Universidad del Valle
Escuela de Ingeniería de Sistemas y Computación
Métodos Números

Taller General

Temas:
Errores de Redondeo y Truncamiento
Raíces y Optimización
Sistemas Lineales y No Lineales
Ajuste de Curvas (Regresión e Interpolación)
Integración y Derivación
Scilab

Evaluación
En la evaluación del taller se tendran en cuenta los siguientes lineamientos:
Cada estudiante debe realizar su propio envio al juez virtual
Podran conformar grupos de trabajo para la solución de los ejercicios pero cada estudiante deberá ser capaz de sustentar los ejercicios de forma individual si asi se requiere
La sustentación tendrá un factor de 0 a 1 sobre la nota del taller.

Tener en cuenta
Para los ejercicios que realice debera tener en cuenta usar las plantillas que se indican. No se debe modificar el nombre de las variables ni las funciones. No se debe hacer uso de mensajes de texto con disp() o printf() en la solución final, si hace uso de ellos en la solución final deberan quedar comentados (//).

Ejercicios
T01P01. Crear un función que reciba una matriz y un escalar y retorne como resultado la multiplicación de la matriz por el escalar.

function matrizR = **multiplicarEscalar**(escalar, matrizA)
// Escriba su código aquí.
endfunction

T01P02. Crear un función que reciba un tiempo de inicio, un valor de stepsize (intervalo) y un tiempo final y retorne como resultado un vector de tiempos y un vector con la evaluación de la función seno para el vector de tiempos.

function [y, t] = **funcionSeno**(tinicio, intervalo, tfin)
// Escriba su código aquí
endfunction

Nota: y y **t** son vectores fila.

T01P03. Crear una función que reciba dos vectores y un entero que corresponde a una opción. Si la opción es 1 deberá retornar la suma elemento por elemento de los vectores, si la opción es 2 deberá retornar la resta elemento por elemento de los vectores y si la opción es 3 debera retornar la multiplicación elemento por elemento, si la opcion no es válida debera retornar vector = %nan.

Si se ingresan vectores de distinta dimensión la función debera retornar vector = %nan

function vector = **operarVectores**(a, b, opcion)
// Escriba su código aquí.
endfunction

T01P04. Crear una función que reciba dos vectores y retorne como resultado la distancia entre ellos. La distancia entre vectores se define como:

Sean a y b vectores en R^n donde $a=[a_1\ a_2\ ...a_n]$ y $b=[b_1\ b_2\ ...b_n]$. La distancia $d(a,b)$ entre a y b esta definida por:

$$d(a,b)=\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+...+(a_n-b_n)^2}$$

Si se ingresan vectores de distinta dimensión la función debera retornar distancia = %nan

function distancia = **calcularDistancia**(a, b)
// Escriba su código aquí.
endfunction

T01P05. Crear un función que reciba dos matrices y retorne como resultado la multiplicación de las matrices. Si las dimensiones de las matrices no permiten su multiplicación debera retornar como resultado %nan.

function matrizR = **multiplicarMatrices**(matrizA, matrizB)
// Escriba su código aquí.
endfunction

T01P06. La deflexión de una viga uniforme sujeta al incremento linealmente distribuido de carga puede ser representada por la ecuación:

$$y=\frac{w_0}{120EIL}(-x^5+2L^2x^3-L^4x)$$

Donde L = 600 cm, E = 50,000 kN/cm^2m, I = 30,000, cm^4 y w0 = 2.5 kN/cm.

Implementar una función para evaluar la ecuación anterior para valores de x. La función debe recibir un valor de x y debe retornar como resultado el valor de y.

// Funcion deflexionBarra
//function [y] = deflexionBarra(x)
//Entrada:
//x : valor en que la ecuacion es evaluada
//Salida:
//y : valor evaluado
function [y] = **deflexionBarra**(x)
// Escriba su código aquí.
endfunction

T01P07. Las funciones por tramos son útiles cuando la relación entre una variable dependiente y una variable independiente no puede ser representada adecuadamente por una sola ecuación. Por ejemplo la velocidad de un cohete puede ser descrita de la siguiente forma:

$$v(t)=10t^2-5t \quad 0\leqslant t\leqslant 8$$
$$v(t)=624-3t \quad 8<t\leqslant 16$$
$$v(t)=36t+12(t-16)^2 \quad 16<t\leqslant 26$$
$$v(t)=2136e^{-0.1(t-26)} \quad t>26$$
$$v(t)=0 \quad otherwise$$

Cree una función que reciba un valor de t y retorne como resultado el valor de v.

```
// Funcion por partes para obtener la velocidad de un cohete
//function v = obtenerVelocidad(t)
//Entrada:
//t : valor en que la funcion es evaluada
//Salida:
// v : velocidad del cohete en el instante t
function v = obtenerVelocidad(t)
// Escriba su código aquí.
endfunction
```

T02P01. Crear un función que solucione la ecuación diferencial de Newton para enfriamiento por medio del método de Euler. La función debera recibir una temperatura ambiente, una temperatura inicial del objeto de interés, una constante de proporcionalidad, un valor para el stepsize (intervalo) y un tiempo final. La función debera retornar dos vectores: un vector de tiempos y un vector de Temperaturas correspondiente a cada uno de los tiempos.

```
// Leyes de Newton de Enfriamiento con Temperatura del Ambiente Constante
// function [T, t] = cambioTemperatura(Tambiente, Tinicial, k, stepsize, tfinal)
// Entrada:
// Tambiente : Temperatura del ambiente en °C
// Tinicial : Temperatura inicial del objeto de interes en °C
// k : constante de proporcionalidad en 0.019/min
// Salida:
// t : vector de tiempo en minutos
// T : vector temperatura del objeto de interes
function [T, t] = cambioTemperatura(Tambiente, Tinicial, k, stepsize, tfinal)
// Escriba su código aquí.
endfunction
```

$$\frac{dT}{dt}=-k(T-T_a)$$

Nota: T es un vector columna y **t** es un vector fila.

T02P02. Crear un función que solucione la ecuación diferencial para el cambio de volumen en un tanque por medio del método de Euler. La función debera recibir un flujo de entrada, un valor de area superficial, un valor para el stepsize (intervalo) y un tiempo final. La función debera retornar dos vectores: un vector de tiempos y un vector de profundidades correspondientes a cada uno de los tiempos. Asuma como condición inicial y = 0 (El tanque esta medio lleno)

$$\frac{dy}{dt}=3\frac{Q}{A}\sin^2(t)-\frac{Q}{A}$$

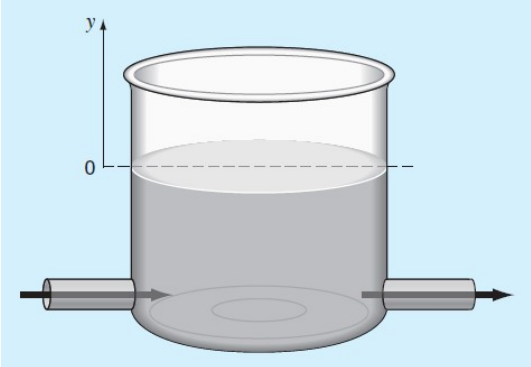


Gráfico Tanque

```
// Estimacion de estado de llenado de un tanque
// function [y,t] = llenadoTanque1(Q, A, stepsize, tfinal)
// Entrada:
// Q : flujo de entrada en m^3/s
// A : area de la superficie en m^2
// Salida:
// t : vector tiempo en segundos
// y : vector profundidad
function [y,t] = llenadoTanque1(Q, A, stepsize, tfinal)
// Escriba su código aquí.
endfunction
```

Nota: y es un vector columna y **t** es un vector fila.

T02P03. Crear un función que solucione la ecuación diferencial para la tasa de descomposición de un material radioactivo en un reactor. La función debera recibir una constante de descomposición, un valor para el stepsize (intervalo) y un tiempo final. La función debera retornar dos vectores: un vector de tiempos y un vector de concentraciones del material radioactivo correspondiente a cada uno de los tiempos.

$$\frac{dc}{dt} = -kc$$

```
// Estimación de la descomposición de un contaminante en un reactor
// function [c, t] = descomposicion(Cinicial, k, stepsize, tfinal)
// Entrada:
// Cinicial : Concentracion inicial del material radioactivo
// k : constante de proporcionalidad
// Salida:
// t : vector de tiempo en minutos
// c : vector concentracion del material radioactivo
function [c, t] = descomposicion(Cinicial, k, stepsize, tfinal)
// Escriba su código aquí.
endfunction
```

Nota: c es un vector columna y **t** es un vector fila.

