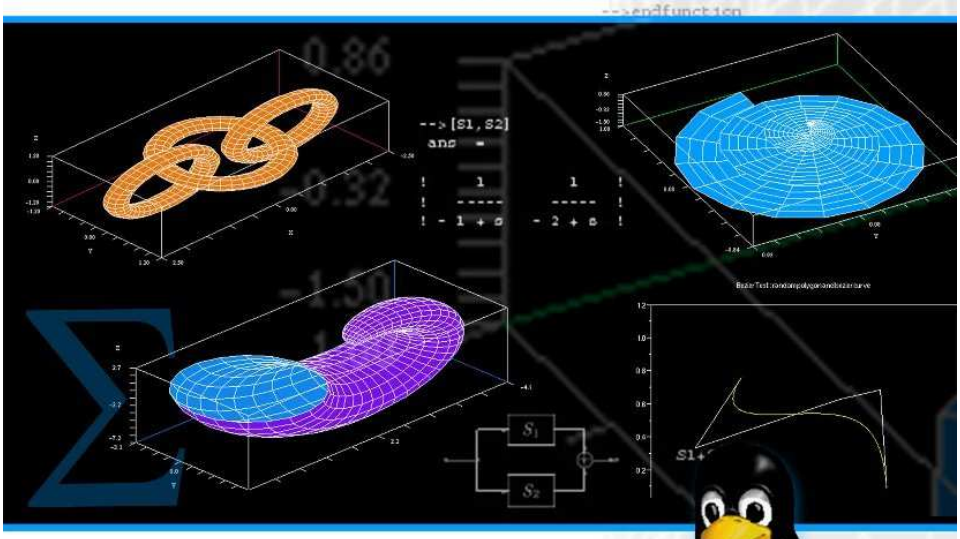
FUNDAMENTOS DE SCILAB Y APLICACIONES



Versión 0.1

Andrés Alfonso Caro
Cesar Valero Sepúlveda

Linux
&
WINDOWS

FUNDAMENTOS DE SCILAB Y APLICACIONES

Andrés Alfonso Caro
Cesar Valero Sepúlveda

Índice general

INTRODUCCIÓN

SCILAB es una extensión del clásico MATLAB®, desarrollado por INRIA-Unité de recherche de Rocquencourt, en el año de 1990, se elaboró para la solución de sistemas de control, procesamiento de señales y otras aplicaciones matemáticas con la filosofía del software libre y amparado por la licencia GPL.

Su publicación se realizó en 1994 en Internet en la URL http://www-rocq.inria.fr/scilab o http://www.scilab.org y desde la época ha recibido contribuciones que lo han permitido crecer y fortalecerse hasta ser hoy un software muy robusto y confiable.

Una característica principal de scilab es su sintaxis, ésta habilita:

El manejo de matrices y todo tipo de operaciones con ellas, números complejos, polinomios y muchas otras funciones matemáticas.

Contiene una gran variedad de funciones primitivas para el análisis de sistemas no lineales .

Provee de un entorno de programación bastante robusto, con el cual podemos escribir programas llamados scripts u objetos que podemos ejecutar en scilab.

La filosofía de SCILAB es:

- Proveer el manejo de datos de forma natural y con una sintaxis sencilla.
- Tener un entorno de programación con licencia GPL, fácil de manipular y escalar.
- Soporte para librerías de desarrollo o toolboxes aplicados a: control lineal, procesamiento de señales, análisis de redes, control no lineal, etc

La instalación de scilab no requiere de grandes conocimientos en programación y configuración de software, éste funciona en equipos cuyo sistema operativo puede ser LINUX o windows . La versión nativa o con el código fuente requiere aproximadamente de 130 Mb de espacio libre en disco y compiladores de C y Fortran para su ejecución. Por otra parte También existe una versión reducida, sin el código fuente cuyo tamaño aproximado es de 40 Mb, la cual no necesita compiladores para ejecutarla.

Este documento tiene como objetivo introducir al lector en el ambiente de SCILAB y mostrar cuales son las características que lo hacen fuerte y una gran alternativa al software propietario. La creación de este libro se ha realizado con SCILAB 3.0, en

su versión para GNU/LINUX, se harán aclaraciones sobre las diferencias que puedan existir con respecto al sistema operativo Windows, las figuras se han generado usando Xfig y algunas capturas con el comando import y Gimp. La composición se ha hecho en LaTeX, usando LYX en LINUX FC1.

Agradecimientos.

Deseamos expresar nuestro agradecimiento a las personas que han estado presentes en la creación de este libro, al Ingeniero Iván Luna, nuestro director. Agradecimientos especiales para el Ingeniero Luis Guillermo Gómez por sus asesorías en SCILAB y su entusiamo a la hora de mostrarnos sus creaciones, para Pablo Munevar y Jimmy Ramirez por permitirnos aplicar algunos capítulos en sus cursos. A Wilmer Sepúlveda por el diseño de la carátula, a ederaam por hacer que este libro este disponible en Gigax, a ODDG por ceder un espacio en LANeros.com y al profesor Carlos franco por dejar un espacio para este documento en su página de química, finalmente y de manera especial a nuestros padres.

Capítulo 1

INSTALACIÓN

1.1. Instalar SCILAB en GNU/LINUX

Paso 1

Descargar el archivo de instalación del sitio http://scilabsoft.inria.fr/, en el momento que escribimos este libro, la versión disponible es la 3.0.

Paso 2

existen varias formas de instalar SCILAB para LINUX, eso lo determina el tipo de paquete descargado, puede ser un fichero .rpm, un binario, o, el archivo fuente (.src).

Caso 1: archivo RPM

Este procedimiento tiene que hacerse como usuario root

```
# rpm -ivh NombreDelArchivo.rpm
```

Sí en la preparación de la instalación; exige dependencias como tk o tcl, tendrá que instalarlas y volver a intentar la instalación de SCILAB.

Caso 2: binario

Cuando las versiones son resientes, es muy común que no haya un paquete rpm disponible, así que es necesario instalar desde los binarios o las fuentes.

Por ejemplo, la versión 3.0 en el momento que escribimos este texto estaba disponible en la web de SCILAB de la siguiente forma:

Scilab-3.0 binary file version for Linux

Scilab-3.0 source version (tar.gz file)

El primer vinculo nos lleva a descargar el archivo binario scilab-3.0.bin.linux-i686.tar.gz. La instalación se puede hacer como un usuario diferente a root. Abriendo

una terminal externa como konsole, gnome-terminal o xterm, escribir la ruta en la que guardó el archivo al descargarlo, en nuestro caso, la ruta es: /home/sci/scilab/scilab-3.0.bin.linux-i686.tar.gz, entonces:

```
$ cd /home/sci/scilab
$ tar zxvf scilab-3.0.bin.linux-i686.tar.gz
```

al descomprimir, crea un directorio con el nombre scilab-3.0

```
$ cd scilab-3.0
$ make
$ cd bin
$ ./scilab
```

y puede comenzar a trabajar con SCILAB. Sí quiere evitar el proceso de ejecución entrando siempre al directorio bin para escribir ./scilab, puede incluir toda la ruta en el archivo .bash_profile de su usuario. En nuestro caso hacemos lo siguiente en la linea PATH del archivo .bash_profile :

```
PATH=$PATH:$HOME/bin:$HOME/scilab/scilab-3.0/bin/
```

NOTA: $HOME es una variable de entorno y el valor que toma es /home/sci

Con el procedimiento anterior, podemos ejecutar SCILAB desde una terminal, escribiendo scilab y oprimiendo enter, o usando ALT+F2 como se explica en la sección 2.1.

Caso 3: fuente(src)

Para la versión 3.0, el archivo fuente tiene el nombre scilab-3.0.src.tar.gz, el primer paso es descomprimirlo. Como ejemplo la ruta en la que guardamos el archivo es: /home/sci/scilab/src, para descomprimirlo:

```
$ cd /home/sci/scilab/src
$ tar zxvf scilab-3.0.src.tar.gz
```

Al descomprimir crea un directorio con el nombre Scilab-3.0-RC

```
$ cd scilab-3.0
$ ./configure
```

Para mas información sobre los modificadores de ./configure, puede consultar el manual de instalación (README_Unix) que está contenido dentro del directorio scilab-3.0.

Si no hubo errores, como usuario root puede ejecutar:

```
# make all
```

Ahora esta listo para trabajar con SCILAB usando GNU/LINUX, La sección 2.1 explica como ejecutar el programa.

1.2. Instalar SCILAB en WINDOWS

La instalación en el Sistema Operativo WINDOWS es simple, el primer paso es descargar el archivo ejecutable de SCILAB para WINDOWS, en nuestro caso instalamos usando el archivo scilab272.exe, al dar doble click aparece la ventana de bienvenida. Dar un click en el botón Next y aparece la ventana que hace referencia al acuerdo de licencia como aparece en la figura 1.1

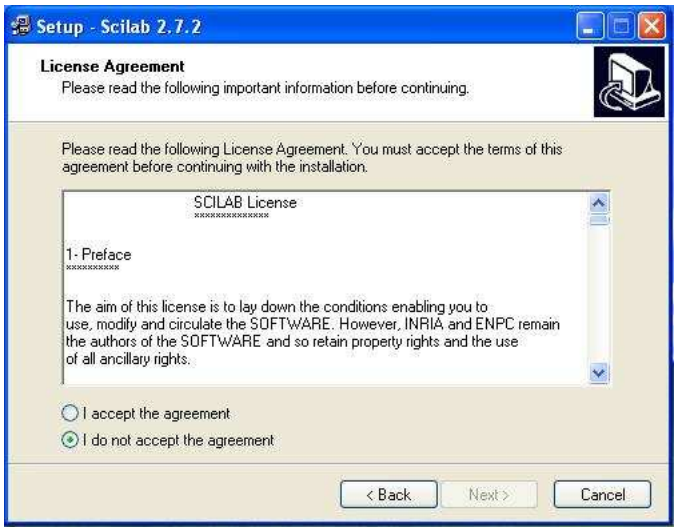


Figura 1.1: Acuerdo de Licencia

Para continuar debe dar click en el campo de selección I accept the agreement y despues en el botón Next, aparece la ventana que se muestra en la figura 1.2, en la que debe seleccionar el directorio o ruta en la que SCILAB quedará instalado.

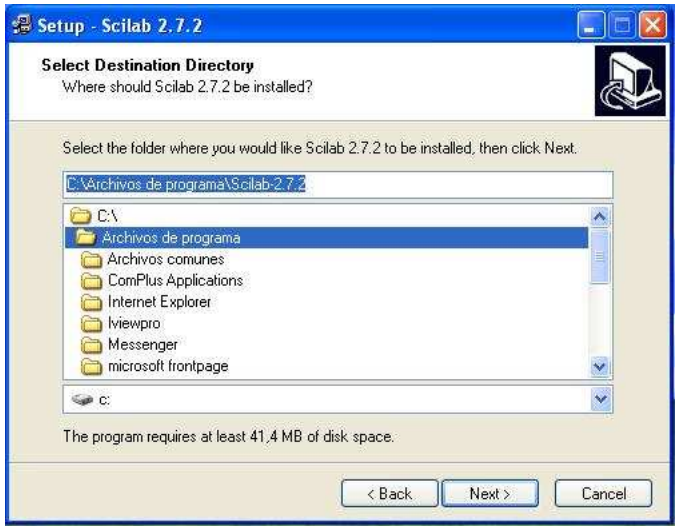


Figura 1.2: Selección de la ruta de instalación

Puede seleccionar la ruta que desee, o usar la ruta predeterminada C:\Archivos de programa\Scilab-2.7.2 , para continuar de click en Next.

Después aparece la ventana en la que puede seleccionar los componentes que quiere incluir en su instalación, seleccione marcando en los campos correspondientes y pulse Next, aparecerá una ventana que le permite seleccionar la carpeta del menú inicio en la que estará disponible el acceso directo para ejecutar SCILAB, si no selecciona las opciones que aparecen, el acceso a SCILAB quedará en el menú programas del menú inicio.

La siguiente ventana le mostrara un resumen de las opciones que usted selecciono en las ventanas anteriores, como el directorio de destino, el tipo de instalación y los componentes seleccionados, al dar click en Next aparece la ventana con la barra de progreso de la instalación.

Al terminar aparece una ventana con dos opciones que le permiten seleccionar si usted quiere que se abra el archivo LEAME de SCILAB y la segunda opción habilita la creación de un icono de SCILAB en el escritorio de WINDOWS.

Para terminar de click en Finish, ahora puede ejecutar SCILAB para WINDOWS como se explica en la sección 2.2

Capítulo 2

PRIMEROS PASOS

Ejecutando SCILAB

2.1. GNU/LINUX

Después de haber instalado SCILAB puede ser ejecutado de varias formas: estando en ambiente gráfico como GNOME o KDE, oprimir ALT+F2, se abre la ventana Ejecutar comando, escribir scilab en minúsculas y oprimir ALT+r si esta trabajando en GNOME, o ALT+e si lo hace en KDE, ó, dar un click en el botón Ejecutar como se muestra en la figura 2.1.

Figura 2.1: Ventana de ejecución

otra forma de ejecutar SCILAB es a través de una terminal externa de la siguiente forma:

```
$ scilab
```

También se puede ejecutar el programa desde la ruta donde está instalado, en la mayoría de las distribuciones LINUX la ruta es /usr/bin/scilab :

```
$ cd /usr/bin
$ ./scilab
```

Nota: el símbolo $ es el prompt del shell de LINUX.

Si no dispone de un ambiente gráfico, SCILAB se puede ejecutar en ambiente texto de la siguiente forma:

```
$ scilab -nw
```

El parámetro -nw significa "no-window"

2.2. SCILAB en MICROSOFT WINDOWS

en MS WINDOWS la instalación de SCILAB crea un icono en el escritorio (Figura 2.2), al hacer doble click se ejecuta SCILAB, otra forma es a través del menú INICIO, dentro del menú PROGRAMAS buscar la carpeta con el nombre Scilab.

Figura 2.2: Icono generado por Scilab en el escritorio de WINDOWS

2.3. Inicio de SCILAB

Después de ejecutar SCILAB se abre la ventana del programa con el siguiente encabezado y el prompt -->, el símbolo del prompt (-->) indica que SCILAB esta listo para recibir comandos y ejecutarlos.

```
==========

scilab-2.7

Copyright (C) 1989-2003 INRIA/ENPC

==========
```

```
Startup execution:
  loading initial environment
-->
```

Los comandos se escriben después del prompt y se ejecutan presionando la tecla Enter. Usando las teclas de flecha derecha e izquierda se puede mover de un lado al otro de la linea de comandos que se ha escrito, con las teclas de flecha hacia arriba y hacia abajo se visualizan las lineas de comando escritas anteriormente, existen combinaciones de teclas usando la tecla CTRL, por ejemplo: para volver al principio de una linea de comandos basta con oprimir CTRL+a, presionando CTRL+e, el cursor se mueve al final de la linea de comandos, CTRL+c, interrumpe el proceso después de haber ejecutado una linea de comandos en SCILAB.

La ventana de SCILAB tiene un menú de 5 botones como se muestra en la figura 2.3

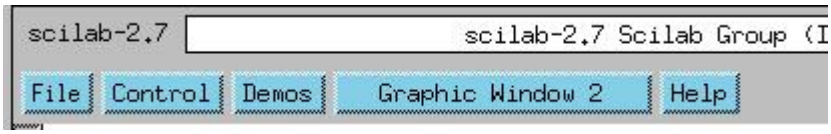


Figura 2.3: Botones de control

al dar click con el ratón sobre el botón File, Control, Demos, Graphic Window 0, ó, Help, se despliega un menú en el que aparecen opciones. Para escoger alguna de las opciones del alguno de los menús es necesario mantener el botón derecho del ratón presionado y arrastrar el puntero hasta la opción que quiera ejecutarse y soltar el botón.

2.3.1. Descripción de las opciones de cada menú

2.3.1.1. File

-Menu Files Operations: carga funciones o datos en SCILAB, o sirve para ejecutar archivos que tienen lineas de comandos llamados guiones o scripts.

-Kill: mata el interprete de guiones o lineas de comandos de SCILAB.

-Quit: salir de SCILAB

2.3.1.2. Control

-Resume: sirve para continuar con un proceso que ha sido detenido con Stop o con ctrl-c.

-Abort: aborta el proceso después de una o varias pausas en el proceso usando Stop o ctrl-c

-Restart: borra todas las variables creadas

-Stop: sirve para detener o dar pausa a los procesos.

2.3.1.3. Demos

Aparece un menú con diferentes demostraciones de tareas y aplicaciones que se pueden hacer con SCILAB .

2.3.1.4. Graphic Window

se usa para mostrar las ventanas de los gráficos activos, explicaremos el uso de este botón posteriormente.

2.3.1.5. Help

al dar click aparece una ventana de ayuda con una estructura de pequeños manuales de los correspondientes items que aparecen en la lista.

2.4. COMENZANDO CON SCILAB

Esta sección de nuestro libro es una introducción para los usuarios nuevos, trataremos de ser explícitos en nuestras explicaciones y ejemplos, los comandos de SCILAB no son de alto nivel de complejidad, todo el trabajo y el aprendizaje requiere de un poco de práctica.

2.4.1. Comentarios

El entorno de SCILAB es similar a los entornos de programación, es por eso que para mantener el orden en las lineas de código que se escriben es recomendable usar comentarios; explicando lo que representan algunos conjuntos de lineas. Para escribir comentarios, se usa //, así:

```
-->//tradicionalmente: este es un comentario
```

2.4.2. Constantes

Para declarar una variable de tipo escalar se introduce el nombre que va representarla (en este caso el nombre es varc), se iguala al valor que va tener para el siguiente caso es 5 y se presiona la tecla Enter.

```
-->//varc tendra valor 5
-->varc=5
  varc =
   5.
```

SCILAB discrimina mayúsculas y minúsculas

```
-->a=5
  a =
   5.
-->A=7
  A =
   7.
```

SCILAB discrimina mayúsculas y minúsculas, como en este caso, a es diferente de A, ahora bien, si nos damos cuenta, vemos que después de haber escrito a=5 y tecleado enter aparece:

```
a =
  5.
```

lo mismo pasa cuando se declara A, si queremos que esto no suceda escribimos un ";" al final de la linea que queramos ejecutar.

```
-->a=5;
-->
```

si después de estar trabajando quiere saber el valor de alguna variable que se declaró anteriormente simplemente teclea el nombre de la variable y Enter.

```
-->A
 A =
   7.
```

2.4.3. Operaciones Básicas

Ahora que ya sabe declarar variables de tipo escalar, puede hacer operaciones con ellas.

2.4.3.1. Suma.

```
-->a+A
 ans =
   12.
```

2.4.3.2. Resta.

```
-->a-A
 ans =
   -2.
```

2.4.3.3. Producto.

```
-->a*A
 ans =
   35.
```

2.4.3.4. División.

```
-->a/A
 ans =
  0.7142857
```

Obviamente la división por cero no se puede realizar y si se intenta, SCILAB muestra un error.

```
!--error 27
division by zero...
```

Para elevar una variable a una potencia determinada, basta con usar el símbolo "^", o doble asterisco "**" así:

```
-->a^3
 ans =
   125.
-->A**2
 ans =
```

```
   49.
```

para hallar la raíz cuadrada se puede usar el comando sqrt seguido de un par de paréntesis con los que se encierra la variable sobre la cual desea realizar la operación. La variable o valor que se pone dentro de los paréntesis suele llamarse argumento, para este ejemplo se puede crear una variable "b" con el resultado de elevar A al cuadrado, así:

```
-->b=A**2;
```

ahora se obtiene la raíz cuadrada de b usando sqrt:

```
-->sqrt(b)
ans =
   7.
```

2.4.4. La función exponencial e^x

para obtener la funcion exponencial se usa el comando exp de la siguiente forma:

```
-->exp(x)
```

x puede ser una constante, un vector, o una matriz con componentes reales o complejas.

2.4.5. Logaritmo

Usamos el comando log para hallar logaritmo natural y log10 para hallar logaritmo en base 10, también es posible hallar logaritmo en base 2, usando log2.

```
-->log(2.7182818)
ans =
1.0000000
-->log10(1000)
ans =
3.
-->log2(1024)
ans =
10.
```

2.4.6. Eliminar variables

Para eliminar una variable creada anteriormente se usa el comando clear:
Ejemplo:

```
-->a=1
 a =
  1.
-->clear a
```

Después de eliminarla, puede llamarla para comprobar que ya no existe y SCILAB muestra un error:

```
-->a
  !--error    4
undefined variable : a
```

2.4.7. FUNCIONES TRIGONOMÉTRICAS

SCILAB tiene la posibilidad de aplicar funciones Trigonométricas sobre las variables, igual que con `sqrt` se usa el comando que representa a la función trigonométrica seguido de un paréntesis "()" y dentro del paréntesis la variable o argumento de la función trigonométrica.
SCILAB no reconoce el número del argumento en grados sino en radianes, así que para obtener el seno de un ángulo como por ejemplo 45°, tendrá que hacer la conversión a radianes, por ejemplo:

$a = \frac{45^o \cdot \pi}{180^o}$

usando SCILAB:

```
-->a=(45*%pi)/180
  a  =
    0.7853982
-->sin(a)
ans  =
   0.7071068, ó, √2/2
```

En la siguiente tabla se muestran los comandos de algunas de las funciones trigonométricas:

Función	Comando	Ejemplo
seno	sin	`-->VarSn=sin(variable)`
arcoseno	asin	`-->VarRs=asin(variable)`
coseno	cos	`-->VarC=cos(variable)`
arcocoseno	acos	`-->VarAc=acos(variable)`
tangente	tan	`-->VarTa=tan(variable)`
arcotangente	atan	`-->VarArt=atan(variable)`
cotangente	cotg	`-->VarCtg=cotg(variable)`

Cuadro 2.1: funciones trigonométricas en SCILAB

2.4.8. CONSTANTES ESPECIALES

SCILAB contiene varias constantes especiales y comienzan con el símbolo porcentaje (%), algunas variables predefinidas por SCILAB son: %i y representa sqrt(-1) o raíz cuadrada de -1($\sqrt{-1}$), posteriormente la usaremos en los casos donde es necesario trabajar con números que tienen parte real y parte imaginaria. %pi (π) (representa el numero 3.1415...), otra variable predefinida y usada con frecuencia es %e (igual a 2.7182...), podemos comprobar el valor de cada una haciendo lo siguiente:

```
-->%pi
%pi  =
     3.1415927
-->%e
%e  =
     2.7182818
-->sqrt(-1)
 ans  =
    i
```

Otras constantes especiales son:

%eps, representa un número muy pequeño, si se escribe %eps:

```
-->%eps
%eps  =
    2.220E-16
```

Ejemplo:

```
-->Bn=1+%eps
 Bn =
    1.
```

%inf representa un número muy grande.

```
-->%inf
%inf  =
   Inf
```

Sí se divide un número pequeño por un valor muy grande, el resultado es un numero muy pequeño

```
-->Np=1/%inf
 Np  =
    0.
```

%nan significa "not a number" (no es un número), al operar esta constante con cualquier valor el resultado es Nan.

```
-->not=%pi+%nan
 not  =
    Nan
```

%t y %f, representan las constantes booleanas verdadero y falso respectivamente.

```
-->Si=%t
 Si  =
    T
-->No=%f
 No  =
    F
```

2.4.9. VECTORES

Así como en varias aplicaciones físicas se usan cantidades escalares para representar ciertas variables, también se hace necesario el tratamiento de ciertas cantidades que se representan con vectores, como la velocidad, la fuerza o en el caso de la electrónica, la corriente

para introducir un vector en SCILAB es necesario darle un nombre al vector como en el caso de las variables, el nombre puede ser en mayúsculas o minúsculas, la disposición del vector puede ser en filas o columnas y los componentes del vector deberán estar encerrados por paréntesis cuadrados “[ ]”:

NombreDelVector=[componentes]

2.4.9.1. VECTOR FILA

Si queremos disponer nuestro vector en forma de fila, separamos cada uno de los componentes con espacios o comas "," así:

```
-->VecF1=[2 4 6]
 VecF1  =
! 2. 4. 6. !
```

Usando comas:

```
-->VecF2=[5,%pi,%e]
VecF2 =
! 5. 3.1415927 2.7182818 !
```

En el segundo ejemplo vemos como los componentes del vector también pueden ser constantes especiales, posteriormente veremos como las componentes pueden ser números complejos e incluso polinomios.

Otra forma de crear vectores fila es escribiendo el valor inicial y el valor final del vector, el rango de valores que hay desde la primera componente hasta la ultima sera de orden lineal, separada cada una por la unidad.

```
-->Vec3=1:10
Vec3 =
! 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. !
```

Sí queremos disminuir o aumentar el valor por el cual van a estar separadas las componentes entre el valor final e inicial, debe indicarse el valor en medio del valor inicial y el valor final así:

```
-->Vec4=1:2:10
Vec4 =
! 1. 3. 5. 7. 9. !

-->Vec5=1:0.8:10
Vec5 =
! 1. 1.8 2.6 3.4 4.2 5. 5.8 6.6 7.4 8.2 9. 9.8 !
```

2.4.9.2. VECTOR COLUMNA

Para disponer el vector en forma de columna se escribe el nombre del vector, seguido por el símbolo igual "=", se abren paréntesis cuadrados "[", después se separar cada componente tecleando Enter, después de escribir la ultima componente se cierra con

"]".

```
-->VecCol=[1
-->2
-->3
-->4
-->5
-->6]
VecCol =
! 1. !
! 2. !
! 3. !
! 4. !
! 5. !
! 6. !
```

La segunda forma es separando las componentes del vector columna usando punto y coma ";" así:

```
-->VecCol2=[3.8; 5.8*%i; 6.3]
VecCol2 =
!  3.8  !
!  5.8i !
!  6.3  !
```

Otra alternativa para generar un vector columna es a apartir de un vector fila y aplicando la transpuesta:

```
-->VecFil=1:0.5:5
VecFil =
!  1. 1.5 2. 2.5 3. 3.5 4. 4.5 5.  !
-->VecCo=VecFil'
VecCo = ! 1. !
! 1.5 !
! 2.  !
! 2.5 !
! 3.  !
! 3.5 !
```

```
! 4.  !
! 4.5 !
! 5.  !
```

2.4.9.3. Transpuesta

Para aplicarle la transpuesta a una expresión simplemente se pone una comilla " ' " simple al lado derecho de la variable que representa la expresión:

```
-->Expr=[3.3  %pi  2*%pi]
Expr =
!  3.3  3.1415927  6.2831853  !
-->Expr'
ans =
! 3.3       !
! 3.1415927 !
! 6.2831853 !
```

2.4.10. MATRICES

Matriz: una matriz X de `m x n` es un arreglo rectangular de mn números dispuestos en m renglones y n columnas. Como ejemplo la matriz B:

$$B = \begin{bmatrix} 4 & 8 & 9 & 6 \\ 5 & 2 & 1 & 7 \\ 3 & 5 & 7 & 9 \end{bmatrix}$$

En SCILAB una matriz se construye utilizando los siguientes parámetros:

nombre de la matriz, símbolo igual `"="`, paréntesis cuadrados ó definición de Matriz `"[ ]"`

La matriz quedara de la siguiente forma dentro de el paréntesis: los elementos de las filas serán separados por comas o espacios, para una nueva fila utilizaremos punto y coma ";",

Ejemplo, la matriz B la podemos representar en SCILAB de la siguiente forma:

```
-->B=[4 8 9 6;5 2 1 7;3 5 7 9]
B =
!  4. 8. 9. 6.  !
!  5. 2. 1. 7.  !
!  3. 5. 7. 9.  !
```

o separado por comas:

```
-->B=[4,8,9,6;5,2,1,7;3,5,7,9]
B =
!  4. 8. 9. 6.  !
!  5. 2. 1. 7.  !
!  3. 5. 7. 9.  !
```

donde el primer vector fila tiene los elementos:

```
4 8 9 6
```

también podemos construir nuestra matriz como un conjunto de vectores de la siguiente forma:

vector x

```
-->x=[1 2 3]
x =
!  1. 2. 3.  !
```

vector y

```
-->y=[4 8 9]
y =
! 4. 8. 9. !
```

vector z

```
-->z=[8 5 2]
z =
! 8. 5. 2. !
```

vector c

```
-->c=[x;y;z]
c =
! 1. 2. 3. !
! 4. 8. 9. !
! 8. 5. 2. !
```

Teniendo los vectores x , y y z, formamos la matriz c

2.4.11. POLINOMIOS

El manejo de polinomios con SCILAB es bastante sencillo, basta con identificar si queremos crear la función a partir de la introducción de los coeficientes y la variable o si queremos que el polinomio sea creado a partir de introducir las raíces y la variable.

La sintaxis para representar polinomios es la siguiente:

Como coeficientes

nombre del polinomio, símbolo igual "=" seguido de la palabra "poly", paréntesis redondo "(" coeficientes entre paréntesis cuadrado "[ ]"seguidos de coma "," en comillas simples la variable base del polinomio seguida de coma ",", en comillas simples la letra "c" que significa coeficiente. Cierra paréntesis ")".

