grandes grupos: aritméticos, relacionales y lógicos.
- También por la cantidad de operandos que afectan, los operadores pueden ser *unarios*, cuando operan sobre un solo elemento, o *binarios*, cuando actúan sobre dos elementos.

Operadores aritméticos

- Estos operadores son los más conocidos, y están formados por los símbolos *+, -, *, /*, y ocasionalmente algún otro símbolo que realiza alguna operación especial.
- Estos operadores realizan, como su nombre lo indica, operaciones aritméticas: suma, resta, multiplicación y división.
- El - y él *+*, además funcionan como operadores unarios, cambiando o dejando igual el signo del operando.

Operadores aritméticos

Símbolo	Operación	Tipo	Ejemplo de uso	Resultado
+	suma (adición)	binario	a + b	a+b
-	resta (sustracción)	binario	a - b	a-b
*	multiplicación (producto)	binario	a * b	a*b
/	división (cociente)	binario	a / b	a/b
+	mismo signo	unario	+a	A
-	cambio de signo	unario	-a	-a

Operadores relacionales

- Se trata de los operadores $>$, $<$, $\geq$, $\leq$, etc., los cuales comparan los valores relativos de los operandos.
- El resultado de una operación relacional es uno de los valores CIERTO o FALSO, los cuales son conocidos como valores lógicos.
- En general, indistintamente del lenguaje de programación puede afirmarse que el valor CIERTO es cualquier número distinto de cero (típicamente 1) mientras que FALSO siempre es cero.

Operadores relacionales

Símbolo	Operación	Tipo	Ejemplo de uso	Resultado
>	mayor a	binario	a > b	CIERTO o FALSO
≥	mayor o igual a	binario	a ≥ b	CIERTO o FALSO
<	menor a	binario	a < b	CIERTO o FALSO
≤	menor o igual a	binario	a ≤ b	CIERTO o FALSO
=	igual a	binario	a = b	CIERTO o FALSO
≠	distinto a	binario	a ≠ b	CIERTO o FALSO

Operadores lógicos

- Los menos utilizados en la actividad cotidiana, pero ampliamente conocidos en el ámbito computacional.
- Estos operadores actúan sobre valores lógicos y el resultado de una operación lógica es un valor lógico.
- Existe una amplia variedad de operadores lógicos, pero los más utilizados se muestran en la tabla de la lámina siguiente.
- Los símbolos para representar un operador lógico varían en los distintos sistemas de cómputo, por lo que se muestra el nombre en inglés en lugar de un símbolo particular.
- Los nombres entre paréntesis son los equivalentes en español.

Operadores lógicos

Nombre	Operación	Tipo	Ejemplo de uso	Resultado
AND (Y)	Y lógico	binario	a AND b	CIERTO si a y b son ciertos, FALSO en cualquier otro caso
OR (O)	O lógico	binario	a OR b	FALSO si a y b son falsos, CIERTO en cualquier otro caso
NOT (NO)	Negación	unario	NOT a	CIERTO si a es FALSO, FALSO si a es CIERTO
XOR (XOR)	O exclusivo	binario	a XOR b	CIERTO si a y b tienen el mismo valor, FALSO en cualquier otro caso.

– Otra representación del resultado de la acción de estos operadores es (I):

AND			**OR**		
a b	CIERTO	FALSO	a b	CIERTO	FALSO
CIERTO	CIERTO	FALSO	CIERTO	CIERTO	CIERTO
FALSO	FALSO	FALSO	FALSO	CIERTO	FALSO

– Otra representación del resultado de la acción de estos operadores es (II):

NOT		**XOR**		
a	-----	a b	CIERTO	FALSO
CIERTO	FALSO	CIERTO	CIERTO	FALSO
FALSO	CIERTO	FALSO	FALSO	CIERTO

Opciones para representar a los algoritmos

Diagramas de bloque

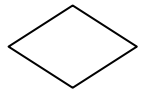
Los diagramas de bloque son una representación gráfica de los algoritmos.