T02P04. Crear un función que solucione la ecuación diferencial para el cambio de volumen en un tanque por medio del método de Euler. La función debera recibir un flujo de entrada, un valor de área superficial, una constante alfa, un valor para el stepsize (intervalo) y un tiempo final. La función debera retornar dos vectores: un vector de tiempos y un vector de profundidades correspondientes a cada uno de los tiempos. Asuma como condición inicial y = 0 (El tanque esta medio lleno)

$$\frac{dy}{dt} = 3\frac{Q}{A}\sin^2(t) - \frac{\alpha(1+y)^{1.5}}{A}$$

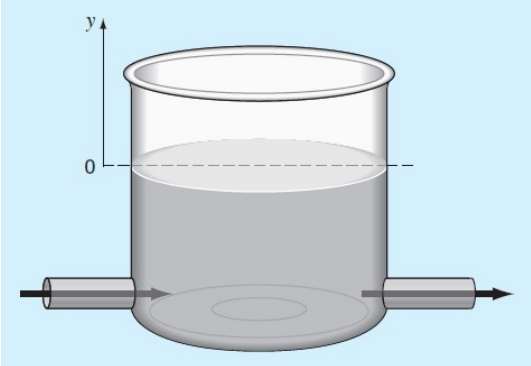


Gráfico Tanque

```
// Estimacion de estado de llenado de un tanque
// function [y,t] = llenadoTanque2(Q, A, alfa, stepsize, tfinal)
// Entrada:
// Q : flujo de entrada en m^3/s
// A : area de la superficie en m^2
// alfa: constante que relaciona la profundidad del tanque con el flujo de salida
// Salida:
// t : vector tiempo en segundos
// y : vector profundidad
function [y,t] = llenadoTanque2(Q, A, alfa, stepsize, tfinal)
// Escriba su código aquí.
endfunction
```

Nota: y es un vector columna y **t** es un vector fila.

T02P05. Crear una función que resuelva la siguiente ecuación diferencial empleando el método de Euler

$$\frac{dy}{dx} = \frac{\sqrt{y}}{2x+1} \quad \text{con} \quad y(0) = 4$$

```
// function [x, y] = obtenerSolucion(valorInicial, stepsize, xfinal)
// Entrada:
// valorInicial: valor inicial para x
// stepsize: tamaño de los pasos
// xfinal: valor final para x
```

// Salida:
// x: vector fila de valores de la variable independiente
// y: vector columna de valores de la variable dependiente
function [x, y] = **obtenerSolucion**(valorInicial, stepsize,xfinal)
// Escriba su código aquí.
endfunction

T02P06 (Opcional). La conservación del volumen indica: cambio en volumen = flujo entrada – flujo de salida

$$\frac{dV}{dt}=Q_{entrada}-Q_{salida}$$

para obtener la ecuación diferencial que representa el nivel de liquido en un tanque de almacenamiento cónico. El flujo de liquido de entrada y de salida es de:

$$Q_{entrada}=3\sin^2(t)$$

$$Q_{salida}=3(y-y_{salida})^{1.5} \quad y>y_{salida}$$

$$Q_{salida}=0 \quad y\leqslant y_{salida}$$

Por tanto:

$$\frac{dV}{dt}=3\sin^2(t)-3(y-y_{salida})^{1.5} \quad y>y_{salida}$$

$$\frac{dV}{dt}=3\sin^2(t) \quad y\leqslant y_{salida}$$

El volumen del cono es: $V=\frac{1}{3}\pi r^3 y$

Donde el flujo tiene unidades de m^3/d y y = elevacíón de la superficie de agua con relación al piso del tanque en metros.
La ecuación diferencial para el cambio de volumen por medio del método de Euler para encontrar el valor de y es:

$$\frac{d(1/3\pi r^3 y)}{dt}=3\sin^2(t)-3(y-y_{salida})^{1.5} \quad y>y_{salida}$$

$$\frac{dy}{dt}=\frac{3\sin^2(t)-3(y-y_{salida})^{1.5}}{1/3\pi r^3}$$

$$y_{(i+1)}=y_{(i)}+\frac{3\sin^2(t_{(i)})-3(y_{(i)}-y_{salida})^{1.5}}{1/3\pi r^3}(t_{(i+1)}-t_{(i)})$$

$$\frac{d(1/3\pi r^3 y)}{dt}=3\sin^2(t) \quad y\leqslant y_{salida}$$

$$\frac{dy}{dt}=\frac{3\sin^2(t)}{1/3\pi r^3}$$

$$y_{(i+1)}=y_{(i)}+\frac{3\sin^2(t_{(i)})}{1/3\pi r^3}(t_{(i+1)}-t_{(i)})$$

Observando el gráfico del tanque se puede deducir la siguiente relación de triangulos:

$$\frac{r}{y}=\frac{r_{top}}{y_{top}}$$

$$r=y\left(\frac{r_{top}}{y_{top}}\right)$$

$$r=y_{(i)}\left(\frac{r_{top}}{y_{top}}\right)$$

Implementar una función que solucione la ecuación diferencial para el cambio de volumen por medio del método de Euler para encontrar el valor de y en un intervalo de tiempo. La función debera recibir un valor de radio superficial (rtop), un valor para la altura del tanque (ytop), un valor para la altura de la tuberia de salida (yout), el stepsize (intervalo) y un tiempo final (tfinal). La función debera retornar dos vectores: un vector de tiempos (t) y un vector de profundidades (y) correspondientes a cada uno de los tiempos. Asuma que el nivel y es inicialmente por debajo del tubo de salida y tiene un valor inicial de $y(0)=0.8\text{m}$

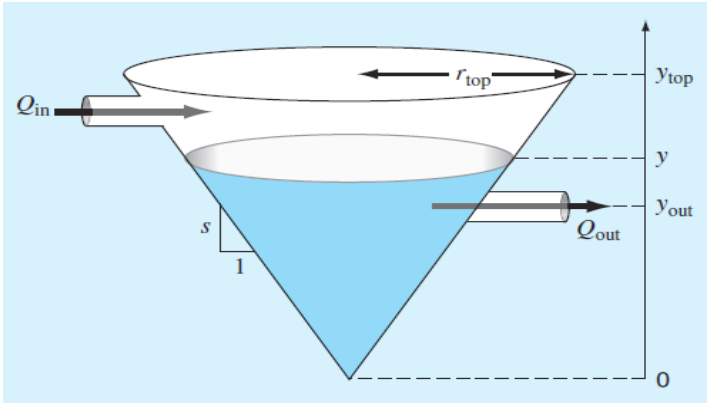


Gráfico Tanque

```
// Estimacion de estado de llenado de un tanque cónico
// function [y,t] = llenadoTanque3(rtop, ytop, yout, stepsize, tfinal)
// Entrada:
// Q : flujo de entrada en m^3/s
// A : area de la superficie en m^2
// Salida:
// t : vector tiempo en segundos
// y : vector profundidad
function [y,t] = llenadoTanque3(rtop, ytop, yout, stepsize, tfinal)
// Escriba su código aquí.
endfunction
```

Nota: y es un vector columna y **t** es un vector fila.

T03P01. Crear un función que estime e^x por medio de la serie de maclaurin. La función debera recibir un valor de x y un valor que corresponde a la cantidad de términos a emplear de la serie. La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones.

```
//Serie de Maclaurin de la funcion exponencial
//function [et, ea, aprox] = funcionExpTaylor(x, iter)
//Entrada:
//x : valor en que la serie sera evaluada
//iter : cantidad de terminos de la serie
//Salida:
```

```
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximados de la serie
function [et, ea, aprox] = funcionExpTaylor(x, iter)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** sin valor absoluto.
El primer dato del vector **ea** debe ser igual a %nan.

T03P02. Crear un función que estime sen(x) por medio de la serie de maclaurin. La función debera recibir un valor de x y un valor que corresponde a la cantidad de términos a emplear de la serie. La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones.

```
//Serie de Maclaurin de la funcion seno
//function [et, ea, aprox] = funcionSenoTaylor(x, iter)
//Entrada:
//x : valor en que la serie sera evaluada
//iter : cantidad de terminos de la serie
//Salida:
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximado de la serie
function [et, ea, aprox] = funcionSenoTaylor(x, iter)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** sin valor absoluto.
El primer dato del vector **ea** debe ser igual a %nan.

T03P03. Crear un función que estime cos(x) por medio de la serie de maclaurin. La función debera recibir un valor de x y un valor que corresponde a la cantidad de términos a emplear de la serie. La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones.

```
//Serie de Maclaurin de la funcion coseno
//function [et, ea, aprox] = funcionCosenoTaylor(x, iter)
//Entrada:
//x : valor en que la serie sera evaluada
//iter : cantidad de terminos de la serie
//Salida:
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximados de la serie
function [et, ea, aprox] = funcionCosenoTaylor(x, iter)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** sin valor absoluto.
El primer dato del vector **ea** debe ser igual a %nan.

T03P04. Crear un función que estime $\frac{1}{1-x}$ para $|x|<1$ por medio de la serie de maclaurin. La función debera recibir un valor de x y un valor que corresponde a la cantidad de términos a emplear de la serie. La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones.

```
//Serie de Maclaurin de la serie geométrica
//function [et, ea, aprox] = funcionSgMaclaurin(x, iter)
//Entrada:
//x : valor en que la serie sera evaluada
//iter : cantidad de terminos de la serie
//Salida:
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximado de la serie
function [et, ea, aprox] = funcionSgMaclaurin(x, iter)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** sin valor absoluto.
El primer dato del vector **ea** debe ser igual a %nan.

