

Tutorial Archinformal de Octave 0.8.1

Diego Bravo Estrada

Abril 2024

Este es un tutorial confeccionado durante algunas sesiones de auto-aprendizaje; con suerte puede ser útil a algunos lectores. Escribir al autor a “diegobravoestrada arroba hotmail punto com” para correcciones y sugerencias.

Contents

1 Intro

“GNU Octave” es un programa que proporciona un lenguaje orientado al cálculo numérico basado en operaciones matriciales. La sintaxis básica es muy similar a la de Matlab, por lo que es posible desarrollar funciones que se ejecutan idénticamente en ambos aplicativos, lo que se ha puesto en práctica en este documento¹ (con algunas excepciones muy puntuales.)

Matlab es un excelente producto comercial que suele percibirse superior a Octave en diversos aspectos, especialmente cuando se incluyen sus extensiones (denominadas *toolboxes*.) Sin embargo, en nuestra opinión Octave proporciona una funcionalidad más que suficiente para la gran mayoría de tópicos de cálculo numérico empleados en ingeniería, métodos numéricos y matemáticas aplicadas, salvo requerimientos extremadamente especializados.

A diferencia de Matlab, Octave se proporciona gratuitamente a los interesados (por el contrario, Matlab puede ser muy costoso dependiendo del contexto de uso.) Además, y no menos importante, el código fuente de Octave está totalmente disponible para quienes desean estudiar cómo funciona, modificarlo y/o extenderlo.

Los usuarios de Windows pueden descargar y ejecutar un instalador de Octave desde <https://octave.org/download.html>; en las distribuciones Linux normalmente se proporciona como uno de los paquetes disponibles en sus respectivos repositorios de software.

1.1 Iniciar Octave

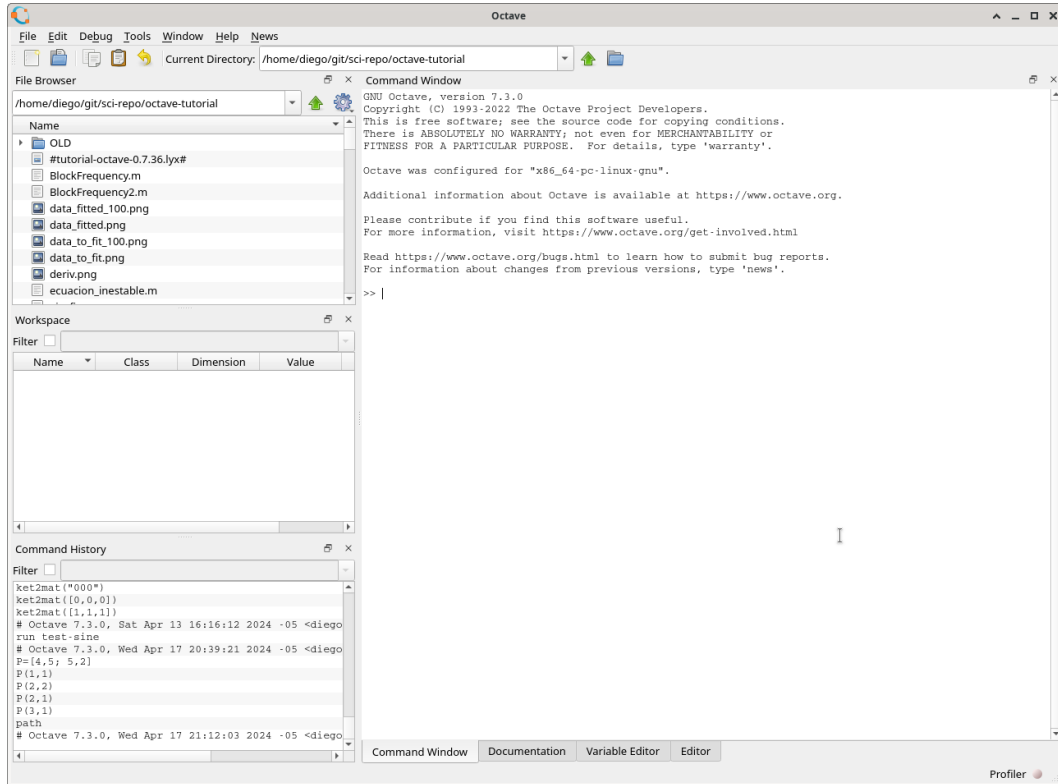
La forma de iniciar Octave depende del sistema operativo en el que se instala. En los entornos Windows usualmente se lanza desde un acceso directo creado por el programa de instalación, o una entrada en el menú de programas. En los entornos Linux también suele estar presente en el menú de programas, y adicionalmente puede lanzarse desde la línea de comandos (shell):

```
$ octave
```

La forma estándar (y usualmente más conveniente) de utilizar Octave es a través de su entorno gráfico²:

¹En realidad, este documento también se puede considerar un tutorial básico de Matlab.

²Es posible forzar el “modo gráfico” o el “modo texto” de Octave proporcionando las opciones `--gui` y `--no-gui` en la línea de comandos, respectivamente.



Observar la ruta indicada en “Current Directory”, la cual puede modificarse para que apunte al directorio donde el usuario desea almacenar su trabajo. La región denotada “Command Window” permite introducir los comandos de Octave que se discuten en el resto de este documento (a continuación del indicador “>>”). En este documento utilizaremos el prompt “octave>” para hacer referencia a los comandos que debe introducir el usuario.

1.2 Usando Matrices

Nuestro primer ejemplo consiste en crear la matriz:

$$P = \begin{pmatrix} 4 & 5 \\ 5 & 2 \end{pmatrix}$$

La notación correspondiente requiere delimitar los elementos de cada fila con comas (o simplemente espacios), y las columnas con el signo “punto y coma”:

```
octave> P=[4,5; 5,2]
P =
4 5
5 2
```

En Octave los elementos existentes en una matriz se obtienen mediante el par (fila, columna), donde estos valores se inician desde el número uno³:

```
octave> P(1,1)
ans = 4
octave> P(2,1)
ans = 5
octave> P(2,2)
ans = 2
octave> P(3,1)
error: P(3,_): out of bound 2 (dimensions are 2x2)
```

Si se agrega un punto y coma al final, no se imprime el valor recién asignado:

```
octave> Q=[1,1; 1,3];
```

Octave es “case sensitive”; por ejemplo, “q” no es lo mismo que “Q”:

```
octave> q
error: `q' undefined near line 3 column 1
```

Imprimir la matriz recién creada:

```
octave> Q
Q =
1 1
1 3
```

Otras funciones incorporadas permiten crear matrices. Por ejemplo, una matriz de números aleatorios⁴ de 5 filas y 2 columnas:

```
octave> ALEATORIOS = rand(5,2)
ALEATORIOS =
0.422600 0.044483
0.567010 0.917974
0.401833 0.954165
```

³En relación a otros lenguajes de programación, la estructura de datos correspondiente a las matrices usualmente se denomina “array” (y más precisamente, array de dos dimensiones.) El índice de los elementos de los arrays suele partir de cero o uno, dependiendo de cada lenguaje de programación. Por ejemplo, en FORTRAN es uno (por omisión), mientras que en C y relacionados (como C++, Java, Javascript, Php, Rust, etc.) es cero.