Estos emplean una serie de bloques y flechas, cada una de las cuales representa una operación o paso del algoritmo.

Las flechas representan la secuencia de implementación de las operaciones.

Algunos símbolos de uso frecuente y su significado se muestran en la tabla presentada en la lámina siguiente.

Opciones para representar a los algoritmos

Símbolo	Significado	Símbolo	Significado
	Terminador. Inicio o fin de programa		Decisión. Indica bifurcación del flujo del algoritmo
	Proceso. Indica cálculo o manipulación de datos		Conector. Enlaza diagramas en la misma página
	Datos. Indica entrada o salida de datos		Conector. Enlaza diagramas en distintas páginas

Seudo-código o seudo-lenguaje

Esta técnica emplea enunciados en inglés o español en lugar de los símbolos gráficos de los diagramas de bloque.

En general, no existe un estándar aceptado internacionalmente para representar algoritmos en seudo-lenguaje.

Sin embargo, para entenderse con otros es necesario fijar una convención.

Los enunciados forman estructuras que se clasifican en tres grandes grupos: secuencias, repetición y selección o decisión.

Opciones para representar a los algoritmos

Las palabras que denotan enunciados de selección y repetición se escriben normalmente en mayúsculas, mientras que el resto de los enunciados se escribe en minúsculas.

Cualquier algoritmo en seudo-lenguaje estará compuesto de palabras reservadas e identificadores.

Las palabras reservadas son aquellas que se reserva la convención elegida para representar a las estructuras de control, tales como condicionales y lazos.

Los identificadores son los nombres de las variables que emplea el algoritmo, en el ejemplo 2 estas serían los nombres *a*, *b* y *c* de los coeficientes del polinomio.

En las tablas siguientes se muestran las estructuras de control de la convención propuesta.

Opciones para representar a los algoritmos

Secuencias:

Estructura	Significado
Paso 1; Paso 2; Paso 3; . . . Paso n; *//...* Paso 1, Paso 2, Paso n;	1.Las secuencias son tareas específicas que debe realizar el programa, tal como leer un valor de la consola o evaluar una fórmula. 2.Estos enunciados son ejecutados en orden secuencial, de izquierda a derecha y de arriba hacia abajo, es decir en el mismo orden en el que leemos. 3.El operador coma "," permite representar en una sola línea varias secuencias cuando estas son lo suficientemente sencillas.

Ejemplo MATLAB

```
% Programa Secuencia

% Demuestra como se programan secuencias

disp([' Hallar la velocidad de un móvil en caída libre al contacto con el piso'])

g = -9.8;           % m/s^2

v0 = 0;             % m/s

y0 = 20;            % m

y = 0;              % m

disp([' Vo = ' num2str(v0) ' m/s ']);

disp([' y0 = ' num2str(y0) ' m ']);

disp([' y = ' num2str(y) ' m ']);

vf = sqrt(v0^2 + 2*g*(y-y0));

disp([' Vf = ' num2str(vf) ' m/s ']);
```

Repeticiones (1)

MIENTRAS (Condición) Sentencias 1; Sentencias 2; . . . Sentencias n; **FIN MIENTRAS**	1. Las sentencias son ejecutadas secuencialmente mientras la condición sea cierta. 2. Se utiliza cuando no se sabe de antemano cuantas repeticiones serán requeridas. 3. Siempre verifica la condición al principio de cada iteración. 4. Si la primera vez que se verifica la condición esta resulta FALSA, las sentencias no se ejecutan nunca.