T03P05. Crear un función que estime $\ln(1+x)$ para $|x|<1$ por medio de la serie de maclaurin. La función debera recibir un valor de x y un valor que corresponde a la cantidad de términos a emplear de la serie. La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones.

```
//Serie de Maclaurin de la funcion logaritmo natural
//function [et, ea, aprox] = funcionLnMaclaurin(x, iter)
//Entrada:
//x : valor en que la serie sera evaluada
//iter : cantidad de terminos de la serie
//Salida:
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximados de la serie
function [et, ea, aprox] = funcionLnMaclaurin(x, iter)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** sin valor absoluto.
El primer dato del vector **ea** debe ser igual a %nan.

T03P06. Crear un función que estime cos(x) por medio de la serie de maclaurin. La función debera recibir un valor de x (x), un valor limite para el error de aproximación (eads: entero indicando el porcentaje de error) y la cantidad maxima de iteraciones (maxit: la cantidad máxima de iteraciones, el programa termina si no se alcanza el error deseado en la cantidad máxima de iteraciones). La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones, tambien se debe retornar la cantidad de iteraciones realizadas para alcanzar el error deseado.

```
//Serie de Maclaurin de la funcion coseno
//function [et, ea, aprox, niter] = funcionCosenoTaylor2(x, eads, maxit)
//Entrada:
//x : valor en que la serie sera evaluada
// eads: valor limite para el error de aproximacion
//maxit : cantidad maxima de iteraciones
//Salida:
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximados de la serie
//niter : cantidad de iteraciones realizadas para alcanzar el error deseado
function [et, ea, aprox, niter] = funcionCosenoTaylor2(x, eads, maxit)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** en valor absoluto.
El primer dato del vector ea debe ser igual a %nan.

T03P07. Crear un función que estime $\tan(x)$ para $|x|<\frac{\pi}{2}$ por medio de la serie de maclaurin. La función debera recibir un valor de x y un valor que corresponde a la cantidad de términos a emplear de la serie (menor al tamaño de términos de Bernoulli disponibles). La función debera retornar tres vectores: un vector con el valor de los errores relativos y un vector con el valor de los errores aproximados y un vector con el valor de las aproximaciones. Para los términos pares de Bernoulli define un vector por dentro de la función de por lo menos 12 términos. Si lo desea puede hacer uso de la función factorial de Scilab.

```
//Serie de Maclaurin de la Tangente
//function [et, ea, aprox] = funcionTangenteMaclaurin(x, iter)
//Entrada:
//x : valor en que la serie sera evaluada
//iter : cantidad de terminos de la serie
//Salida:
//et : vector de errores relativos
//ea : vector de errores aproximados
//aprox : vector con valores aproximado de la serie
function [et, ea, aprox] = funcionTangenteMaclaurin(x, iter)
// Escriba su código aquí.
endfunction
```

Nota:
et, **ea** y **aprox** son vectores columnas.
et y **ea** sin valor absoluto.
El primer dato del vector **ea** debe ser igual a %nan.

T04P01. Crear una funcion que reciba un valor de punto flotante en el formato IEEE-754 de 32 bits y retorne como resultado la conversión en formato decimal. Tenga en cuenta los casos especiales vistos en clase.

```
// Convierte un numero de punto flotante en formato IEEE-754 de 32 bits a decimal
//function valor = convertir(signo, mantisa, exponente)
// Entrada:
// signo : valor entero 0 ó 1
// mantisa : string de 23 caracteres '0' ó '1'
// exponente: string de 8 caracteres '0' ó '1'
// Salida:
// valor: valor en formato decimal
function valor = convertir(signo, mantisa, exponente)
// Escriba su código aquí.
endfunction
```

T04P02. Crear una funcion que reciba un valor de punto flotante en formato decimal y retorne como resultado la conversión en formato IEEE-754 de 32 bits. **NO** tenga en cuenta los casos especiales vistos en clase.

```
// Convierte un numero decimal a punto flotante en formato IEEE-754 de 32 bits
//function [signo,mantisa,exponente] = convertirIEEE(valor)
// Salida:
// signo : string 0 ó 1
// mantisa : string de 23 caracteres '0' ó '1'
// exponente: string de 8 caracteres '0' ó '1'
// Entrada:
// valor: valor en formato decimal
function [signo,mantisa,exponente]= convertirIEEE(valor)
// Escriba su código aquí.
endfunction
```

T05P01. La velocidad hacia arriba de un cohete puede ser calculada por medio de la siguiente formula:

$$v = u \ln\left(\frac{m_0}{m_0 - qt}\right) - g\,t$$

Donde v es la velocidad hacia arriba, u es la velocidad en que el combustible es expulsado en relación con el cohete, m_0 es la masa incial del cohete en el tiempo $t=0$, q es la tasa de consumo de combustible y g es la aceleración de la gravedad (asumirla como constante $9.81\,m/s^2$). Si u =1800 m/s, $m_0 = 160{,}000\,kg$ y $q = 2600\,kg/s$.

Expresar la función como un problema de raices que permita visualizar el tiempo en el cual $v = 750\,m/s$

```
// function fx = tiempoCohete(t)
// Entrada:
// t: vector de tiempos
// Salida:
// fx: vector columna con los valores de la funcion evaluados en el vector de tiempos.
```

function fx = **tiempoCohete**(t)
// Escriba su código aquí.
endfunction

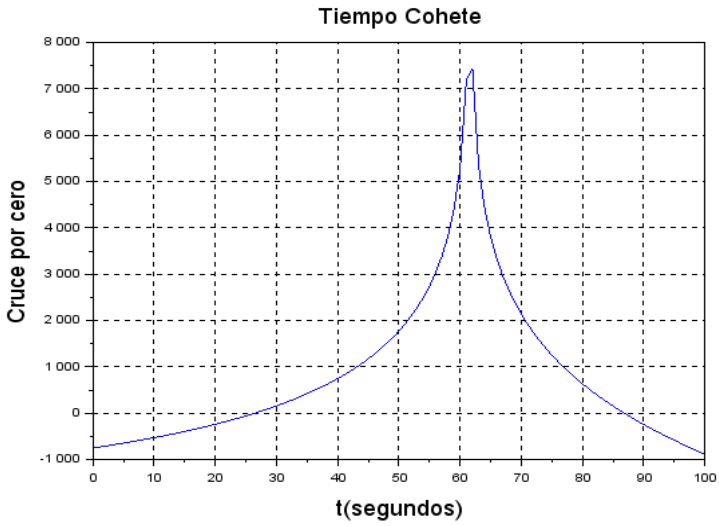


Grafico de la función para el vector de tiempo t=0:1:100.
Nota: Este gráfico permite seleccionar los valores iniciales (x_l, x_u) para los algoritmos de cálculo de las raíces.

T05P02. Se desea diseñar un tanque esferico para guardar agua para una pequeña población. El volumen del liquido que puede guardar el tanque y su relación con la profundidad de agua en el tanque se calcula por medio de la siguiente fórmula:

$$V = \pi h^2 \frac{(3\mathrm{R} - h)}{3}$$

Donde V es el volumen en $[m^3]$, h es la profundidad de agua en el tanque $[m]$ y R el radio del tanque en $[m]$.

Expresar la función como un problema de raices que permita visualizar la profundidad en la cual $V = 30\,m^3$ con $R = 3\mathrm{m}$

// function fx = profundidadTanque(h)
// Entrada:
// h: vector de profundidades
// Salida:
// fx: vector columna con los valores de la funcion evaluados en el vector de profundidades.
function fx = **profundidadTanque**(h)
// Escriba su código aquí.
endfunction

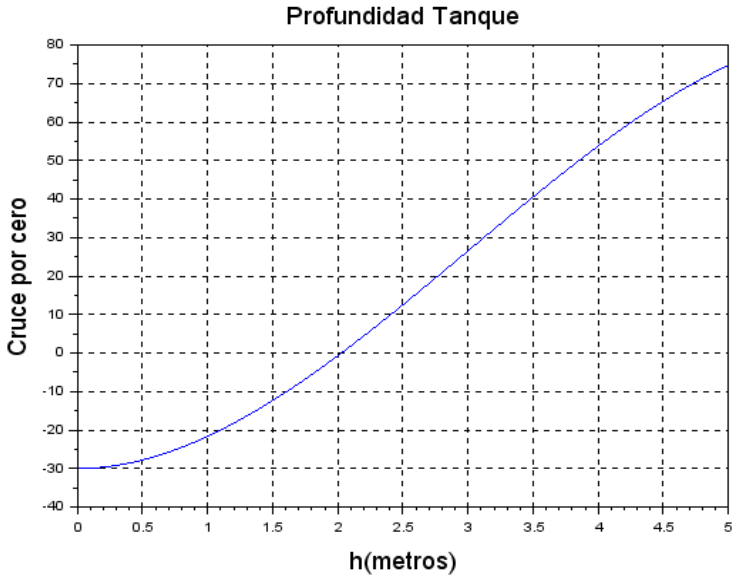


Grafico de la función para el vector de profundidad h=0:0.01:5.
Nota: Este gráfico permite seleccionar el valor inicial (x_r) para los algoritmos de cálculo de las raíces.