Ejemplo:

```
-->Pol1=poly([1 3 5 7],'x','c')
Pol1 =
```

$1 + 3x + 5x^2 + 7x^3$

coeficientes = 1, 3, 5, 7

Como raíz

nombre del polinomio, símbolo igual "=", seguido de la palabra "poly", paréntesis "(", raíces del polinomio entre paréntesis cuadrado "[ ]", seguidos de coma "," en comillas simples la variable base del polinomio, Cierra paréntesis ")".

Ejemplo:

```
-->b=poly([1 3 5 7],'x')
b =
```

$105 - 176x + 86x^2 - 16x^3 + x^4$

raíces =1, 3, 5, 7. coeficientes= 105, 176, 86, 16, 1.

2.4.12. ELABORACIÓN DE GRÁFICOS BÁSICOS

comandos básicos

plot, plot2d

2.4.12.1. PLOT

Este comando nos permite elaborar gráficos simples, de la siguiente forma:

```
plot(x, y, "titulo eje x," titulo eje y", "titulo gráfico")
```

en donde x y y son vectores del mismo tamaño y que la gráfica a realizar es: y con respecto a x.

NOTA: recordemos que una gráfica entre más puntos tenga mayor resolución tendrá.

Ejemplo:

```
-->x=[1:1:10];

-->plot(x, cos(x),"Números de 1 a 10","coseno de x","COSENO")
```

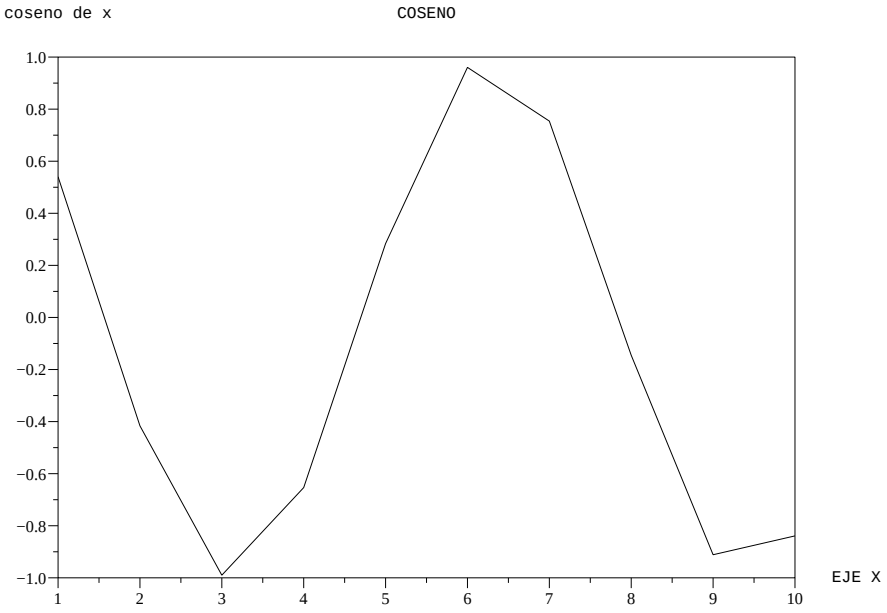


Figura 2.4: Gráfica de la función coseno con pocos puntos

si nuestra muestra es mayor, el resultado es el siguiente:

```
-->x=[1:0.01:10];

-->plot(x, cos(x),"EJE X","coseno de x","COSENO")
```

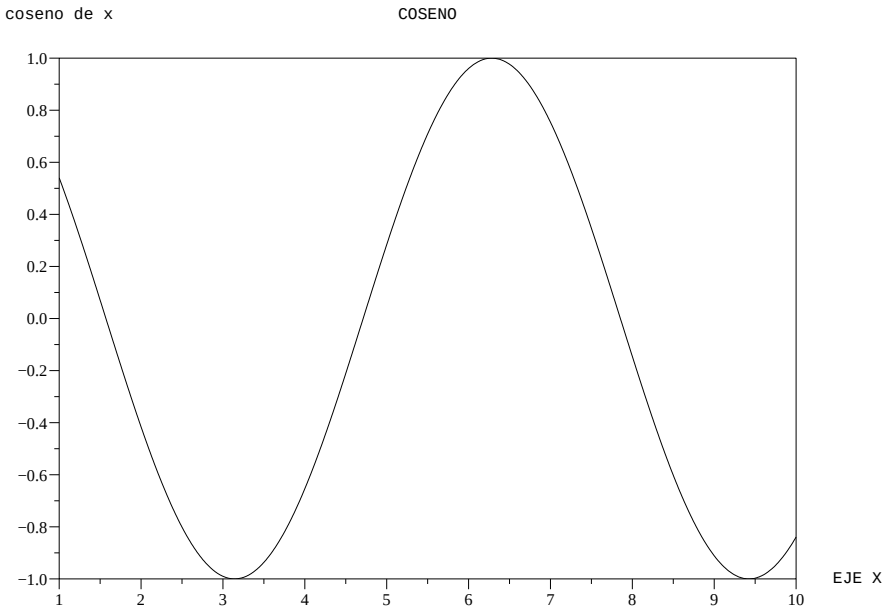


Figura 2.5: Gráfica de la función coseno con más componentes en el vector.

2.4.12.2. PLOT2D

El comando plot2d, nos permite elaborar gráficos con mejores características de la siguiente forma:

```
plot2d(variable indep,variable depend, "argumentos")
```

Ejemplo:

```
-->x=[-6:0.05:6]; -->plot2d(x,sin(2*x),rect=[-4,-1.1,4,1.1])
```

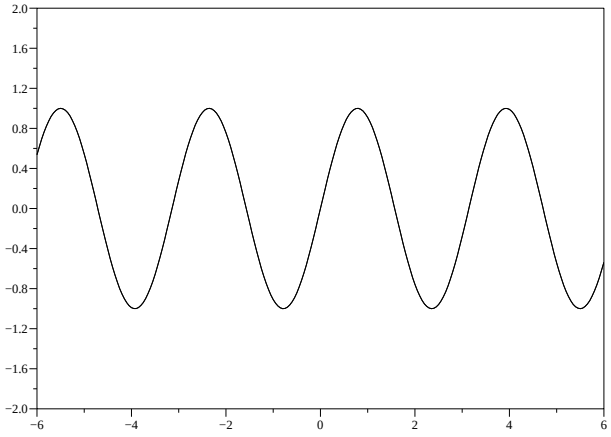


Figura 2.6: Gráfica con plot2d

Otras formas de mostrar esta misma gráfica:

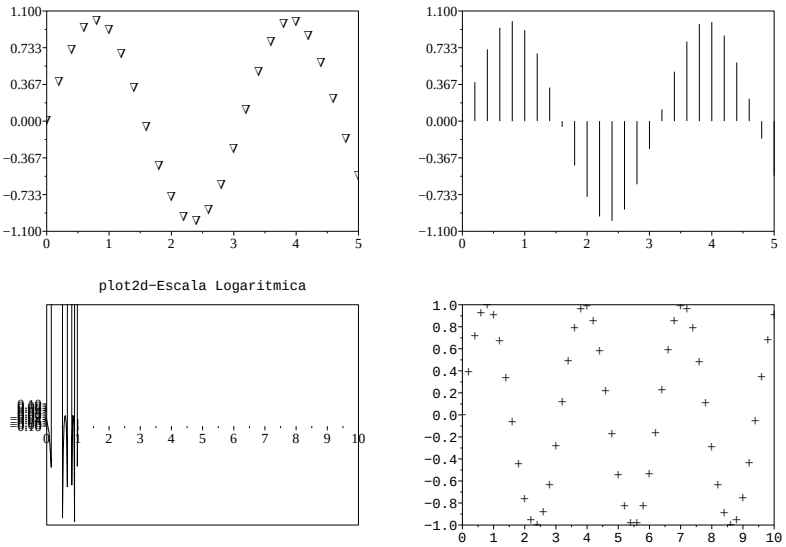


Figura 2.7: Otra gráfica con plot2d

2.4.13. MANEJO DE FUNCIONES

Consideremos el problema en donde debemos dar solución varias veces a una función en donde las operaciones son complicadas, y podemos tener un margen de error grande

al equivocarnos simultáneamente en la solución de la misma; SCILAB nos permite dar solución a este tipo de funciones, con el solo hecho de definir la función e ingresar los parámetros a solucionar. Veamos un ejemplo:
la función:

$$nombrefuncion = sen(x) + cos(y) + xy^2 - \sqrt{48^x}$$

evaluada en:
x=4
y=3

```
-->deff('[R]=nombrefuncion(x,y)','R=sin(x)+cos(y)+x*y**2-sqrt(48**x)')
-->nombrefuncion(4,3)
ans =
- 2269.7468
```

en donde "R" es la variable de salida y "x" , "y" las variables de entrada, "nombrefuncion" el nombre con la que llamaremos la función.

Capítulo 3

MATEMÁTICAS CON SCILAB

3.1. ÁLGEBRA LINEAL

A través de los capítulos de este libro hemos ido introduciendo conceptos elementales que son necesarios para la comprensión en el uso de SCILAB, el capitulo anterior muestra como introducir variables y hacer operaciones con ellas, trabajar con constantes del programa y otras tareas, sobre las cuales iremos profundizando en los capítulos posteriores.

Este capítulo aborda el Álgebra Lineal como parte fundamental para el trabajo con SCILAB, debido a la importancia que este tema tendrá en los capítulos siguientes intentaremos mostrar las distintas formas que se emplean para desarrollar tareas relacionadas con el manejo de vectores, ecuaciones lineales y matrices

3.1.1. VECTORES

dentro del trabajo en ciencias las cantidades son de vital importancia, así cómo regularmente el análisis de algunos fenómenos depende de cantidades escalares como la temperatura, la resistencia eléctrica, la longitud, etc. y existen métodos para hacer operaciones con esas cantidades como la suma o el producto, también es necesario trabajar con cantidades que además de magnitud poseen dirección y sentido, es el caso de los vectores que pueden representar un comportamiento detallado de una gran variedad de fenómenos tales como la fuerza, la velocidad, el desplazamiento, velocidad angular,el campo y la corriente eléctrica. Dichas cantidades pueden ser representadas en el plano o en el espacio.

3.1.1.1. VECTOR FILA DE N COMPONENTES.

Se define a un vector fila de n componentes como un conjunto ordenado de n números escritos de la siguiente manera:

$(x_1, x_2, x_3 \cdots x_n)$

En SCILAB:

```
-->vecfil=[7 2.5 3 5.6 2.2 4.8];
```

3.1.1.2. VECTOR COLUMNA DE N COMPONENTES.

Un vector columna de n componentes es un conjunto ordenado de n números escritos de la siguiente manera:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix}$$

3.1.1.3. COMPONENTES DE UN VECTOR

En los anteriores vectores, se llama a x1 primera componente, a x2 segunda componente , y asi sucesivamente. En general, xk se llama a la k-ésima componente del vector. Los componentes de un vector pueden ser: escalares, funciones, polinomios, complejos, etc.

En SCILAB:

```
-->x=poly(0,'x');
-->a=[1 6];
-->b=[1/x, a*x, x**2]
b =
!           2 !
! 1  x  6x  x ! ,o,   b = (1/x, x/1, 6x/1, x^2/1)
! -  - ---  - !
! x  1  1   1 !
```

3.1.1.4. VECTOR CERO

Es el vector cuyos elementos sean todos cero.Para crear un vector con esta característica se usa el comando zeros de SCILAB, así:

```
-->zeros(1,3)
```

```
ans =
! 0. 0. 0. !
```

3.1.1.5. VECTOR UNOS

Este vector se caracteriza por que todas sus componentes son uno's, se realiza de la siguiente forma:

```
-->ones(0:5)
ans =
! 1. 1. 1. 1. 1. 1. !
```

también podemos cambiar todas las componentes de un vector por uno's con el mismo comando.

```
-->j=[1 4 7];
-->ones(j)
ans =
! 1. 1. 1. !
```

OPERACIONES CON VECTORES

3.1.1.6. ADICIÓN

Sean `a` y `b` dos vectores:

$a = [a_1, a_2, \cdots, a_n]$
$b = [b_1, b_2, \cdots, b_n]$

la suma de `a` y `b`, escrita `a+b`, es el vector que se obtiene sumando las componentes correspondientes:

$$a + b = [a_1 + b_1, a_2 + b_2, \cdots, a_n + b_n]$$

en SCILAB, la suma de vectores se realiza de la siguiente forma:

Declaración de los vectores.

Declaración de la operación a realizar:

Resultado.

Ejemplo:

```
-->a=[1:1:5];
-->b=[4 3 4 7 8];
-->a+b
ans =
! 5. 5. 7. 11. 13. !
```

recordemos que para sumar dos vectores, estos deben ser del mismo tamaño, y forma, de lo contrario podemos obtener un error.

```
--->n=[4 5 6]
n =
! 4. 5. 6. !
-->m=[1;5;7]
m =
! 1. !
! 5. !
! 7. !
-->n+m
  !--error 8
inconsistent addition
```

El error se produce al intentar sumar un vector fila con un vector columna

3.1.1.7. MULTIPLICACIÓN POR UN ESCALAR

El producto de un numero real k por el vector a, escrito ka, es el vector que se obtiene multiplicando cada componente de a por k:

$ka = (ka_1, ka_2, \cdots, ka_n)$

Ejemplo:

```
-->n=[4 5 6]
n =
! 4. 5. 6. !
-->5n
ans =
! 20. 25. 30. !
```

3.1.1.8. PRODUCTO PUNTO

El producto punto o interno de n y m y denotado por n*m, es el escalar que se obtiene multiplicando las componentes de los vectores fila n y columna m y sumando luego los productos resultantes.

Usando SCILAB

->n=[4 8 6];

->m=[2;6;8];

->n*m

ans =

104.

Hay que tener en cuenta que el producto punto no es conmutativo, es decir, no es igual multiplicar :

vector fila * vetor columna $\neq$ vector columna * vector fila.

```
-->n*m
ans = 104.
-->m*n
ans =
! 8.  16. 12. !
! 24. 48. 36. !
! 32. 64. 48. !
```

3.1.1.9. PRODUCTO CRUZ

El producto cruz es el vector resultante de la multiplicación termino a termino de dos vectores fila ó columna de la misma dimensión.

En SCILAB:

```
-->a=[4 5 9];
-->b=[7 1 2];
-->a.*b
ans =
! 28. 5. 18. !
```

3.1.1.10. NORMA DE UN VECTOR

la norma o magnitud $\|a\|$ de un vector a está dada por la distancia entre los extremos de este; y su definición matemática es:

$a = [a_1, a_2]$
$\|a\| = \sqrt{a_1^2 + a_2^2}$

en Scilab utilizamos el siguiente comando para hallarla: "norm(vector)"
Ejemplo:

```
-->a=[4 5 9];
-->norm(a)
ans =
11.045361
```

3.1.1.11. VECTOR UNITARIO

el vector unitario se calcula tomando un vector y dividiéndolo por su norma, de esta manera obtenemos un nuevo vector, cuya magnitud es uno. Y dirección es igual al vector inicial.
En SCILAB:

```
-->x=[4 1 3];
-->vunit=(x/norm(x))
vunit =
! 0.7844645 0.1961161 0.5883484 !
```

podemos utilizar el vector vunit y comprobar su norma:

```
-->norm(vunit)
ans =
  1.
```

OTROS OPERADORES

3.1.1.12. POSICIÓN DE UNA COMPONENTE

podemos saber cual es la componente de un vector preguntando por su posición dentro del vector, introduciendo el valor j que determina la columna en la que se encuentra

la componente.
$a(j_n)$

para el vector "a" cuyas componentes van creciendo 0.33 desde -5 hasta 10, hallar la posición (15). utilizamos : vector([posición fila,columna])

```
-->a=[-5:0.33:10];
-->a(15)
ans =
  - 0.38
```

-0.38 es el valor de la componente del vector a que se encuentra en la posición o columna 15.

3.1.1.13. CAMBIO DE UNA COMPONENTE

para cambiar una componente por otra, igualamos la posición de la componente al valor que queremos cambiar.
Ejemplo:

```
-->n=[1 8 7 6 3];
-->n(3)=0
n =
! 1. 8. 0. 6. 3. !
```

en el ejemplo cambiamos la posición tres que tenía valor "7" por "0"

3.1.1.14. TRANSPUESTA DE UN VECTOR

la transpuesta de un vector es el vector resultante del cambio de filas por columnas, así la transpuesta de un vector fila es un vector columna con las mismas componentes, y viceversa.
Para transponer un vector colocamos una comilla simple al nombre de este, de la siguiente manera:
En SCILAB:

```
-->j=[6 3 2];
-->j'
ans =
! 6. !
! 3. !
```

```
! 2. !
```

3.2. MATRICES

Una matriz X de m x n es un arreglo rectangular de mn números dispuestos en m renglones y n columnas. El símbolo m x n se lee "m por n". el vector renglón $(a_{i1}, a_{i2}, \cdots, a_{in})$ se llama renglón i

y el vector columna $\begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}$ Se llama columna j.

3.2.1. COMPONENTES DE UNA MATRIZ

La componente o elemento ij de X denotado por a_{ij}, es el número que aparece en la fila i columna j de X. Por lo general las matrices se denotan con letras mayúsculas.

Si X es una matriz m x n con m=n, entonces X se llama matriz cuadrada. Una matriz m x n con todos los elementos iguales a cero se llama matriz cero de m x n.El tamaño de una matriz esta dado por m, n y se denota como matriz de m x n.

Para SCILAB, una matriz se construye, declarando una variable igualada a vectores fila del mismo tamaño, separados por punto y coma ";" entre paréntesis cuadrados "[ ]".
Usando SCILAB:

```
-->M=[4 6 7;1 5 3;7 5 3]
M =
! 4. 6. 7. !
! 1. 5. 3. !
! 7. 5. 3. !
```

También podemos declarar los vectores con anterioridad:

```
-->a=[4,6,7]; b=[1,5,3]; c=[7,5,3];
-->M=[a;b;c]
M =
! 4. 6. 7. !
! 1. 5. 3. !
```

```
! 7. 5. 3. !
```

Las componentes de la matriz pueden ser polinomios o expresiones matemáticas:

```
-->x=poly(0,'x');
-->a=[x/4,6/x,7x]; b=[1x**2,5x,3/x**2]; c=[7,5,3];
-->M=[a;b;c]
M =
! 0.25x  6   7x !
! -----  -  --- !
!   1    x   1  !
!               !
!  2            !
! x    5x     3 !
! -   ---     - !
!              2!
! 1    1      x !
!               !
! 7    5      3 !
! -    -      - !
! 1    1      1 !
```

para mejor entendimiento de la matriz:

$$M = \begin{bmatrix} \frac{0{,}25x}{1} & \frac{6}{x} & \frac{7x}{1} \\ \frac{x^2}{1} & \frac{5x}{1} & \frac{3}{x^2} \\ \frac{7}{1} & \frac{5}{1} & \frac{3}{1} \end{bmatrix}$$

Igual que con los vectores, también podemos ubicar una componente y cambiarla:

```
-->a=[4,6,7]; b=[1,5,3]; c=[7,5,3];
-->M(1,3)=0
M =
! 4. 6. 0. !
! 1. 5. 3. !
! 7. 5. 3. !
```

cambiamos la componente de la fila 1 columna 3 "7" por "0".

3.2.2. OPERACIONES BÁSICAS:

3.2.2.1. TRANSPUESTA DE UNA MATRIZ.

Se obtiene al cambiar filas por columnas y columnas por filas, así la matriz M=[aij] de i filas por j columnas, quedaria expresada como M'=[aji] de j filas por i columnas.

Ejemplo en SCILAB:

```
-->M
M =
! 4. 6. 7. !
! 1. 5. 3. !
! 7. 5. 3. !
```

Para obtener la transpuesta de la matriz M , la declaramos seguida de una comilla M', o comillas M"

```
-->M'
ans =
! 4. 1. 7. !
! 6. 5. 5. !
! 7. 3. 3. !
```

3.2.2.2. ADICIÓN Y SUSTRACCIÓN

la suma y resta de matrices tiene como resultado una nueva matriz de igual dimensión a las operadas, en donde sus componentes equivalen a la suma de las componentes de igual posición.

Ejemplo:

A=[aij]

B=[bij]

C=[(a+b)ij]

En SCILAB:

1)

```
-->a=[4,6,7]; b=[1,5,3];
-->a+b
ans =
! 5. 11. 10. !
```

```
-->a-b
ans =
! 3. 1. 4. !
```

2)

```
-->mA=[%pi 3.2; 4.7 8.5];
-->mB=[2.6 7.4; 7.9 6.8];
-->sumAB=mA+mB
sumAB =
! 5.7415927  10.6 !
! 12.6       15.3 !
```

3.2.2.3. MULTIPLICACIÓN POR UN ESCALAR

La multiplicación de una matriz M por un escalar K, da como resultado una matriz F, de la siguiente forma:

$F_{nxm} = kM_{nxm}$, *donde* $f_{ij} = km_{ij}$

Ejemplo en SCILAB:

```
-->M=[4 8 5; 4 7 3; 1 0 7];
-->K=3;
-->F=K*M
F =
! 12. 24. 15. !
! 12. 21.  9. !
!  3.  0. 21. !
```

3.2.2.4. MULTIPLICACIÓN DE MATRICES

Para multiplicar dos matrices A y B se requiere que el número de columnas de la primera matriz "A" sea igual al número de filas de la segunda matriz "B". El resultado de esta operación es una matriz con igual numero de filas de A, e igual numero de columnas de B y cuyos componentes serán el resultado de la siguiente expresión.

$$A_{2x3} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \qquad B_{3x2} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

Haciendo A*B

$$C_{2x2} = A*B = \begin{bmatrix} a_{11}*b_{11} + a_{12}*a_{21} + a_{13}*a_{31} & a_{11}*b_{12} + a_{12}*b_{22} + a_{13}*b_{32} \\ \\ a_{21}*b_{11} + a_{22}*b_{21} + a_{23}*b_{31} & a_{21}*b_{12} + a_{22}*b_{22} + a_{23}*b_{32} \end{bmatrix}$$

Usando SCILAB.

1)

```
-->A=[4 8 5; 4 7 3];
-->B=[7 8; 4 6; 4 3];
-->A*B
ans =
! 80. 95. !
! 68. 83. !
```

2.

```
-->A=[4 8 5];
-->B=[7 8; 4 6; 4 3];
-->A*B
ans =
! 80. 95. !
```

3.2.2.5. MATRIZ INVERSA

Se dice que una matriz A es inversible cuando cumple con las siguientes condiciones:

* Que la matriz sea cuadrada de tamaño n x m, donde n=m.

* Que Exista una matriz B con la propiedad de que :
A*B=B*A=I donde I es la matriz identidad. Tal matriz B es única
llamamos a la matriz B la inversa de A y la denotamos por A^{-1}. Obsérvese que la relación anterior es simétrica; esto es, si B es la inversa de A, entonces A es la inversa de B.

Ejemplo usando un método tradicional para hallar la inversa:

$$A = \begin{bmatrix} 4 & 5 & 9 \\ 7 & 4 & 3 \\ 8 & 6 & 3 \end{bmatrix} \qquad A^{-1} = \frac{1}{det\, A}(C_{ij})^T$$

Donde $(C_{ij})^T$ es la matriz de cofactores transpuesta y $C_{ij} = (-1)^{i+j} * M_{ij}$.

M_{ij} es el determinante de la matriz de cofactores.
usando SCILAB podemos hallar el determinante de la matriz A:

```
-->A=[4 5 9; 7 4 3; 8 6 3];
A =
! 4. 5. 9. !
! 7. 4. 3. !
! 8. 6. 3. !
-->de=det(A)
de =
  81.
```

ahora podemos determinar las matrices de cofactores:

```
-->co11=[4 3; 6 3]
co11 =
! 4. 3. !
! 6. 3. !
-->co12=[7 3; 8 3]
co12 =
! 7. 3. !
! 8. 3. !
-->co13=[7 4; 8 6]
co13 =
! 7. 4. !
! 8. 6. !
-->co21=[5 9; 6 3]
co21 =
! 5. 9. !
! 6. 3. !
-->co22=[4 9; 8 3]
```

```
co22 =
! 4. 9. !
! 8. 3. !
-->co23=[4 5; 8 6]
co23 =
! 4. 5. !
! 8. 6. !
-->co31=[5 9; 4 3]
co31 =
! 5. 9. !
! 4. 3. !
-->co32=[4 9; 7 3]
co32 =
! 4. 9. !
! 7. 3. !
-->co33=[4 5; 7 4]
co33 =
! 4. 5. !
! 7. 4. !
```

Teniendo las matrices de cofactores, podemos hallar el determinante de cada una para obtener

C_{ij} y $(C_{ij})^T$.

Usando SCILAB para allar C_{ij}.

```
-->c11=det(co11)*(-1)**2
c11 =
- 6.
-->c12=det(co12)*(-1)**3
c12 =
3.
-->c13=det(co13)*(-1)**4
c13 =
10.
-->c21=det(co21)*(-1)**3
c21 =
```

```
39.
-->c22=det(co22)*(-1)**4
c22 =
- 60.
-->c23=det(co23)*(-1)**5
c23 =
16.
-->c31=det(co31)*(-1)**4
c31 =
- 21.
-->c32=det(co32)*(-1)**5
c32 =
51.
-->c33=det(co33)*(-1)**6
c33 =
- 19.
Cij=[c11 c12 c13; c21 c22 c23; c31 c32 c33]
Cij =
! - 6.    3.   10. !
!  39. - 60.   16. !
! -21.   51. - 19. !
```

Usando SCILAB para transponer Cij y hallar $(C_{ij})^T$.

```
-->CijT=Cij'
CijT =
! -6.  39. -21. !
!  3. -60.  51. !
! 10.  16. -19. !
```

Usando SCILAB hallamos $\frac{(C_{ij})^T}{det\,A}$ o $MainvA = \frac{CijT}{de}$:

```
-->MainvA=CijT/de
MainvA =
! -0.0740741  0.4814815 -0.2592593 !
!  0.0370370 -0.7407407  0.6296296 !
!  0.1234568  0.1975309 -0.2345679 !
```

En SCILAB la matriz inversa puede ser calculada utilizando la función "inv" y evitar todo el desarrollo del ejercicio anterior asi:

"inv(nombre de la matriz)"

Ejemplo:

```
-->A=[4 5 9;7 4 3;8 6 3];
-->B=inv(A)
B =
! -0.0740741  0.4814815 -0.2592593 !
!  0.0370370 -0.7407407  0.6296296 !
!  0.1234568  0.1975309 -0.2345679 !
-->C=fix(A*B)
ans =
! 1. 0. 0. !
! 0. 1. 0. !
! 0. 0. 1. !
-->D=fix(B*A)
ans =
! 1. 0. 0. !
! 0. 1. 0. !
! 0. 0. 1. !
```

con el anterior ejemplo, obtenemos la inversa de la matriz A, $A^{-1} = B$ igual a la matriz B, y comprobamos la propiedad, en donde A por su inversa es igual a la matriz identidad.

NOTA: algunas matrices no tiene inversa, por ejemplo aquellas en donde todos los elementos de una de sus filas o columnas son cero, o son múltiplos de otra fila o columna respectivamente y se les conoce como matrices singulares.

3.2.2.6. MATRIZ IDENTIDAD

La matriz Identidad es aquella n-matriz cuadrada, con unos como elementos de su diagonal principal y ceros como elementos que no están en la diagonal, denotada por I_n.

Ejemplo:

$$I_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

En SCILAB podemos crear este tipo de matrices con la función "eye" asi:
"eye(número de filas, número de columnas)"
Ejemplo:

```
-->In=eye(2, 2)
In =
! 1. 0. !
! 0. 1. !
-->In2=eye(3, 3)
In2 =
! 1. 0. 0. !
! 0. 1. 0. !
! 0. 0. 1. !
```

3.2.2.7. TAMAÑO DE UN VECTOR O UNA MATRIZ

para determinar el numero de filas y/o columnas de un vector o una matriz, utilizamos en SCILAB la función "size()" así:
"size(nombre de la matriz o vector)"
Ejemplo:

```
-->Z=[4 8 9 6;7 9 6 2;4 7 8 8;1 2 4 5];
-->size(Z)
ans =
! 4. 4. !
```

3.2.2.8. EXTRACCIÓN Y CAMBIO DE ELEMENTOS DE UNA MATRIZ

En SCILAB, podemos saber cual es elemento que hay en una posición determinada de una matriz, y modificarlo si lo queremos
Ejemplo:

```
-->Z=[4 8 9 6;7 9 6 2;4 7 8 8;1 2 4 5];
-->Z(3,2)=0
Z =
! 4. 8. 9. 6. !
! 7. 9. 6. 2. !
```

```
! 4. 0. 8. 8. !
! 1. 2. 4. 5. !
```

el elemento fila 3 columna 2 de Z es siete "7", pero lo cambiamos por cero "0".

3.2.2.9. GENERACIÓN DE MATRICES CON NÚMEROS ALEATORIOS

En SCILAB utilizamos la función "rand" para generar números aleatorios con un intervalo entre "0 y 1", lo que nos lleva a poder construir vectores y matrices de la misma forma.
Ejemplo en SCILAB:

```
-->N=rand(3,3)
N =
! 0.0683740 0.7263507 0.2320748 !
! 0.5608486 0.1985144 0.2312237 !
! 0.6623569 0.5442573 0.2164633 !
```

en donde N es el nombre de la matriz, y rand(3,3) es la matriz aleatoria de tres filas por tres columnas.
También podemos construir una matriz aleatoria, combinando la función "rand" con la función "int".
Ejemplo en SCILAB:

```
-->H=int(6*rand(4,3))
H =
! 5. 1. 3. !
! 3. 1. 2. !
! 1. 2. 1. !
! 5. 1. 3. !
```

en donde "H" es el nombre de la matriz, "int" nos presenta un valor entero, "6" entero entre 0 y 6 , "rand(4,3)" la matriz aleatoria de 4 filas por 3 columnas.