⁴Los números aleatorios de `rand()` están uniformemente distribuidos en el intervalo (0,1); existen otras variantes, notablemente `randn()` permite obtener números aleatorios normalmente distribuidos.

```
0.897529 0.601186
0.321889 0.726979
```

Multiplicación por un escalar:

```
octave> P
P =
4 5
5 2
octave> 2*P
ans =
8 10
10 4
octave> 3 * P
ans =
12 15
15 6
```

Multiplicación y potencia de matrices:

```
octave> P*Q
ans =
9 19
7 11
octave> Q*P
ans =
9 7
19 11
octave> Q^2
ans =
2 4
4 10
octave> Q^3
ans =
6 14
14 34
```

En ocasiones es necesario multiplicar las matrices “elemento a elemento”, así como elevar a una potencia a cada elemento, lo que se consigue mediante la notación `.*` y `.^` como se muestra a continuación:

```
octave> P.*Q
ans =
4 5
5 6
octave> Q.^2
ans =
1 1
1 9
octave> Q.^3
ans =
```

```
1 1
1 27
```

Matriz “conjugada transpuesta”⁵:

```
octave> (P*Q)'  
ans =  
9 7  
19 11
```

1.3 Resolución de ecuaciones Lineales

Al doble de lo que posee Pablito le falta 6 para alcanzar al quintuple de lo que tiene Juanita. Pero a Juanita le falta 9 para alcanzar a Pablito. Cuánto tienen?

Solución:

$$\begin{aligned} 2 * P + 6 &= 5 * J \\ J + 9 &= P \end{aligned}$$

En matrices:

$$\begin{pmatrix} 2 & -5 \\ -1 & 1 \end{pmatrix} * X = \begin{pmatrix} -6 \\ -9 \end{pmatrix}$$

Usando el operador “backslash” de Octave:

```
octave> A=[2, -5; -1, 1]  
A =  
2 -5  
-1 1  
octave> B=[-6; -9]  
B =  
-6  
-9  
octave> A\B  
ans =  
17  
8
```

⁵La función `transpose()` proporciona la transpuesta sin conjugación.

Respuesta Pablito tiene 17, Juanita 8.

Asimismo, matricialmente esta solución se puede expresar con ayuda de la matriz inversa:

$$\begin{pmatrix} 2 & -5 \\ -1 & 1 \end{pmatrix} * X = \begin{pmatrix} -6 \\ -9 \end{pmatrix} \rightarrow X = \begin{pmatrix} 2 & -5 \\ -1 & 1 \end{pmatrix}^{-1} * \begin{pmatrix} -6 \\ -9 \end{pmatrix}$$

Lo que se puede implementar en Octave del siguiente modo⁶:

```
octave> inv(A)*B
ans =
    17
     8
```

Así como en los escalares se tiene $ab^{-1} = \frac{a}{b}$, para las matrices se tiene que $A/B=A*\text{inv}(B)$. En este orden de ideas, se define el operador ./ para efectuar la división de los elementos correspondientes entre dos matrices.

1.4 Intervalos numéricos

Una manera sencilla de crear una matriz de $1 \times N$ cuyos elementos están en progresión aritmética...

```
octave> N=1:10
N =
 1 2 3 4 5 6 7 8 9 10
```

Para que el paso corresponda a dos unidades en vez de una:

```
octave> N=1:2:10
N =
 1 3 5 7 9
octave> N=4:5:80
N =
 4 9 14 19 24 29 34 39 44 49 54 59 64 69 74 79
```

1.5 Búsqueda de elementos

Test de igualdad con un elemento:

```
octave> N == 69
ans =
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

⁶Computacionalmente el cálculo de la matriz inversa es muy costoso, por lo que para este caso se prefiere el operador de backslash.

Como se ve, se retorna una matriz de ceros y unos; los unos corresponden al elemento que es igual a lo especificado.

También se puede buscar los elementos distintos del valor en cuestión⁷:

```
octave> N ~= 69
ans =
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
```

La función `find()` permite ubicar los índices de elementos distintos de cero de una matriz. Por ejemplo:

```
octave> find([0,0,9,0,3,0,1,2,0,0])
ans =
3 5 7 8
```

Más específicamente ¿Cómo ubicar el índice en la matriz que contiene el valor 69?

```
octave> find(N==69)
ans = 14
octave> find(N==68)
ans = [] (1x0)
```

La función `any()` retorna 1 (correspondiente a “true”) si algún elemento de una matriz es distinto de cero. Combinado con `find()` es posible responder directamente a la pregunta ¿el valor 69 está contenido en la matriz?

```
octave> any(find(N==69))
ans = 1
octave> any(find(N==68))
ans = 0
```

Sin embargo, la función `ismember()` proporciona la misma funcionalidad (así como otras opciones):

```
octave> ismember(69,N)
ans = 1
octave> ismember(68,N)
ans = 0
```

1.6 Extracción de elementos de una matriz

Partimos de un “cuadrado mágico” con la función `magic()`:

⁷El operador `~=` (que se puede leer como “no es igual a”) es también válido en Matlab. Adicionalmente Octave permite emplear el operador `!=` con el mismo propósito, el cual es compartido por el lenguaje C y relacionados, pero -lamentablemente- no por Matlab.


```
octave> K=magic(4)
K =
  16   2   3  13
   5  11  10   8
   9   7   6  12
   4  14  15   1
```

Para obtener la segunda fila:

```
octave> K(2,:)
ans =
   5  11  10   8
```

La tercera columna:

```
octave> K(:,3)
ans =
   3
  10
   6
  15
```

El “cuadrado interno”:

```
octave> K(2:3,2:3)
ans =
  11  10
   7   6
```

Las filas 2 a 4, sólo la tercera columna:

```
octave> K(2:4,3)
ans =
  10
   6
  15
```

La fila 2, desde la posición 3 hasta “el final” simbolizado con “end”:

```
octave> K(2,3:end)
ans =
  10   8
```

Otra forma de generar rangos, con `linspace()`. Genera N elementos igualmente espaciados en un intervalo. Por ejemplo, de 2 a 4, 11 elementos:

```
octave> linspace(2,4,11)
ans =
 2.0000  2.2000  2.4000  2.6000  2.8000  3.0000  3.2000  3.4000  3.6000  3.8000  4.0000
octave> linspace(2,4,10)
ans =
 2.0000  2.2222  2.4444  2.6667  2.8889  3.1111  3.3333  3.5556  3.7778  4.0000
```

1.7 Unión de matrices

Dadas:

```
octave> A=[1,2;3,4]
A =
  1  2
  3  4
octave> B=[5,5;6,6;7,7]
B =
  5  5
  6  6
  7  7
```

Podemos obtener una matriz total uniendo las filas de ambas (esto funciona debido a que ambas tienen el mismo número de columnas.)

```
octave> [A;B]
ans =
  1  2
  3  4
  5  5
  6  6
  7  7
```

Para unir las columnas de ambas, es lógico que el número de filas debe ser el mismo:

```
octave> [A,B]
error: number of rows must match (3 != 2) near line 14, column 4
```

Pero podemos unir A con la (conjugada) traspuesta de B:

```
octave> [A,B']
ans =
  1  2  5  6  7
  3  4  5  6  7
```

En resumen, con $[X, Y]$ unimos las matrices a lo largo de las columnas; con $[X; Y]$ se unen a lo largo de sus filas.

La función `zeros()` proporciona una matriz de ceros:

```
octave> zeros(3,7)
ans =
  0  0  0  0  0  0  0
  0  0  0  0  0  0  0
  0  0  0  0  0  0  0
```

La función `eye()` proporciona una matriz identidad:

```
octave> eye(3)
ans =
```

```

Diagonal Matrix
1 0 0
0 1 0
0 0 1

```

Podemos combinar lo anterior así (notar la unión de tres matrices):

```

octave> [[A,B',A];zeros(3,7)]
ans =
1 2 5 6 7 1 2
3 4 5 6 7 3 4
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

```

Otro ejemplo:

```

octave> Q=ones(4)
Q =
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
octave> R=rand(3,2)
R =
0.391289 0.853896
0.534633 0.739454
0.052724 0.515341
octave> Q(1:2,:)
ans =
1 1 1 1
1 1 1 1
octave> [Q(1:2,:),R']
ans =
1.000000 1.000000 1.000000 1.000000 0.391289 0.534633 0.052724
1.000000 1.000000 1.000000 1.000000 0.853896 0.739454 0.515341

```

1.8 Tipos de datos numéricos

Usualmente Octave utiliza números de punto flotante de doble precisión (también conocido como “double” en lenguaje C) que es la representación usual para los números reales en cálculo científico. Ocasionalmente puede ser conveniente forzar el uso de otros tipos de datos más restringidos:

```

octave> R=(100*(rand(4,4).*rand(4,4))).^2
R =
1.6480e+02 2.9748e+00 7.1926e+00 4.1811e+01
5.6098e+03 8.5593e+01 2.3651e+00 3.2293e+02
3.5274e+01 9.6808e+01 2.1813e+03 2.5410e+03

```

```

3.4960e+02 7.5738e+00 1.2520e-02 6.1105e+03
octave> RI8=int8(R)
RI8 =
  127  3   7  42
  127 86   2 127
   35 97 127 127
  127  8   0 127
octave> RI16=int16(R)
RI16 =
  165   3   7  42
 5610 86   2 323
   35  97 2181 2541
  350   8   0 6111
octave> RI32=int32(R)
RI32 =
  165   3   7  42
 5610 86   2 323
   35  97 2181 2541
  350   8   0 6111

```

El tipo se conserva:

```

octave> class(RI16)
ans = int16
octave> 2.3+RI16
ans =
  167  5   9  44
 5612 88   4 325
   37 99 2183 2543
  352 10   2 6113

```

Aquí se aprecia que el valor 2.3 (todas las constantes numéricas son `double` por omisión) es promovido al nivel inferior del otro operando⁸ (en este caso, `int16`.)

Para forzar a `double`:

```

octave> 2.3+double(RI16)
ans =
  167.3000  5.3000   9.3000  44.3000
 5612.3000 88.3000   4.3000 325.3000
   37.3000 99.3000 2183.3000 2543.3000
  352.3000 10.3000   2.3000 6113.3000

```

Octave opera internamente en doble precisión, pero los resultados se muestran en precisión simple por omisión, por ejemplo:

```

octave> a=rand(2,3);

```

⁸Esto difiere de muchos lenguajes de programación -como el C y relacionados- que promueven hacia el tipo de datos de mayor capacidad de los operandos.

```
octave> a
a =
  0.67448 0.71599 0.48106
  0.17609 0.97179 0.65884
```

Para mostrar los valores en doble precisión:

```
octave> format long
octave> a
a =
  0.674481809361969 0.715987465166574 0.481057898720585
  0.176085436236263 0.971787720305162 0.658836932094040
octave> format short
octave> a
a =
  0.67448 0.71599 0.48106
  0.17609 0.97179 0.65884
```

Para eliminar variables de la memoria utilizamos “clear”⁹. El comando “whos” permite apreciar las variables actualmente definidas:

```
octave> clear a
octave> a
error: `a' undefined near line 17 column 1
octave> whos
Variables in the current scope:
Attr Name Size Bytes Class
==== =====
      M   7x1     56 double
      NT  7x1     56 double
      T   7x1     56 double
      ans 1x30     30 char
Total is 51 elements using 198 bytes
```

1.9 Tamaño de las Matrices

Para obtener las dimensiones de una matriz utilizamos la función `size()`:

```
octave> N=ones(2,3)
N =
  1 1 1
  1 1 1
octave> size(N)
ans =
  2 3
octave> length(N)
ans = 3
```

⁹clear sin argumentos elimina todas las variables de memoria.

```

% numero de filas
octave> size(N,1)
ans = 2

% numero de columnas
octave> size(N,2)
ans = 3

```

Notar que `length()` devuelve la dimensión de mayor extensión¹⁰.
El rango de una matriz se puede obtener con `rank()`:

```

octave> A=[2,3,4;4,6,8;4,9,12]
A =
 2 3 4
 4 6 8
 4 9 12
octave> rank(A)
ans = 2
octave> fprintf("El rango de A es %d\n", rank(A));
ans = 2

```

En el ejemplo mostrado, la segunda fila es igual al doble de la primera, por lo que éstas no son linealmente dependientes (lo que implica que el rango debe ser menor que tres.)

Observar también que hemos introducido la función de impresión de texto formateado `fprintf()`, inspirada en el lenguaje C¹¹. Una función similar corresponde a la función `disp()` que permite mostrar un único valor a la vez:

```

octave> disp('El rango de A es'); disp(rank(A));
El rango de A es
2

```

De otro lado, las funciones `min()` y `max()` calculan los valores mínimo y máximo respectivamente de un vector¹². Para una matriz, encuentra los

¹⁰Octave proporciona las funciones `rows()` y `columns()` equivalentes a `size(:,1)` y `size(:,2)` respectivamente; éstas no deben emplearse si se busca compatibilidad con Matlab

¹¹La función `fprintf()` de Octave y Matlab subsume la funcionalidad de `printf()` y `fprintf()` del lenguaje C. Octave también soporta la función `printf()`, no disponible en Matlab.

¹²Con la palabra “vector” aquí nos referimos simplemente a una “tupla”, y más precisamente a una matriz de $N * 1$ o de $1 * N$. Continuaremos con este uso informal de “vector como tupla”, puesto que este documento está más orientado hacia la ingeniería que a las ciencias matemáticas.

valores mínimos y máximos de las columnas (y devuelve un vector fila con estos valores.) Por ejemplo:

```
octave> min(A)
ans =
2 3 4
octave> max(A)
ans =
4 9 12
octave> max(max(A))
ans = 12
```

Para ordenar los elementos de un vector utilizamos `sort()`. En una matriz, ordena los elementos de cada columna:

```
octave> B=int8(10*rand(5,6))
B =
7 7 7 7 4 9
4 6 8 4 2 2
8 4 1 8 2 1
9 10 9 0 7 2
4 9 2 2 6 10
octave> sort(B)
ans =
4 4 1 0 2 1
4 6 2 2 2 2
7 7 7 4 4 2
8 9 8 7 6 9
9 10 9 8 7 10
octave> sort(B(1,:))
ans =
4 7 7 7 7 9
octave> sort(B(1,:), 'descend')
ans =
9 7 7 7 7 4
```

Octave proporciona sentencias de control de flujo típicas de otros lenguajes de programación. Por ejemplo el loop `while...end`:

```
octave> f = 1; while f <= 5 ; fprintf('F es %g\n', f) ; f = f + 1; end
F es 1
F es 2
F es 3
F es 4
F es 5
```

Imprimir las filas de una matriz una a la vez, mediante la conversión de cada fila en una “cadena de texto”, esta vez con un loop `for...end`:

```
octave> A=magic(3)
A =
```

```

      8   1   6
      3   5   7
      4   9   2
octave> for i=[1:3] ; fprintf("Fila %d -> %s\n", i, num2str(A(i,:))) ; end
Fila 1 -> 8   1   6
Fila 2 -> 3   5   7
Fila 3 -> 4   9   2

```

Ejercicio

1 Resolver el sistema de ecuaciones lineales

$$89x - 21y - 31z = 421$$

$$12.3x + 91z = 12$$

$$-x - y + z = 94$$

Solución: Formamos las matrices A y B:

```

octave> A=[89,-21,-31 ; 12.3,0,91 ; -1,-1,1]
A =

```

```

      89.00000   -21.00000   -31.00000
      12.30000     0.00000    91.00000
      -1.00000   -1.00000     1.00000

```

```

octave> B=[421;12;94]
B =

```

```

      421
       12
       94

```

Obtenemos la solución:

```

octave> A\B
ans =

```

```

    -13.2117
   -78.8707
     1.9176

```

Otra forma menos eficiente:

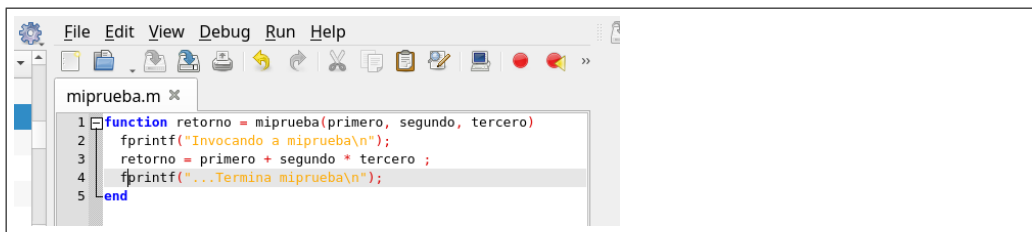
```

octave> inv(A)*B
ans =
    -13.2117   -78.8707     1.9176

```


2 Definiendo y utilizando funciones

A continuación definimos una función denominada `miprueba()` que recibe tres argumentos de entrada, los cuales permiten obtener un resultado (valor de retorno de la función.) A diferencia de muchos lenguajes de programación que proporcionan una sentencia “de retorno” para este fin¹³, aquí los valores se retornan asignándolos a ciertas variables señaladas en la definición de la función (en nuestro ejemplo es la que denominamos “`retorno`”.) La función debe ser escrita en un archivo que tiene su mismo nombre y extensión “.m” (para este ejemplo, será `miprueba.m`), el cual debe ubicarse en el directorio de trabajo desde donde se ejecuta Octave. Esta escritura puede hacerse desde el entorno gráfico de Octave, que proporciona un editor incorporado con “syntax highlighting”:



Como se aprecia, contenido de la función corresponde a un bloque encerrado entre `function` y `end`¹⁴. Una vez grabado este archivo, la función se puede ejecutar desde Octave:

```
octave> miprueba(1,2,3)
Invocando a miprueba
...Termina miprueba
ans = 7
```

Asignando el resultado a una variable:

```
octave> a = miprueba(1,2,3);
Invocando a miprueba
...Termina miprueba
octave> a
a = 7
```

¹³En Octave también se dispone de una sentencia “`return`” (sin argumentos) que termina inmediatamente la ejecución de la función, retornándose el contenido de la “variable de retorno”.

¹⁴Octave proporciona la posibilidad de emplear “`endfunction`” para hacer más explícito que se está terminando el cuerpo de una función; asimismo, el cierre de un bucle “`for`” se puede señalar con “`endfor`”, el cierre de un bloque “`if`” con “`endif`”, etcétera. En todos estos casos Matlab sólo acepta `end`.

Nota para usuarios en modo consola: Si se emplea el modo consola, es usual que el usuario utilice su propio editor de texto, lo que se suele hacer con ayuda de una sesión adicional a la que se usa para Octave. Si esto no es posible (cosa muy improbable en los computadores modernos), el editor también puede desplegarse mediante el comando `edit` de Octave. En tal caso puede ser necesario configurar el editor a ser desplegado, lo que se consigue creando o modificando el archivo `$HOME/.octaverc`. Por ejemplo, para lanzar el editor `vi`:

```
edit editor "vi %s"
```

Menos frecuente es el uso de un editor externo en entorno gráfico, el cual se debería lanzar asincrónicamente a la sesión Octave (como una tarea independiente que no suspende la ejecución de este último):

```
edit editor "mousepad %s"  
edit mode "async"
```

En modo consola también suele ser útil el comando `type`, que permite visualizar el texto completo de un archivo sin recurrir a un editor externo.

Recalamos que estas indicaciones no son necesarias para los usuarios que emplean Octave en su entorno gráfico usual.

2.1 Funciones con argumentos variables

Esta sección puede obviarse en una primera lectura. A nuestro criterio, son contados los casos en los que esta sintaxis es beneficiosa versus la complejidad adicional que conlleva.

Para máxima flexibilidad en los argumentos “de entrada”, utilizar la variable especial `varargin`:

```
function retorno = prueba(varargin)  
    v = [varargin{:}];  
    retorno(1) = sum(v);  
    retorno(2) = prod(v);  
end
```

`varargin` es un “cell array”, es decir una colección que soporta valores de diverso tipo. Con `{:}` se obtiene una lista de todos los valores, y con `[]` se crea una matriz de esta lista.

```
octave> prueba(3,4)  
ans =  
    7    12
```

La variable especial `nargin` se puede utilizar para validar que el usuario ha introducido cierto número mínimo de parámetros. El siguiente ejemplo permite generalizar la función `kron()` para ejecutar el producto tensorial de un número arbitrario de matrices:

```
function r=gkron(varargin)
    r=varargin{1};
    for z = [2:nargin]
        r=kron(r,varargin{z});
    end
end
octave> A=kron(kron(magic(2),eye(2)),magic(2))
A =
    16    12     0     0    12     9     0     0
     4     8     0     0     3     6     0     0
     0     0    16    12     0     0    12     9
     0     0     4     8     0     0     3     6
     4     3     0     0     8     6     0     0
     1     2     0     0     2     4     0     0
     0     0     4     3     0     0     8     6
     0     0     1     2     0     0     2     4

octave> B=gkron(magic(2),eye(2),magic(2))
B =
    16    12     0     0    12     9     0     0
     4     8     0     0     3     6     0     0
     0     0    16    12     0     0    12     9
     0     0     4     8     0     0     3     6
     4     3     0     0     8     6     0     0
     1     2     0     0     2     4     0     0
     0     0     4     3     0     0     8     6
     0     0     1     2     0     0     2     4

octave> isequal(A,B)
ans = 1
```

2.2 Número variable de valores de retorno

Consideramos esta funcionalidad algo inusual,¹⁵ pero se usa ampliamente en Octave (y Matlab.) El usuario puede solicitar un número variable de valores de respuesta en el momento de la invocación mediante una sintaxis como esta:

¹⁵Al menos si se compara con la mayoría de lenguajes de propósito general donde la función únicamente opera en base a los argumentos de entrada, siguiendo una estructura más fiel a la definición de las funciones matemáticas.

```
[x,y,z] = f(...) # solicita 3 argumentos de salida
[x,y] = f(...) # solicita sólo 2
```

Para controlar esta situación la función puede emplear la variable especial `nargout`. Ejemplo:

```
function [ a,b,c,d ] = prueba3 ()
    fprintf("Se solicitado %d valores\n", nargout);
    a=3;
    b=4;
    c=5;
    d=6;
end
```

```
octave> prueba3
Se solicitado 0 valores
ans = 3
octave> [x,y] = prueba3
Se solicitado 2 valores
x = 3
y = 4
octave> [x,y] = prueba3;
Se solicitado 2 valores
octave> [x,y,z,w,k] = prueba3
Se solicitado 5 valores
x = 3
y = 4
z = 5
w = 6
error: element number 5 undefined in return list
octave> [x,y,z,w] = prueba3
Se solicitado 4 valores
x = 3
y = 4
z = 5
w = 6
```

2.3 Introducir comentarios y proporcionar ayuda

En Octave existen distintas formas de introducir comentarios¹⁶, pero la forma más portable (con Matlab) consiste en añadir líneas que se inician con un signo de porcentaje (%). Un grupo ininterrumpido de líneas de comentario se considera un bloque de comentarios.

¹⁶Ver la documentación oficial de Octave más detalles; al momento de escribir estas líneas, esta información se encuentra en <https://docs.octave.org/v8.2.0/Comments.html>.

El primer bloque de comentarios dentro de una función se considera “la ayuda” de ésta; por ejemplo:

```
function [ ret ] = prueba_ayuda (x,y)
% Utilizar: prueba_ayuda(x,y)
%
% Esta funcion permite hallar la media armonica de dos
% numeros

% Aqui mas comentarios de un segundo bloque
ret = 2*x*y/(x+y);

end
```

luego, esta ayuda puede ser obtenida mediante el comando “help”:

```
octave> help prueba_ayuda
`prueba_ayuda' is a function from the file /home/diego/octave/prueba_ayuda.m

Utilizar: prueba_ayuda(x,y)

Esta funcion permite hallar la media armonica de dos
numeros
```

Ejemplo: redondeo de valores Es frecuente la necesidad de redondear los resultados de las operaciones. A tal efecto se puede emplear una función de redondeo a “*n*” dígitos, que se puede construir a partir de la función `round()` de Octave; esto se puede combinar con el comando `format` para controlar más aspectos de la salida (como la notación exponencial, etc.)