T05P03. Varios campos de la ingeniería requieren una estimación lo mas exacta posible de la población. Por ejemplo, en relación con el transporte es necesario determinar por separado las tendencias en crecimiento de la población en las áreas urbana y suburbana.

La población en el área urbana decrece con el tiempo de acuerdo con:

$$P_u(t) = P_{u,max}\, e^{-k_u t} + P_{u,min}$$

La población en el área suburbana crece de acuerdo con:

$$P_s(t) = \frac{P_{s,max}}{1 + (P_{s,max}/P_0 - 1)\, e^{-k_s t}}$$

Crear una función (problema de raices) a partir de las dos anteriores que permita visualizar el tiempo y los valores respectivos para P_u y P_s cuando el área suburbana es 20% superior al área urbana. Los valores de los parametros son: $P_{u,max} = 80{,}000$ personas, $k_u = 0.05/yr$, $P_{u,min} = 110{,}000$ personas, $P_{s,max} = 320{,}000$ personas, $P_0 = 10{,}000$ personas, $k_s = 0.09/yr$.

```
// function Purb = funcion1(t)
// Entrada:
// t: vector de tiempos
// Salida:
// PUrb: vector con los valores de la funcion evaluados en el vector de tiempos.
function Purb = funcion1(t)
// Escriba su código aquí.
endfunction

// function Psub = funcion2(t)
// Entrada:
// t: vector de tiempos
```

```
// Salida:
// PSub: vector con los valores de la funcion evaluados en el vector de tiempos.
function Psub = funcion2(t)
// Escriba su código aquí.
endfunction

// funcion que expresa la relación "cuando el área suburbana es 20% superior al área urbana"
// function fx = tiempoPoblacion(t)
// Entrada:
// t: vector de tiempos
// Salida:
// fx: vector columna con los valores de la funcion evaluados en el vector de tiempos.
function fx = tiempoPoblacion(t)
// Escriba su código aquí.
endfunction
```

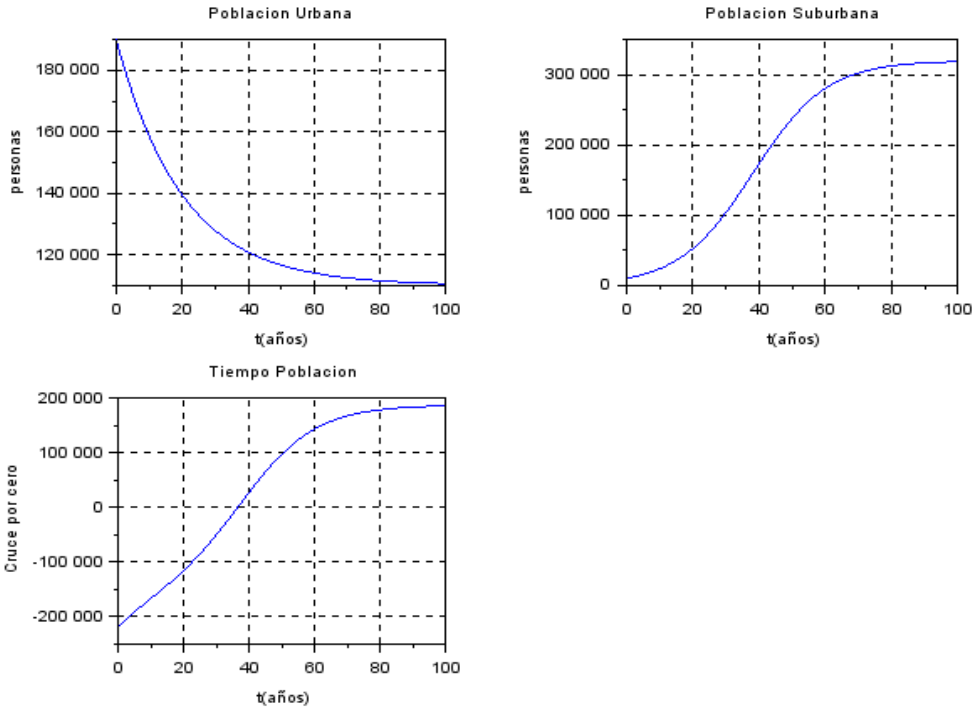
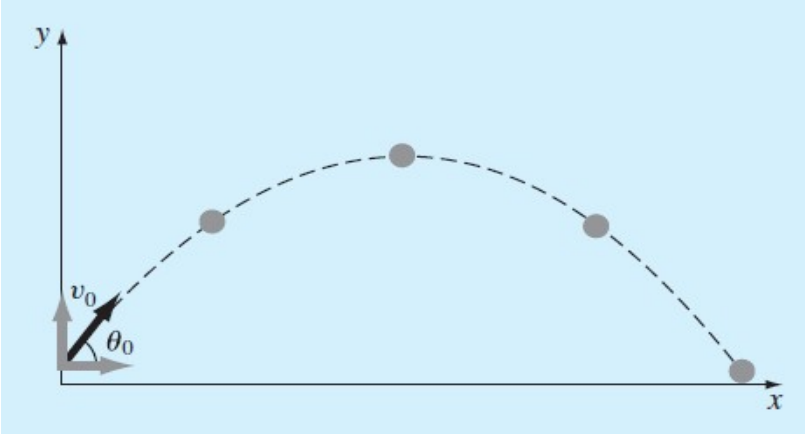


Grafico de las funciones: funcion1, funcion2 y tiempoPoblacion para el vector de tiempo t=0:1:100.
Nota: Este gráfico permite seleccionar los valores iniciales (x_l, x_u) para los algoritmos de cálculo de las raíces.

T05P04. Los ingenieros aeroespaciales calculan la trayectoria de proyectiles como por ejemplo los cohetes. Un problema relacionado trata con la trayectoria de una bola. La trayectoria de una bola arrojada por una persona esta definida por las coordenadas (x,y). La trayectoria puede ser modelada como:

$$y = \tan(\theta_0)\, x - \frac{g}{2 * v_0^2 \cos^2 \theta_0} x^2 + y_0$$



Coordenadas (x,y) de la Trayectoria de una Bola

Expresar la función como un problema de raices que permita visualizar el angulo inicial apropiado θ_0 para el cual la altura $y=1\,m$ con $v_0=30\,m/s$, la distancia al receptor $x=90\,m$ y la bola abandola la mano del lanzador a una altura de $y_0=1.8\,m$.

```
// function fx = anguloProyectil(theta0)
// Entrada:
// theta0: vector de angulos
// Salida:
// fx: vector columna con los valores de la funcion evaluados en el vector de profundidades.
function fx = anguloProyectil(theta0)
// Escriba su código aquí.
endfunction
```

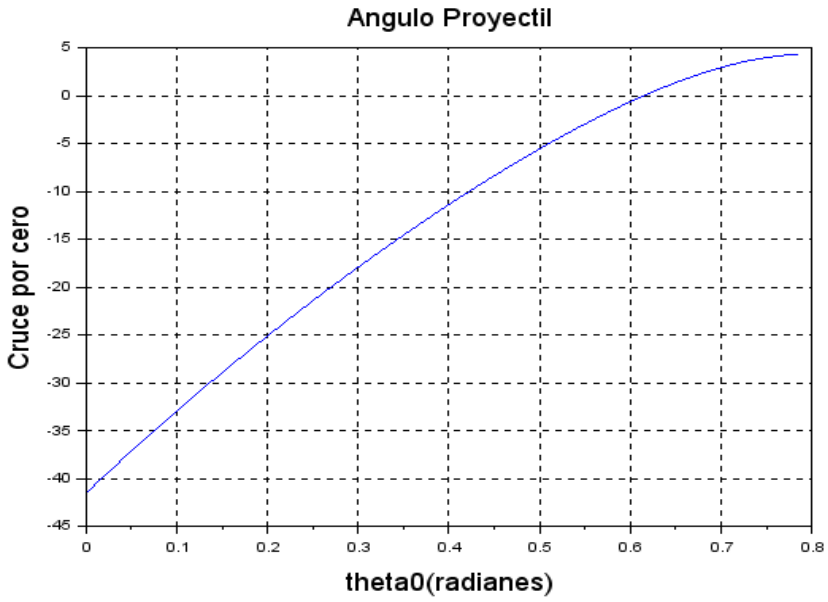


Grafico de la función para el vector de angulos theta0=0:%pi/180:%pi/4.
Nota: Este gráfico permite seleccionar el valor inicial (x_r) para los algoritmos de cálculo de las raíces.