Ejemplo MATLAB

```
%  Programa mientras
%  Demostración de la estructura de repetición while
num = input(' numero  (>0) = ')
cont=0;
while num > 1
    num = num/2;
    cont = cont + 1;
end
disp([' Resultado = ' num2str(num)]);
disp([' Numero de divisiones = ' num2str(cont)]);
```

Repeticiones (2)

HACER Sentencias 1; Sentencias 2; . . . Sentencias n; **MIENTRAS** (Condición)**;**	1. Las sentencias son ejecutadas secuencialmente mientras que la condición sea cierta. 2. Se utiliza cuando no se conoce de antemano cuantas iteraciones serán requeridas. 3. Esta estructura ejecuta las sentencias al menos una vez antes de verificar la condición, es decir, la verificación siempre se hace al final de cada iteración.

Repeticiones (3)

PARA Inicio **Hasta** Fin **Incrementar** Inc Sentencias 1; Sentencias 2; . . . Sentencias n; **FIN PARA**	1. Siempre se compone de tres partes, a saber: *Inicio* (inicialización del contador de control del lazo), *Fin* (condición de salida del lazo, mientras esta sea cierta se continúa iterando) y *Inc* (que representa el incremento o decremento del contador de control). 2. Las sentencias son ejecutadas secuencialmente una cantidad predefinida de veces. 3. La condición del lazo se verifica al comienzo de cada iteración, si esta resulta falsa a priori nunca se ejecutaran las sentencias respectivas.

Ejemplo MATLAB

```
%  Programa para
%  Demostración de la estructura de repetición para
disp([' Hallar la suma de los primeros n enteros'])


n = input(' numero  (entero) = ');
acum = 0;
for i=1:n
  acum=acum +i;
end
disp([' n = ' num2str(n)]);
disp([' Suma = ' num2str(acum)]);
```

Opciones para representar a los algoritmos

Decisiones (1)

//Condicional de una sola alternativa **SI** (Condición) Sentencias 1; Sentencias 2; . . . Sentencias n; **FIN SI** *//Sintaxis corta* **SI** (Condición) Sentencia;	1. Si la condición es cierta, se ejecutan secuencialmente las sentencias. 2. Si la condición es no cierta, la ejecución del algoritmo continúa en el primer enunciado que siga a la palabra reservada *FIN SI*. 3. La versión corta ahorra la necesidad de dibujar una línea y emplear la palabra *FIN SI*, pero sólo aplica sobre sentencias sencillas. Es decir aquellas que pueden ser separadas por el operador coma o porque se trata de un solo estamento.

Decisiones (2)

//Condicional de doble alternativa **SI** (Condición) Sentencias 1; Sentencias 2; . . Sentencias n; **SINO** Sentencias n + 1; Sentencias n + 2; . . Sentencias n + m; **FIN SI**	1. Si la condición es cierta, se ejecutan secuencialmente únicamente los enunciados que se encuentran entre las palabras *SI* y *SINO*. 2. Si la condición no es cierta, se ejecutan secuencialmente únicamente los enunciados que se encuentran entre las palabras *SINO* y *FIN SI*. 3. En cualquiera de los dos casos, la ejecución del algoritmo continúa en el primer enunciado que siga a la palabra *FIN SI*. 4. Esta estructura provee la bifurcación típica en computación donde debe decidirse que camino tomar al llegar a una intersección.

Ejemplo MATLAB

```
%  Programa SolEcSegGr
%  SolEcSegGr halla las raices de una
% ecuación de segundo grado
%  a x^2 + b x + c = 0

clear all;
disp(['  Solucion de una ecuacion de segundo grado  ']);
disp(['  Introduccion de los coeficientes A,B y C  ']);
a = input('a = ');
b = input('b = ');
c = input('c = ');

% Calculo del discriminante

d = b^2 - 4*a*c;

if d>=0
   x1=(-b+sqrt(d))/(2*a);
   x2=(-b-sqrt(d))/(2*a);
   disp([' x1 =  ' num2str(x1)]);
   disp([' x2 =  ' num2str(x2)]);
else
   xr=-b/(2*a);
   xi=sqrt(abs(d))/(2*a);
   disp([' x1 =  ' num2str(xr)  ' + i '
num2str(xi)]);
   disp([' x2 =  ' num2str(xr)  ' - i '
num2str(xi)]);
end
```

Opciones para representar a los algoritmos

En todas las estructuras de control anteriores se enfatizan las palabras reservadas en negritas y mayúsculas

El texto en cursivas precedido por un doble slash (*"//"*) corresponde a los comentarios opcionales que pueden insertarse en el seudo-código.