3.2.2.10. RANGO DE UNA MATRIZ

el rango de una matriz A, denotado como rang(A), es el valor común de su rango fila y su rango columna. Así, el rango de una matriz es el máximo número de filas independientes, y también el máximo de columnas independientes.

En SCILAB, Podemos obtener el rango de una matriz utilizando la función "rank(A)". Ejemplo en SCILAB:
hallar el rango de la matriz J.

```
-->J=[1 2 0 -1;2 6 -3 -3;3 10 -6 -5]
J =
! 1.  2.  0. -1. !
! 2.  6. -3. -3. !
! 3. 10. -6. -5. !
-->rank(J)
ans =
2.
```

3.2.2.11. DETERMINANTE DE UNA MATRIZ

Sea la matriz $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ de (2 x 2), el determinante de esta matriz esta dado por:

$det\, A = |A| = a_{11}a_{22} - a_{21}a_{12}$

y se denota $|A|$

en SCILAB el determinante de una matriz lo obtenemos con la función "det(matriz)"
ejemplo En SCILAB:

```
-->A=[4 9 7; 1 5 9; 7 5 2];
-->det(A)
ans =
199.
```

Un ejemplo mas sencillo:

```
-->B=[4 9 ; 5 2];
-->det(B)
ans =
37.
```

3.2.2.12. MATRIZ SINGULAR

Se dice que si el determinante de una matriz cuadrada es igual cero, la matriz es singular y no invertible .

En SCILAB podemos conocer si una matriz es o no singular mediante la función "cond(matriz)", si la respuesta es un valor demasiado grande cuyo determinante es igual a cero, la matriz es singular de lo contrario es no singular.
Ejemplo en SCILAB.

1)

```
-->// Singular, No Invertible
-->D=[1 6 2;-2 3 5; 7 12 -4];
-->cond(D)
ans =
1.202E+16
-->det(D)
ans =
6.899E-15
```

2)

```
-->//No singular, Invertible
-->F=[1 -3 0 -2;3 -12 -2 -6;-2 10 2 5;-1 6 1 3];
-->cond(F)
ans =
135.68111
-->det(F)
ans =
- 1.
```

3.2.2.13. VALORES PROPIOS y VECTORES PROPIOS

Para la matriz A de n x m con componentes reales, el numero z(real o complejo) se llama valor propio de A si existe un vector diferente de cero v en Cn tal que:
Av=zv. El vector v$\neq$0 se llama vector propio de A correspondiente al valor propio z
con SCILAB podemos hallar estos valores utilizando la funcion "spec(matriz)".

Ejemplo en SCILAB:

```
-->//matriz de 2x2 con valores propios complejos conjugados
-->v=[3 -5;1 -1]
v =
! 3. - 5. !
! 1. - 1. !
-->spec(v)
```

```
ans =
! 1. + i !
! 1. - i !
```

valores propios= (1+i),(1-i)

```
-->//matriz de 3x3 con valores propios distintos.
-->f=[1 -1 4; 3 2 -1; 2 1 -1]
f =
! 1. -1.  4. !
! 3.  2. -1. !
! 2.  1. -1. !
-->spec(f)
ans =
!  3. !
!  1. !
! -2. !
```

3.2.3. SOLUCIÓN DE ECUACIONES LINEALES

linsolve:

la función linsolve en SCILAB, nos permite dar solución a un sistema de ecuaciones lineales, resolviéndolo como un sistema matricial, la forma general de esta función es:

[x0, kerA]=linsolve(A,b [,x0])

donde A es la matriz de n_a filas por m_a columnas, b es un vector de n_a filas, x0 el vector solución, kerA es la matriz de ma * k soluciones.

Todas las posibles soluciones de linsolve son de la forma:

A*x +b =0

Dependiendo del número de incógnitas y ecuaciones, la función linsolve producirá diferentes resultados.

Ejemplo:

resolver el sistema:

$$\begin{aligned} 2x + y - 2z &= 10 \\ 3x + 2y + 2z &= 1 \\ 5x + 4y + 3z &= 4 \end{aligned}$$

igualamos a "0".

$$\begin{aligned} 2x + y - 2z - 10 &= 0 \\ 3x + 2y + 2z - 1 &= 0 \\ 5x + 4y + 3z - 4 &= 0 \end{aligned}$$

lo transformamos de la forma $(Ax + b = 0)$

$$A = \begin{matrix} 2 & 1 & -2 \\ 3 & 2 & 2 \\ 5 & 4 & 3 \end{matrix}, \quad x = \begin{matrix} x \\ y \\ z \end{matrix}, \quad c = \begin{matrix} -10 \\ -1 \\ -4 \end{matrix}$$

En SCILAB:

```
-->A=[2 1 -2;3 2 2;5 4 3], c=[-10; -1; -4]
A =
! 2. 1. -2. !
! 3. 2.  2. !
! 5. 4.  3. !
c =
! -10. !
! -1.  !
! -4.  !
-->[x0,kerA]=linsolve(A,c)
kerA =
[]
x0 =
!  1. !
!  2. !
! -3. !
```

aquí la solución es:

x=1, y=2, z=-3

como "kerA=0" indica que solo hay una única solución.

APLICACIONES

SOLUCIÓN DE MÚLTIPLES ECUACIONES CON COEFICIENTES IGUALES.

Solucionar los siguientes sistemas de ecuaciones lineales:

5x + 2y +4z +w =10

4x +7y +z + 2w =7

7x + y +9z +w =5

x + 4y +9Z+2w =2

5x + 2y +4z +w =7
4x +7y +z + 2w =7
7x + y +9z +w =9
x + 4y +9Z+2w =4

5x + 2y +4z +w =6
4x +7y +z + 2w =7
7x + y +9z +w =9
x + 4y +9Z+2w =14

Para la solución múltiple, el primer paso es transformar los sistemas a la forma Ax=b

$$A = \begin{bmatrix} 5 & 2 & 4 & 1 \\ 4 & 7 & 1 & 2 \\ 7 & 1 & 9 & 1 \\ 1 & 4 & 9 & 2 \end{bmatrix} \qquad X = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \qquad B = \begin{bmatrix} 10 & 7 & 6 \\ 7 & 7 & 7 \\ 5 & 9 & 9 \\ 2 & 4 & 14 \end{bmatrix}$$

Usando SCILAB:

```
-->A=[5 2 4 1;4 7 1 2;7 1 9 1;1 4 9 2], B=[10 7 6;7 7 7;5 9 9;2 4 14],
A =
! 5. 2. 4. 1. !
! 4. 7. 1. 2. !
! 7. 1. 9. 1. !
! 1. 4. 9. 2. !
B =
! 10. 7. 6. !
!  7. 7. 7. !
!  5. 9. 9. !
! 2. 4. 14. !
-->X=inv(A)*B
X =
! 1.6111111   1.0972222 -0.1666667 !
! -9.2777778 -0.4305556 -0.8333333 !
! -3.5       -0.125      0.5       !
! 34.5        2.875      6.5       !
```

la matriz X resultante contiene los valores de las variables x , y, z, w por columnas de cada uno de los sistemas.

3.2.4. Breve ejemplo de aplicación usando circuitos eléctricos

Hallar el valor de las corrientes, del circuito de la siguiente figura.

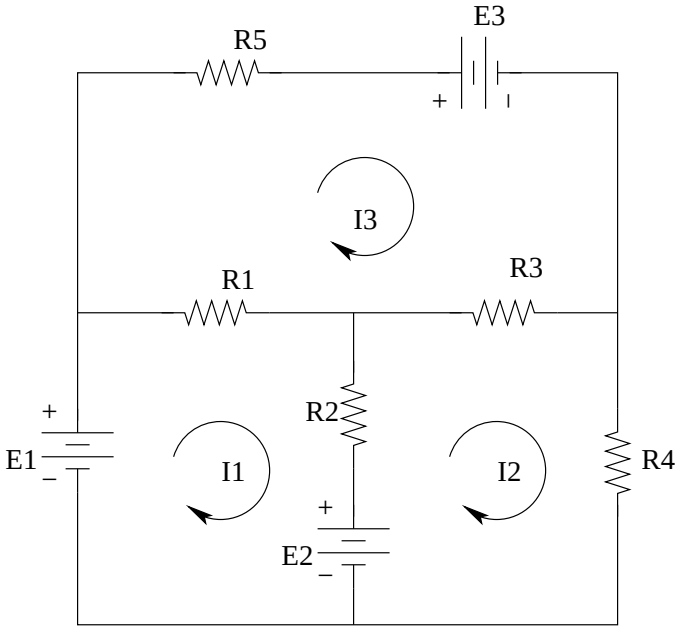


Figura 3.1: Circuito con 3 mallas

Usando LVK (Ley de Voltajes de Kirchhoff), hallamos la matriz de voltaje.
$-E_1$+E_2 +(R1+R2)i_1 -R2i_2 -R1i_3= 0
$-E_2$ -(R2)i_1 +(R2+R3+R4)i_2 -(R3)i_3= 0
E_3 -(R1)i_1 -(R3)i_2 +(R1+R3+R5)i_3= 0

Ordenando

$$\begin{array}{rrrl} (\text{R1+R2})i_1 & \text{-R2}i_2 & \text{-R1}i_3 = & E_1 - E_2 \\ \text{-(R2)}i_1 & +(\text{R2+R3+R4})i_2 & \text{-(R3)}i_3 = & E_2 \\ \text{-(R1)}i_1 & \text{-(R3)}i_2 & +(\text{R1+R3+R5})i_3 = & -E_3 \end{array}$$

suponemos los siguientes valores para:

$R_1 = 4\Omega$
$R_2 = 8\Omega$
$R_3 = 7\Omega$
$R_4 = 9\Omega$

$R_5 = 2\Omega$

$E_1 = 9V$
$E_2 = 6V$
$E_3 = 8V$

Organizamos en la forma Ax=B

$$\begin{bmatrix} 12 & -8 & -4 \\ -8 & 24 & 7 \\ 4 & -7 & 13 \end{bmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} = \begin{bmatrix} 3 \\ 6 \\ -8 \end{bmatrix}$$

En SCILAB:

```
-->R=[12 -8 -4;-8 24 -7;-4 -7 13], V=[3;6;-8]
R =
! 12. -8. -4. !
! -8. 24. -7. !
! -4. -7. 13. !
V =
!  3. !
!  6. !
! -8. !
-->I=inv(R)*V
I =
!  0.2446381 !
!  0.2064343 !
! -0.4289544 !
```

solución:

i_1= 0.2446381 A
i_2= 0.2064343 A
i_3= -0.4289544 A

3.3. POLINOMIOS

3.3.1. CONSTRUCCIÓN DE POLINOMIOS

En SCILAB un polinomio es representado como un vector fila, pero declarado en forma de polinomio a través de la función "poly", el siguiente vector:

$a = [1\,2\,3\,7\,8]$

es una es una secuencia de datos en la que cada componente del vector puede ser usada como coeficiente de un polinomio en terminos de una variable x.

$1 + 2x + 3x^2 + 7x^3 + 8x^4$

para mas claridad usando SCILAB:

```
-->//declaracion de a como vector fila
-->a=[1 2 3 7 8]
a =
! 1. 2. 3. 7. 8. !
-->//declaracion de fx como polinomio, cuyos coeficientes son las
-->//componentes de a
-->fx=poly(a,'x','coeff')
fx =
1 + 2x + 3x^2 + 7x^3 + 8x^4
```

Otra forma de introducir un polinomio es creando con anterioridad la variable a partir de la raíz "0" y el comando "poly" así:

```
-->x=poly(0,'x')
x =
x
-->pol1=x**2+5*x**5+7*x**3
pol1 =
x^2 + 7x^3 + 5x^5
```

suponiendo que las componentes del siguiente vector rz son las raices de un polinomio, con poly podemos crear un polinomio a partir de sus raices:

```
-->rz=[-1 -2]
rz = ! - 1. - 2. !
-->pol2=poly(rz,'p')
pol2 =
2 + 3p + p^2
```

3.3.2. RAÍCES DE UN POLINOMIO

para encontrar las raíces de un polinomio en SCILAB, utilizamos la función "roots(polinomio)",usando el polinomio pol2 del ejemplo anterior, podemos comprobar que las raíces son las componentes del vector rz:

```
-->pol2
pol2 =
2 + 3p + p^2
-->ra1=roots(pol2)
ra1 =
! - 1. !
! - 2. !
```

Otro ejemplo con el siguiente polinomio:

$pol3 = 10 + 22s + 4s^2 + 26s^3 + 15s^5 + s^7$

```
-->//vector de coeficientes
-->coe=[10 22 4 26 0 15 0 1]
coe =
! 10. 22. 4. 26. 0. 15. 0. 1. !
-->//creacion del polinomio "pol3"
-->pol3=poly(coe,'s','coeff')
pol3 =
10 + 22s + 4s^2 + 26s^3 + 15s^5 + s^7
-->r=roots(pol3)
r =
! -0.4006244              !
! -0.3364228 + 1.1203582i !
! -0.3364228 - 1.1203582i !
!  0.5475124 + 1.0427372i !
!  0.5475124 - 1.0427372i !
! -0.0107774 + 3.626405i  !
! -0.0107774 - 3.626405i  !
```

nótese que "pol3" tiene una raíz real y varias raíces complejas

3.3.3. EVALUACIÓN DE UN POLINOMIO

podemos evaluar un polinomio con la función "horner" de la siguiente forma:
horner(polinomio,valor a evaluar)
para el ejemplo anterior, podemos evaluar el polinomio usando las cuatro primeras raíces.

```
-->ev1=horner(pol3,r(1,1))
ev1 =
0
-->ev2=horner(pol3,r(2,1))
ev2 = 5.329E-15 - 6.585E-16i
-->//usando round para aproximar a cero.
-->ev2=round(horner(pol3,r(2,1)))
ev2 =
0
-->ev3=round(horner(pol3,r(3,1)))
ev3 =
0
-->ev4=round(horner(pol3,r(4,1)))
ev4 =
0
```

3.3.4. DIVISIÓN DE POLINOMIOS

En SCILAB podemos utilizar la función "pdiv()" para efectuar divisiones entre polinomios. Agregando algunos argumentos podemos también hallar el residuo de dicha división.
Ejemplo:

```
-->//Division de polinomios usando "pdiv"
-->x=poly(0,'x')
x =
x
-->p=1+x**2+2*x**3
p =
1 + x^2 + 2x^3
-->q=1-2*x
q = 1 - 2x
-->pdiv(p,q)
ans =
- 0.5 - x - x^2
```

en el ejemplo anterior dividimos el polinomio p entre el polinomio q, sin obtener el residuo de la división, pero para obtenerlo, igualamos la función a dos variables, la

primera será el cociente y la segunda el residuo.

[residuo, cociente]=pdiv(dividendo,divisor)

Ejemplo:

```
-->//division con pdiv, mostrando residuo y cociente
-->s=poly(0,'s')
s =
s
-->poli1=2+3*s+s**2
poli1 =
2 + 3s + s^2
-->poli2=3*s+1
poli2 = 1 + 3s
-->[residuo, cociente]=pdiv(poli1,poli2)
cociente =
0.8888889 + 0.3333333s
residuo =
1.1111111
```

Y de esa forma se obtiene el cociente y el residuo de la división de polinomios con "pdiv"

3.3.5. DERIVADA DE UN POLINOMIO

En SCILAB, podemos conocer la derivada de una función polinómica, utilizando la función "derivat()"
Usando SCILAB.

```
-->//Declaracion de un polinomio
-->x=poly(0,'x')
x =
x
-->num=-5-2*x
num =
- 5 - 2x
-->den=25*x**2+10*x**3+x**4
den =
```

```
25x^2 + 10x^3 + x^4
-->p=num/den
p =
  - 5 - 2x
---------------
25x^2 + 10x^3 + x^4
-->//Derivada del polinomio p
-->Dx=derivat(p)
Dx =
    250x + 200x^2 + 60x^3 + 6x^4
---------------------------------
625x^4 + 500x^5 + 150x^6 + 20x^7 + x^8
```

Podemos también evaluar el polinomio derivado Dx

```
-->//vector fila con componentes para evaluar Dx
-->val=1:6
val = ! 1. 2. 3. 4. 5. 6. !
-->evalDx=horner(Dx,val)
evalDx =
! 0.3981481 0.0488338 0.0140336 0.0057013 0.0028 0.0015513 !
```

También podemos obtener las raíces de Dx.

```
-->//raices de Dx
-->rDx=roots(numer(Dx))
rDx =
!  0                !
! -2.5  +1.4433757i !
! -2.5  -1.4433757i !
! -5.               !
```

NOTA: no siempre la raiz de un polinomio, es evaluable y considerable en cero, puesto que para el anterior ejemplo se convierte en (0/0) y este problema no esta definido, por ,lo anterior es mejor seguir las siguientes instrucciones:

reducir el polinomio a una expresión mas manejable, para ello extraemos el numerador y el denominador y los dividimos entre si

```
-->Dx
Dx =
   250x + 200x^2 + 60x^3 + 6x^4
---------------------------------
625x^4 + 500x^5 + 150x^6 + 20x^7 + x^8
-->//numerador de Dx usando numer()
-->numDx=numer(Dx)
numDx =
250x + 200x^2 + 60x^3 + 6x^4
-->//denominador de Dx usando denom()
-->denDx=denom(Dx)
denDx =
625x^4 + 500x^5 + 150x^6 + 20x^7 + x^8
-->//dividiendo los dos polinomios
-->repol=numDx/denDx
repol =
     50 + 30x + 6x^2
-------------------------
125x^3 + 75x^4 + 15x^5 + x^6
```

el paso a seguir es hallar la raíz del numerador del nuevo polinomio (repol).

```
-->//extraer el numerador de repol
-->nurepol=numer(repol)
nurepol =
50 + 30x + 6x^2
-->//hallar raices de nurepol
-->roots(nurepol)
ans =
! - 2.5 + 1.4433757i !
! - 2.5 - 1.4433757i !
```

3.4. FUNCIONES

En SCILAB podemos definir una función y evaluarla en, el o los valores que necesitemos. Para definir una función utilizaremos el comando deff:

deff('[resp1,resp2,...]=nombre_función(valorin1,valorin2,...)','declarar función')
resp1 y resp2 variables de salida
valorin1 y valorin2 variables de entrada.

Como ejemplo podemos definir la siguiente función:

$a = x + 5x^2 + x^3 + \frac{x}{3} + x^7$

En SCILAB:

–>//definir funcion con deff()
–>deff('a=fun(x)','a=x+5*x**2+x**3+(x/3)+x**7')

Ahora bien, para evaluar la función se escribe el nombre de la funcion con el argumento que queremos para evaluarla, el argumento puede ser un valor único o un vector para obtener diversos valores en un determinado dominio representado por el vector.

```
-->//usando valores unicos
-->fun(1)
ans =
8.3333333
-->fun(-1)
ans =
1.6666667
-->fun(-2)
ans =
- 118.66667
-->//usando un vector como argumento
->vec=-10:10;
-->revec=fun(vec)
revec =
column 1 to 5
! - 10000513. - 4783305. - 2097354.7 - 823650.33 - 279980. !
column 6 to 11
! - 78131.667 - 16373.333 - 2173. - 118.66667 1.6666667 0. !
column 12 to 17
! 8.3333333 158.66667 2263. 16533.333 78381.667 280340. !
column 18 to 21
```

```
! 824140.33 2097994.7 4784115. 10001513. !
```

Cada componente desde la columna 1 hasta la 21 de revec, es el resultado de evaluar cada componente del vector vec como argumento de fun.

3.5. NÚMEROS COMPLEJOS

Generalmente cuando se realizan operaciones matemáticas y cálculos para determinar resultados de algunas variables, manipulamos números reales como 0.891, $\pi\,(3{,}1415927)$, $\exp(2{,}7182818)$, o $-\frac{3}{8}$ y así sucesivamente, con estos números se pueden medir ciertos resultados relacionados con fenómenos como el voltaje o la corriente, sin embargo, las variables relacionadas con algunos comportamientos nos pueden llevar a resultados diferentes, que nos obligan a manipular números complejos, nótese que en el siguiente planteamiento la solución para x es:

$x^2 + 40{,}96 = 0$
$x^2 = -40{,}96$
$\sqrt{x^2} = \sqrt{-40{,}96}$
$x = \sqrt{-1} \cdot \sqrt{40{,}96}$

La solución no es un número real y generalmente es llamado número imaginario, para manipular números imaginarios, esta definida una unidad imaginaria, denotada por j en los libros de ingeniería o i en los textos de matemáticas (en SCILAB se usa i).

$i = \sqrt{-1}$

En SCILAB:

```
-->//unidad imaginaria
-->sqrt(-1)
ans =
   i
```

Y podemos deducir que:

$i^2 = -1, \quad i^3 = -i, \quad i^4 = 1...$ etc.

En SCILAB podemos representar la unidad imaginaria i con la constante especial " %i" y a partir de ella crear números imaginarios y números complejos.

```
-->//representacion de la unidad imaginaria usando%i
-->%i
%i =
i
-->%i**2
ans =
- 1.
-->%i**3
ans =
- i
-->%i**4
ans =
1.
```

Cómo crear números imaginarios y números complejos:

Para crear un número imaginario se realiza el producto entre un escalar y la unidad imaginaria representada por "%i"

```
-->//creacion de numeros imaginarios
-->4.8*%i
ans =
4.8i
-->imag1=2*%i
imag1 =
2.i
-->imag2=%pi*%i
imag2 =
3.1415927i
```

Generalmente un número complejo es la suma de un número real y un número imaginario

$z = x + yi$

z es un número complejo, donde x e y son reales, z tiene parte real x y parte imaginaria y, expresada de la siguiente manera:

$Re(z) = x$
$Im(z) = y$

La representación cartesiana de z equivaldría a:

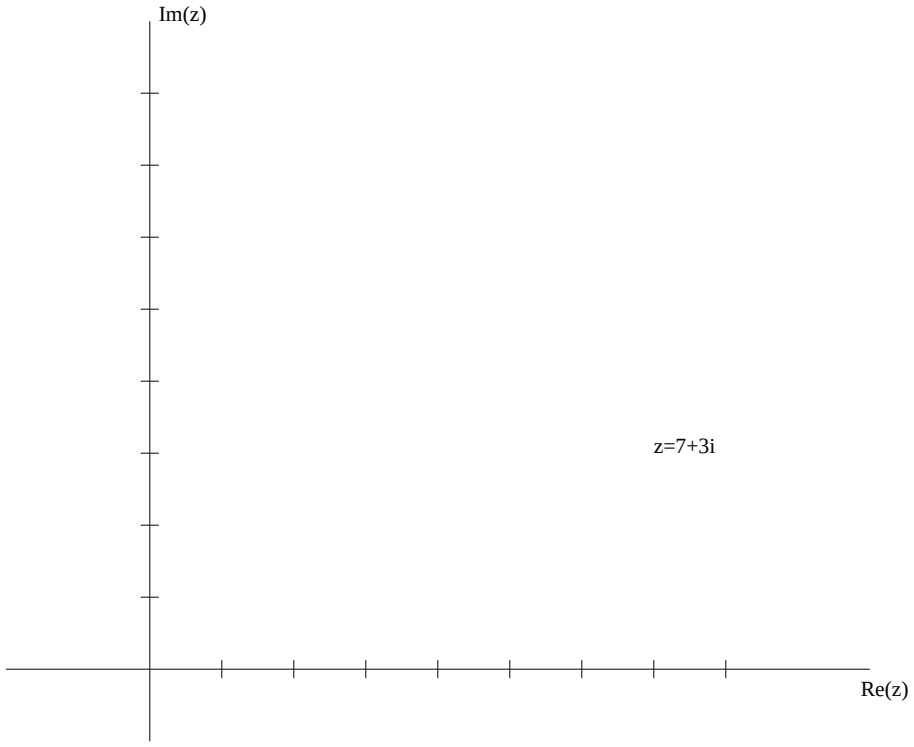


Figura 3.2: plano con Re(z) e Im(z)

La variable independiente o eje x = Re(z) Variable dependiente o eje y =Im(z). En la figura 3.2 se indica la forma cartesiana del número complejo a la que también se le conoce como forma rectangular.

$z = 7 + 3i$
$Re(z) = 7$
$Im(z) = 3$

Teniendo un numero complejo en forma rectangular, usando SCILAB podemos extraer su parte real y su parte imaginaria así:

```
-->//extraccion de la parte real e imaginaria de un numero complejo
-->A1=12.5+6.3*%i
A1 =
```

```
12.5 + 6.3i

-->real(A1)

ans =

12.5

-->imag(A1)

ans =

6.3

-->//otro ejemplo

-->A2=%pi+2*%pi*%i

A2 =

3.1415927 + 6.2831853i

-->real(A2)

ans =

3.1415927

-->imag(A2)

ans =

6.2831853
```

si tomamos el plano cartesiano y trazamos una linea desde el origen hasta el punto final podemos determinar la distancia r y el ángulo theta que se forma entre la linea y el eje real.

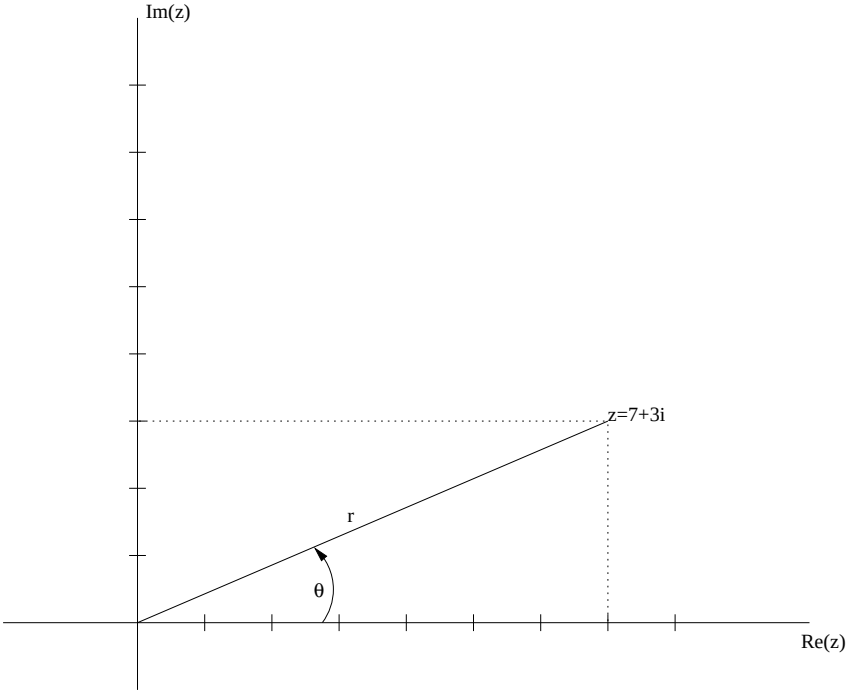


Figura 3.3: Forma polar de z

y podemos determinar r y θ.

$r = \sqrt{x^2 + y^2}$
$\theta = \tan^{-1} \frac{b}{a}$

$x = r * cos\theta$
$y = r * sen\theta$

Teniendo r y θ podemos representar un número complejo en forma polar de la siguiente forma:

$A = r\angle\theta$

La representación gráfica de A se muestra en la figura 3.3.

Con SCILAB podemos comprobar lo anterior:

```
-->//ingresar numero complejo en forma rectangular
-->A=4+3*%i
```

```
A =
4. + 3.i
-->//extraer Re(A) e Im(A)
-->reA=real(A);
imA=imag(A);
-->//distancia de r
-->r=sqrt(reA**2+imA**2)
r =
5.
-->//hallar el valor de theta
-->thetarad=atan(imA/reA); theta=thetarad*180/%pi
theta =
36.869898
-->//r y theta
-->r,theta
r =
5.
theta =
36.869898
```

3.5.1. El conjugado de un número complejo

Se realiza con el comando conj().

```
-->//conjugado de B
-->B=3+2.8*%i
B =
3. + 2.8i
-->cB=conj(B)
cB =
3. - 2.8i
```

Sí se hace la transpuesta de una matriz en la que hay componentes imaginarios o números complejos, SCILAB hace el conjugado de estas componentes así:

```
-->//matriz im con componentes complejos
-->im=[2, 3+2*%i; 3*%i 3.8]
im =
```

```
! 2.   3.+ 2.i !
! 3.i  3.8     !
-->//transpuesta de im
-->im'
ans =
! 2.       -3.i !
! 3. - 2.i  3.8 !
```

Para evitar esto, podemos realizar la conjugada de la matriz y después transponer, o, usar el comando mtlb_0().

```
-->//usando conjugada y transponiendo
-->co1=conj(im)
co1 =
! 2.    3.- 2.i !
! -3.i  3.8     !
-->co1'
ans =
! 2.       3.i !
! 3.+ 2.i  3.8 !
-->//conjugado de im usando el comando mtlb_0
-->mtlb_0(im)
ans =
! 2.       3.i !
! 3.+ 2.i  3.8 !
```

3.5.2. Suma y resta de números complejos.

```
-->//suma de numeros complejos
-->nima=1+%i
nima =
1. + i
-->nima2=3+3*%i
nima2 = 3. + 3.i
-->suma=nima+nima2
suma =
4. + 4.i
```

```
-->//resta de numeros complejos

-->resta=nima2-nima

resta =

2. + 2.i
```

3.5.3. Producto y división de números complejos.

```
-->//producto de numeros complejos

-->im2=4+3*%i; im3=6+4*%i;

-->re=im2*im3

re =

12. + 34.i

-->//division de numeros complejos

-->re1=im2/im3

re1 =

0.6923077 + 0.0384615i
```

3.5.4. La función exponencial compleja

En ingeniería electrónica, eléctrica y control los números complejos son de uso frecuente, debido a la relación entre los senos y los cosenos y la función exponencial compleja $e^{j\theta}$

$e^{j\theta} = cos\theta + jsen\theta$ Forma rectangular de la fórmula de Euler

A partir de la formula de Euler, en SCILAB podemos modelar el movimiento gradual de un fasor, para observar el comportamiento de forma clara, cambiaremos la longitud r del vector que se moverá desde un ángulo igual a 5^o hasta 85^o.