T05P05. Crear un función que implemente el método de Newton para **optimización** con diferencias finitas (Este método es similar al método de la secante para encontrar raíces). Este método se

caracteriza por alcanzar un valor óptimo en pocas iteraciones aunque tambien puede diverger. Las fórmulas para implementar el método se presentan a continuacíón:

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

$$f'(x) = \frac{f(x_1 + \delta x_1) - f(x_1 - \delta x_1)}{2\delta x_1}$$

$$f''(x) = \frac{f(x_1 + \delta x_1) - 2f(x_1) + f(x_1 - \delta x_1)}{(\delta x_1)^2}$$

La función debe recibir como parametros: la función a evaluar (funcion), la fracción de perturbación (pt), un valor inicial de estimación de la raiz (xr), un valor limite para el error de aproximación (eads: entero indicando el porcentaje de error) y la cantidad maxima de iteraciones (maxit: la cantidad máxima de iteraciones permite que en caso de error). La función debe retornar los valores de la aproximación actual (xr) y la aproximación anterior (xrold), el error aproximado y la cantidad de iteraciones realizadas (niter) para alcanzar el error deseado.

```
// function [xrold, xr, ea, niter] = NewtonOptimo(funcion, pt, xr, eads, maxit)
// Entrada:
// funcion: funcion a evaluar su optimo
// pt: fraccion de perturbacion
// xr: valor inicial
// eads: error deseado
// maxit: cantidad maxima de iteraciones a realizar
// Salida:
// xrold: valor anterior al optimo encontrado
// xr: valor optimo encontrado
// ea: valor del error de aproximacion
// niter: cantidad de iteraciones realizadas para alcanzar el error deseado.
function [xrold, xr, ea, niter] = NewtonOptimo(funcion, pt, xr, eads, maxit)
// Escriba su código aquí.
endfunction
```

T05P06. Crear un función que implemente el método de la interpolación parabólica. La función debe recibir como parametros: la función a evaluar (funcion), tres valores de entrada para el algoritmo de interpolación (x1,x2,x3), el error deseado (eads) y la cantidad máxima de iteraciones (maxit). La función debe retornar el valor del punto minimo ó máximo (x4), el error aproximado (ea) y la cantidad de iteraciones realizadas (niter).

```
// Algoritmo de interpolacion parabolica
// function [x4, ea, niter] = interpolacionParabolica(funcion, x1, x2, x3, eads, maxit)
// Entrada:
// funcion: funcion a encontrar el minimo o maximo
// x1: valor de entrada para el algoritmo de interpolacion
// x2: valor de entrada para el algoritmo de interpolacion
// x3: valor de entrada para el algoritmo de interpolacion
// eads: error deseado
// maxit: cantidad maxima de iteraciones
```

```
// Salida:
// x4: valor optimo
// ea: error aproximado en la iteracion niter
// niter: cantidad de iteraciones para alcanzar el error aproximado retornado
function [x4, ea, niter] = interpolacionParabolica(funcion, x1, x2, x3, eads, maxit)
// Escriba su código aquí.
endfunction
```

T05P07. Crear una función que implemente el método de bisección. La función debe recibir como parametros: la función a evaluar (funcion), el rango a evaluar (xl y xu) y el error deseado (eads: entero indicando el diferencial de error). El error deseado se debe emplear para obtener el numero de iteraciones a realizar de acuerdo a la siguiente formula: $n = \log_2(\frac{(\Delta x^0)}{E_{(a,d)}})$

La función debe retornar como parametro los ultimos valores del rango (xl y xu), el valor de estimación de la raíz (xr), el valor del error aproximado (ea: entero indicando el porcentaje de error) y la cantidad de iteraciones realizadas (niter).

```
// function [xl, xu, xr, ea, niter] = biseccion2(funcion, xl, xu, eads)
// Entrada:
// funcion: funcion a evaluar su optimo
// xl: valor del limite inferior
// xu: valor limite superior
// eads: error deseado
// Salida:
// xl: ultimo valor del limite inferior
// xu: ultimo valor del limite superior
// xr: valor optimo encontrado
// ea: valor del error de aproximacion
// niter: cantidad de iteraciones realizadas para alcanzar el error deseado.
function [xl, xu, xr, ea, niter] = biseccion2(funcion, xl, xu, eads)
// Escriba su código aquí.
endfunction
```

T05P08. Crear una función que implemente el método de falsa posición. La función debe recibir como parametros: la función a evaluar (funcion), el rango a evaluar (xl y xu), el error deseado (eads: entero indicando el diferencial de error) y la cantidad máxima de iteraciones (maxit). La función debe retornar como parametro los ultimos valores del rango (xl y xu), el valor de estimación de la raíz (xr), el valor del error aproximado (ea: entero indicando el porcentaje de error) y la cantidad de iteraciones realizadas (niter).

```
// function [xl, xu, xr, ea, niter] = falsaposicion2(funcion, xl, xu, eads, maxit)
// Entrada:
// funcion: funcion a evaluar su optimo
// xl: valor del limite inferior
// xu: valor limite superior
// eads: error deseado
// Salida:
// xl: ultimo valor del limite inferior
// xu: ultimo valor del limite superior
// xr: valor optimo encontrado
// ea: valor del error de aproximacion
```

// niter: cantidad de iteraciones realizadas para alcanzar el error deseado.
function [xl, xu, xr, ea, niter] = **falsaposicion2**(funcion, xl, xu, eads, maxit)
// Escriba su código aquí.
endfunction

T05P09. Crear un función que implemente el método de Newton-Raphson para **raíces**. La función debe recibir como parametros: la función a evaluar (funcion), la derivada de la función (dfuncion), un valor inicial de estimación de la raiz (xr), un valor limite para el error de aproximación (eads: entero indicando el porcentaje de error) y la cantidad maxima de iteraciones (maxit). La función debe retornar los valores de la aproximación actual (xr) y la aproximación anterior (xrold), el error aproximado y la cantidad de iteraciones realizadas.

// function [xrold, xr, ea, niter] = newraphraiz(funcion, dfuncion, xr, eads, maxit)
// Entrada:
// funcion: funcion a evaluar su optimo
// pt: fraccion de perturbacion
// xr: valor inicial
// eads: error deseado
// maxit: cantidad maxima de iteraciones a realizar
// Salida:
// xrold: valor anterior al optimo encontrado
// xr: valor optimo encontrado
// ea: valor del error de aproximacion
// niter: cantidad de iteraciones realizadas para alcanzar el error deseado.
function [xrold, xr, ea, niter] = **newraphraiz**(funcion, dfuncion, xr, eads, maxit)
// Escriba su código aquí.
endfunction

T05P10. Crear un función que implemente el método de la secante. La función debe recibir como parametros: la función a evaluar (funcion), la fracción de perturbación (pt), un valor inicial de estimación de la raiz (xr), un valor limite para el error de aproximación (eads: entero indicando el porcentaje de error) y la cantidad maxima de iteraciones (maxit). La función debe retornar los valores de la aproximación actual (xr) y la aproximación anterior (xrold), el error aproximado y la cantidad de iteraciones realizadas.

// function [xrold, xr, ea, niter] = secante2(funcion, pt, xr, eads, maxit)
// Entrada:
// funcion: funcion a evaluar su optimo
// pt: fraccion de perturbacion
// xr: valor inicial
// eads: error deseado
// maxit: cantidad maxima de iteraciones a realizar
// Salida:
// xrold: valor anterior al optimo encontrado
// xr: valor optimo encontrado
// ea: valor del error de aproximacion
// niter: cantidad de iteraciones realizadas para alcanzar el error deseado.
function [xrold, xr, ea, niter] = **secante2**(funcion, pt, xr, eads, maxit)
// Escriba su código aquí.
endfunction

T06P01. Crear un función que reciba una matriz de coeficientes A (nxn) y un vector de constantes b (nx1), y retorne como resultado un vector con los valores desconocidos x (nx1) y una constante con el número de condición de la matriz de coeficientes. Si la dimension de las filas o columnas de la matriz A es distinta a la dimension del vector b retorne como resultado %nan.

```
// Calcula la respuesta de un sistema de ecuaciones lineales
// function [vectorx, condicion] = obtenerRespuesta(matrizA, vectorb)
// Entrada:
// matrizA: matriz de coeficientes de dimension nxn
// vectorb: vector de constantes de dimension nx1
// Salida:
// vectorx: vector de valores desconocidos de dimension nx1
// condicion: numero de condición
function [vectorx, condicion] = obtenerRespuesta(matrizA, vectorb)
// Escriba su código aquí.
endfunction
```

T06P02. Crear un función que reciba una matriz A (nxm) y retorne como resultado la inversa/pseudoinversa de la matriz, si la matriz A no tiene inversa debera retornar como resultado %nan.

```
// Calcula la inversa/pseudoinversa de una matriz
// function matrizR =obtenerInversa(matrizA)
// Entrada:
// matrizA : matriz de dimensiones nxm
// Salida:
// matrizR: matriz inversa/pseudoinversa
function matrizR = obtenerInversa(matrizA)
// Escriba su código aquí.
endfunction
```