Siempre deben escribirse comentarios en el código, de manera de documentar el algoritmo lo suficiente como para que pueda ser entendido por otra persona.

Las condiciones lógicas de los lazos siempre deben ir entre paréntesis, esta condición es obligatoria

Las líneas verticales delimitan el conjunto de sentencias que están subordinadas a la estructura de control que las precede.

Ejemplo

Desarrollar un algoritmo en seudo-lenguaje que calcule

$$Sum = \sum_{i=1}^{n} x_i = x_1 + x_2 + \ldots + x_{n-1} + x_n \ ,$$

donde n y los números x_1, x_2,...,x_n son conocidos.

```
ENTRADA: n, x1, x2,...,xn
SALIDA: suma de los valores xi
//Inicialización del acumulador
Sum = 0;
PARA i = 1 HASTA n INCREMENTAR i EN 1
  Sum = Sum + xi; //Acumular la suma
FIN PARA
IMPRIMIR Sum;
```

Ejemplo

Para constatar que el algoritmo cumple correctamente con su cometido, puede realizarse un proceso de verificación llamado "*corrida en fríó*".

La misma consiste en suponer unos valores de entrada y ejecutar el algoritmo, anotando los valores intermedios de las variables de manera que se pueda observar su evolución.

A continuación se muestra la corrida en frío para el algoritmo anterior, suponiendo que *n = 3, x1 = 1, x2 = 2, x3 = 3*.

Corrida en frío: n = 3; x1 = 1, x2 = 2, x3 = 3

Sum = 0, i = 1 (i = 1 al comienzo de la estructura *PARA – FIN PARA*)

Pregunta: $(i = 1) \leq 3$? $\Rightarrow$ si $\Rightarrow$ Sum = 0 + 1 = 1, i = 1 + 1 = 2 ,

Pregunta: $(i = 2) \leq 3$? $\Rightarrow$ si $\Rightarrow$ Sum = 1 + 2 = 3, i = 2 + 1 = 3 ,

Pregunta: $(i = 3) \leq 3$? $\Rightarrow$ si $\Rightarrow$ Sum = 3 + 3 = 6, i = 3 + 1 = 4 ,

Pregunta: $(i = 4) \leq 3$? $\Rightarrow$ no $\Rightarrow$ fin de estructura *PARA – FIN PARA* ,

Mostrar Sum = 6 .

Ejemplo

Construya un algoritmo que tenga como entrada un número real $x \in [0, \pi/4]$, el valor de *sen(x)*, una tolerancia de error TOL y un número máximo de iteraciones M y que muestre a la salida cuántos términos de la serie de MacLaurin de la función seno se necesitan para aproximar el valor del *sen(x)* con la tolerancia de error TOL. Si se requiere más de M términos, entonces se debe detener el algoritmo y mostrar un mensaje de error.

La serie de MacLaurin para la función seno es

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{i=1}^{n} \frac{-1^{i+1} \cdot x^{2i-1}}{(2i-1)!} \ .$$

Ejemplo

ENTRADA: x, sen(x), TOL, M

SALIDA: Número de términos o mensaje de error

```
//Inicialización de los parámetros
n = 0, sum = 0, F = sin(x), signo = 1;
//lazo de cálculos
HACER
    | n = n + 1;
    | sum = sum + (signo * x^(2n-1)) / (2 * n - 1)!;
    | error = | sum – F |;
    | signo = -1 * signo;
MIENTRAS ((n ≤ M) Y (error ≥ TOL));
SI (n ≤ M)
   | Mostrar “Se requieren “, n, “ términos”;
SINO
   | Mostrar “El método fallo”;
FIN SI
```