```
octave> type r2.m
function ans=r2(x)
ans=round(x*100)/100;
end
```

```
octave> 1./magic(4)
ans =
    0.062500    0.500000    0.333333    0.076923
    0.200000    0.090909    0.100000    0.125000
    0.111111    0.142857    0.166667    0.083333
    0.250000    0.071429    0.066667    1.000000
```

```
octave> format shortg
```

```
octave> 1./magic(4)
ans =
    0.0625         0.5         0.33333         0.076923
         0.2         0.090909         0.1         0.125
    0.11111         0.14286         0.16667         0.083333
```

```

0.25      0.071429      0.066667      1
octave> r2(1./magic(4))
ans =
0.06      0.5      0.33      0.08
0.2      0.09      0.1      0.13
0.11      0.14      0.17      0.08
0.25      0.07      0.07      1

```

Ejercicio

Proporcionar una secuencia de 10000 números aleatorios binarios a partir de `rand()`. Aplicar los tres primeros tests de calidad recomendados por NIST/CSRC¹⁷:

a) Sesgo binario Dada la secuencia de números binarios pseudo-aleatoria e_1, e_2, \dots, e_n evaluar los valores $X_i = 2e_i - 1$ (que produce +1 y -1)

Evaluar

$$S = \frac{\sum X_i}{\sqrt{n}}$$

Evaluar la función de error complementario

$$P = \operatorname{erf}\left(\frac{S}{\sqrt{2}}\right)$$

Si $P < 0.01$ entonces se concluye que la secuencia no es aleatoria.

b) Test de frecuencia por bloque Dada la secuencia de números binarios pseudo-aleatoria e_1, e_2, \dots, e_n y un tamaño de bloque “ M ”, se particiona la secuencia en $N = \lceil n/M \rceil$ bloques sin “overlapping”, descartándose los elementos no utilizados¹⁸.

Determinar la proporción J_i de “unos” dentro de cada M-bloque usando la ecuación:

¹⁷Tomado de “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications” (NIST/CSRC) versión 2008; existe una versión actualizada en <https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final>. El que una secuencia de números pseudoaleatorios apruebe estos tests no garantiza su calidad; estos tests sólo funcionan en modo negativo: permiten detectar ciertas secuencias presumiblemente “de mala calidad”.

¹⁸ $\lceil x \rceil$ se refiere al máximo entero, implementado por la función `floor()` de Octave.

$$J_i = \frac{\sum_{j=1}^M e_{i-1}M + j}{M}$$

para $i = 1 \dots N$.

Se calcula

$$\chi^2 = 4M \sum_{i=1}^N \left(J_i - \frac{1}{2}\right)^2$$

para $i = 1 \dots N$.

Calcular $P = igamc\left(\frac{\chi^2}{2}, N/2\right)$, donde $igamc(x, a)$ se obtiene de la función gamma incompleta:

$$igamc(x, a) = 1 - gammainc(x, a)$$

Si $P < 0.01$, la secuencia no es aleatoria.

c) Tests de “carreras” Una “carrera” de longitud k consiste de k bits idénticos y está precedida y sucedida por bits de valor opuesto. Dados e_1, e_2, \dots, e_n se obtiene la proporción:

$$J = \frac{\sum_{j=1}^n e_j}{n}$$

Se calcula los valores $r(k)$ con $k \in [1..(n-1)]$, donde $r(k) = 0$ si $e(k) = e(k+1)$ y $r(k) = 1$ en caso contrario.

$$V_n = 1 + \sum_{k=1}^{n-1} r(k)$$

Se obtiene el valor P :

$$P = \operatorname{erfc} \left(\frac{|V_n - 2nJ(1-J)|}{2\sqrt{2nJ(1-j)}} \right)$$

Si $P < 0.01$, la secuencia no es aleatoria.

Solución: Sea N un conjunto de números obtenidos con `rand()` como en los ejemplos:

```
octave> N=rand(10000,1);
```

Obtenemos datos binarios:

```
octave> E=N>0.5;
```

a) Crearemos la función `rfrequency()` que recibe estos valores:

```
function [ret] = rfrequency(E)
    X=2*E-1;
    suma = sum(X);
    S=abs(suma)/sqrt(size(E,1));
    P=erfc(S/sqrt(2));
    ret = P;
end
```

```
octave> rfrequency(E)
ans = 0.40091
```

Como $P > 0.01$ entonces no descartamos la aleatoridad de la secuencia.

b) Creamos la función `BlockFrequency(E,M)` que recibe la secuencia y el tamaño de bloque. El aspecto más importante de esta implementación consiste en la creación de la matriz EM que contiene todos los N intervalos en columnas de M filas. Para esto empleamos `reshape()`, sin embargo, esta función requiere que la fuente de los datos sea exactamente de tamaño $M * N$, por lo que previamente obtenemos el vector E2 con esta cantidad de elementos, posiblemente descartándose los del final de E.

```
function [ ret ] = block_frequency (E,M)
    n = size(E, 1);
    N = floor(n/M);
    E2 = E(1:N*M);
    EM = reshape(E2,M,N);
    j = sum(EM)/M;
    CHI2 = 4 * M * sum((j' - 1/2).^2);
    ret = 1 - gammainc(CHI2 / 2, N / 2);
end
```

Probaremos la función con $M = 150$ (el documento de NIST aconseja $M > 0.01 * n$):

```
octave> BlockFrequency(E,150)
ans = 0.16724
```

como $P > 0.01$ entonces la aleatoridad de la secuencia no se descarta.

Una forma alternativa menos eficiente pero que puede ser más flexible y simple consiste en iterar los N intervalos (en lugar de crear la matriz auxiliar con `reshape`):

```
function [ ret ] = block_frequency2 (E,M)
    n = size(E, 1);
    N = floor(n/M);
    for i = [1:N]
        j(i) = sum(E([(i - 1) * M + 1: (i - 1) * M + M]))/M;
    end
    CHI2 = 4 * M * sum((j - 1/2).^2);
    ret = 1 - gammainc(CHI2 / 2, N / 2);
end
```

c) Crearemos la función `Runs(E)` que recibe la secuencia.

Aquí el paso clave consiste en aprovechar la no-igualdad de elementos (operador `~=`) para obtener unos y ceros, evitando un loop. Para esto creamos dos matrices desplazadas en un elemento extremo al inicio y al final: $[E; 0]$ y $[0; E]$. El resultado produce una matriz de $(n + 1)$ elementos, descartándose los elementos extremos (tomamos desde “2” hasta “n”).