T06P03. Crear un función que reciba una matriz A (nxn) simétrica y retorne como resultado la factorización de cholesky (matriz U), si la matriz A no es simétrica debera retornar como resultado %nan.

```
// Calcula la factorización de cholesky de una matriz
// function matrizU = obtenerFactorizacion(matrizA)
// Entrada:
// matrizA: matriz de dimensiones nxn
// Salida:
// matrizU: matriz U resultado de la factorización
function matrizU = obtenerFactorizacion(matrizA)
// Escriba su código aquí
endfunction
```

T06P04. Crear un función que reciba dos vectores a y b y un valor entero p, y retorne como resultado la norma-p de la resta de los vectores. El valor de p corresponde al valor de la norma a calcular. Si los vectores a y b son de diferente tamaño retorne como resultado %nan

```
// Calcular la norma-p de la resta de dos vectores
// function norma = obtenerNorma(vectora, vectorb, p)
// Entrada:
// vectora: vector de dimensiones nx1
// vectorb: vector de dimensiones nx1
```

// Salida:
// norma: valor que corresponde a la norma de la diferencia entre los vectores de entrada
function norma = **obtenerNorma**(vectora, vectorb, p)
// Escriba su código aquí
endfunction

T07P01. Crear una función que implemente el método de Newton-Raphson para la solución de sistemas no lineales de dos ecuaciones con dos incógnitas. La funcion debe recibir dos ecuaciones en forma de funciones **u** y **v**, un vector **x0** con la solución inicial, un valor **es** que corresponde al criterio de parada para el error y un número máximo de iteraciones **maxit**. La función debe retornar el vector solución **x**, un vector **f** con la evaluación en las funciones **u** y **v** del vector **x**, el valor del error de aproximación **ea** y el número de iteraciones **iter**. El valor para la fracción de perturbación para la estimación de las derivadas parciales debe ser 0.000001.

// Calcula el jacobiano para un sistema de dos ecuaciones con dos incognitas.
//Entrada:
// u : ecuación uno como función
// v : ecuación dos como función
// x : vector con la solución
// Salida
// J : Jacobiano
// f : vector de las funciones u, v y w evaluadas en x
function [J, f]=**jacob**(u, v, x)
// Escriba su código aquí
endfunction

// Método de Newton-Raphson para la solución de sistemas de ecuaciones no lineales
// function [x,f,ea,iter]=NewtonRaphson1(u, v, x0, es, maxit)
// Entrada:
// u : ecuación uno como función
// v : ecuación dos como función
// x0 : vector con la solución inicial
// es : criterio de parada para el error
// maxit : número máximo de iteraciones
// Salida:
// x : vector solucion
// f : vector con la evaluación de x en las funciones u y v
// ea : error de aproximación como porcentaje
// iter : número de iteraciones realizadas
function [x,f,ea,iter]=**NewtonRaphson1**(u, v, x0, es, maxit)
// Escriba su código aquí
endfunction

Tome como base la sección 12.2.2 del libro Applied Numerical Methods with MATLAB for Engineers and Scientists del autor Steven Chapra y el caso de estudio del capítulo.

T07P02. Crear una función que implemente el método de Newton-Raphson para la solución de sistemas no lineales de tres ecuaciones con tres incógnitas. La funcion debe recibir tres ecuaciones en forma de funciones **u**, **v** y **w** un vector **x0** con la solución inicial, un valor **es** que corresponde al criterio de parada para el error y un número máximo de iteraciones **maxit**. La función debe retornar el vector solución **x**, un vector **f** con la evaluación en las funciones **u**, **v** y **w** del vector **x**, el valor del error de aproximación **ea** y el número de iteraciones **iter**. El valor para la fracción de perturbación

para la estimación de las derivadas parciales debe ser 0.000001.

```
// Calcula el jacobiano para un sistema de tres ecuaciones con tres incognitas.
//Entrada:
// u : ecuación uno como función
// v : ecuación dos como función
// w: ecuación tres como función
// x : vector con la solución
// Salida
// J : Jacobiano
// f : vector de las funciones u, v y w evaluadas en x
function [J, f]=jacob(u, v, w, x)
// Escriba su código aquí
endfunction

// Método de Newton-Raphson para la solución de sistemas de ecuaciones no lineales
// function [x,f,ea,iter]=NewtonRaphson2(u, v, w, x0, es, maxit)
// Entrada:
// u : ecuación uno como función
// v : ecuación dos como función
// w: ecuación tres como función
// x0 : vector con la solución inicial
// es : criterio de parada para el error
// maxit : número máximo de iteraciones
// Salida:
// x : vector solucion
// f : vector con la evaluación de x en las funciones u, v y w
// ea : error de aproximación como porcentaje
// iter : número de iteraciones realizadas
function [x,f,ea,iter]=NewtonRaphson2(u, v, w, x0, es, maxit)
// Escriba su código aquí
endfunction
```

T07P03 (Opcional). Valide la respuesta del ejercicio anterior por medio de la herramienta SOLVER de LibreOffice Calc. En un archivo comprimido incluya una captura de pantalla de la configuración de SOLVER y la hoja de calculo usada para encontrar la respuesta. Envie el archivo comprimido por medio del campus virtual.

T08P01. Crear una función que implemente el método de Gauss-Seidel. La función debe recibir una matriz de coeficientes **A** (coeficientes del sistema de ecuaciones), un vector columna **b**, un valor **es** que corresponde a un criterio de parada para el error y un número máximo de iteraciones **maxit**. La función debe terminar si se alcanza el número máximo de iteraciones **maxit** o si se alcanza un error menor o igual a **es**. La función debe retornar el valor del vector solución **x**, el valor del error de aproximación **ea** y el número de iteraciones **iter**.

```
// Método de Gauss-Seidel para la solución de sistemas lineales
// function [x, ea, iter] = GaussSeidel(A, b, es, maxit)
// Entrada:
// A : Matrix de coeficientes
// b: vector columna
// es : criterio de parada para el error
// maxit : número máximo de iteraciones
// Salida:
```

```
// x : vector columna con la solución
// ea : error de aproximación como porcentaje
// iter : número de iteraciones realizadas
function [x, ea, iter] = GaussSeidel(A, b, es, maxit)
// Escriba su código aquí
endfunction
```

T08P02. Crear una función que implemente el método de Gauss-Seidel con relajación. La función debe recibir una matriz de coeficientes **A** (coeficientes del sistema de ecuaciones), un vector columna **b**, un valor **lambda** que corresponde al coeficiente de relajación, un valor **es** que corresponde al criterio de parada para el error y un número máximo de iteraciones **maxit**. La función debe terminar si se alcanza el número máximo de iteraciones **maxit** o si se alcanza un error menor o igual a **es**. La función debe retornar el valor del vector solución **x**, el valor del error de aproximación **ea** y el número de iteraciones **iter**.

```
// Método de Gauss-Seidel para la solución de sistemas lineales
// function [x, ea, iter] = GaussSeidelR(A, b, lambda, es, maxit)
// Entrada:
// A : Matrix de coeficientes
// b: vector columna
// lambda : coeficiente de relajación
// es : criterio de parada para el error
// maxit : número máximo de iteraciones
// Salida:
// x : vector columna con la solución
// ea : error de aproximación como porcentaje
// iter : número de iteraciones realizadas
function [x, ea, iter] = GaussSeidelR(A, b, lambda, es, maxit)
// Escriba su código aquí
endfunction
```

T09P01. Crear una función que reciba un vector **y**, y retorne como resultado el **promedio**, la **mediana** y al **desviación estandar**.

```
// Calcula estadísticas a partir de un vector de datos
// function [promedio, mediana, desv] = obtenerEstadisticas(y)
// Entrada:
// y: vector de valores registrados a partir de un experimento
// Salida:
// promedio: promedio del vector de datos de entrada y
// mediana: mediana del vector de datos de entrada y
// desv: desviación estándar del vector de datos de entrada y
function [promedio, mediana, desv] = obtenerEstadisticas(y)
// Escriba su código aquí.
endfunction
```

T09P02. Crear un función que reciba un vector **x** con los datos de la variable independiente y un vector **y** con los datos de la variable dependiente. La función debe retornar en un vector **a** el valor de la *pendiente* y el *intercepto* [pendiente, intercepto] que resultan de aplicar la técnica de mínimos cuadrados y el coeficiente de determinación.

```
// Aplica el método de mínimos cuadrados para calcular la pendiente y el intercepto de una recta que
```

sigue la tendencia de un conjunto de datos
// function [a, r2] = regresionLineal(x, y)
// Entrada:
// x: vector con el valor de la variable independiente
// y: vector con el valor de la variable dependiente
// Salida:
// a: vector con el valor de la pendiente y el intercepto
// r2: coeficiente de determinación
function [a, r2] = **regresionLineal**(x, y)
// Escriba su código aquí.
endfunction

T09P03. Crear una función que implemente el método de regresión polinomial. La función debe recibir un vector **x** con los datos de la variable independiente, un vector **y** con los datos de la variable dependiente y un valor **n** que corresponde al orden del polinomio a ajustar para los datos de entrada. La función debe retornar en un vector **a** el valor de los coeficientes (Ej: para un polinomio de orden dos [pendiente, intercepto]) y el coeficiente de determinación.