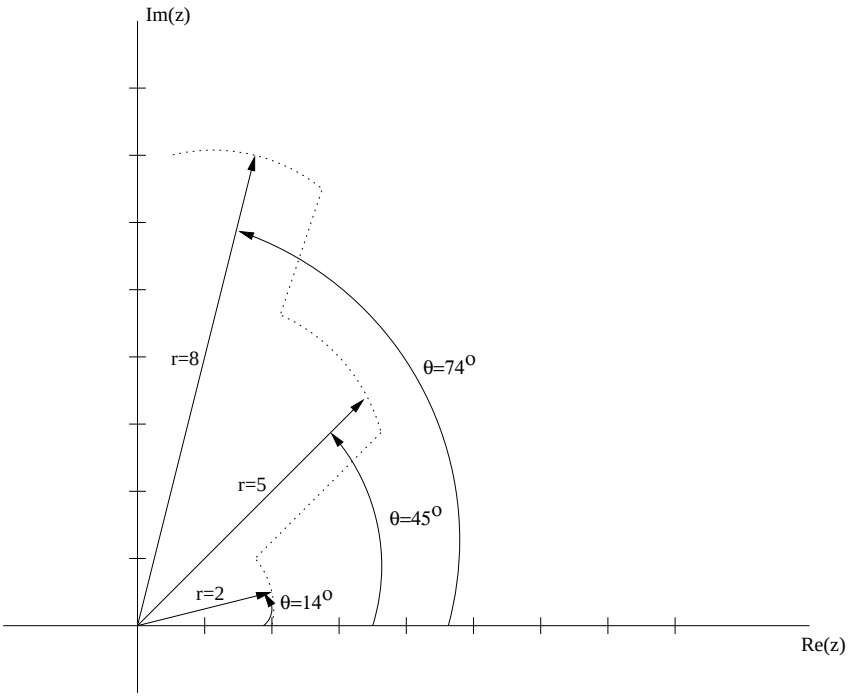


Figura 3.4: vector girando con diferente magnitud

Primero debemos declarar el vector `r` que representará la longitud cambiante del fasor, haremos que tenga 3 magnitudes: 2, 5 y 8, la gráfica numero 3.4 muestra 3 momentos diferentes del fasor con sus respectivos ángulos y magnitudes.

Crear el vector `r` implica manipular cada una de las componentes que le darán al fasor la magnitud cambiante a medida que gire gradualmente, para cada uno de los movimientos tendrá una magnitud, así que haremos `r` de 1 fila por 80 columnas y usaremos vectores 1's para generar los cambios de magnitud.

```
-->//creando vector r con magnitudes 2, 5 y 8
-->r=[2*ones(1,27), 5*ones(1,27), 8*ones(1,26)];
-->size(r)
ans =
! 1. 80. !
```

Para generar cada uno de las posiciones en grados según el movimiento del vector tendremos que la posición inicial es 5 grados y la posición final 85 grados, haremos la conversión a radianes y generaremos un vector theta de 1 fila por 80 columnas que represente cada uno de los grados.

```
-->//valor de 5 y 85 grados en radianes
```

```
-->vin=(5*%pi)/180

vin =

0.0872665

-->vfi=(85*%pi)/180

vfi =

1.4835299

-->//generar rangos de division entre componentes del vector theta

-->d=vfi-vin

d =

1.3962634

-->ran=d/79

ran =

0.0176742

-->//generar vector theta de 1x80

-->theta=vin:ran:vfi;

-->size(theta)

ans =

! 1. 80. !

-->//generando funcion exponencial compleja

-->fexp=r.*cos(theta)+r.*sin(theta)*%i;

-->//grafica del fasor

-->plot2d(real(fexp),imag(fexp),rect=[0,0,15,9])
```

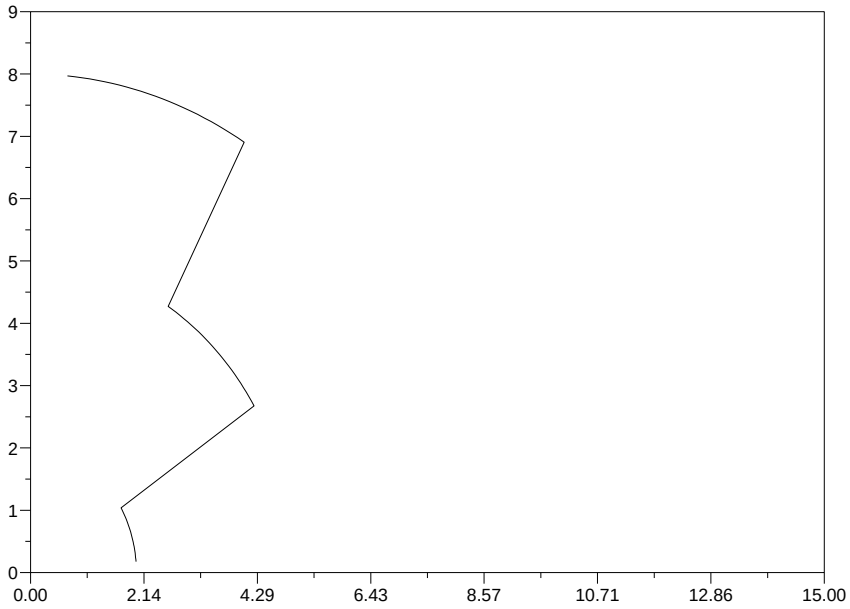


Figura 3.5: Gráfica del fasor generada en SCILAB

Podemos observar en la figura 3.6, como se va moviendo el vector de 5 a 85 grados creando vectores para cada grado de la siguiente forma:

->//crear vectores entre 5 y 85 grados

->matx=[zeros(1,80); real(fexp)];

->maty=[zeros(1,80); imag(fexp)];

->plot2d4(matx,maty,rect=[0,0,15,9])

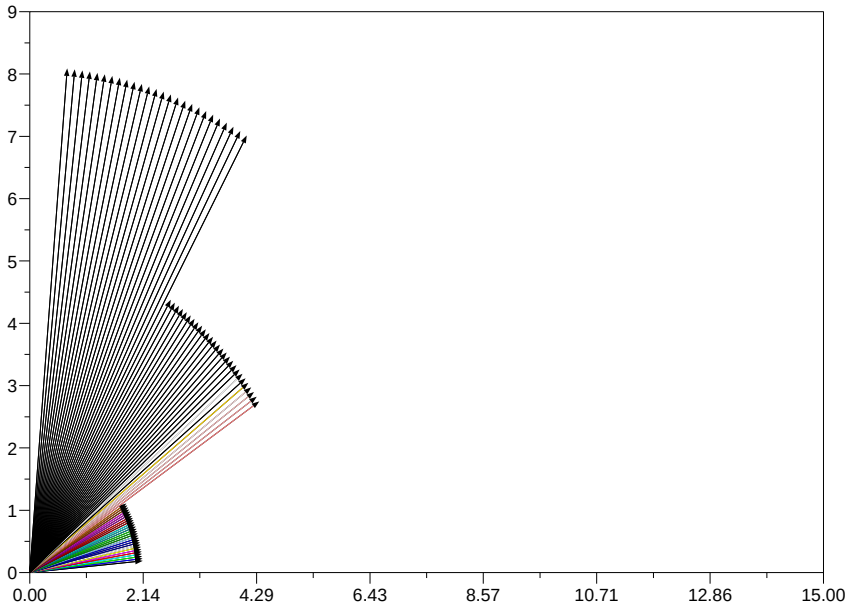


Figura 3.6: Gráfico con vectores

NOTA: las características que se han agregado a los gráficos usando plot2d y plot2d4 se explican en el capitulo GRÁFICOS EN SCILAB.

3.6. ECUACIONES DIFERENCIALES ORDINARIAS (EDO)

SCILAB, utiliza la función "ode()", para solucionar numéricamente ecuaciones diferenciales ordinarias. El uso de ésta es:

y=ode(y0,t0,t,f).

Donde:

y0 : es un vector o una matriz con las condiciones iniciales de la función.

t0 : tiempo inicial.

t : tiempo en el que se va a evaluar.

f : expresión o función a evaluar.

Ejemplo:

Para la edo siguiente encontrar los valores que satisfagan la ecuación entre:

$t_0 = 0;$

$1 < t < 8;$

$y_0 = 1;$

$$\frac{dy}{dt} = y - ycos(t)$$

Solución:

utilizamos "deff()", luego ingresamos los valores iniciales t_0,t,y_0 y por último llamamos la función "ode()", si queremos ver la respuesta en un gráfico utilizamos "plot2d()".

Usando SCILAB:

```
-->deff("[soledo]=f(t,y)","soledo=y-y.*cos(t)");

-->y0=1;t0=0;t=1:0.05:8;

-->y=ode(y0,t0,t,f);

-->plot2d(t,y,2);

-->xtitle('Solución a la ode y=y - y*cos(t)' ,'tiempo','y(t)');
```

Respuesta en la Figura 3.7:

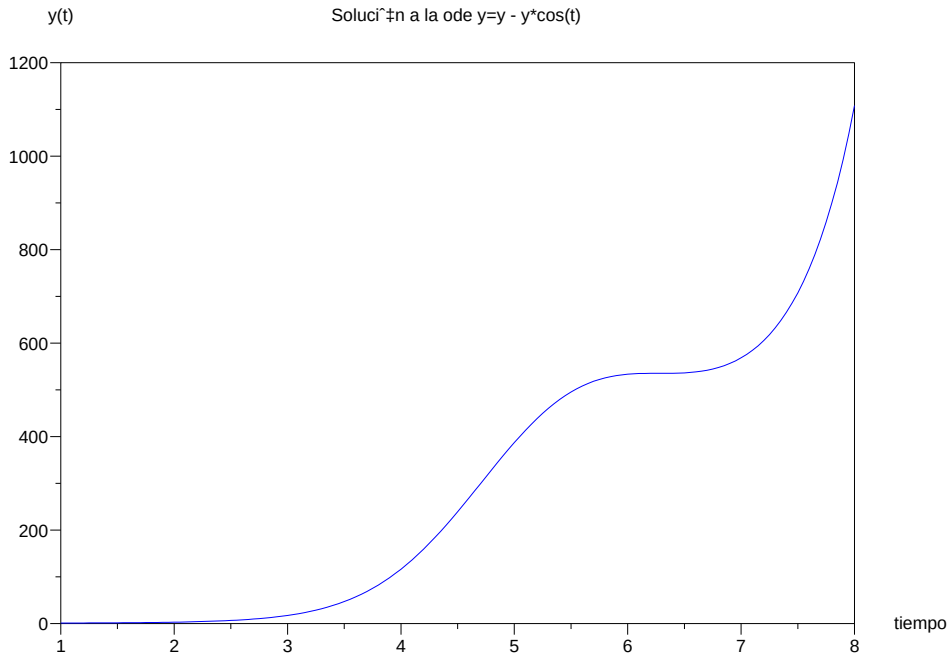


Figura 3.7: Solución Númerica con ode

Capítulo 4

GRÁFICOS EN SCILAB

En el capítulo 2, habíamos hablado de como se construía un gráfico sencillo con el comando plot y plot2d, en el presente capitulo nos dedicaremos a mostrar como podemos generar gráficos de mayor calidad y modificarlos, además de utilizar otros comandos para poder realizar gráficos ajustados a nuestras necesidades.

4.1. GRÁFICOS CON plot

plot es un comando de SCILAB que sirve para realizar gráficos de funciones sencillas representadas en el plano x,y.

El manejo del comando 'plot' es el siguiente:

plot(x,y,'xcap','ycap','caption')

x, y: son vectores de la misma dimensión.

xcap: titulo del eje 'x'.

ycap: titulo del eje'y'.

caption: titulo de el gráfico.

Ejemplo en SCILAB:

```
//graficar 5+sen(x) entre 0 y 4*pi.
-->x=0:0.05:4*%pi;
-->y=5+sin(x);
-->plot(x,y,'eje x','eje y','5sen(x)')
```

El resultado de plot se muestra en la Gráfica 4.1

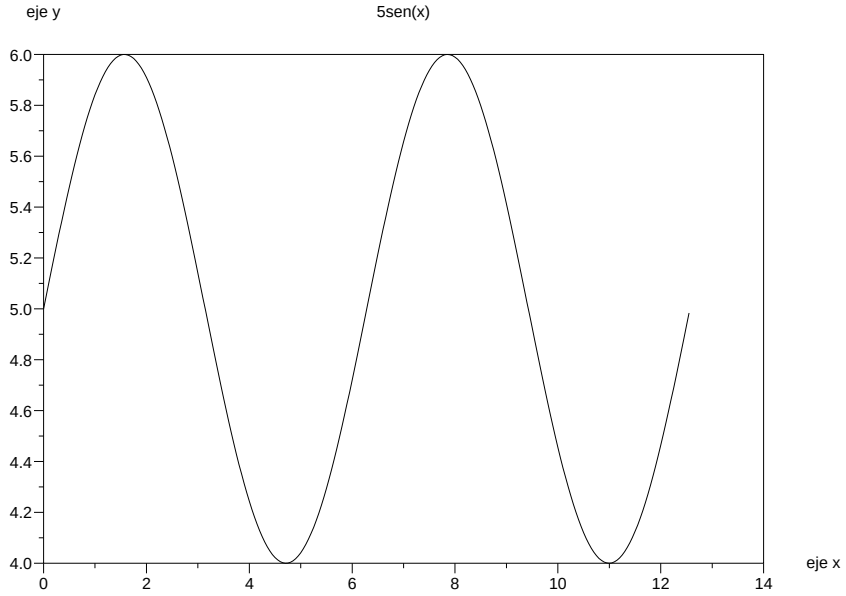


Figura 4.1: Gráfico de y=5+sin(x)

Podemos representar varias funciones en la misma gráfica , por ejemplo:

```
//graficar 5+sen(x) y 4+cos(x)

-->x=0:0.05:4*%pi;

-->y=5+sin(x);

-->w=4+cos(x);

-->plot([w;y])
```

El resultado de plot se muestra en la Gráfica 4.2

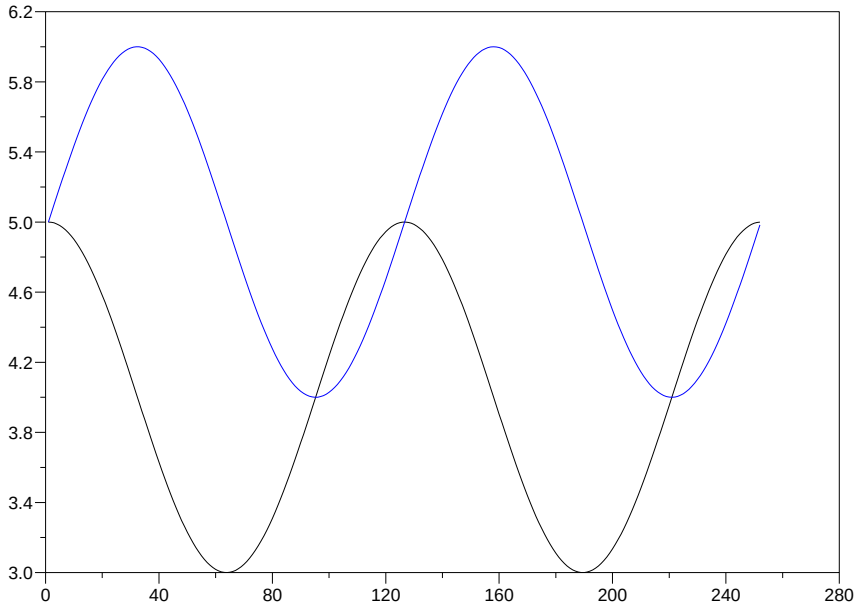


Figura 4.2: Gráfico de 5+sen(x) y 4+cos(x)

Podemos poner consecutivamente una función como 5+sen(x)seguida de 4+cos(x):

```
//grafica de 5+sen(x) seguida de 4+cos(x)

-->x=0:0.05:4*%pi;

-->y=5+sin(x);

-->w=4+cos(x);

-->plot([y,w])
```

El resultado de plot se muestra en la Gráfica 4.3

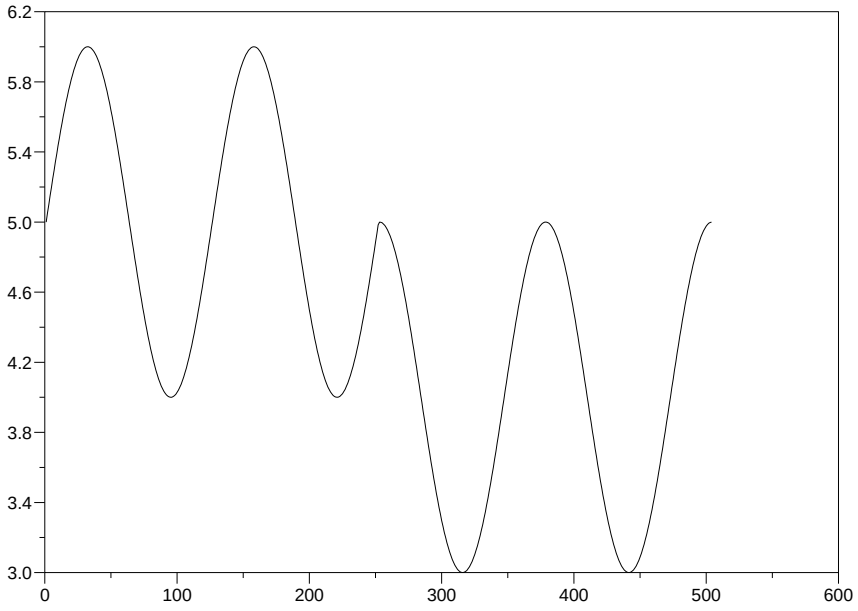


Figura 4.3: Gráfico de 5+sen(x) seguida de 4+cos(x)

4.2. Varias ventanas de gráficos

Cuando generamos el primer gráfico en SCILAB, el titulo predeterminado de la ventana de la gráfica es "scilab graphic (0)" como puede notar en la figura 4.4 y 4.5 .

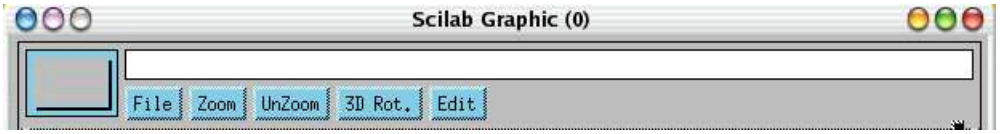


Figura 4.4: Título de ventana de gráfico de SCILAB en LINUX.

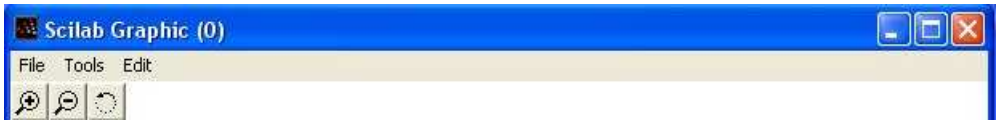


Figura 4.5: Título de ventana de gráfico de SCILAB en MS WINDOWS

Sí queremos crear más ventanas de gráficos en una tarea que demanda la visualización separada de más de una gráfica podemos crear el conjunto de ventanas que necesitemos usando el comando scf, con el siguiente ejemplo podemos entender cómo funciona scf.

```
-->//crear 3 ventanas de graficos diferentes

-->x=0:%pi/60:2*%pi;

-->y0=tan(x); y1=sin(x); y2=cos(x);

-->scf; plot(x,y0); scf; plot(x,y1); scf; plot(x,y2);
```

El resultado de scf y plot se muestra en la figura 4.6, 4.7 y 4.8.

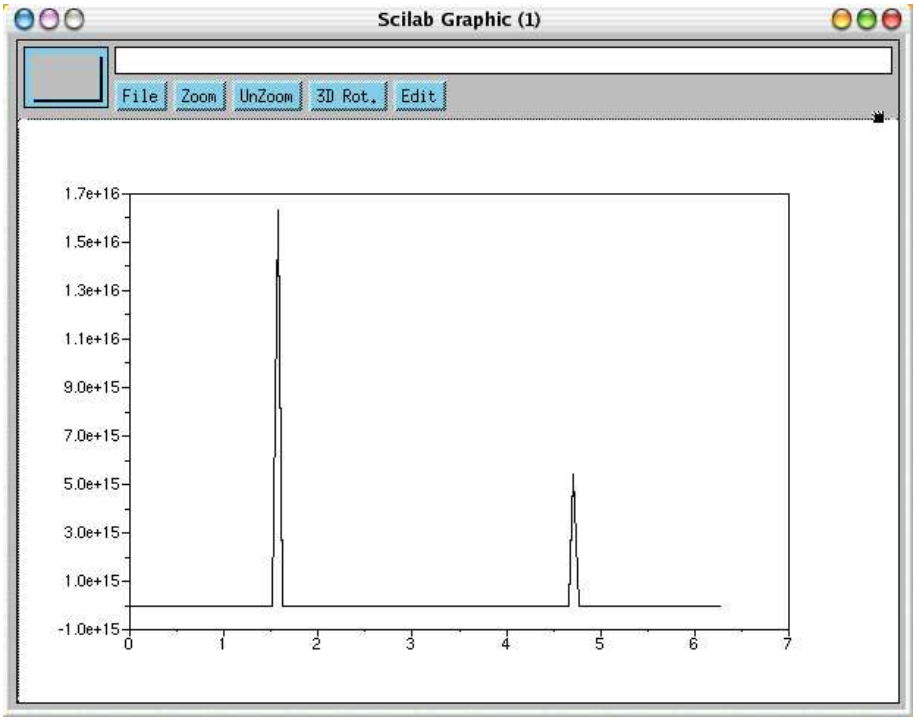


Figura 4.6: ventana N° 1

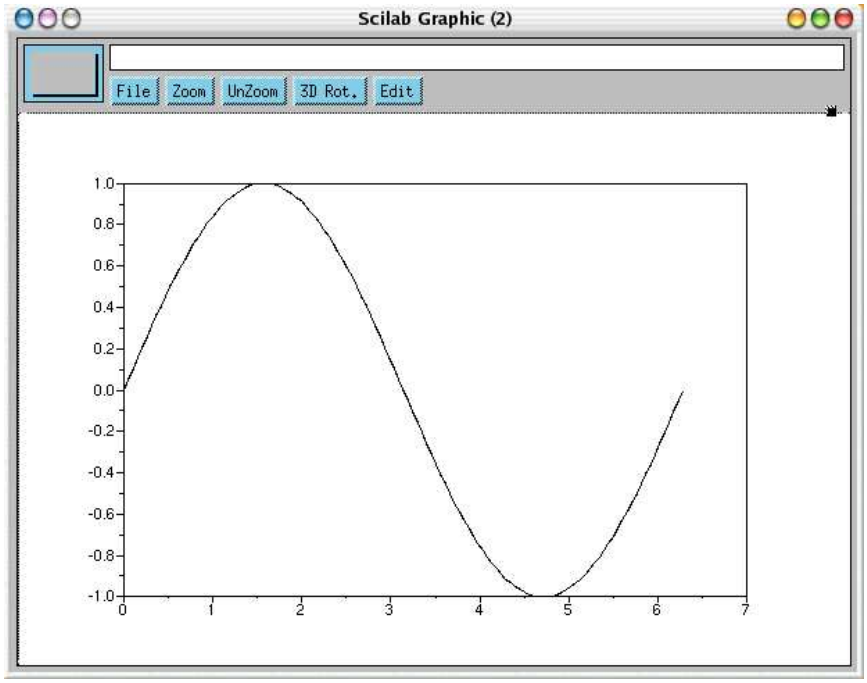


Figura 4.7: ventana Nº 2

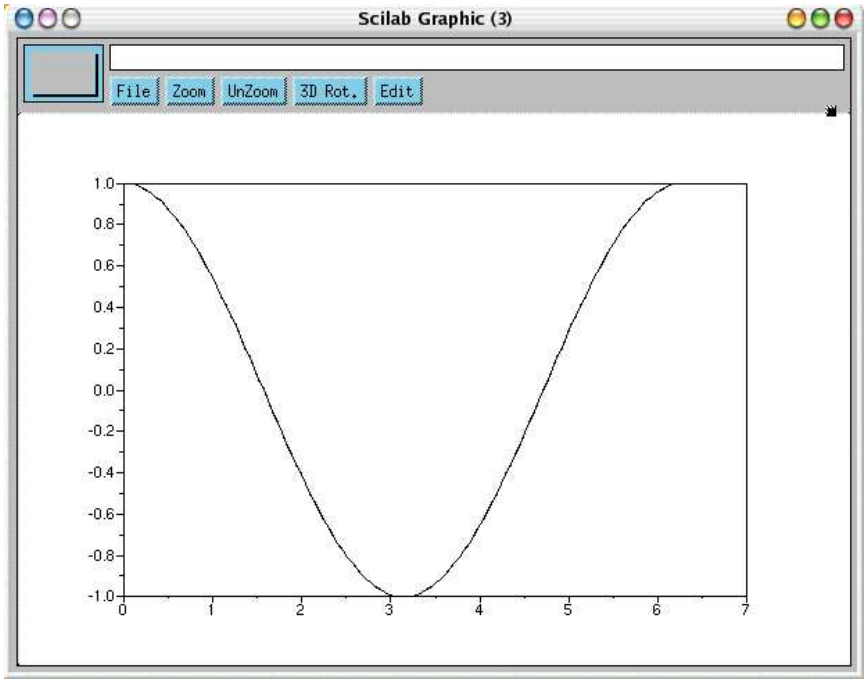


Figura 4.8: Ventana Nº 3

4.3. COMANDOS DE BORRADO

4.3.1. clf y xdel

Después de haber creado un gráfico podemos borrar la ventana "scilab graphic" o borrar el contenido de la ventana, es decir la figura que muestra SCILAB. Para borrar toda la ventana podemos usar xdel() y para borrar el contenido de la ventana usamos clf(). Usando xdel podemos borrar la ventana que deseemos poniendo el número de la ventana en medio del paréntesis de xbasc(), lo mismo para clf().

Podemos eliminar la ventana número 3, creada en el ejemplo anterior así:

```
-->xdel(3)
```

Y borrar el contenido de la ventana 2.

```
-->clf(2)
```

4.4. COMANDO xgrid()

Esta función se utiliza en SCILAB para colocar una malla guia en el gráfico que realizamos, en el paréntesis colocamos un valor numérico que representa el color en el que queremos mostrar la malla, por default () es negro pero podemos utilizar:

Número	Color
0 y1	negro
2	azul
3	verde
4	cían
5	rojo
6	violeta
7	amarillo
8	blanco
9	azul oscuro
19	café
...	...
32	ocre

Cuadro 4.1: Colores para xgrid()

Ejemplo:

```
-->// x^2 -4 , con malla color rojo
```

```
-->x=-4:0.01:4;
-->plot(x,x^2-4,'x','y','titulo');
-->xgrid(5)
```

El resultado de plot y la malla o cuadricula con color rojo se muestra en la Gráfica 4.9.