Ejemplo: Fortran (1)

```
!*****************************************************************************
!
!  PROGRAM: Num_Ter_Seno
!
!  PURPOSE:  Calcula el número de términos para aproximar la función seno
!            utilizando el desarrollo en serie de Mclaurin
!
!  ENTRADA:  x, sen(x), TOL, M
!  SALIDA:   Número de términos o mensaje de error
!*****************************************************************************

            program Num_Ter_Seno
            implicit none
            integer M,n,fact,i
            double precision  x,TOL,sum,F,signo,error,term

!   Datos de entrada
    write(*,*)" x (radianes) = "
    read(*,*) x
            write(*,*)" Tolerancia = "
    read(*,*) TOL
            write(*,*)" Número de términos = "
    read(*,*) M
```

Ejemplo: Fortran (2)

```
!   Inicialización de los parámetros

            n = 0
            sum = 0.0
            F = sin(x)
            signo = 1.0

!   Primera aproximación

            error = 2.*TOL

!   lazo de cálculos

    do while ((error.gt.TOL).and.(n.le.M))
                n = n + 1;
!   calculo del factorial
                fact=1
        do i=1,2*n-1
                  fact=fact*i
                enddo
                sum = sum + (signo * (x**(2*n-1))/fact)
                term = (signo * (x**(2*n-1))/fact)
               error = abs(sum-F)
                write(*,*)n,"  ",sum,"   ",error,"  ",term
                signo = (-1.)*signo
            enddo
```

Ejemplo: Fortran (3)

```
if(n.le.M) then
  write(*,*)"Se requieren", n," términos"
else
  write(*,*)"El método falló"
endif

end program Num_Ter_Seno
```

Representación de números en el computador

Palabra: unidad fundamental de representación de la información

- Arreglo de dígitos binarios o bits que son representados en la memoria de la computadora. Ejemplo: 1110101; 101110010
- Los números se almacenan típicamente en una palabra o más.

Sistema de números: manera de representar cantidades numéricas

–Base: número utilizado como referencia para construir el sistema

–Sistema de números más común es el decimal o en base 10.

Representación de números en el computador

10 símbolos (0-9) son utilizados para la representación de las cantidades

Ejemplo: $409 = 4 \bullet 10^2 + 0 \bullet 10^1 + 9 \bullet 10^0$

Sistemas de uso extendido: binario (base 2), hexadecimal (base 16)

Ejemplo Binario = 110

La equivalencia entre un sistema y otro se obtiene de manera sencilla

Ejemplos: $(110)_2 = 1*2^2 + 1*2^1 + 0*2^0 = 4+2+0 = (6)_{10}$

$(83)_{10} = 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = (1010011)_2$

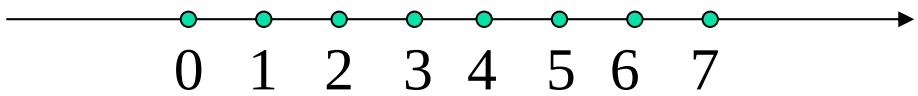
Representación de números en el computador

–Dependiendo de la cantidad de bits (o longitud de palabra) las máquinas tienen un limite para almacenar números enteros.

Ejemplo: máquina de siete bits, el limite es:
$(\pm 111111)_2 = (\pm\ 63)_{10}$

El conjunto de números enteros que pueden ser representados (obviando el signo) es entonces:

000	$0=0x2^2+0x2^1+0x2^0$	100	$4=1x2^2+0x2^1+0x2^0$
001	$1=0x2^2+0x2^1+1x2^0$	101	$5=1x2^2+0x2^1+1x2^0$
010	$2=0x2^2+1x2^1+0x2^0$	110	$6=1x2^2+1x2^1+0x2^0$
011	$3=0x2^2+1x2^1+1x2^0$	111	$7=1x2^2+1x2^1+1x2^0$



Representación de punto flotante

–El número es representado como una parte fraccional o mantisa (m) , y la base (b) elevada a un exponente (e):

$$n = m \cdot b^{e} \quad ,$$

– La mantisa es usualmente normalizada y escogida en el rango [0;1).