//function [a, r2] = RegresionPolinomial(x, y, n)
// Entrada:
// x: vector con el valor de la variable independiente
// y: vector con el valor de la variable dependiente
// n : orden del polinomio a ajustar
// Salida:
// a: vector con el valor de la pendiente y el intercepto
// r2: coeficiente de determinación
function [a, r2] = r**egresionPolinomial**(x, y, n)
// Escriba su código aquí
endfunction

T10P01. Crear una función que reciba un vector **x** con los datos de la variable independiente, un vector **y** con los datos de la variable dependiente y un valor **xx** en el cual la interpolación es calculada. La función debe retornar como resultado el valor interpolado **yint** para **xx** empleando el método de interpolación de Newton.

// Aplica el método de interpolación de Newton a partir de un conjunto de datos
// function yint = interpolacionNewton(x, y, xx)
// Entrada:
// x: vector con el valor de la variable independiente
// y: vector con el valor de la variable dependiente
// xx: valor de la variable independiente para el cual la interpolacion es calculada
// Salida:
// yint: valor interpolado
function yint = **interpolacionNewton**(x, y, xx)
// Escriba su código aquí
endfunction

T10P02. Crear una función que reciba un vector **x** con los datos de la variable independiente, un vector **y** con los datos de la variable dependiente y un valor **xx** en el cual la interpolación es calculada. La función debe retornar como resultado el valor interpolado **yint** para **xx** empleando el método de interpolación de Lagrange.

```
// Aplica el método de interpolación de Lagrange a partir de un conjunto de datos
// function yint = interpolacionLagrange(x, y, xx)
// Entrada:
// x: vector con el valor de la variable independiente
// y: vector con el valor de la variable dependiente
// xx: valor de la variable independiente para el cual la interpolacion es calculada
// Salida:
// yint: valor interpolado
function yint = interpolacionLagrange(x, y, xx)
// Escriba su código aquí
endfunction
```

T11P01. Crear una función que reciba una **función**, un valor de limite inferior **a**, un valor de limite superior **b** y un número de segmentos **n**. La función debe retornar el valor de la integral **I** que resulta de aplicar el método de la regla trapezoidal compuesta.

```
// Aplica el método de la regla trapezoidal compuesta para un límite inferior, un límite superior y un numero de segmentos
// function I = trapezoidalCompuesta(funcion, a, b, n)
// Entrada:
// funcion: funcion a integrar
// a: límite inferior
// b: límite superior
// n: cantidad de segmentos
// Salida:
// I: valor de la integral
function I = trapezoidalCompuesta(funcion, a, b, n)
// Escriba su código fuente aquí
endfunction
```

T11P02. Crear una función que implemente el método de integración de Romberg. La función debe recibir la función **func** a integrar, los limites de integración **a** y **b**, un valor **es** que corresponde al criterio de parada para el error y un número máximo de iteraciones **maxit**. La función debe retornar el valor estimado de la integral **q**, el valor del error de aproximación **ea** y el número de iteraciones **iter**.

```
//function [q,ea,iter]= IntegracionRomberg(func,a,b,es,maxit)
// Entrada:
// func: función a integrar
// a: límite inferior de integración
// b: límite superior de integración
// es : criterio de parada para el error
// maxit : número máximo de iteraciones
// Salida:
// q : estimación de la integral
// ea : error de aproximación como porcentaje
// iter : número de iteraciones realizadas
function [q,ea,iter]= integracionRomberg(func,a,b,es,maxit)
// Escriba su código aquí
endfunction
```

T11P03. Crear una función que reciba una **función**, un valor de limite inferior **a**, un valor de limite

superior **b** y un número de segmentos **n**. La función debe retornar el valor de la integral **I** que resulta de aplicar el método de la regla de simpson de 1/3 compuesta.

```
// Aplica el método de la regla de simpson de 1/3 compuesta para un límite inferior, un límite superior y un numero de segmentos
// function I = tercioCompuesta(funcion, a, b, n)
// Entrada:
// funcion: funcion a integrar
// a: límite inferior
// b: límite superior
// n: cantidad de segmentos
// Salida:
// I: valor de la integral
function I = tercioCompuesta(funcion, a, b, n)
// Escriba su código fuente aquí
endfunction
```

T11P04. Crear una función que reciba una **función**, un valor de limite inferior **a** y un valor de limite superior **b**. La función debe retornar el valor de la integral **I** que resulta de aplicar el método de cuadratura de Gauss con 6 puntos.

```
// Aplica el método de la cuadratura de Gauss con 6 puntos para un límite inferior y un límite superior
// function I = cuadraturaGauss(funcion, a, b)
// Entrada:
// funcion: funcion a integrar
// a: límite inferior
// b: límite superior
// Salida:
// I: valor de la integral
function I = cuadraturaGauss(funcion, a, b)
// Escriba su código fuente aquí
endfunction
```

T12P01. Crear una función que reciba una **función**, un **valor de inicio**, un **valor de fin**, un **valor de stepsize**. La función debe retornar el valor de la derivada **dy** para la función de entrada en el rango de datos inicio:stepsize:fin. Emplee la función *diff* de scilab.

```
// Emplea la función diff de scilab para encontrar el valor de la derivada de una función en un rango de datos
// function dy = calcularDerivada(funcion, inicio, stepsize, fin)
// Entrada:
// function: función a derivar
// inicio: inicio para el rango de datos
// stepsize: tamaño de paso
// fin: fin para el rango de datos
// Salida:
// dy: valor de la derivada de la función en el rango
function dy = calcularDerivada(funcion, inicio, stepsize, fin)
// Escriba su código aquí
endfunction
```

Anexo

Funciones de Interés
bin2dec(str) : convierte un string en binario a decimal
mstr2sci(str) : convierte un string a un vector