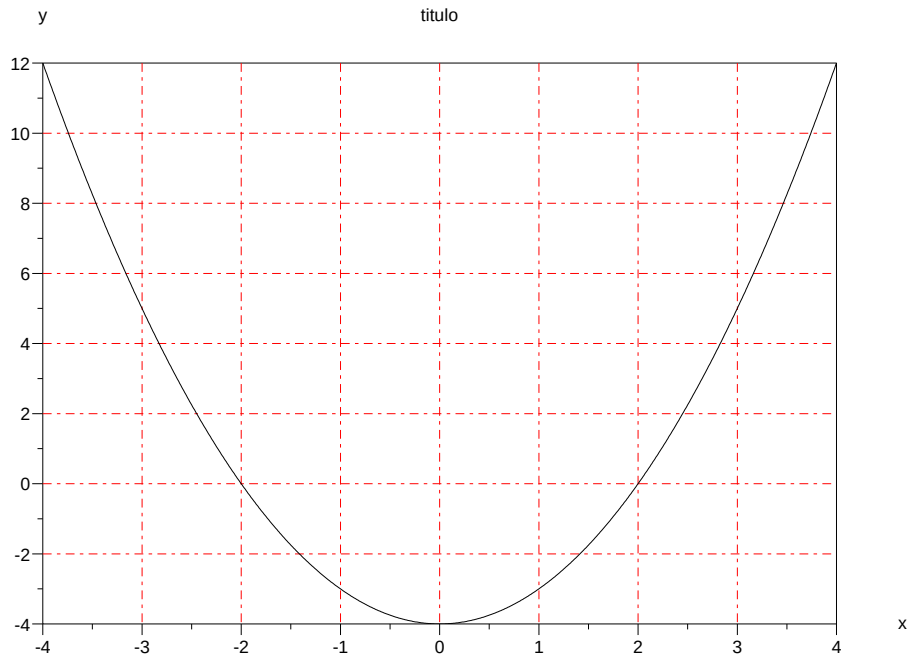


Figura 4.9: gráfico con malla o cuadricula roja

4.5. EL COMANDO plot2d()

El comando plot2d se utiliza en SCILAB para generar gráficos en el plano, la forma de usar este comando es:

plot2d(x,y, 'argumentos')

x : es un vector o una matriz cuyo número de elementos debe ser igual a número de elementos de 'y'.

y : es un vector o una matriz.

'argumentos' están dados por:

style: color o estilo de la curva.

rect: genera una ventana limitando el gráfico a los valores que deseamos. [xmin,ymin,xmax,ymax]

logflag: muestra los ejes en escalas como: natural natural 'nn', natural log 'nl', log natural 'ln', log log 'll'.

axesflag: especifica en que posición se dibujaran los ejes.

leg: escribe la leyenda de la curva realizada

El comando plot2d, debe emplearse con cuidado para que las curvas representadas no sean erróneas, por lo tanto deben seguirse las siguientes indicaciones:

plot2d, permite:

Caso 1. Graficar dos vectores iguales, vector fila con vector fila, vector columna con vector columna.

Caso 2. Graficar una matriz contra un vector fila cuyo número de columnas sea igual al número de filas de la matriz.

Caso 3. Graficar una matriz A contra una matriz B donde i,j de A sea igual a i,j de B, es decir que el numero de filas y columas de A sea igual al numero de filas y columnas de B.

```
-->//vector fila con vector fila

-->t=-3:0.01:3;

-->//(t^2-3)^2-3

-->f=(t^2-3)^2-3;

-->plot2d(t,f)
```

El resultado de plot2d se muestra en la Gráfica 4.10.

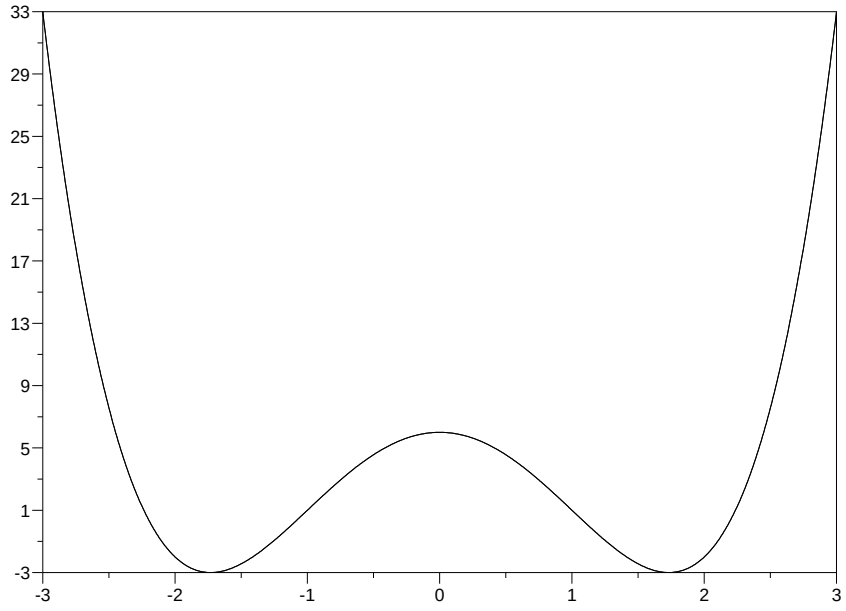


Figura 4.10: Gráfica de un vector contra otro vector de la misma dimensión

```
-->//vector fila con matriz.

-->//número columnas del vector fila = número de filas de la matriz.

-->t=-3:0.01:3;

-->f=(t^2-3)^2-3;

-->g=2*t+2;

-->plot2d(t,[f',g'])
```

El resultado de plot2d se muestra en la Gráfica 4.11.

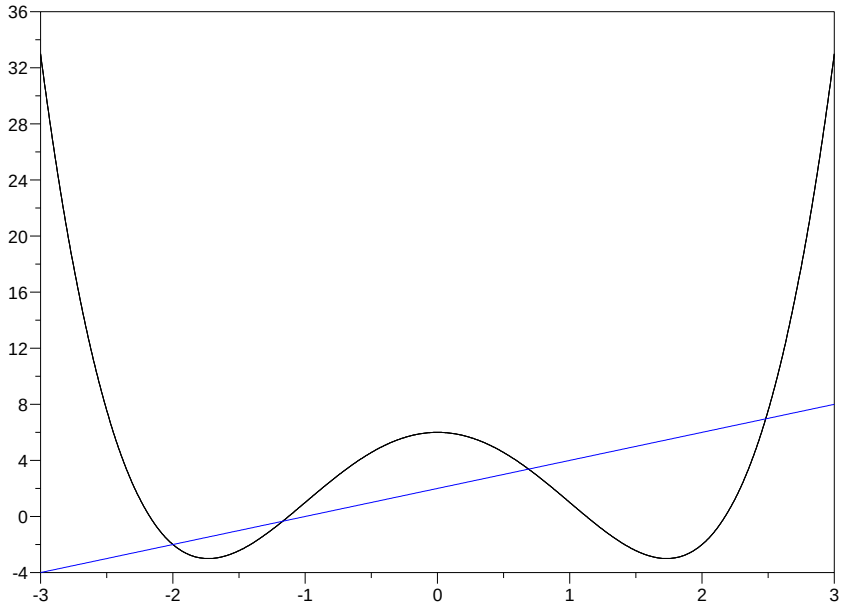


Figura 4.11: Gráfica de vector fila contra matriz

```
-->//vector Columna con matriz
-->//y=-x^2.
-->//x^2-2.
-->//x+1
-->x=-5:0.1:5;
-->y=-x^2;
-->y1=x^2-2;
-->y2=x+1;
-->plot2d(x',[y',y1',y2'])
```

El resultado de plot2d se muestra en la Gráfica 4.12.

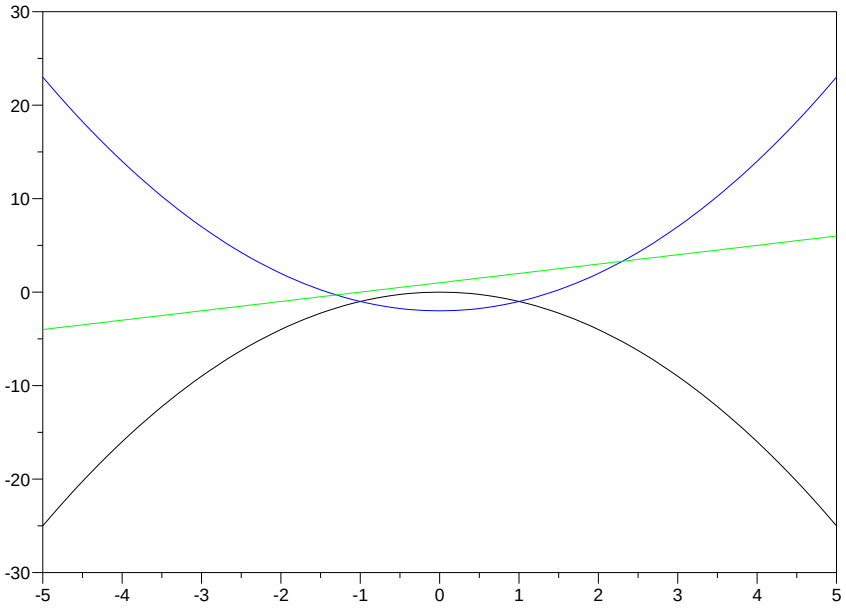


Figura 4.12: Vector columna y matriz

4.5.1. TRABAJANDO CON ARGUMENTOS

Utilizamos el gráfico de la figura 4.7 para aplicar algunos argumentos o caracteristicas a la gráficas.

4.5.1.1. Argumento style

Se utiliza colocando un valor entero positivo para cambiar el color, o uno negativo para el estilo del gráfico. Los siguientes gráficos muestran varias representaciones de style:

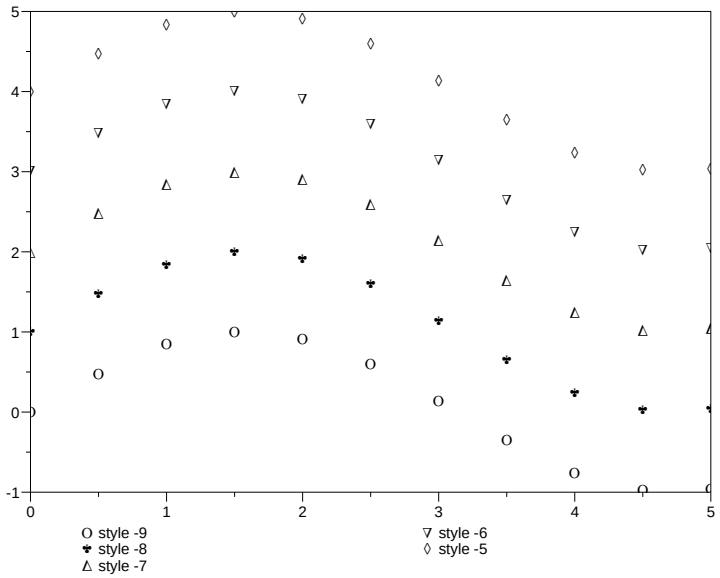


Figura 4.13: style formas

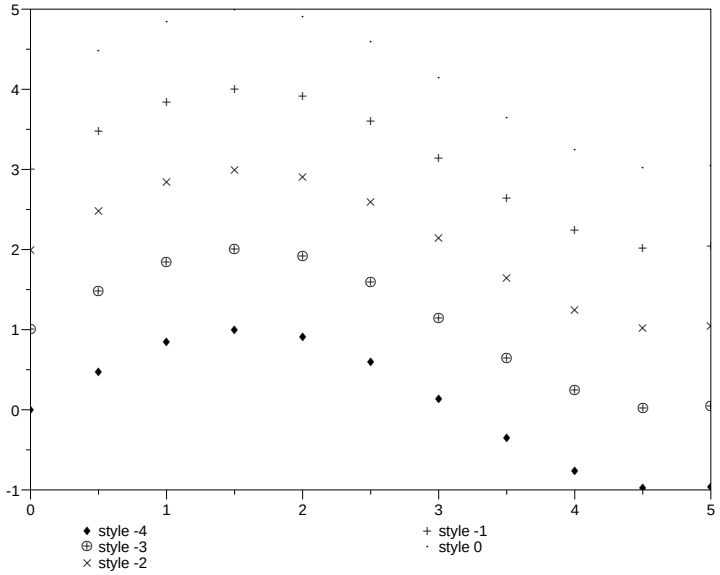


Figura 4.14: style formas

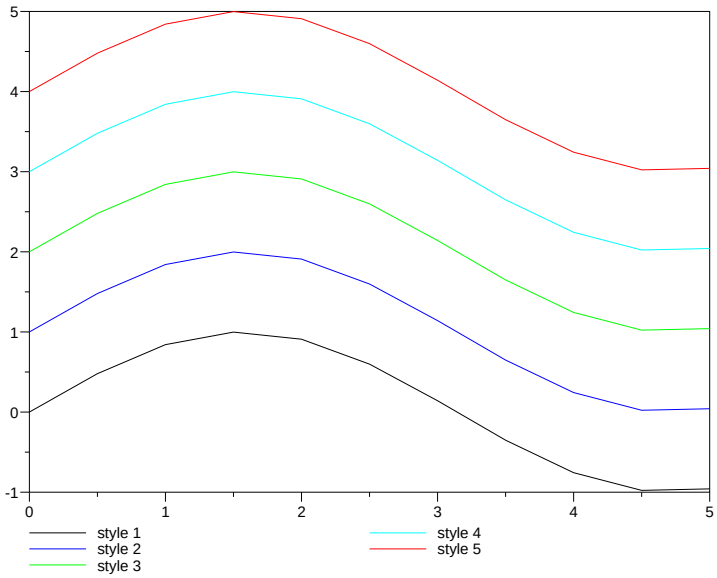


Figura 4.15: style colores

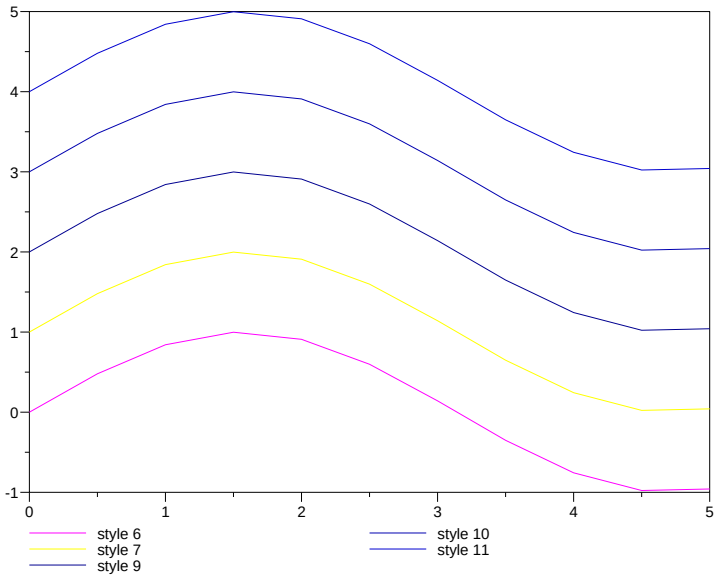


Figura 4.16: style colores

Ejemplo:

```
-->x=-5:0.1:5;

-->y=-x**2;

-->y1=x**2-2;

-->y2=x+1;

-->plot2d(x',[y',y1',y2'],[-2 -4 2])
```

El resultado de plot2d se muestra en la Gráfica 4.17.

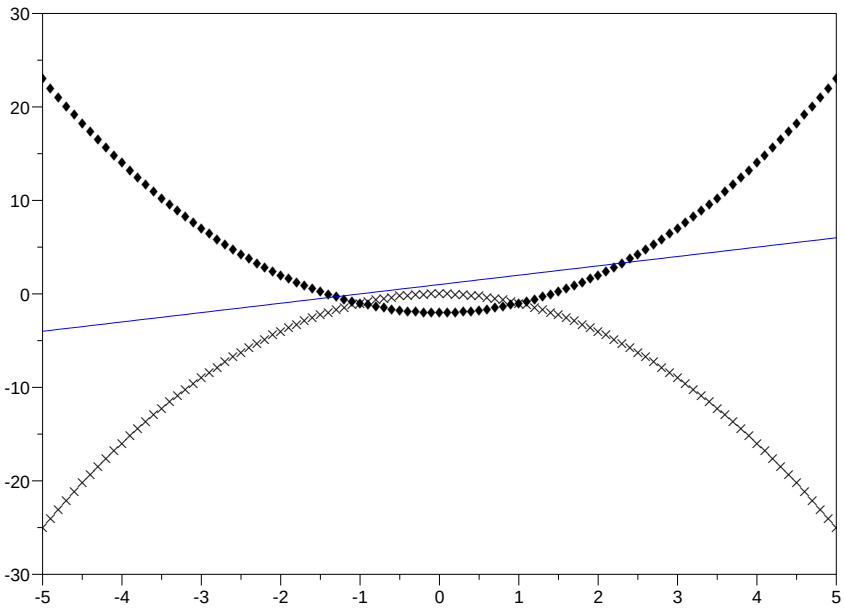


Figura 4.17: usando style

```
-->plot2d(x',[y',y1',y2'],[-9 -4 -8])
```

El resultado de plot2d se muestra en la Gráfica 4.18.

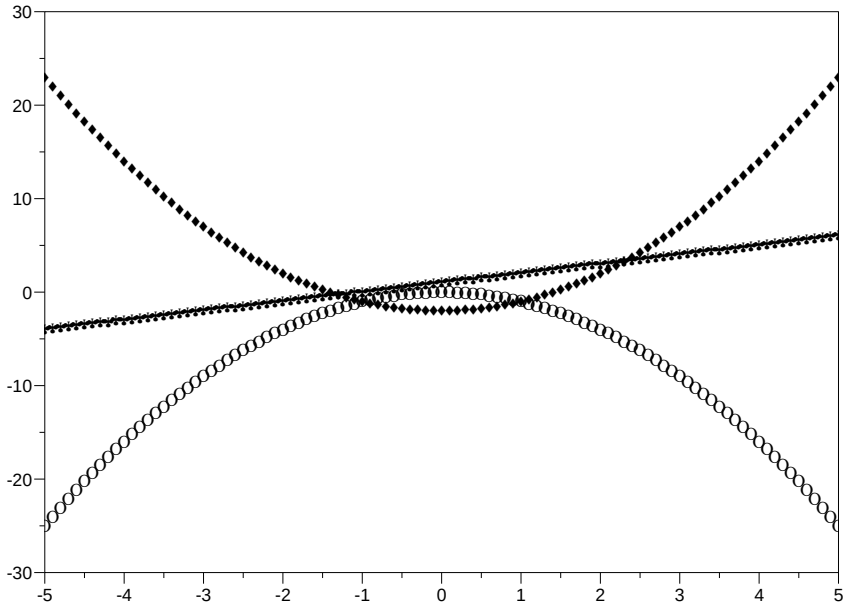


Figura 4.18: cambiando style

4.5.1.2. Argumento rect

rect=[xmin,ymi,xmax,ymax]

Este comando nos permite visualizar los ejes, con las coordenadas descritas por los valores mínimos y máximos.

Ejemplo:

```
-->//del grafico anterior queremos visualizar los ejes x y y entre (0,0)
y (4,5)

-->plot2d(x',[y',y1',y2'],[2 4 9],rect=[0, 0, 4, 5])
```

El resultado de plot2d se muestra en la Gráfica 4.19.

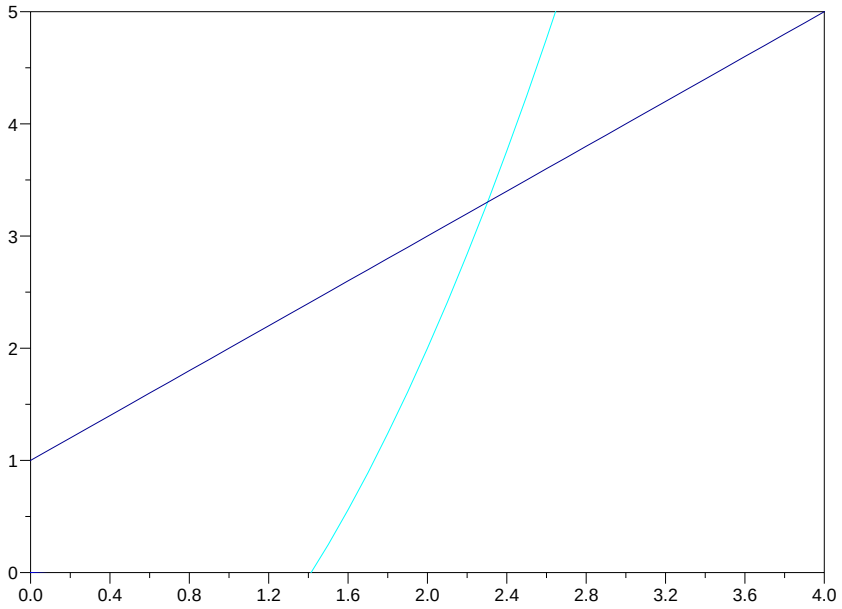


Figura 4.19: usando argumento rect

```
-->//eje x toma valores de -2 hasta 3 y y toma valores desde -2 hasta 3
```

```
-->plot2d(x',[y',y1',y2'],[2 4 9],rect=[-2, -2, 3, 3])
```

El resultado de plot2d se muestra en la Gráfica 4.20.

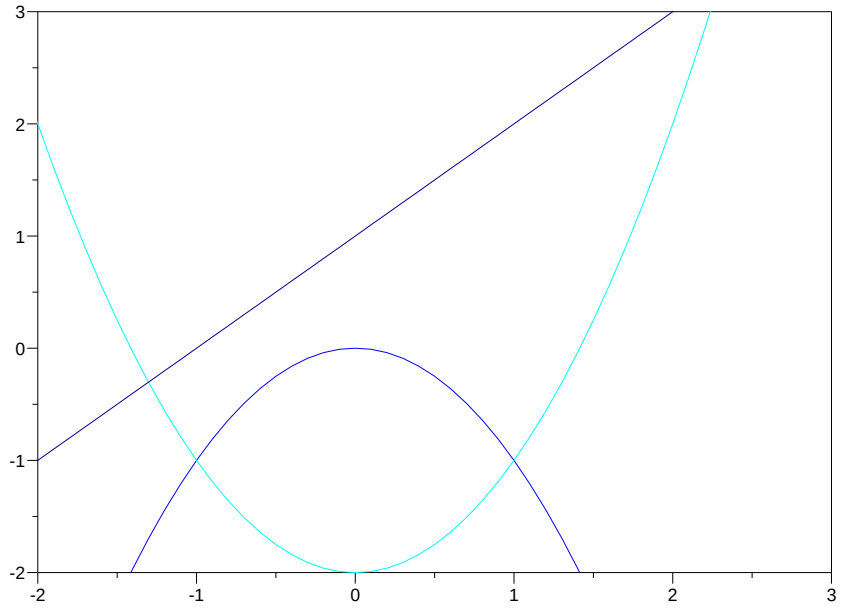


Figura 4.20: eje x entre -2 y 3, eje y entre -2 y 3

4.5.1.3. Argumento leg

Este argumento se utiliza para colocar una leyenda a cada curva según sea su estilo o color, esta leyenda quedará ubicada en la parte inferior de la gráfica.

Si queremos referenciar con leyendas un grafico donde hay varias curvas en el argumento leg separamos las etiquetas con `@`.

```
//uso de leyendas.
-->>clf()
-->x=-5:0.1:5;
-->y=(-x**2);
-->y1=(x**2)-2;
-->y2=x+1;
-->plot2d(x',[y',y1',y2'],[2 4 9],rect=[-2, -2, 3, 3],leg="(-x**2)@((x**2)-2)@(x+1)")
```

El resultado de plot2d se muestra en la Gráfica 4.21.

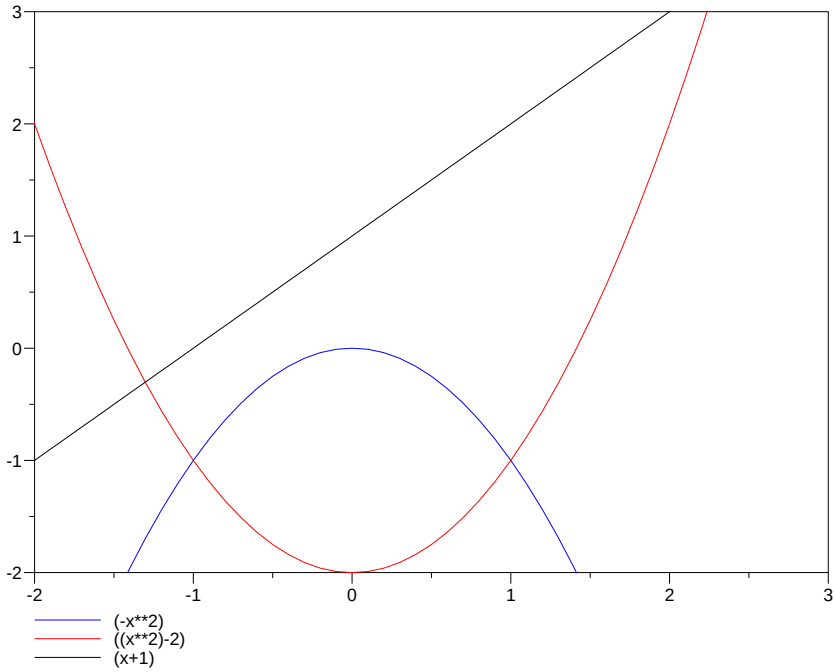


Figura 4.21: uso de leyendas con leg

4.5.2. MEJORANDO LA VISUALIZACIÓN CON "gca()"

gca() es un comando que nos permite mejorar notablemente las caracteristicas de nuestros gráficos. De este comando trataremos algunos parámetros que nos permitirán adecuar ciertas variables de los gráficos construidos previamente.

Cómo usar gca() :

Después de haber creado una grafica con una función o modelo determinado es necesario crear una variable e igualarla a gca() así: variable=gca().

Ejemplo:

```
-->plot2d(x',[y',y1',y2'],[2 5 1],rect=[-2,-2,3,3],leg='(-x**2)@((x**2)-2)@(x+1)')
-->a=gca();
```

Utilizar cualquiera de las siguientes funciones antecedidas de la variable definida y seguidas de un punto, así: a.(variable a cambiar)

4.5.3. VARIABLES ÚTILES USANDO gca():

4.5.3.1. grid=[y,x]

muestra una malla en el gráfico y 'x' e 'y' son el color de las respectivas proyecciones de los ejes.
Ejemplo:

```
-->//malla con color azul las proyecciones horizontales (2) y rojo las verticales(5)
-->plot2d(x',[y',y1',y2'],[2 5 1],rect=[-2,-2,3,3],leg='(-x**2)@((x**2)-2)@(x+1)')
-->a=gca();
-->a.grid=[5,2];
```

El resultado se muestra en la Gráfica 4.22.

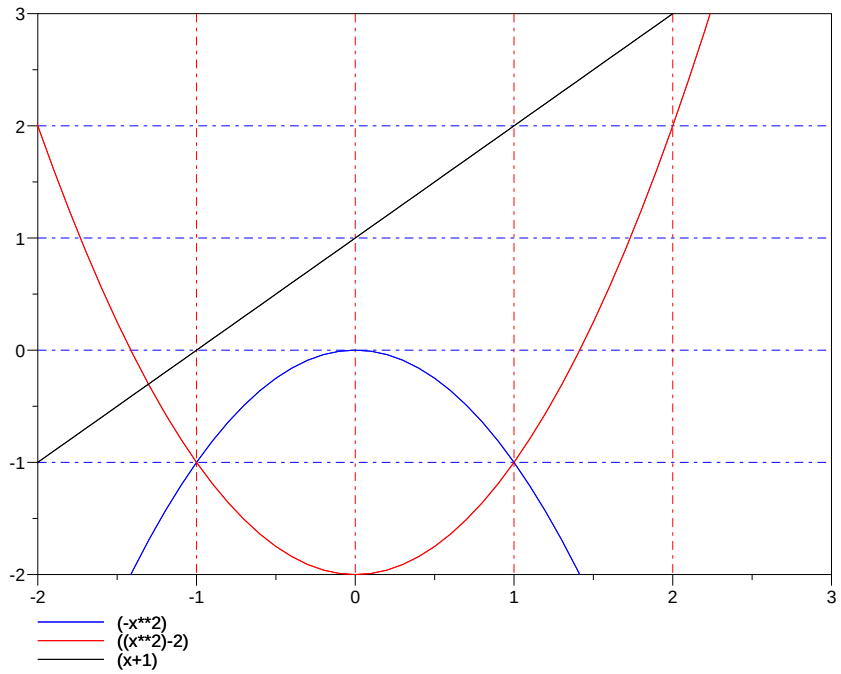


Figura 4.22: a.grid=[5,2] usando a=gca()

4.5.3.2. log_flags

Por las características de algunos modelos matemáticos a veces es necesario que las gráficas resultantes sean mostradas en una escala diferente a la convencional, como por ejemplo semilogarítmica, SCILAB tiene la capacidad de mostrar gráficos usando

la escala que se necesite y para conseguir hacerlo se usa la variable log_flags así:

```
-->x=0:0.1:4;
-->y=x^2+x^3;
-->plot2d(x,y,5,leg='x**2+x**3',rect=[0.8,0,10,80])
-->a=gca();
-->a.grid=[1,2];
-->a.log_flags = "ln";
```

El resultado se muestra en la Gráfica 4.23.