Ejemplo: 1/56 =0.01785714...se expresa como $0.1785*10^{1}$

No hay ceros a la derecha del punto decimal

–En sistema binario tendremos que por ejemplo

$$0.1010 = 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$$

$$(0.1010)_2 = \left[\left(\frac{1}{2}\right) + \left(\frac{1}{8}\right)\right]_{10} = (0.625)_{10}$$

O, convirtiendo de decimal a binario

$$12.875 = (8 + 4 + 0 + 0 + 0.5 + 0.25 + 0.125)_{10}$$

$$1x2^3 + 1x2^2 + 0x2^1 + 0x2^0 + 1x2^{-1} + 1x2^{-2} + 1x2^{-3}$$

$$1 \quad 1 \quad 0 \quad 0. \quad 1 \quad 1 \quad 1$$

$$1100.111$$

Representación de números en el computador

- Los limites de la representación están dados por la cantidad de bits disponible para almacenar números.
- Con N bits tenemos: 1 signo número, 1 signo del exponente, 2 ó 3 para el exponente y N-4 ó N-3 para la mantisa. Ejemplo: con 10 bits tenemos:

(a) ± ± EEMMMMMM ó (b) ± ± EEEMMMMM

- Estas consideraciones establecen no solo un limite sino el conjunto de números que se pueden representar
- Ejemplo: el conjunto de números que se puede representar con aritmética de 7 bits es: ± ± EEMMM

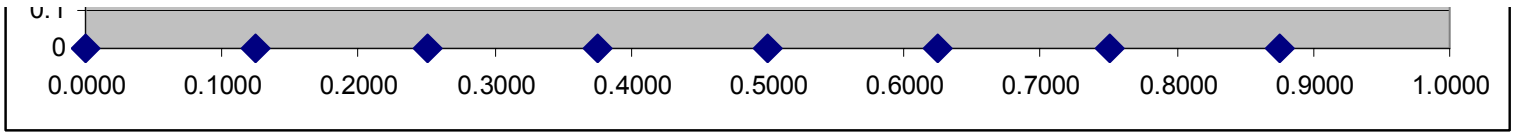
Representación de números en el computador

–Estas consideraciones establecen no solo un limite sino el conjunto de números que se pueden representar

–Ejemplo: el conjunto de números que se puede representar con aritmética de 7 bits es: ± ± EEMMM

–Veamos las mantisas que podemos representar con tres bits

.000	$0=0x2^{-1}+0x2^{-2}+0x2^{-3}$	.100	$0.5=1x2^{-1}+0x2^{-2}+0x2^{-3}$
.001	$0.125=0x2^{-1}+0x2^{-2}+1x2^{-3}$	.101	$0.625=1x2^{-1}+0x2^{-2}+1x2^{-3}$
.010	$0.25=0x2^{-1}+1x2^{-2}+0x2^{-3}$	.110	$0.75=1x2^{-1}+1x2^{-2}+0x2^{-3}$
.011	$0.375=0x2^{-1}+1x2^{-2}+1x2^{-3}$	.111	$0.875=1x2^{-1}+1x2^{-2}+1x2^{-3}$



–Luego, el número de números que podemos representar sobre la recta reales limitado y, pueden darse los siguientes casos:

(a)Se requiere un número mayor de bits

(b)El número no puede representarse ni aún utilizando un número infinito de bits (por ejemplo 1/3 en sistema decimal 0.333333....)

Como consecuencia de esto, usted verá que en algunos casos, los instrumentos que usted utiliza (programas o calculadoras) introducen o modifican dígitos durante las operaciones.

En algunas plataformas es posible incrementar el número de bits (doble precisión).

MECÁNICA COMPUTACIONAL I

Capítulo 1

Introducción a la Computación