Instrucciones de Prueba

// Ejercicio T01P01
matrizR = multiplicarEscalar(4, [1 2 3;1 2 3;1 2 3])
// Ejercicio T01P02
[y,t] = funcionSeno(-%pi,%pi/180,%pi)
// Ejercicio T01P03
vector = operarVectores([1 3 4],[4 5 6],1)
// Ejercicio T01P04
distancia = calcularDistancia([1 3 4], [1 2 3])
// Ejercicio T01P05
matrizR = multiplicarMatrices([1 2 3;1 2 3;1 2 3], [1 2 3;1 2 3;1 2 3])
// **Ejercicio T01P06**
[y] = deflexionBarra(100)
[y] = deflexionBarra(400)
// **Ejercicio T01P07**
[v] = obtenerVelocidad(5)
[v] = obtenerVelocidad(9)
[v] = obtenerVelocidad(20)
[v] = obtenerVelocidad(21)
// Ejercicio T02P01
[T, t] = cambioTemperatura(20, 70, 0.019, 2, 20)
// Ejercicio T02P02
[y,t] = llenadoTanque1(400, 1200, 0.5, 10)
// Ejercicio T02P03
[c,t] = descomposicion(100, 0.175, 0.1, 1)
// Ejercicio T02P04
[y,t] = llenadoTanque2(450, 1250, 150, 0.5, 10)
// **Ejercicio T02P05**
[x,y] = obtenerSolucion(4,0.5,5)
// **Ejercicio T02P06**
[y,t] = llenadoTanque3(2.5, 4, 1, 0.5, 10)
[y,t] = llenadoTanque3(2.5, 4, 1, 0.5, 5)
// Ejercicio T03P01
[et, ea, aprox] = funcionExpTaylor(0.5, 10)
// Ejercicio T03P02
[et, ea, aprox] = funcionSenoTaylor(%pi/4, 10)
// Ejercicio T03P03
[et, ea, aprox] = funcionCosenoTaylor(%pi/4, 10)
// Ejercicio T03P04
[et, ea, aprox] = funcionSgMaclaurin(0.5, 10)
// Ejercicio T03P05
[et, ea, aprox] = funcionLnMaclaurin(0.5, 10)
// **Ejercicio T03P06**
[et, ea, aprox,niter] = funcionCosenoTaylor2(%pi/6, 0.01, 10)

```
[et, ea, aprox,niter] = funcionCosenoTaylor2(%pi/3, 0.0000001, 5)
// Ejercicio T03P07
[et, ea, aprox] = funcionTangenteMaclaurin(%pi/4, 5)
[et, ea, aprox] = funcionTangenteMaclaurin(%pi/6, 5)
// Ejercicio T04P01
convertir(0,'11111111111111111111111','11111110') // 3.403D+38
// Ejercicio T04P02
[signo,mantisa,exponente]=convertirIEEE(100)
[signo,mantisa,exponente]=convertirIEEE(3400)
// Ejercicio T05P01
t = 0:1:100
fx = tiempoCohete(t)
plot(t,fx)
xtitle("Tiempo Cohete","t(segundos)","Cruce por cero")
a=get("current_axes");
t=a.title;
t.font_size=4;
x_label=a.x_label;
x_label.font_size= 4;
y_label=a.y_label;
y_label.font_size= 4;
set(gca(),"grid",[1 1])
// Ejercicio T05P02
h = 0:0.01:5
fx = profundidadTanque(h)
plot(h,fx)
xtitle("Profundidad Tanque","h(metros)","Cruce por cero")
a=get("current_axes");
t=a.title;
t.font_size=4;
x_label=a.x_label;
x_label.font_size= 4;
y_label=a.y_label;
y_label.font_size= 4;
set(gca(),"grid",[1 1])
// Ejercicio T05P03
t = 0:1:100
Purb = funcion1(t)
Psub = funcion2(t)
// Las graficas de Purb y Psub se deben intersectar entre 33 y 34
fx = tiempoPoblacion(t)
subplot(221)
plot(t,Purb)
set(gca(),"grid",[1 1])
xtitle("Poblacion Urbana","t(años)","personas")
subplot(222)
plot(t,Psub)
set(gca(),"grid",[1 1])
xtitle("Poblacion Suburbana","t(años)","personas")
subplot(223)
plot(t,fx)
set(gca(),"grid",[1 1])
```

```
xtitle("Tiempo Poblacion","t(años)","Cruce por cero")
```
// Ejercicio T05P04
```
theta0=0:%pi/180:%pi/4
fx = anguloProyectil(theta0)
plot(theta0,fx)
xtitle("Angulo Proyectil","theta0(radianes)","Cruce por cero")
a=get("current_axes");
t=a.title;
t.font_size=4;
x_label=a.x_label;
x_label.font_size= 4;
y_label=a.y_label;
y_label.font_size= 4;
set(gca(),"grid",[1 1])
```
// **Ejercicio T05P05**
```
function fx = funcion(x)
    fx = 3 + 6*x + 5*x^2 + 3*x^3 + 4*x^4
endfunction
pt = 0.01, xr = -1, eads = 0.01, maxit = 10; // Aqui el error es un porcentaje
[xrold, xr, ea, niter] = NewtonOptimo(funcion, pt, xr, eads, maxit)
```
// Ejercicio T05P06
```
function fx=funcion(x)
    fx = (x^2)/10 - 2*sin(x);
endfunction
x1 = 0, x2 = 1, x3 = 4, eads=0.001, maxit = 20;
[x4, ea,niter] = interpolacionParabolica(funcion, x1, x2, x3, eads, maxit)
```
// Ejercicio T05P07
```
function fx = funcion(x)
    v = 36, t = 4, cd = 0.25, g = 9.81;
    fx = sqrt(g*x/cd).*tanh(sqrt(g*cd./x)*t) - v;
endfunction
xl = 50, xu = 200, eads = 0.5859;
[xl, xu, xr, ea, niter] = biseccion2(funcion, xl, xu, eads)
```
// Ejercicio T05P08
```
function fx = funcion(x)
    v = 36, t = 4, cd = 0.25, g = 9.81;
    fx = sqrt(g*x/cd).*tanh(sqrt(g*cd./x)*t) - v;
endfunction
xl = 40, xu = 200, maxit = 40, eads = 0.0000535; // llega en 30 iteraciones
[xu, xl, fxl, fxu, xr, ea, niter] = falsaposicion2(funcion, xl, xu, eads, maxit)
```
// Ejercicio T05P09
```
function fx = funcion(x)
    v = 36, t = 4, cd = 0.25, g = 9.81;
    fx = sqrt(g*x/cd).*tanh(sqrt(g*cd./x)*t) - v;
endfunction
function dfx = dfuncion(x)
    v = 36, t = 4, cd = 0.25, g = 9.81;
    dfx = (1/2)*sqrt(g/(x*cd))*tanh(sqrt((g*cd)/(x))*t) - (g/(2*x))*t*(sech(sqrt((g*cd)/(x))*t))^2;
endfunction
xr = 140, maxit = 10, eads = 0.00000099; // llega en 4 iteraciones
```

```
[xrold, xr, ea, niter] = newraphraiz(funcion, dfuncion, xr, eads, maxit)
```

// Ejercicio T05P10

```
function fx = funcion(x)
    v = 36, t = 4, cd = 0.25, g = 9.81;
    fx = sqrt(g*x/cd).*tanh(sqrt(g*cd./x)*t) - v;
endfunction
maxit = 10, eads = 0.0000041, pt = 1E-6, xr = 50; // llega en 6 iteraciones
[xrold, xr, ea, niter] = secante2(funcion, pt, xr, eads, maxit)
```

// Ejercicio T06P01

```
[vectorx, condicion] = obtenerRespuesta([2 5 2;3 5 8;3 8 1], [1 5 7]')
[vectorx, condicion] = obtenerRespuesta([2 5 2;3 5 8;3 8 1], [1 5]')  - %nan
```

// Ejercicio T06P02

```
matrizR = obtenerInversa([2 5 2;3 5 8; 2 6 3])
matrizR = obtenerInversa([2 5;3 5; 2 6])
matrizR = obtenerInversa([2 5 1;2 5 1;2 4 5]) - %nan
```

// Ejercicio T06P03

```
matrizU = obtenerFactorizacion([1 2 3;2 5 4;3 4 45])
matrizU = obtenerFactorizacion([1 8 3;2 6 4;3 7 45]) - %nan
```

// Ejercicio T06P04

```
norma = obtenerNorma([1 4 6],[4 7 9], 1)
norma = obtenerNorma([1 4 6],[4 7], 1) - %nan
```

// Ejercicio T07P01

```
// Ecuacion 1
function f=u(x,y)
        f = x^2 - 5 + y^2;
endfunction
// Ecuacion 2
function f=v(x,y)
        f = y + 1 - x^2;
endfunction
x0 = [1.5 1.5]';
[x,f,ea,iter]=NewtonRaphson1(u, v, x0, 0.0001, 50)
```

// Ejercicio T07P02

```
// Ecuacion 1
function f=u(x, y, z)
f = 3*x-cos(y*z)-1/2;
endfunction
// Ecuacion 2
function f=v(x, y, z)
f = x^2-81*(y+0.1)^2+sin(z)+1.06;
endfunction
// Ecuacion 3
function f=w(x, y, z)
f = exp(-x*y)+20*z+(10*%pi-3)/3;
endfunction
// Resultados
x0 = [1.5 1.5 1.5]';
[x,f,ea,iter]=NewtonRaphson2(u, v, w, x0, 0.0001, 50)
```

// Ejercicio T08P01

```
A = [0.8 -0.4 0; -0.4 0.8 -0.4; 0 -0.4 0.8]
b = [41 25 105]'
```

```
[x, ea, iter]  = GaussSeidel(A, b, 0.00001, 50)
// Ejercicio T08P02
A = [10 -2;-3 12]
b = [8 9]';
lambda = 1.2
[x, ea, iter] = GaussSeidelR(A,b,lambda,0.0001,50)
// Ejercicio T09P01
y = [2 8 0 3 7 6 8 7 9 1 6];
[promedio, mediana, desv] = obtenerEstadisticas(y)
// Ejercicio T09P02
x = [10 20 30 40 50 60 70 80];
y = [25 70 380 550 610 1220 830 1450];
[a, r2] = regresionLineal(x,y)
// Ejercicio T09P03
x = [0 1 2 3 4 5]';
y = [2.1 7.7 13.6 27.2 40.9 61.1]';
[a, r2] = regresionPolinomial(x, y, 2);
x = [0 1 2 3 4 5]';
y = [2.1 7.7 13.6 27.2 40.9 61.1]';
[a, r2] = regresionPolinomial(x, y, 3);
// Ejercicio T10P01
x = [1 4 6 5]';
y = log(x);
interpolacionNewton(x,y,2)
// Ejercicio T10P02
T = [-40 0 20 50];
d = [1.52 1.29 1.2 1.09];
density = interpolacionLagrange(T,d,15)
// Ejercicio T11P01
function v = velocidad(t)
v = sqrt(9.81*68.1/0.25)*tanh(sqrt(9.81*0.25/68.1)*t);
endfunction
trapezoidalCompuesta(velocidad,0,3,5)
// Ejercicio T11P02
function f = ecuacion(x)
        f = 0.2+25*x-200*x.^2+675*x.^3-900*x.^4+400*x.^5;
endfunction
[q, ea, iter] = integracionRomberg(ecuacion,0,0.8,0.000001,50)
// Ejercicio T11P03
deff('[y]=mifuncion(x)','y=0.2+25*x-200*x.^2+675*x.^3-900*x.^4+400*x.^5');
I = tercioCompuesta(mifuncion,0,0.8,4)
// Ejercicio T11P04
deff('[y]=mifuncion(x)','y=0.2+25*x-200*x.^2+675*x.^3-900*x.^4+400*x.^5');
I = cuadraturaGauss(mifuncion,0,0.8)
// Ejercicio T12P01
function y=funcion(x)
y = sin(x);
endfunction
calcularDerivada(funcion, 0, 0.001, 0.006)
```