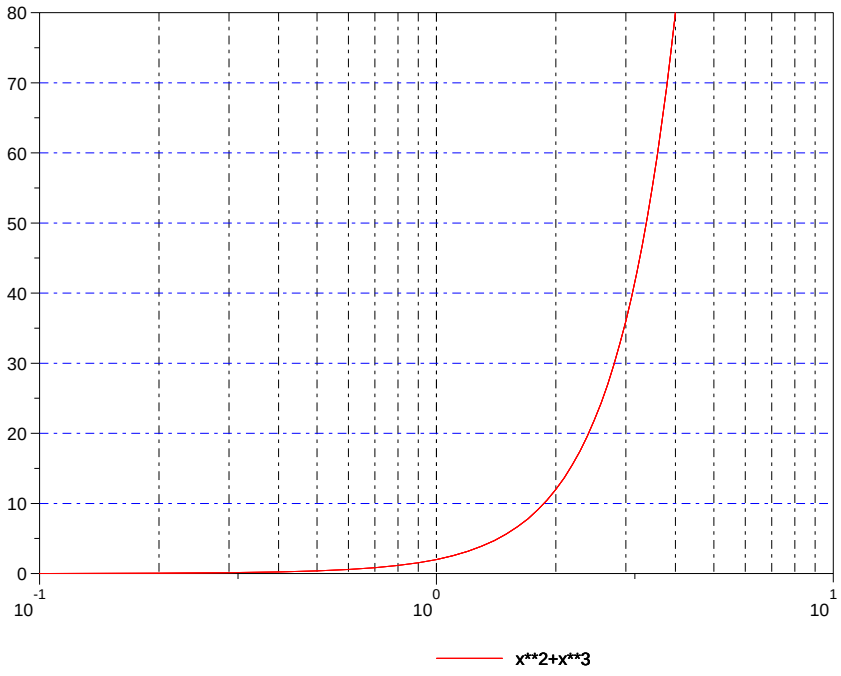


Figura 4.23: usando a.log_flags="ln", con a=gca();

4.5.3.3. x_location, y_location

Podemos ubicar de varias formas, y la forma en que se haga esta dada por:

Centrados="middle"
Izquierda="left"
Derecha ="rigth"

Superior= "top"
inferior ="bottom"

Ejemplo:

```
x=-5:0.1:5;
y=-x**2;
y1=x**2-2;
y2=x+1;
plot2d(x',[y',y1',y2'],[2 5 1],rect=[-3,-3,3,3 ],leg='(-x**2)@((x**2)-2)@(x+1)'
)
a=gca();
a.x_location="middle";
a.y_location="middle";
```

El resultado se muestra en la Gráfica 4.24.

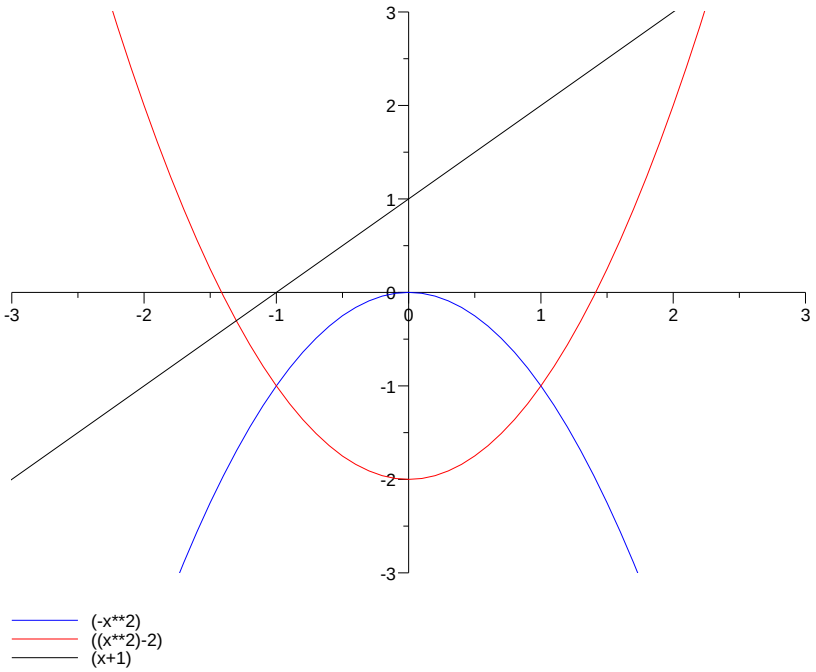


Figura 4.24: usando x_location y y_location

Sí usted desea conocer mas características del comando gca puede escribir en SCILAB:

```
-->help gca
```

4.5.4. ETIQUETAS

4.5.4.1. xtitle para ejes y gráfica

Para presentar las gráficas con información para que sean mas comprensibles, es necesario usar títulos y etiquetas, asi como anteriormente usamos leg para referenciar las curvas, podemos usar xtitle para darle nombre a la gráfica y a los ejes así: xtitle("titulo","eje x","eje y").

Ejemplo:

```
x=-5:0.1:5;

y=-x**2;

y1=x**2-2;

y2=x+1;

plot2d(x',[y',y1',y2'],[2 5 1],rect=[-3,-3,3,3 ],leg='(-x**2)@((x**2)-2)@(x+1)'
)

a=gca();

a.x_location="middle";

a.y_location="middle";

xtitle("ELABORACION DE GRAFICAS"," eje x","eje y")

a.title.font_style=5;

a.title.font_size=4;
```

El resultado se muestra en la Gráfica 4.25.

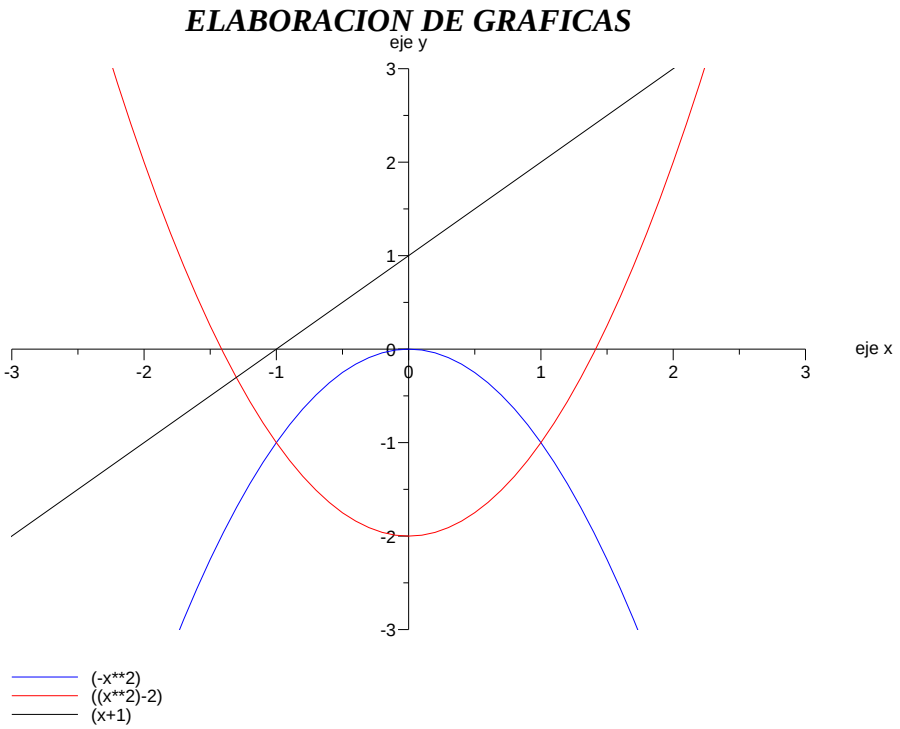


Figura 4.25: xtitle usando a.title.font_style y a.title.font_size de gca()

Si queremos ver nuestra gráfica con guias:

A la gráfica anterior adicionamos:

```
a.grid=[6,6];
```

El resultado se muestra en la Gráfica 4.26.

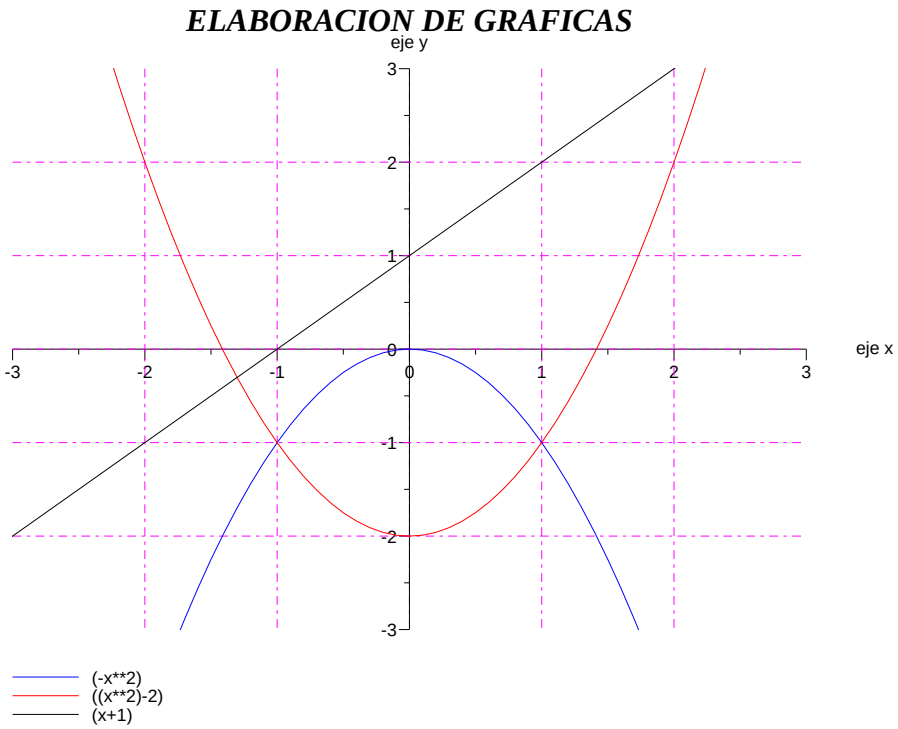


Figura 4.26: usando guias

4.5.4.2. Notas y etiquetas para las curvas

Sí queremos escribir una nota o etiqueta adicional a nuestro gráfico, utilizamos el comando xstring() de la siguiente manera:

```
xstring(x,y,"texto",angulo,box)
```

Donde x,y es es la coordenada donde queremos ubicar la etiqueta, texto la etiqueta a adicionar, ángulo el ángulo al que queremos nuestra etiqueta y box el valor numérico entre 0 y 1 si queremos o no un recuadro.

Ejemplo:

```
x=-5:0.1:5;
y=-x**2;
y1=x**2-2;
y2=x+1;
plot2d(x',[y',y1',y2'],[2 5 1],rect=[-3,-3,3,3 ],leg='(-x**2)@((x**2)-2)@(x+1)'
)
```

```
a=gca();
a.x_location="middle";
a.y_location="middle";
xtitle("ELABORACION DE GRAFICAS"," eje x","eje y")
a.title.font_style=5;
a.title.font_size=4;
a.grid=[6,6];
xstring(0.8,0.8,[" Nota"; "Adicional"],0,1);
```

El resultado se muestra en la Gráfica 4.27.

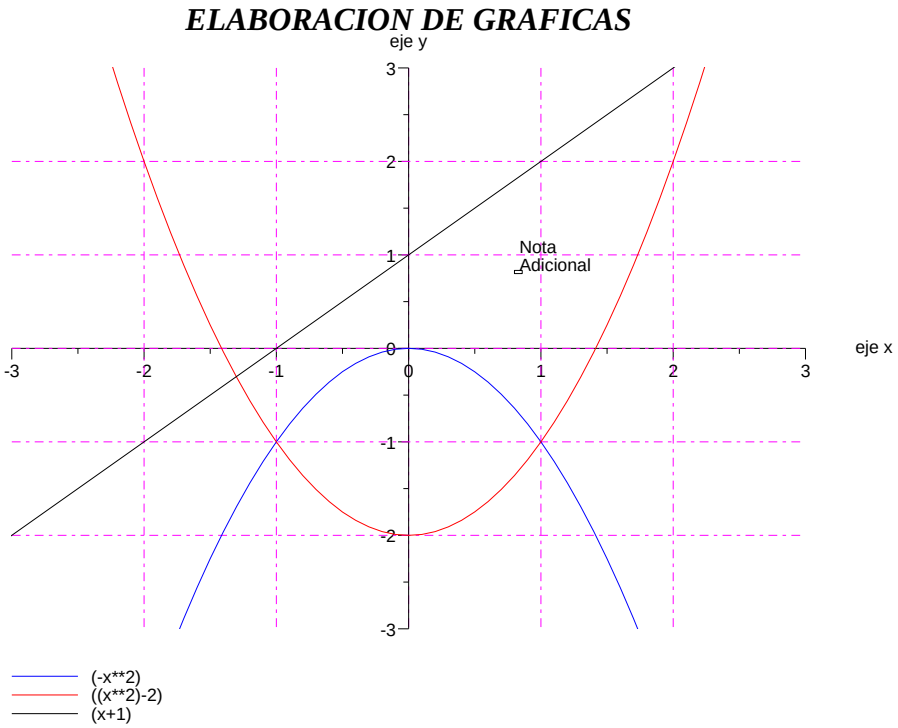


Figura 4.27: usando xstring()

4.6. PLOT2D2

la función plot2d2 se utiliza para elaboración de gráficos de forma escalón. Los escalones están dados por la cantidad de puntos muestreados

La sintaxis y argumentos son iguales a los de la función plot2d().

Ejemplo:

```
-->//funcion x^2 de forma escalon.
-->x=-4:0.5:4;
-->plot2d2(x,x**2,5,leg="step -->x**2");
-->a=gca();
-->a.y_location = "middle";
-->a.box="off";
-->xtitle("PLOT2D2 X**2","x","y");
-->a.title.font_style=5;
-->a.title.font_size=4;
```

El resultado se muestra en la Gráfica 4.28.

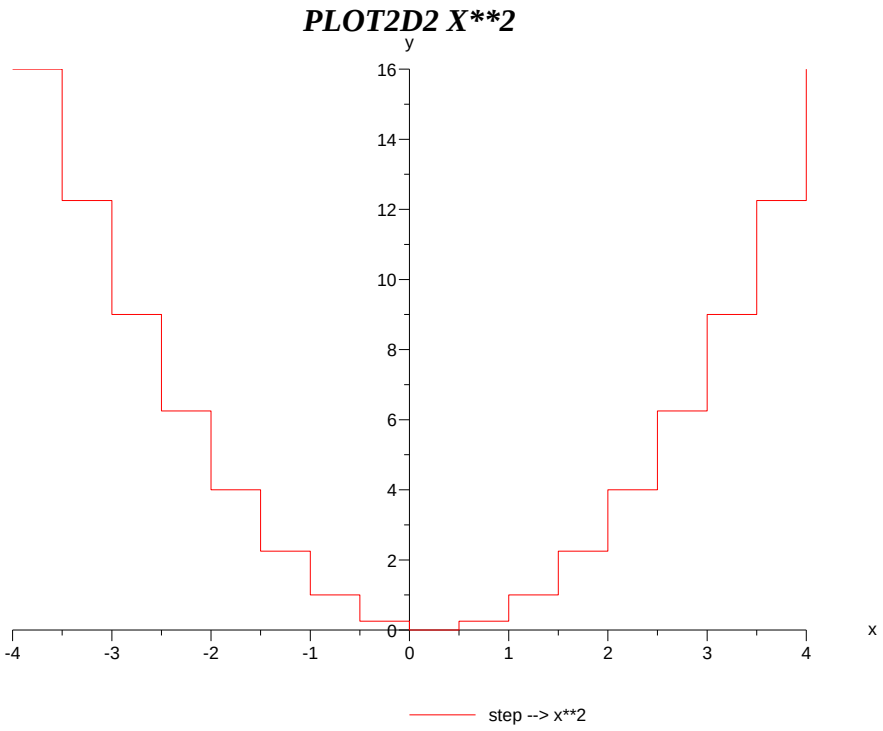


Figura 4.28: Usando plot2d2

4.7. PLOT2D3

Este comando en SCILAB permite generar gráficos en los que los valores se muestran en barras verticales.

La sintaxis y argumentos son iguales a plot2d()

Ejemplo:

```
-->//grafica de x^3 +1
-->x=-3:0.05:3;
-->plot2d3(x,x**3+1,6,leg="barras verticales (x**3+1)");
-->a=gca();
-->a.y_location="middle";
-->a.x_location="middle";
-->xtitle("plot2d3","x","y");
-->a.title.font_style=5;
-->a.title.font_size=3;
```

El resultado se muestra en la Gráfica 4.29.

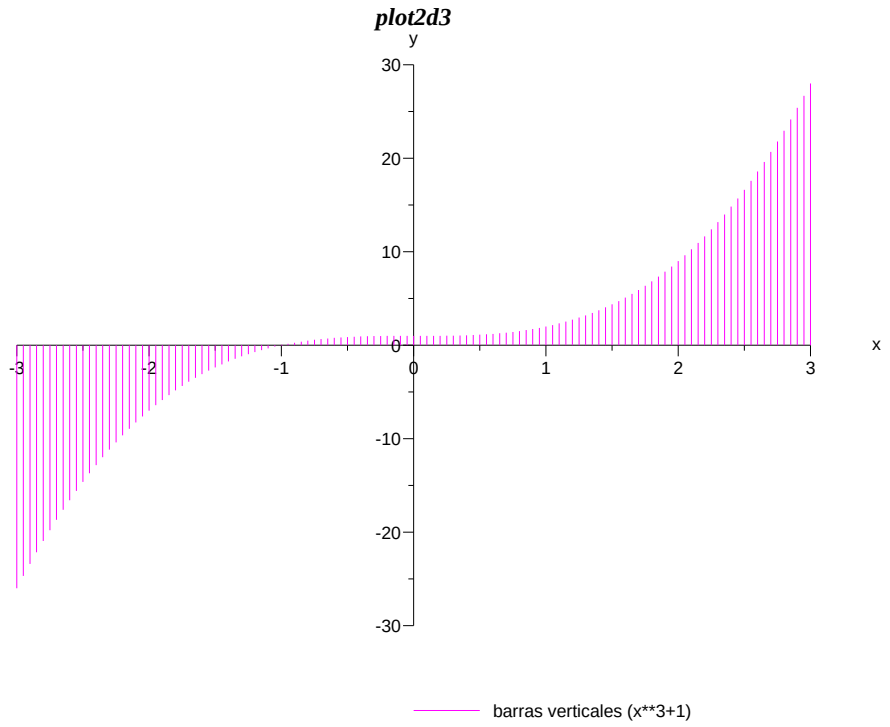


Figura 4.29: Usando plot2d3

4.8. PLOT2D4

Este comando de SCILAB genera gráficos cuyos puntos de la curva están unidos por flechas.

La sintaxis y argumentos son iguales a plot2d()

Ejemplo:

```
-->//(x^3/2)-2x
-->x=-3:0.5:3;
-->plot2d4(x,-2*x+0.5*x**3,leg="flechas (-2x+0.5x**3)");
-->a=gca();
-->a.y_location="middle";
-->a.title.font_style=5;
-->a.title.font_size=3;
-->a.x_location="middle";
-->xtitle("plot2d4","x","y");
```

El resultado se muestra en la Gráfica 4.30.

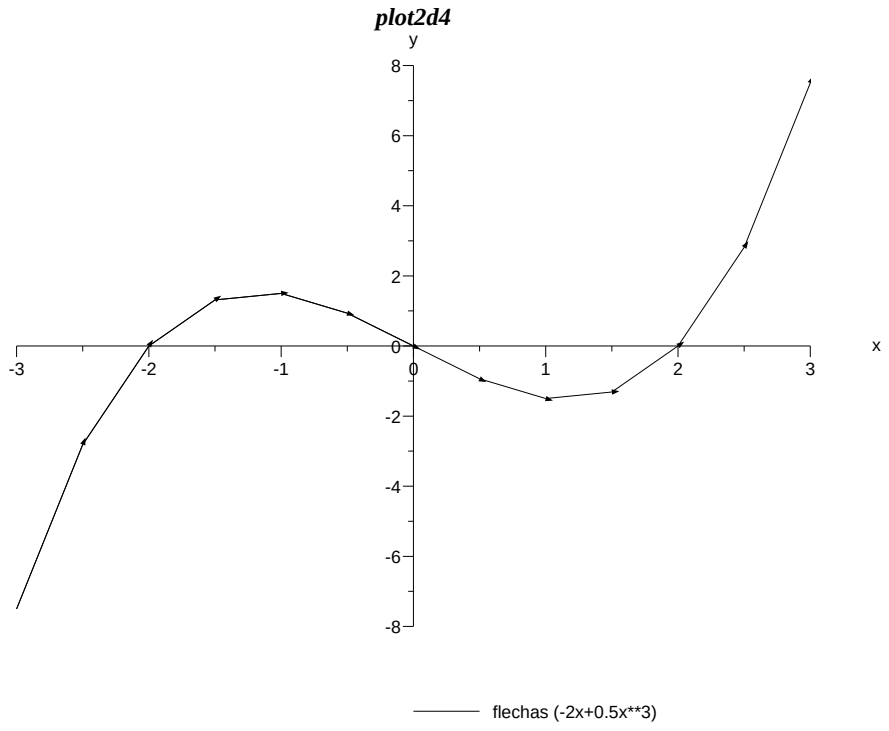


Figura 4.30: Usando plot2d4

4.9. plot2d4 y vectores

4.9.1. Vector

si queremos graficar vectores en el plano, utilizamos esta función de tal forma que la curva mostrada sea un vector. Por ejemplo, si queremos graficar el vector de la figura 4.31:

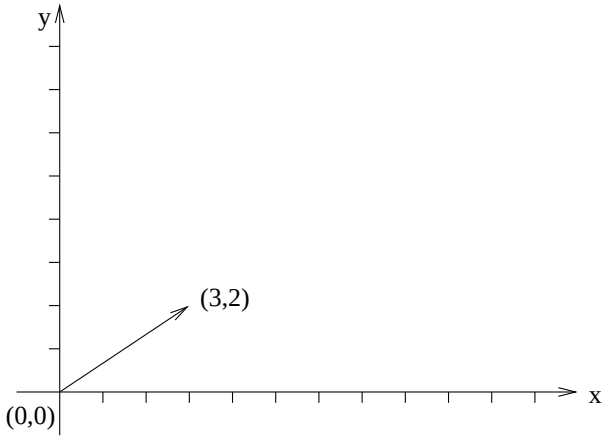


Figura 4.31: vector

Note que para el vector de la figura $x_0 = y_0 = 0$, $x_1 = 3$ y $y_1 = 2$, podemos crear dos vectores para representar esto, así:

$$x = \begin{matrix} x_0 & x_1 \\ \downarrow & \downarrow \\ [0 & 3] \end{matrix} \qquad y = \begin{matrix} y_0 & y_1 \\ \downarrow & \downarrow \\ [0 & 2] \end{matrix}$$

En SCILAB:

```
-->//vector desde el origen (0.0) hasta (3,2).
-->x=[0,3];
-->y=[0,2];
-->plot2d4(x,y,2,rect=[-1,-1,3.5,3.5],leg="vector(0,0),(3,2)");
-->xtitle("grafica de un vector","x","y");
-->xstring(3,2,"(3,2)");
-->a=gca();
-->a.x_location="middle";
-->a.y_location="middle";
-->a.title.font_style=5;
```

```
-->a.title.font_size=4;
```

El resultado se muestra en la Gráfica 4.32.

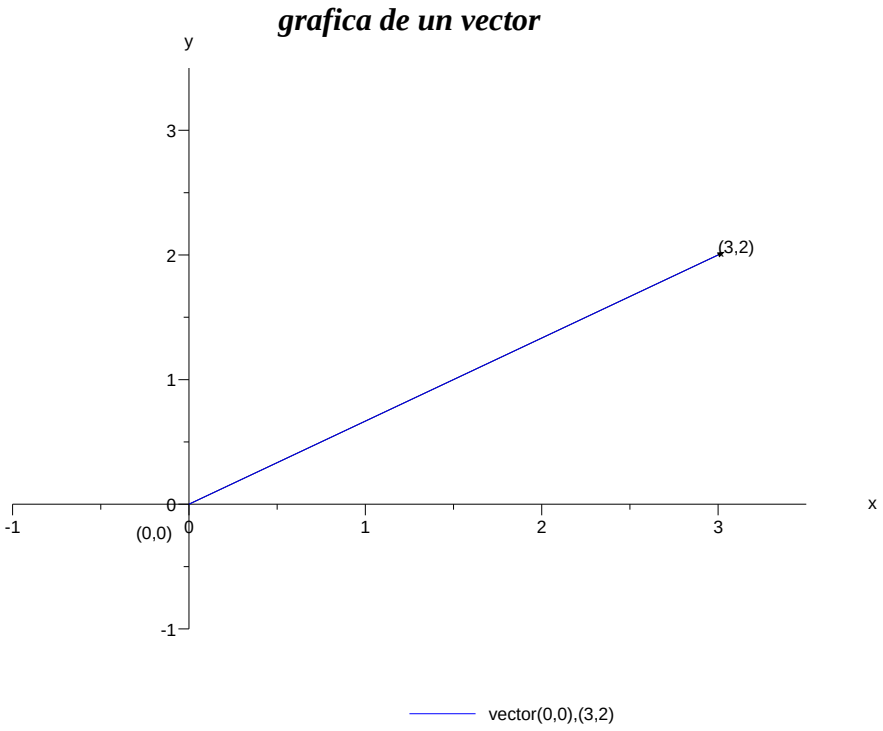


Figura 4.32: vector en SCILAB usando plot2d4

4.9.2. VARIOS VECTORES

para graficar varios vectores, generamos dos matrices de 2 filas por n columnas, en donde la matriz X, tendrá los valores iniciales y finales de los pares "x" y la matriz Y tendrá los valores iniciales y finales de los pares "y" y n es igual al numero de vectores que queremos graficar. Por ejemplo los vectores de la figura 4.33:

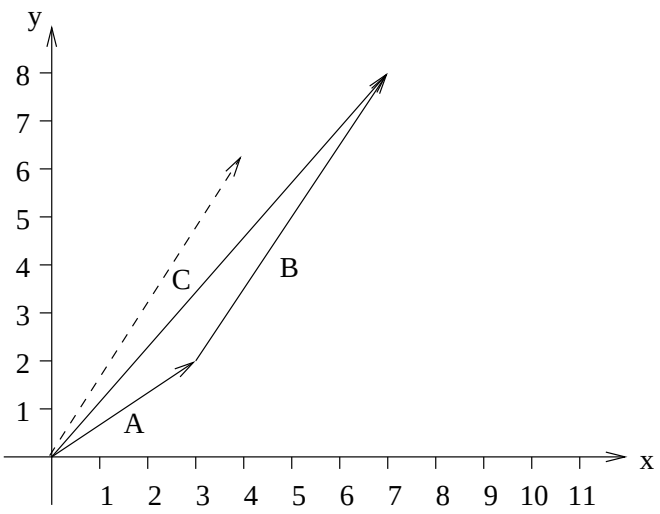


Figura 4.33: varios vectores

C es la suma de A y B, y para realizar la suma desplazamos B hacia el origen y
Usando SCILAB:

```
-->A1=[0;3];
-->B1=[3;7];
-->B1desp=[0;4];
-->C1=[A1+Bdesp];
-->A2=[0;2];
-->B2=[2;8];
-->B2desp=[0;6];
-->C2=[A2+B2desp];
-->x=[A1,B1,C1]; y=[A2,B2,C2];
-->plot2d4(x,y,[2,2,2],rect=[-0.5,-0.5,8,9])
-->xtitle("vectores A,B y C","x","y");
-->xstring(2,0.8,"A")
-->xstring(5,4.5,"B")
-->xstring(3.4,4.5,"C")
-->a=gca();
-->a.x_location="middle";
-->a.y_location="middle";
```

El resultado se muestra en la Gráfica 4.34.

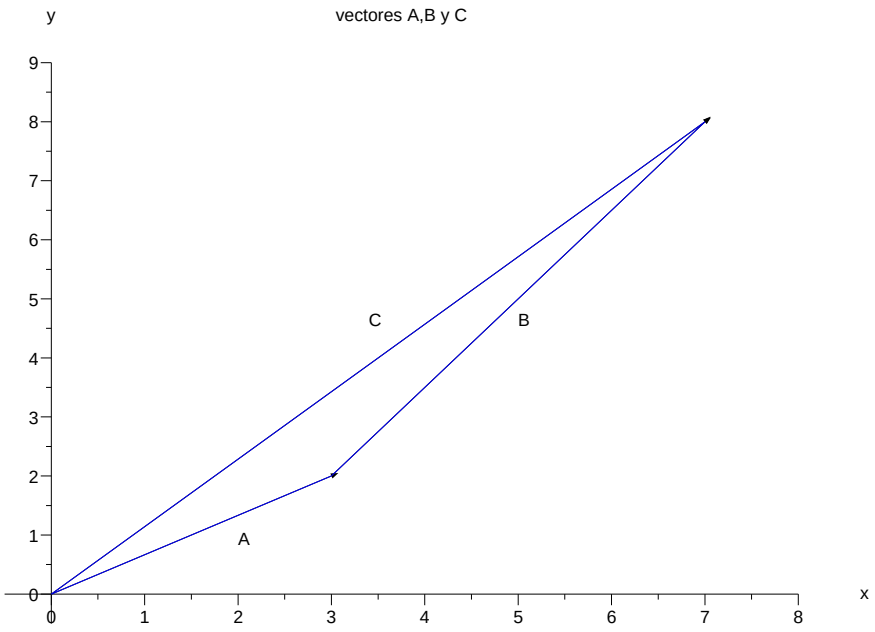


Figura 4.34: vectores en SCILAB

4.10. GRÁFICOS EN 3D CON PLOT3D.

SCILAB puede generar gráficos en 3D utilizando la función plot3d(x,y,z,argumentos). Estos gráficos son realmente una superficie representada por los valores de (x,y,z) para realizar un gráfico con esta función hay que tener en cuenta lo siguiente:
"x" y "y" son vectores de xm y yn dimensión,
z una matriz de dimensión (xm,yn),
z(i,j) son los valores de la superficie en el punto (x(i),y(i)).
Ejemplo en SCILAB:

```
-->//graficar el seno de una funcion en 3D.
-->x=0:0.1:2*%pi;
-->y=sin(x);
-->z=y' *ones(1,63);
-->plot3d(x,x/2,z);
```

El resultado se muestra en la Figura 4.35.

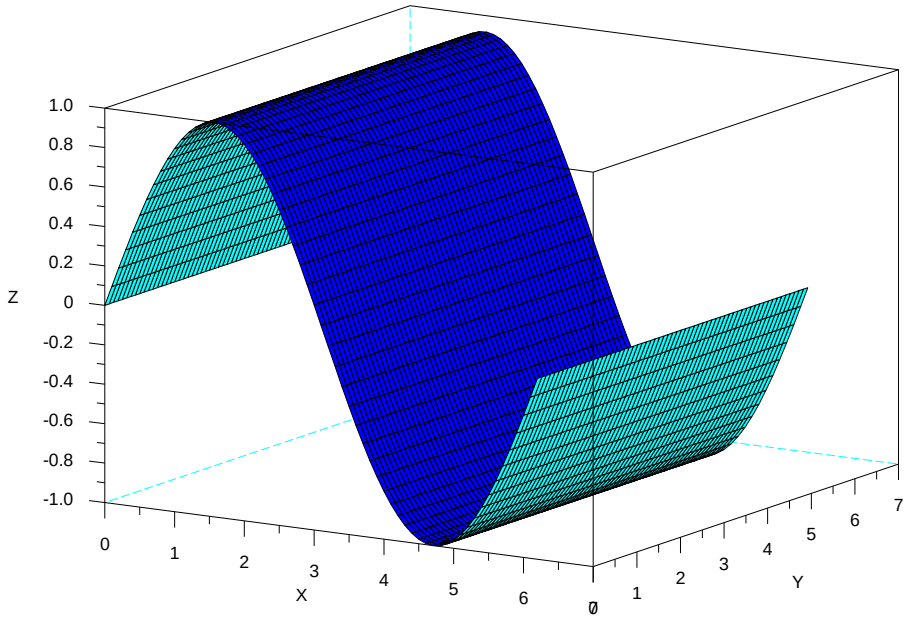


Figura 4.35: seno de una función en 3D

En el anterior gráfico tenemos un vector fila "x" de 63 componentes, la función "y" igual a "seno(x)", con estos datos formamos nuestra matriz cuadrada "z".

Otro ejemplo con vectores de diferente dimensión:

En SCILAB:

```
-->y=-5:0.1:5;

-->b=-3:0.1:3;

-->z=y'*(-b**2+2);

-->plot3d(y,b,z);
```

El resultado se muestra en la Figura 4.36.

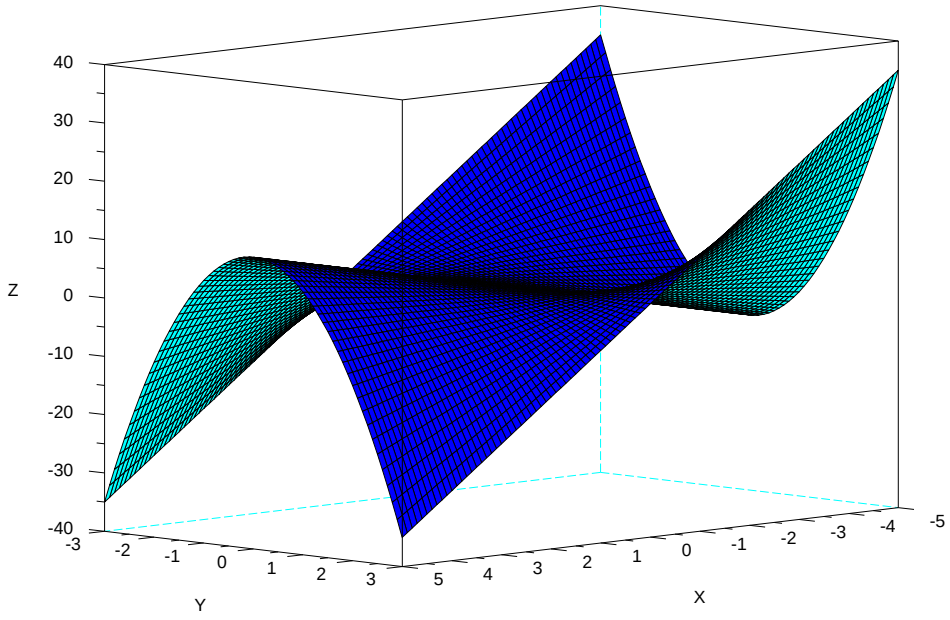


Figura 4.36: Ejemplo con vectores de diferente dimensión

4.10.1. El ARGUMENTO leg=""

leg= leyenda del gráfico, al igual que con plot2d(), podemos escribir una cadena de caracteres para cada eje, separados por el signo @.

```
-->y=-5:0.1:5;

-->b=-3:0.1:3;

-->z=y'*(-b**2+2);

-->plot3d1(y,b,z,leg="eje x@eje y@eje z");
```

El resultado se muestra en la Figura 4.37.

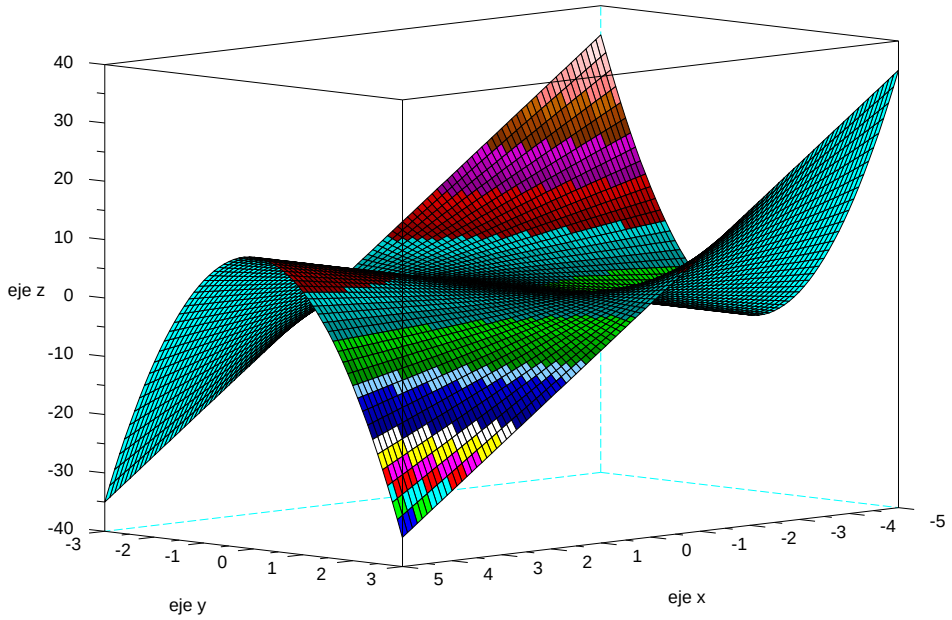


Figura 4.37: Gráfico en 3D usando leg=””

De esta manera terminamos el capitulo de gráficos en SCILAB, aunque tratamos de cubrir el tema de forma amplia, queremos recomendar el uso de los comandos de ayuda help plot y apropos graphics, para observar las características de los comandos de gráficos que no se han nombrado en este libro.

Capítulo 5

PROGRAMACIÓN EN SCILAB

5.1. INTRODUCCIÓN

Otra características de SCILAB, es la capacidad para construir funciones. Esto permite el desarrollo de programas o pequeños guiones (scripts) especializados que pueden ser incluidos dentro del entorno de SCILAB y las rutinas comúnmente usadas junto con el uso de algunas librerias. En este capitulo haremos una introducción a la forma como se construyen algunos programas o scripts para ser usados en SCILAB.

5.1.1. EL EDITOR DE SCILAB

El editor o scipad es un editor de texto incluido dentro del entorno de SCILAB escrito en TCL/TK. Puede abrirse, presionando el botón Editor en la parte superior del entorno de trabajo de SCILAB, o también puede iniciarse desde la linea de comandos de scilab, ejecutando scipad() o scipad f .

Si por ejemplo se invoca scipad f, y se vuelve a invocar nuevamente el mismo comando, aparecerá un mensaje indicando que se debe guardar el archivo contenido en esa sesión de Scipad y abrir una sesión de Scipad con otro nombre. Scipad permite el uso de shorcuts con el teclado como los siguientes:

La tecla de función F2, permite guardar el archivo. La extensión con la que se guardan los archivos en scipad para ser ejecutados en SCILA puede ser: sce, sci y tst, generalmente se usa sce

F5 guarda el archivo y ejecuta las lineas del programa escrito en Scipad dentro de SCILAB.

Doble click en el botón izquierdo del ratón selecciona la palabra que esté cubriendo el cursor.

Tres clicks seguidos con el botón derecho del ratón seleccionan la linea completa que este cubriendo el puntero del cursor.

Con el botón derecho del ratón aparece un menú con varias opciones de edición.

Manteniendo la tecla shift apretada y presionando el botón derecho del ratón aparece un menú que permite seleccionar la ejecución del script o programa dentro de SCILAB.

Manteniendo la tecla Ctrl presionada y apretando el botón derecho del ratón aparece un menú en el que se pueden modificar algunas características de formato y visualización.

Manteniendo la tecla Ctrl apretada y pulsando la tecla mas (+), incrementa el tamaño de la fuente de todo el entorno de Scipad. (el mas "+" no debe ser el del teclado numérico)

Manteniendo la tecla Ctrl apretada y pulsando la tecla menos o guion (-) el tamaño de la fuente del entorno Scipad disminuye.

Para cargar el programa o script dentro de SCILAB se puede usar el siguiente shortcut: Ctrl+Alt+L y es análogo a llamar el archivo con el contenido del script usando el comando exec desde SCILAB.

5.1.2. EJECUTAR UN SCRIPT CON exec()

5.1.2.1. Usando GNU/LINUX

exec ejecuta los programas .sce o .sci para que corran en el entorno de SCILAB, el argumento contiene dos variables, el primero es el path y el segundo es mode, el cual no sera tratado en este libro. Para información sobre lo que hace mode puede consultar en la linea de comandos de SCILAB ejecutando help exec.

exec(path [,mode])

path es la ruta a nivel de directorios en la que se encuentran los archivos que contienen las lineas de programación o los scripts que serán ejecutados en SCILAB. Para saber cual es la ruta o el path predeterminado se usa el comando pwd, en nuestro caso al ejecutar pwd en la linea de comandos de SCILAB bajo LINUX aparece:

```
-->pwd
ans =
/home/elec
```

Para cambiar el directorio o la ruta predeterminada se puede hacer lo siguiente : crear un directorio especifico para guardar los scripts, en nuestro caso el directorio se llama sciprogs. Teniendo el directorio en el que se guardaran los scripts, en SCILAB se ejecuta el comando chdir() así:

```
-->chdir('/home/elec/sciprogs')
```

```
ans =

0.
```

La respuesta igual a cero indica que la ruta se ha cambiado satisfactoriamente. Para comprobarlo, puede ejecutar nuevamente el comando pwd.

El siguiente paso es probar el primer script, para comenzar crearemos algunas lineas en Scipad para ejecutarlas con exec():

```
->scipad()
```

Dentro de la ventana de scipad escribiremos lo siguiente:

```
//primer script

t=linspace(0,2*%pi,180);

w=2;

am=exp(-t);

sig=am.*cos(w*t)+am.*sin(w*t)*%i;

plot2d(real(sig),imag(sig));

xtitle("plot de mi primer script","real", "im")
```

Para guardar el archivo se da un click en file dentro del entorno de Scipad y en el menú que muestra, seleccionamos save as, se busca la carpeta o directorio que pusimos con chdir y en el campo Nombre de archivo le damos un nombre usando la extension .sce (Figura 5.1), para este ejemplo usaremos 1script.sce como nombre. Por ultimo damos un click en salvar y ahora podemos ejecutar el script desde el entorno de SCILAB así:

```
-->exec 1script.sce
```

El resultado del script debe ser la gráfica mostrada en la Figura 5.2.

Notese que para ejecutar el script con exec no fue necesario poner en el argumento la ruta completa en la que se encuentra el archivo 1script.sce; ésto debido al proceso que se hizo con el comando chdir.

Figura 5.1: Ventana guardar como en Scipad

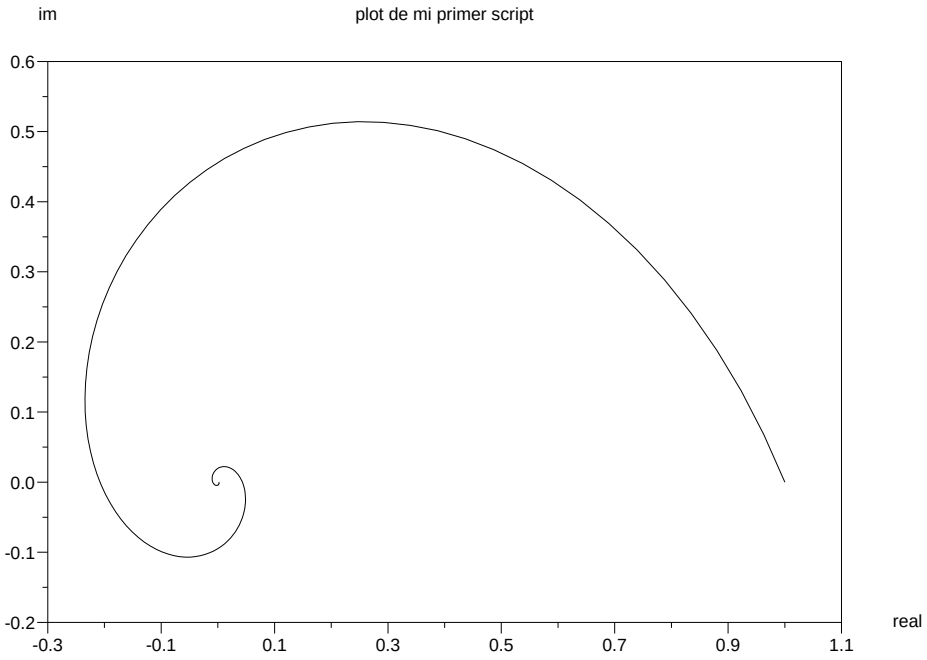


Figura 5.2: Resultado de 1script.sce

5.1.2.2. Usando pwd y chdir MS WINDOWS®

En este sistema al ejecutar pwd en el entorno de SCILAB muestra la ruta:

```
-->pwd
ans =
C:\Documents and Settings\elec\Escritorio
```

Ahora, para cambiarla también puede usar chdir() y creando la carpeta sciprogs en C: se puede hacer:

```
-->chdir('c:\sciprogs')
ans =
0.
```

La respuesta igual a cero indica que la ruta se ha cambiado satisfactoriamente. Para comprobarlo, puede ejecutar nuevamente el comando pwd.

El siguiente paso es probar el primer script, para comenzar crearemos algunas lineas en Scipad para ejecutarlas con exec():

Puede seguir los pasos de la sección 5.1.2.1.

SCILAB puede ser una herramienta efectiva a la hora de programar. Durante este capitulo abordaremos algunos temas como: herramientas de programación , definición y uso de funciones y definición de operadores para nuevas clases de datos.

5.2. HERRAMIENTAS DE PROGRAMACIÓN

SCILAB soporta un buen grupo de rutinas de programación conocidas, que incluyen: ciclos, estructuras condicionales, select case y creación de ambientes, muchas tareas de programación pueden ser llevadas a cabo generando funciones. En este capitulo explicaremos algunas características de programación disponibles en SCILAB.

5.2.1. Operadores Lógicos

Existen cinco métodos para realizar comparaciones entre valores y variables creadas en SCILAB. El Cuadro 5.1 muestra los símbolos que se usan para realizar comparaciones en el entorno de programación de SCILAB.

5.2.2. Ciclos for y while

Existen dos tipos de ciclos en SCILAB: el ciclo for y el ciclo while. for ejecuta las acciones de la estructura del ciclo un numero especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del ciclo. for

símbolo	significado
==	igual a
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
<>	diferente de

Cuadro 5.1: Operadores lógicos

esta delimitado por end.

NOTA: Se asume que los ejemplos de este capitulo y posteriores serán desarrollados en Scipad.

Ejemplo:

```
//ciclo for
x=1
for i=1:4, x=x*i, end
```

En SCILAB:

```
-->exec for.sce
-->//ciclo for
-->x=1
x =
1.
-->for i=1:4, x=x*i, end
x =
1.
x =
2.
x=
6.
x =
24.
```

El ciclo for puede iterar en cualquier vector o matriz, tomando como valores los elementos del vector o las columnas de la matriz.

Ejemplo:

```
//ciclo for con vector
x=1;
for i=[-2 1 4], x=x+i, end;
```

En SCILAB:

```
-->exec forvec.sce
-->//ciclo for con vector
-->x=1;
-->for i=[-2 1 4], x=x+i, end;
x =
- 1.
x =
0.
x =
4.
```

El ciclo while realiza una secuencia de iteraciones como lo hace for, pero solamente cuando se cumplen una serie de condiciones:
Ejemplo:

```
//ciclo while
x=1; while x<14, x=2*x, end;
```

En SCILAB:

```
-->exec while.sce
-->//ciclo while
-->x=1;
-->while x<14,x=2*x, end;
x =
2.
x =
4.
x =
8.
x =
```

16.

El ciclo for y while pueden ser finalizados usando el comando break así:

```
//usando break
a=0;
for i=1:5:100,a=a+1;if i>10 then break, end; end
a
```

En SCILAB

```
-->exec forbreak.sce
-->//usando break
-->a=0;
-->for i=1:5:100,a=a+1;if i>10 then break, end; end
-->a
a =
3.
```

En ciclos anidados, break sale del interior del ciclo:

```
//ciclos anidados
for k=1:3;
  for j=1:4;
    if k+j>4 then break; else disp(k);
    end;
  end;
end
```

En SCILAB:

```
-->exec anidado.sce
-->//ciclos anidados
-->for k=1:3;
-->    for j=1:4;
-->        if k+j>4 then break; else disp(k);
-->        end;
-->    end;
-->end
```

1.
1.
1.
2.
2.
3.

5.3. Condicionales

En SCILAB existen dos tipos de estructuras condicionales: la estructura condicional if-then-else y la estructura select-case. La estructura if-then-else evalua una expresión, si la condición es verdadera, ejecuta las instrucciones entre then y else (o end). Si la condición no es verdadera, lo que este entre else y end es ejecutado. else no siempre se requiere dentro de la estructura if. elseif también se puede usar y es reconocido por el interprete de comandos.

```
//condicionales
x=1;
if x>0 then, y=-x, else, y=x,end
x=-1;
if x>0 then, y=-x, else, y=x,end
```

En SCILAB

```
-->exec condic.sce
-->//condicionales
-->x=1;
-->if x>0 then, y=-x, else, y=x,end
y =
- 1.
-->x=-1;
-->if x>0 then, y=-x, else, y=x,end
y =
- 1.
```

La condición select-case elije una expresión entre varias expresiones posibles y ejecuta una de las n acciones entre cada case.

```
//selec-case
qu=input('escriba un numero');
if qu>1 then,x=%t, else,x=%f,end
select x, case%t, y=qu+5, case%f, y=sqrt(qu), end
```

En SCILAB

```
-->exec scase.sce
-->//selec-case
-->qu=input('escriba un numero');
escriba un numero-->6
-->if qu>1 then,x=%t, else,x=%f,end
x =
T
-->select x, case%t, y=qu+5, case%f, y=sqrt(qu), end
y =
11.
```

5.4. Uso y definición de funciones

Sabemos que es posible definir una función en el entorno de SCILAB, sin embargo, es mas conveniente para el uso futuro de las funciones, crear archivos que contengan una función dentro de Scipad. En esta sección describiremos la estructura de una función y varios comandos de SCILAB que serán usados en la creación de un ambiente de funciones.

5.4.1. Estructura de una función

La forma general de una función para SCILAB es la siguiente:

function [y1,...,yn]=fn(x1,...,xm)

.

.

.

donde fn es el nombre de la función, el conjunto de las x son los m argumentos de entrada de la función, el conjunto de y son los n argumentos de salida de la función, y los 3 puntos verticales bajo la palabra function representan el listado de instrucciones

ejecutados por la función, el siguiente ejemplo calcula el factorial de k (k!):

```
//funcion para calcular factorial
function [x]=fact(k)
  k=int(k)
  if k<1 then k=1, end
  x=1;
  for j=1:k,x=x*j;end
endfunction
```

La función está dentro del archivo fact.sce. Debe ser cargada dentro de SCILAB usando los comandos exec o getf, para comprobar si una función está cargada se usa:

```
-->exists('fact')
ans =
0.
```

La respuesta 0 indica que la función no está cargada.

```
-->exec fact.sce;
-->exists('fact')
ans =
1.
```

Ahora podemos probar nuestra función fact así:

```
->for i=1:5, x=fact(i), end
x =
1.
x =
2.
x =
6.
x =
24.
x =
120.
```

El entorno y las funciones de programación son bastante amplias, por ahora hemos hecho énfasis en cuestiones introductorias, esperamos cubrir profundamente este tema en las próximas versiones de este libro.

Capítulo 6

APLICACIONES

El presente capítulo hace parte de uno de los principales objetivos de este libro y es tratar de extender el uso de SCILAB como alternativa en el modelamiento de varias tareas en el aprendizaje de las matemáticas, la física y las aplicaciones que puede tener en la Ingeniería Electrónica, Mecánica, control y comunicaciones. Para la primera versión de este libro incluiremos algunas aplicaciones dirigidas a los circuitos y control, esperamos que el capitulo crezca gradualmente con ayuda de la comunidad de usuarios de SCILAB que quieran apoyar este proyecto. En la parte final de este capítulo pondremos un listado con los nombres de las personas que hagan contribuciones a este capitulo, con las respectivas referencias.

6.1. FILTRADO Y REPRESENTACIÓN DE FUNCIONES DE TRANSFERENCIA.

Para el filtrado de funciones utilizaremos de SCILAB, las funciones wfir(), syslin(), flts().

6.1.1. La función: 'wfir()' genera la fase lineal de un filtro.

Esta función tiene la siguiente sintaxis:

```
[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)
```

donde:

ftype : el tipo de filtro que queremos 'lp', 'hp', 'bp', 'sb'.

forder : orden del filtro.

cfreq : vector de corte de frecuencia (0<cfreq(1), cfreq(2)<.5)

wtype : tipo de ventana ('re','tr','hm','hn','kr','ch')
fpar : vector de parámetros.

Respuestas:

wft : Coeficientes del filtro en el dominio del tiempo
wfm : Respuesta del filtro en frecuencia.
fr : frecuencia.

6.1.2. La función: 'syslin()' nos define un sistema lineal.

Su sintaxis es:

[sl]=syslin(dom,A,B,C [,D [,x0]])
[sl]=syslin(dom,N,D)
[sl]=syslin(dom,H)

donde:
dom : ('c' , 'd'), especifica el dominio del tiempo del sistema en: continuo 'c', o discreto 'd'.
A,B,C,D : matrices de representación de estados
N, D : matrices polinómicas.
H : matriz racional o representación lineal de estados.
sl : tlist (" syslin " list) representación lineal del sistema.

6.1.3. La función: 'flts()' nos muestra la respuesta en el tiempo.

sintaxis:

flts(u,sl)
donde:
u : matriz o vector de entrada al sistema.
sl : el sistema o respuesta obtenido con syslin

ejemplo en SCILAB:

```
-->[wft,wfm,fr]=wfir('lp',20,[0.2 0], 'hm',[0 0]);
-->t=1:100;
-->x1=sin(%pi*t/10);
```

```
-->x2=sin(2*%pi*t/3);

-->x=x1+x2;

-->s=poly(0,'s');

-->hz=syslin('d',poly(wft,'s','c')./s**20);

-->yhz=flts(x,hz);

-->plot2d(t,yhz);
```

respuesta:

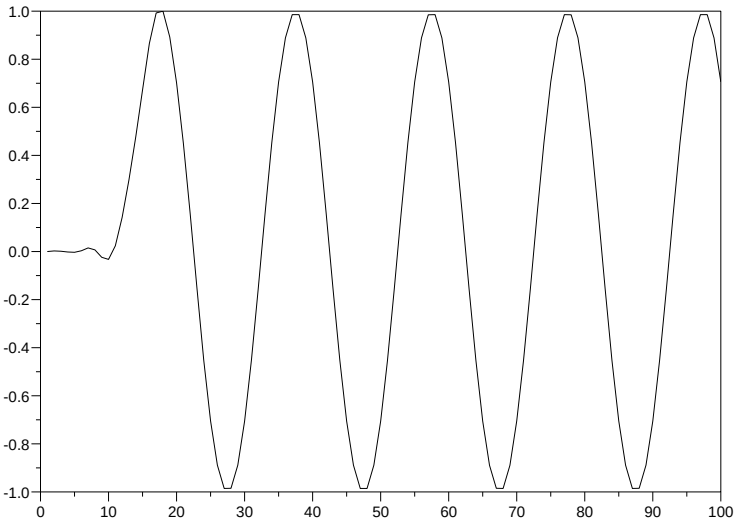


Figura 6.1: Señal de Salida.

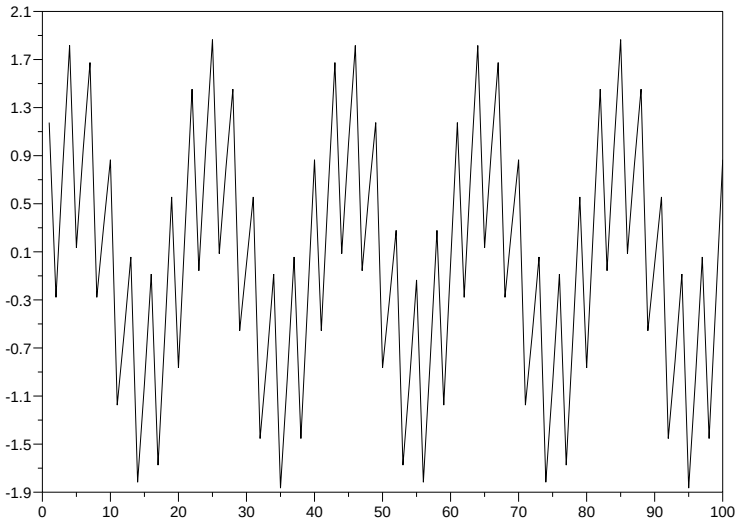


Figura 6.2: Señal de Entrada

6.2. Representación de Polos Y Ceros

Para representar gráficamente los polos y ceros de un filtro utilizamos la función de SCILAB 'plzr()'.

sintaxis:

plzr(sl)

Donde:

sl = respuesta de syslin

Ejemplo en SCILAB:

```
-->//Del anterior ejemplo evaloamos polos y ceros de 'hz'
-->plzr(hz)
```

respuesta:

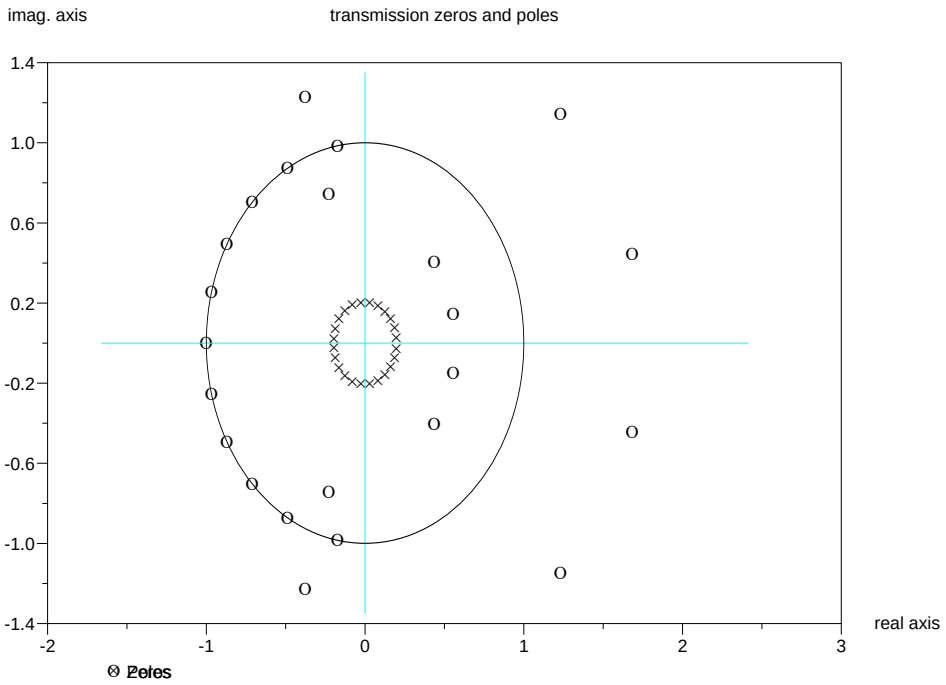


Figura 6.3: Polos y Ceros de hz

para verificar los ceros hallamos en SCILAB las raices de 'hz'

```
//raices de hz
-->roots(numer(hz))
ans =
! 0.4351230 + 0.4049239i !
! 0.4351230 - 0.4049239i !
! - 0.2286276 + 0.7434353i !
! - 0.2286276 - 0.7434353i !
! 0.5552955 + 0.1469770i !
! 0.5552955 - 0.1469770i !
! - 0.1736648 + 0.9848048i !
! - 0.1736648 - 0.9848048i !
! - 0.4875428 + 0.8730991i !
! - 0.4875428 - 0.8730991i !
! - 0.7110511 + 0.7031404i !
! - 0.7110511 - 0.7031404i !
```

```
! - 0.8697111 + 0.4935611i !
! - 0.8697111 - 0.4935611i !
! - 0.3779177 + 1.2288865i !
! - 0.3779177 - 1.2288865i !
! - 0.9670702 + 0.2545098i !
! - 0.9670702 - 0.2545098i !
! - 1. !
! 1.2316125 + 1.1461341i !
! 1.2316125 - 1.1461341i !
! 1.6829415 + 0.4454453i !
! 1.6829415 - 0.4454453i !
```

6.3. Diagramas de Bode.

para gráficar un diagrama de Bode en SCILAB utilizamos la función: 'bode()'.

Sintaxis:

bode(sl,[fmin,fmax] [,step] [,comments])

Donde :
sl : respuesta de syslin en tiempo continuo o discreto.
fmin,fmax : valor real 'frecuencia in Hz'
step : valor del escalón.

Ejemplo:
De la ecuación de estado siguiente generar el diagrama de Bode.

$$x^* = 7\pi x + u$$

$$y = 10\pi x + u$$

En SCILAB:

```
-->//diagrama de bode de 'hz'.
-->a=7*%pi;
-->b=1;
-->c=10*%pi;
-->d=1;
```

```
-->r=syslin('c',a,b,c,d)

r =

r(1) (state-space system:)

!lss A B C D X0 dt !

r(2) = A matrix =

21.991149 dp

r(3) = B matrix =

1.

r(4) = C matrix =

31.415927

r(5) = D matrix =

1.

r(6) = X0 (initial state) =

0.

r(7) = Time domain =

c .

-->bode(r,0.1,100);
```

respuesta:

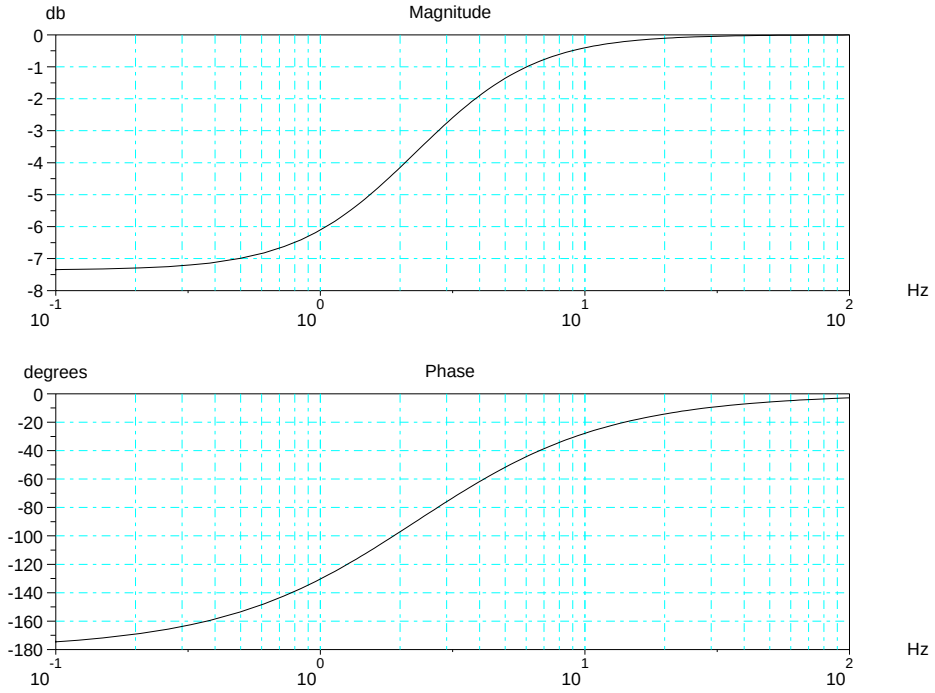


Figura 6.4: bode del sistema

6.4. REPRESENTACIÓN DE SISTEMAS LINEALES.

La función utilizada en SCILAB para definir sistemas lineales es 'syslin()' esta función toma los parámetros de las matrices que definen el sistema lineal en espacio de estados.

Otra función a tener en cuenta es 'ss2tf()' que convierte de espacio de estados a funciones de transferencia.

Su sintaxis es:

[h]=ss2tf(sl)

Donde sl : sistema lineal resultante de 'syslin()'

h : matriz de salida.

Ejemplo:

Definimos un sistema lineal.

$$A = \begin{bmatrix} 1 & -3 \\ 7 & -2 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, C = \begin{bmatrix} 4 & -2 \end{bmatrix};$$

En SCILAB:

```
-->A=[1 -3;7 -2];
-->B=[1;0];
-->C=[4 -2];
-->slin=syslin('c',A,B,C)
slin =
slin(1) (state-space system:)
!lss A B C D X0 dt !
slin(2) = A matrix =
! 1. - 3. !
! 7. - 2. !
slin(3) = B matrix =
! 1. !
! 0. !
slin(4) = C matrix =
! 4. - 2. !
slin(5) = D matrix =
0.
slin(6) = X0 (initial state) =
! 0. !
! 0. !
slin(7) = Time domain =
c
-->//conversion a tf.
-->hs=ss2tf(slin)
hs =
- 6 + 4s
----------
19 + s + s^2
```

6.5. SISTEMAS CONECTADOS.

Uso de SCILAB para los siguientes casos de sistemas conectados.

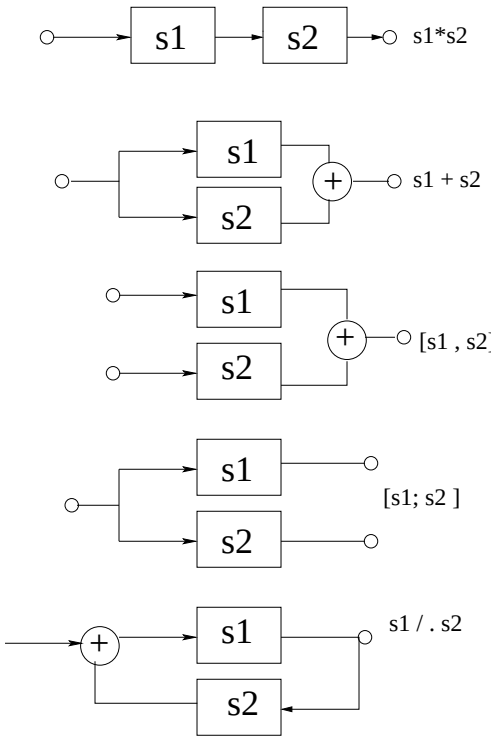


Figura 6.5: Tipo de Sistemas Conectados

Para manipular los anteriores sistemas SCILAB utiliza las siguientes funciones:
'tf2ss()' transforma una función de transferencia a un sistema de estados

sintaxis:

sl=tf2ss(h)

Donde:
h : matriz racional
sl : sistema lineal (sl=[A,B,C,D(s)])

'ssprint ()'

sintaxis:

ssprint(sl [,out])

Donde:

sl : respuesta de syslin
out : respuesta de ssprint

Ejemplo:

$$S1 = \frac{1}{s+2},$$

$$S2 = \frac{1}{3-s}$$

De los sistemas anteriores vamos a realizar :

1. $S1 + S2$
2. $S1 * S2$
3. $\begin{bmatrix} S1 \\ S2 \end{bmatrix}$
4. $[S1\, S2]$
5. $\frac{S1}{S2}$

En SCILAB:

```
-->s=poly(0,'s')
s =
s
-->s1=1/(s+2)
s1 =
  1
-----
2 + s
-->s2=1/(3-s)
s2 =
  1
-----
3 - s
-->s1=syslin('c',s1);
-->s2=syslin('c',s2);
-->ss2=tf2ss(s2);
-->ss1=tf2ss(s1);
```

```
-->ssprint(ss2)
.
x = | 3 |x + | 1 |u
y = |-1 |x
-->rhs=ss2*s1;
-->ssprint(rhs)

.   | 3 1  |    | 0 |
x = | 0 -2 |x + | 1 |u
y = |-1 0  |x
-->ht=ss2tf(rhs)
ht =
    -1
------------
- 6 - s + s^2
```

vimos como podemos operar entre funciones de transferencia y ecuaciones de estado, aunque la respuesta es también directa de la siguiente manera:

```
-->s1*s2
ans =
    1
----------
6 + s - s^2

-->s1+s2
ans =
    5
---------
6 + s - s^2

-->[s1;s2]
ans =
!   1    !
! -----  !
! 2 + s  !
!        !
!   1    !
```

```
! ------ !
! 3 - s  !
-->[s1 s2]
ans =
!   1           1 !
! -------   ------- !
! 2 + s     3 - s !

-->s1/.s2
ans =
 3 - s
---------
7 + s - s^2
```

6.6. LAPLACIANO BIDIMENSIONAL.

El siguiente código para Scilab es una implementación parcial para la solución de un laplaciano Bidimensional.

```
n=50; y=1:2*n; x=1:n;
for i=1:n,
a(i,1)=1000;
a(i,2*n)=1000;
end
for j =1:2*n,
a(1,j)=1000;
a(n,j)=0;
end
errmax=100;
while errmax >1,
errmax=0;
for i =2:n-1,
for j =2:2*n-1,
vr=0.25*(a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1));
err=abs(a(i,j)-vr);
if err >errmax then
errmax =err;
```

```
end,
a(i,j)=vr;
end
end
end
plot3d(x,y,a);
```

En el se puede observar:

- La definición del número de puntos de que constará la matriz $(nX2n)$. Observe que hemos asumido un condensador cuyas dimensiones son n X 2n.
- La definición de las condiciones de borde del condensador $(V1 = 1000, V0 = 0)$.
- La iteración del calculo (mientras el error máximo sea mayor a 1) del valor de cada punto como: $V_{i,j} = \frac{1}{4}(V_{i-1,j} + V_{i,j-1} + V_{i+1,j} + V_{i,j+1})$ manteniendo fijos los valores en los bordes $V_1 = 1000, V_0 = 0$
- El cálculo del error máximo en la matriz de puntos resultante para definir el criterio de la salida (cuando el error máximo sea menor a 1) .
- La representación gráfica del valor de potencial en cada punto.

A continuación encontrará algunos gráficos de la salida de la simulación.

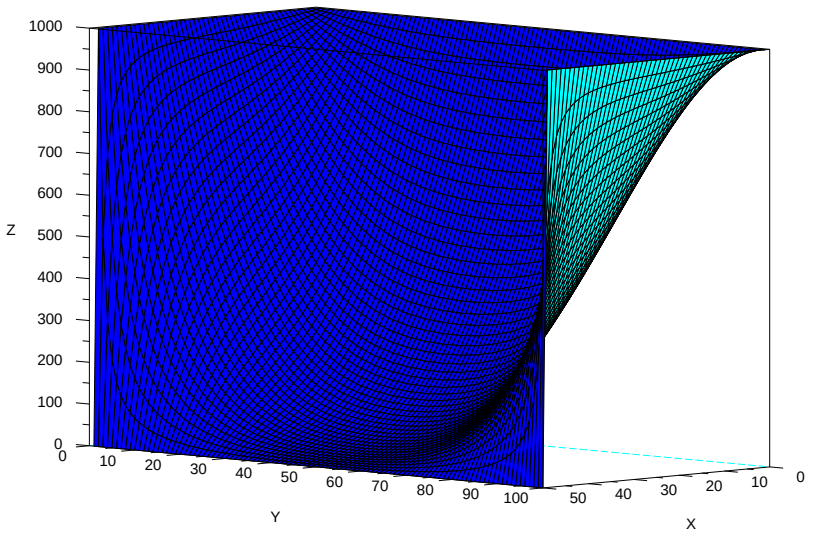


Figura 6.6: Valor de potencial en función de x & y

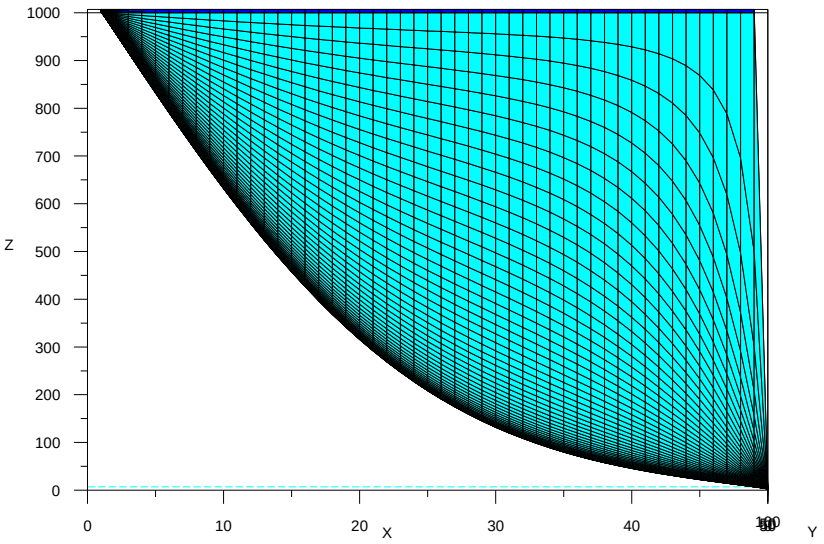


Figura 6.7: Vista lateral

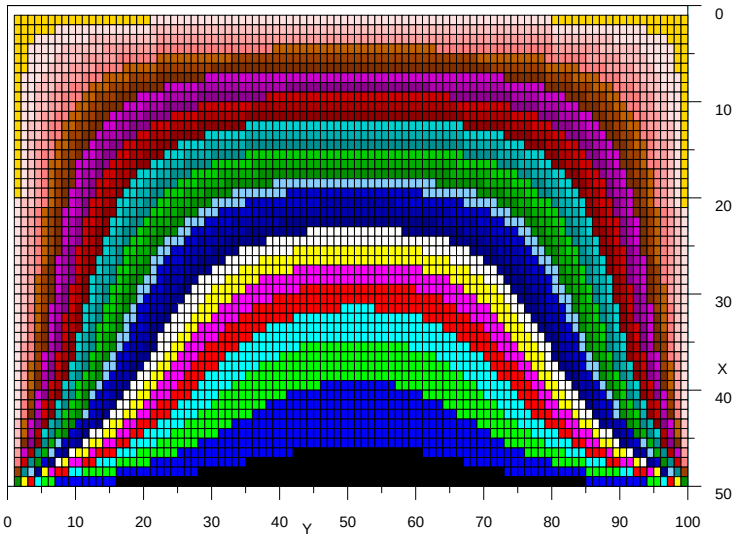


Figura 6.8: Vista superior con plot3d1()

6.7. CONTROLADORES P I D.

En SCILAB podemos simular la respuesta de una planta controlada por P,I,D con una señal escalón de entrada. Para esto utilizamos las funciones "syslin()" y "csim()"
.

"csim()" simula la respuesta en el tiempo de un sistema lineal.

Su sintaxis es:

r=csim(u,t,sl)

Donde:

u : es la función de entrada en este caso step (escalón).

t : el vector tiempo en el que se va a evaluar la función.

sl : la respuesta de la función syslin.

r: respuesta.

Ejemplo:

en SCILAB:

```
-->//Proporcional-Derivativo (PD)
-->s=poly(0,'s');
-->t=0:0.01:6;
-->//Planta
-->pl=1/(s**2+1);
-->//Controlador
-->K=1;
-->D=1;
-->sis=K*(1+D*s);
-->res=sis*pl/(1+sis*pl);
-->y=syslin('c',res)
-->z=csim('step',t,y);
-->plot2d(t,z,2);
-->xtitle('Proporcional.Derivativo (1/(s**2+1)) K=1 D=1','tiempo','y(t)');
```

respuesta:

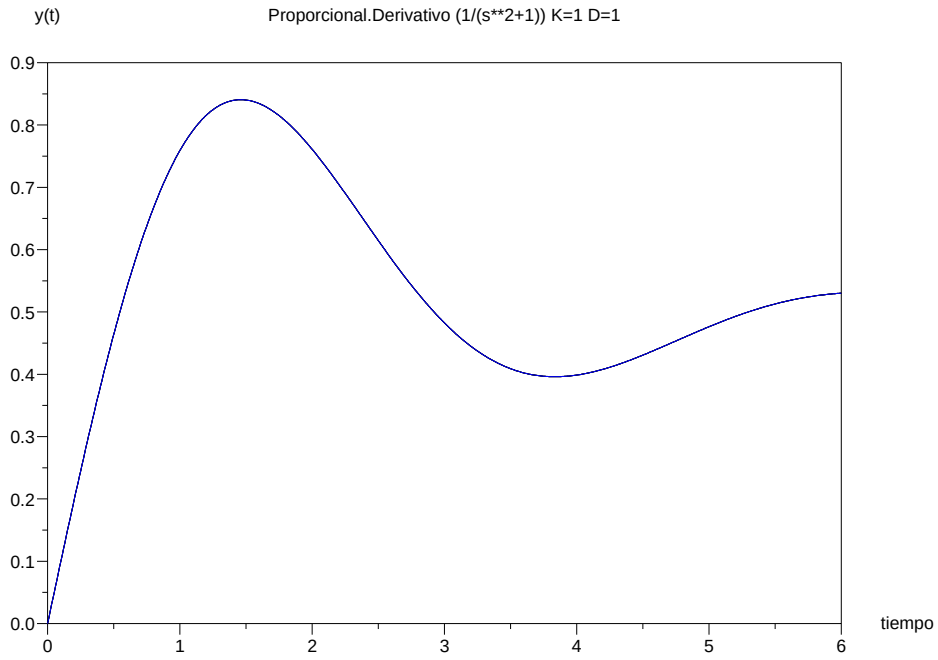


Figura 6.9: Proporcional Diferencial

podemos ver la variación de k en la misma gráfica utilizando "for() ":

Eemplo:

```
-->s=poly(0,'s');
-->t=0:0.01:6;
-->//Planta
-->pl=1/(s**2+1);
-->//Controlador
-->for i=1:5
-->K=i;
-->D=1;
-->sis=K*(1+D*s);
-->res=sis*pl/(1+sis*pl);
-->y=syslin('c',res);
-->z=csim('step',t,y);
-->plot2d(t,z,i);
```

```
-->end
-->xtitle('Proporcional.Derivativo (1/(s**2+1)) K=[1 2 3 4 5] D=1','tiempo','y(t)');
```

Respuesta:

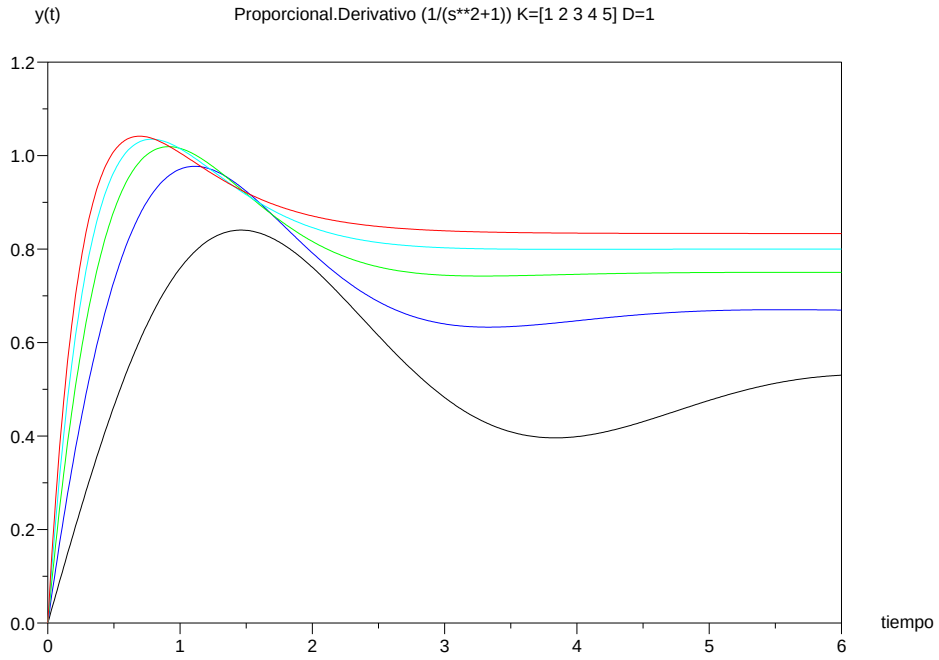


Figura 6.10: K= 1,2,3,4,5

Variando D:

```
-->s=poly(0,'s');
-->t=0:0.01:6;
-->//Planta
-->pl=1/(s**2+1);
-->//Controlador
-->a=0;
-->for i=0.5:0.5:2.5;
-->a=a+1;
-->K=3;
-->D=i;
-->sis=K*(1+D*s);
```

```
-->res=sis*pl/(1+sis*pl);
-->y=syslin('c',res);
-->z=csim('step',t,y);
-->plot2d(t,z,a);
-->end
-->xtitle('Proporcional.Derivativo (1/(s**2+1)) K=3 D=[0.5, 1, 1.5, 2, 2.5]','tiempo
```

Respuesta:

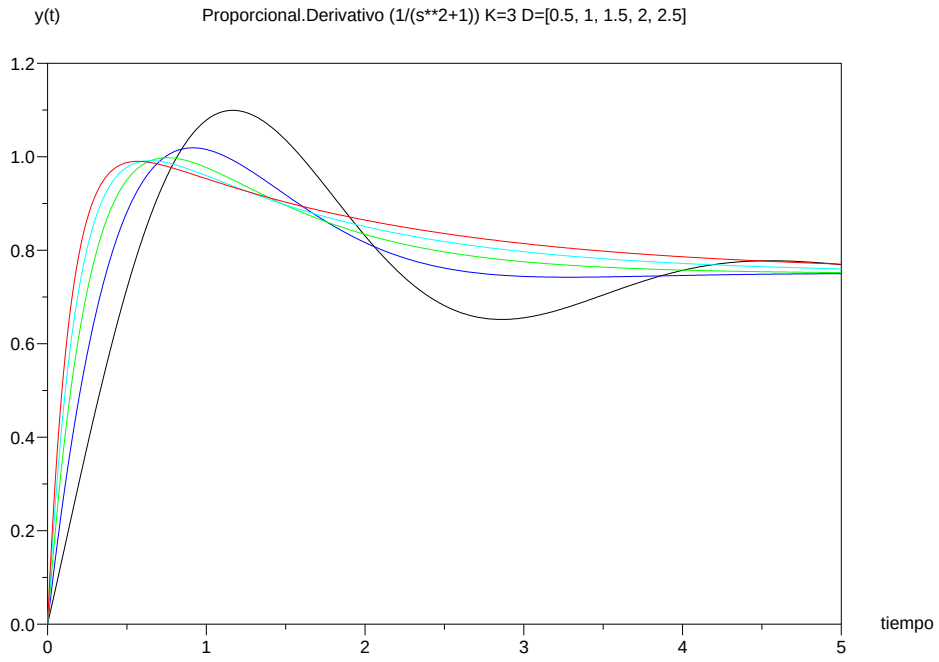


Figura 6.11: D= 0.5, 1, 1.5, 2, 2.5

Proporcional-integrador pi

```
-->xbasc();
-->s=poly(0,'s');
-->t=0:0.01:50;
-->//Planta
-->pl=1/(4*s+6);
-->//Controlador (PI)
-->for i=1:5;
```

```
-->K=i;
-->I=2;
-->sis=K*(1+1/(I*s));
-->res=sis*pl/(1+sis*pl);
-->y=syslin('c',res);
-->z=csim('step',t,y);
-->plot2d(t,z,i);
-->end;
-->xtitle('Proporcional Integrador Planta=1/(4s+6), K=[1 2 3 4 5] I=2','tiempo','y(t)');
```

Respuesta:

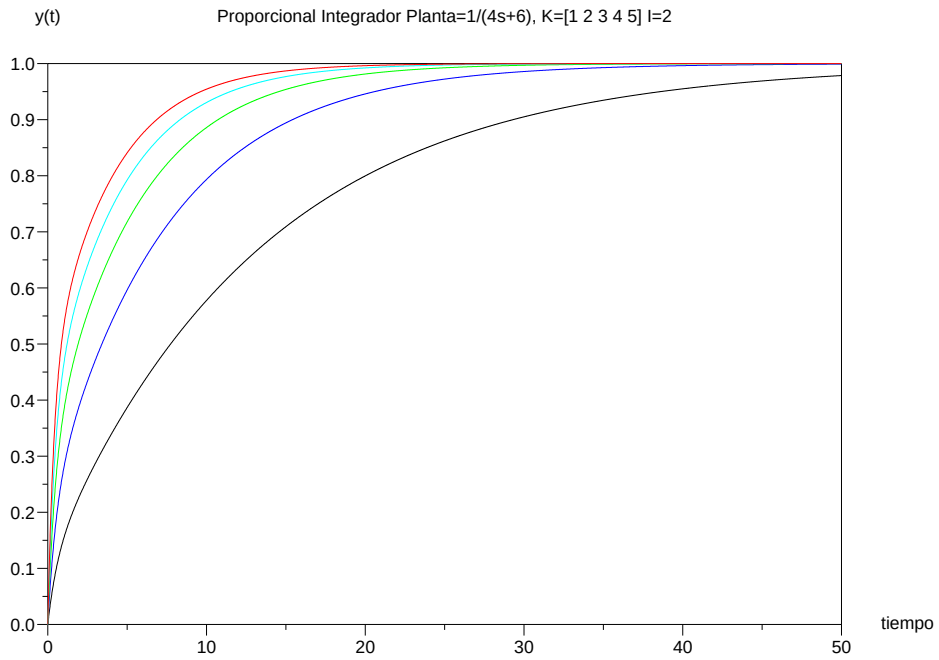


Figura 6.12: PI, P= 1, 2, 3, 4, 5. I=2

Variando I

```
-->xbasc();
-->s=poly(0,'s');
-->t=0:0.01:50;
-->//Planta
```

```
-->pl=1/(4*s+6);
-->//Controlador (PI)
-->a=0;
-->for i=0.4:0.4:2;
-->a=a+1;
-->K=3;
-->I=i;
-->sis=K*(1+1/(I*s));
-->res=sis*pl/(1+sis*pl);
-->y=syslin('c',res);
-->z=csim('step',t,y);
-->plot2d(t,z,a);
-->end;
-->xtitle('Proporcional Integrador Planta=1/(4s+6), K=[1 2 3 4 5] I=2','tiempo','y(t
```

respuesta:

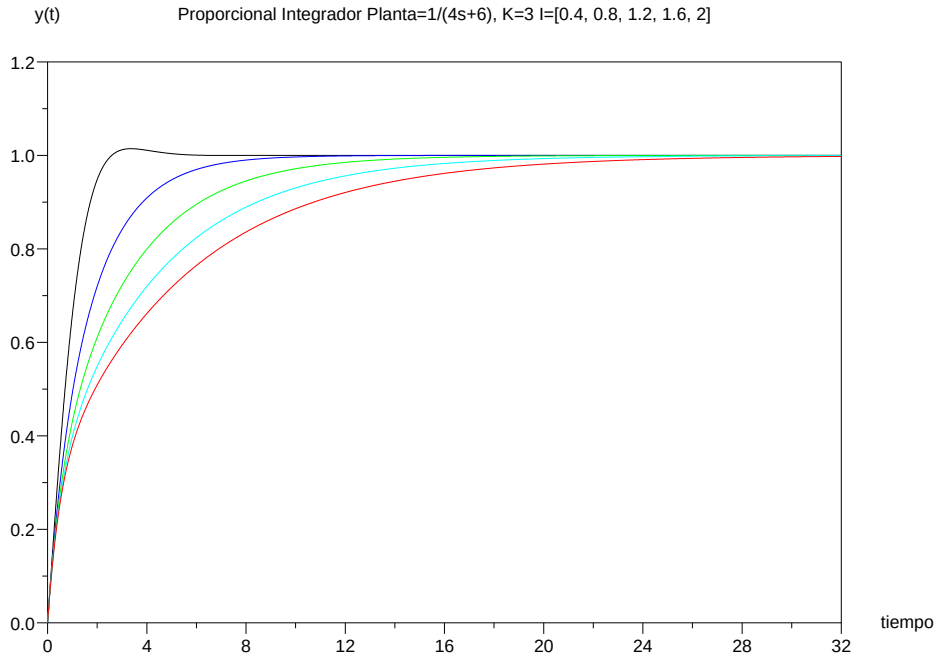


Figura 6.13: PI K=3, I=[0.4, 0.8, 1.2, 1.6, 2.0]

PERSONAS QUE HAN CONTRIBUIDO CON ESTE CAPÍTULO

Ing. Luis Guillermo Gómez , Coordinador Licenciatura en electrónica Universidad Pedagógica Nacional. Bogotá, Colombia. Contribución: LAPLACIANO BIDIMENSIONAL

Índice alfabético