```
function [ ret ] = Runs (E)
    n = size(E, 1);
    J=sum(E)/n;
    rextra = [E;0]!=[0;E];
    r=rextra(2:size(E,1));
    Vn=sum(r)+1;
    P=erfc(abs(Vn-2*n*J*(1-J))/(2*sqrt(2*n)*J*(1-J)));
    ret = P;
end
```

```
octave> Runs(E)
ans = 0.66503
```

como $P > 0.01$ la aleatoridad de la secuencia no se descarta.

3 Gráfico de funciones

3.1 Ploteo de valores

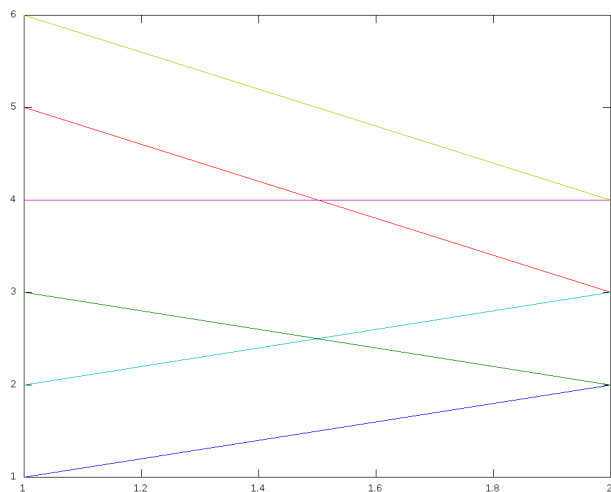
Una lista de valores puede ser trazada a lo largo del eje Y , utilizando los números naturales como eje X . Dicho de otro modo, si se desea dibujar una matriz, los valores se representan en el eje Y , mientras que los índices de éstos se representan en el eje X .

Los valores de una matriz a trazarse normalmente deben ubicarse a lo largo de columnas. En el siguiente ejemplo, `plot` asume seis series de valores, cada una de sólo dos elementos:

```
octave> A=[1,3,5,2,4,6;2,2,3,3,4,4]
A =
```

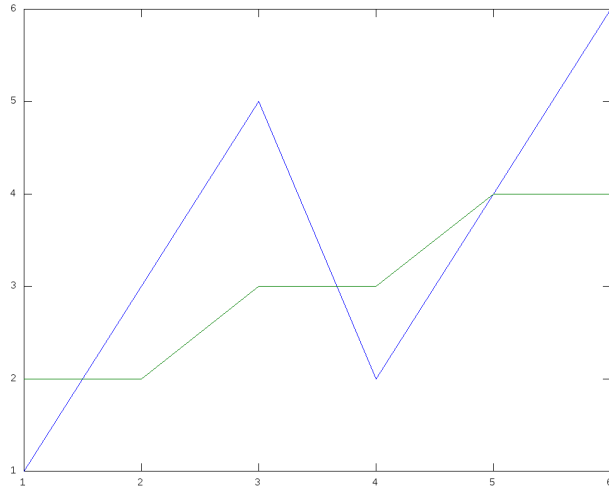
```
  1  3  5  2  4  6
  2  2  3  3  4  4
```

```
octave> plot(A)
```



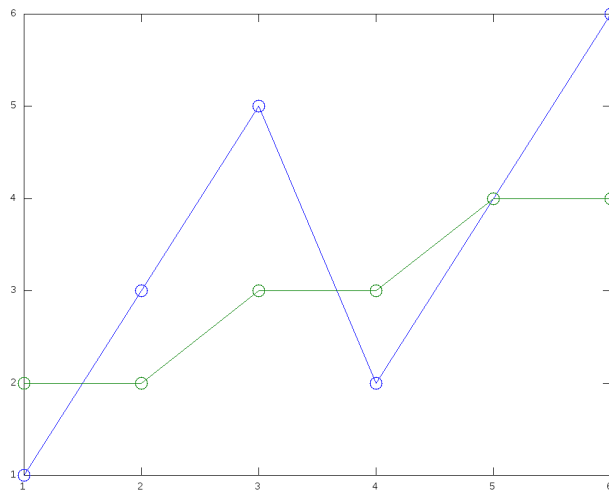
Más frecuente es el ploteo de muchos valores en pocas series:

```
octave> plot(A')
```



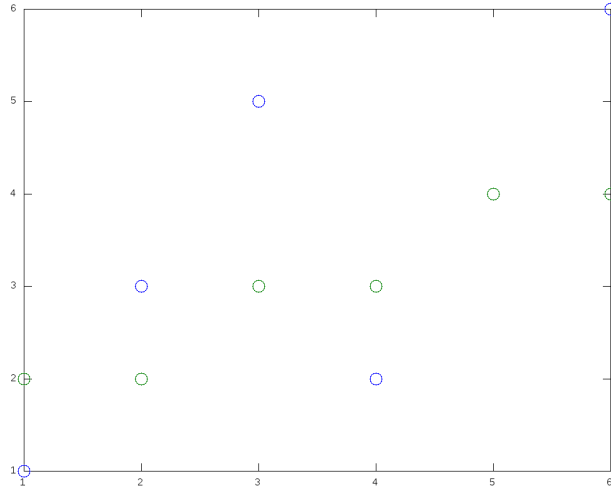
El argumento textual que sigue a los valores determina el formato del gráfico. En este caso es de líneas ("-") y tiene señalización con círculos ("o"). A continuación es posible agregar propiedades, por ejemplo, el tamaño de la señalización (markersize.)

```
octave> plot(A', "-o", "markersize", 20);
```



Eliminar las líneas:

```
octave> plot(A', "o", "markersize", 20);
```



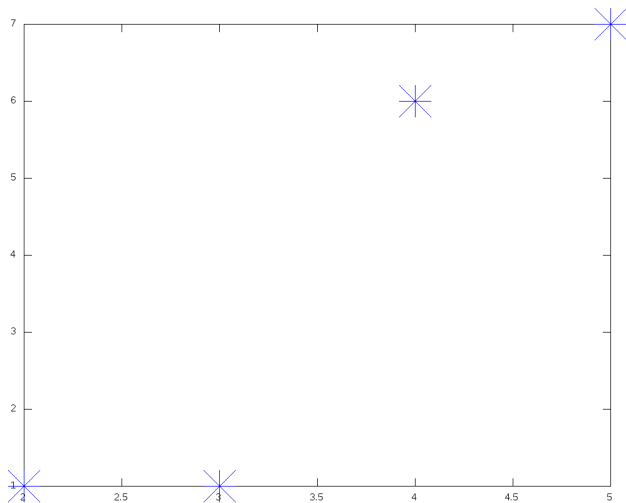
Los gráficos se asocian a una “figura”, que puede considerarse una “ventana física”; por omisión se utiliza la número “1”. Si se hace otro dibujo (por ejemplo con `plot`) se reemplazará la figura actual. Para generar varios gráficos en simultáneo, se requiere activar más figuras utilizando `figure()`. Para grabar la figura actual se puede utilizar `print(‘nombre.png’)`; para grabar la figura “n”, usar `print(n, ‘nombre.png’)`; diversas extensiones y formatos de archivo gráfico son válidas.

3.2 Ejes

Partimos del gráfico de algunos puntos sueltos:

```
octave> A=[2,3,4,5]';
octave> B=[1,1,6,7]';
octave> plot(A,B,'.*',"markersize", 50)
```

Obtenemos puntos que están sobre los ejes y se hacen difíciles de visualizar:

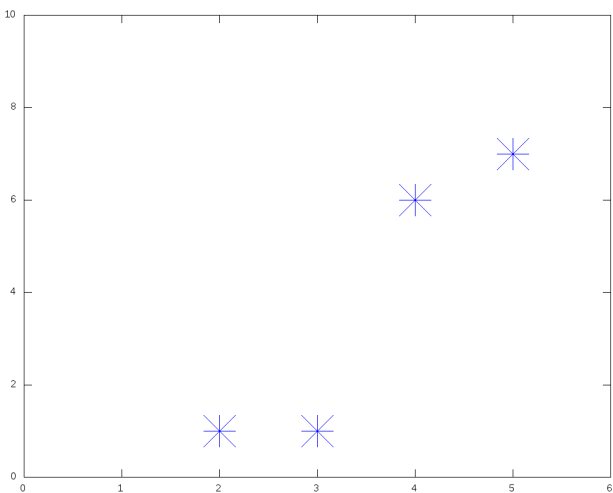


En el ejemplo, hemos forzado esto con un “`markersize`” muy elevado, de lo contrario los puntos serían casi indistinguibles.

Podemos mejorar esto definiendo el rango para los ejes de coordenadas a visualizar utilizando “`axis`”; por ejemplo:

```
octave> axis([0,6,0,10])
```

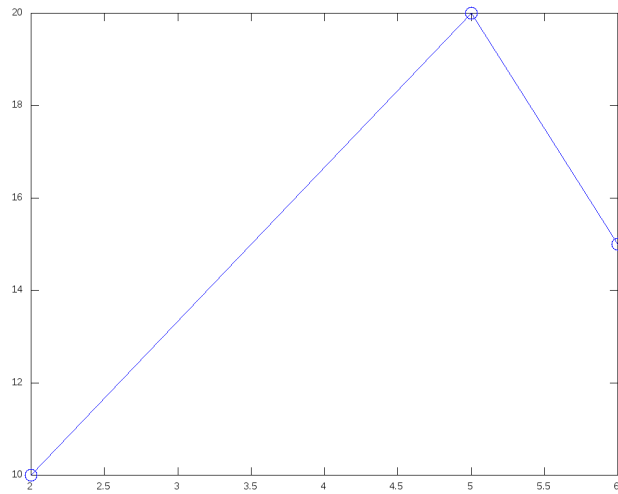
Permite obtener el siguiente gráfico:



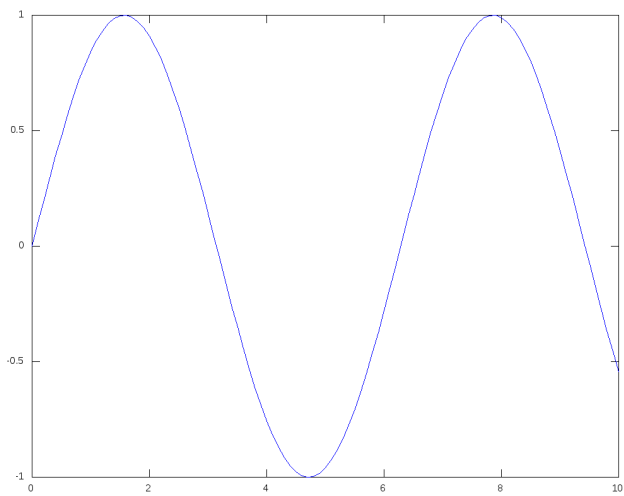
3.3 Ploteo de funciones

Es similar al anterior pero el eje X se fija a los valores de un dominio.

```
octave> plot([2;5;6],[10;20;15],"-o","markersize",20);
```

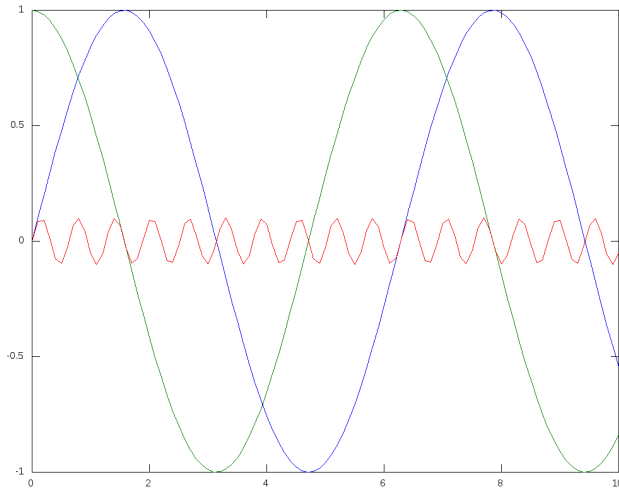


```
octave> x=[0:0.1:10]';  
octave> y=sin(x);  
octave> plot(x,y);
```



Como siempre, las columnas determinan series de valores...

```
octave> plot(x,[sin(x),cos(x),sin(10*x)/10]);
```



3.4 Grafico de varias series de datos

Veamos otro ejemplo: se tiene un sistema de procesamiento de formularios en batch. Se ha hecho pruebas de funcionamiento controlándose el tiempo y el consumo de memoria del proceso para diversos tamaños del lote de datos de entrada. Los resultados se muestran en la tabla siguiente:

Tamaño de muestra	Tiempo de proceso	Memoria consumida (mb)
5000	50s	80
10000	20s	120
20000	50s	165
40000	1m 20s	310
80000	2m 40s	710
120000	3m 50s	900
150000	5m	1200

Se sabe que este proceso puede en el futuro requerir operar con lotes más grandes, por lo que se desea obtener un gráfico que ilustre la tendencia en el tiempo y consumo de memoria.

Para esto, creamos las matrices:

- TM = Tamaño de muestra en millares

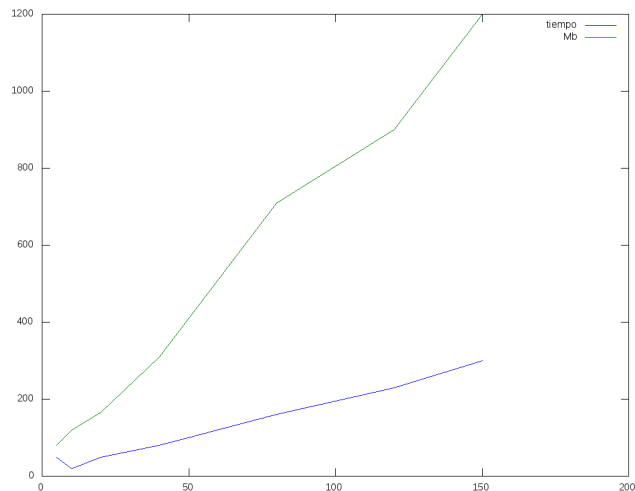
- T = Tiempo en segundos
- M = Memoria en Mb

```
octave> TM=[5; 10; 20; 40; 80; 120; 150];
octave> T=[50 ; 20; 50; 60 + 20; 2*60 + 40; 3*60+50; 5*60];
octave> M=[80; 120; 165; 310; 710; 900; 1200];
```

Finalmente lanzamos el ploteo como dos series de datos con la leyenda respectiva¹⁹:

```
octave> plot (TM, T, '-b', TM, M, '-g');
octave> legend('tiempo', 'Mb');
```

El resultado:



3.5 Gráficos tridimensionales

Malla de dominio (mesh) Los gráficos tridimensionales son esencialmente de dos tipos: líneas y superficies. Para el último caso, es conveniente comprender la malla de dominio (x, y) que permite llevar a cabo la graficación.

Considérese una función $z = z(x, y)$ graficada como superficie. La estrategia de graficación de Octave consiste en aproximar esta superficie mediante

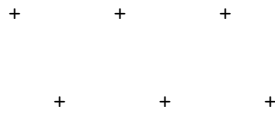
¹⁹Octave soporta una sintaxis alternativa para incluir las leyendas como parte de plot; para este caso correspondería a: `plot(TM, T, '-b;tiempo;', TM, M, '-g;Mb;')`.

una “malla” de puntos representativos (más puntos en la malla, más aproximación.)

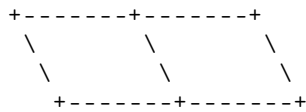
Comoquiera que los puntos z son función de los pares (x, y) , la estructura de la malla se define mediante el conjunto de dichos pares.

Debemos tener en cuenta que la malla consiste no tanto de los puntos (x, y) sino de las interconexiones de éstos, del mismo modo que una red se compone de pequeñas “celdas”.

Lo último significa que no es suficiente disponer de un conjunto de pares sueltos (x, y) , sino de información complementaria acerca de la mutua cercanía entre éstos. De este modo se consigue pasar de un conjunto de vértices con esta estructura:



A una malla con esta estructura:



Para esto Octave exige que los elementos “ x ” y los elementos “ y ” se dispongan separadamente en matrices rectangulares A y B de las mismas dimensiones. Para una misma posición (i, j) , el punto (x, y) es $(A(i, j), B(i, j))$.

Gracias a esta representación, Octave puede deducir los puntos (x, y) mutuamente “adyacentes”, simplemente a través del carácter adyacente de las posiciones (i, j) de las matrices de elementos.

Octave proporciona la función “`meshgrid`” que permite obtener rápidamente las matrices A y B . Por ejemplo, considérese el rango x en $[1, 4]$, e y en $[5, 7]$. Tenemos:

```

octave> [A,B]=meshgrid([1:4],[5:7])
A =

    1    2    3    4
    1    2    3    4
    1    2    3    4

B =

    5    5    5    5
    6    6    6    6
  
```

```
7 7 7 7
```

Nuestra malla es de dimensión 3×4 y tiene por tanto 12 puntos.

Los puntos generados pueden graficarse fácilmente con `plot`:

```
octave> [A,B]=meshgrid([1:4],[5:7])
```

```
A =
```

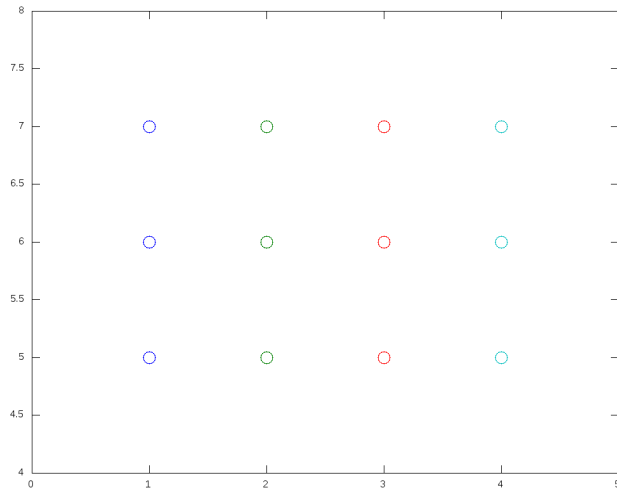
```
1 2 3 4
1 2 3 4
1 2 3 4
```

```
B =
```

```
5 5 5 5
6 6 6 6
7 7 7 7
```

```
octave> plot(A,B,'o',"markersize", 20)
```

```
octave> axis([0,5,4,8])
```



Pero lo interesante es utilizar estos puntos como base para una superficie. Partiremos de una superficie plana con $z = 1$:

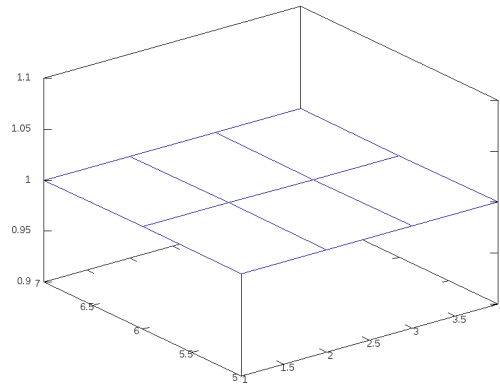
```
octave> Z=ones(size(A))
```

```
Z =
```

```
1 1 1 1
1 1 1 1
1 1 1 1
```

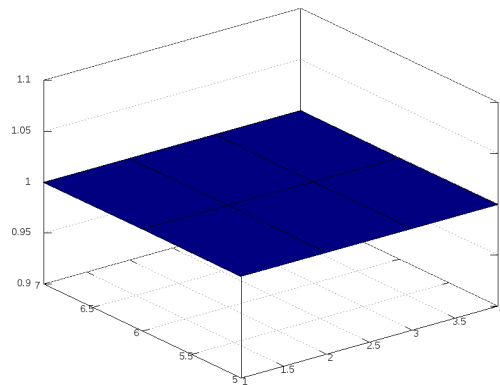
```
octave> mesh(A,B,Z)
```

Con `mesh()` obtenemos la malla tridimensional:



O mediante `surf()`, la malla aparece coloreada (aquí de un único color dado que “Z” es constante):

```
octave> surf(A,B,Z)
```



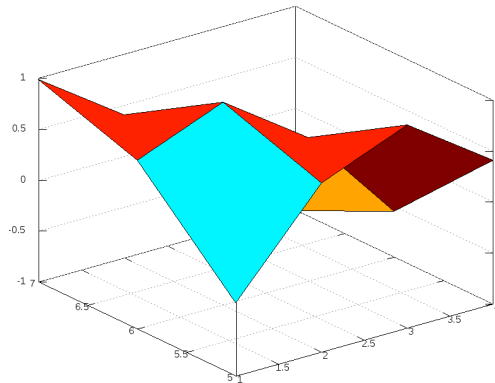
Una función un poco más compleja ilustra la necesidad de una malla más

fina:

```
octave> Z=sin(A+B)
Z =
```

```
 -0.27942    0.65699    0.98936    0.41212
  0.65699    0.98936    0.41212   -0.54402
  0.98936    0.41212   -0.54402   -0.99999
```

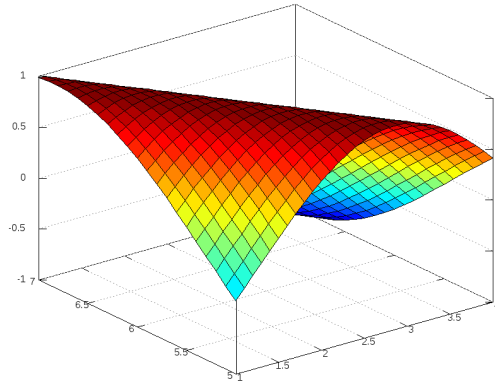
```
octave> surf(A,B,Z)
```



Lo cual se consigue fácilmente mediante un paso más fino en el dominio:

```
octave> [A,B]=meshgrid([1:0.1:4],[5:0.1:7]);
octave> Z=sin(A+B);
octave> surf(A,B,Z)
```

El resultado proporciona información más útil:

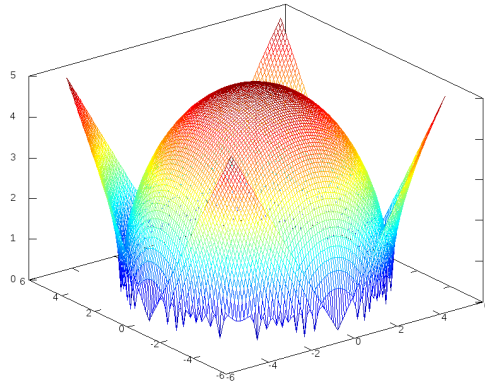


3.6 Dominios más sofisticados

En ocasiones no es conveniente utilizar un dominio rectangular (x, y) como el definido anteriormente. Por ejemplo, para graficar un hemisferio $Z = \sqrt{R^2 - x^2 - y^2}$ podríamos tener problemas:

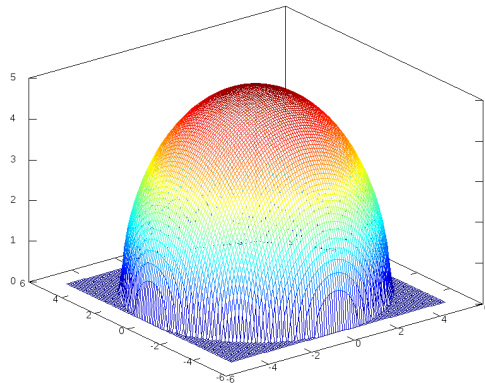
```
octave> [A,B]=meshgrid([-5:0.1:5],[-5:0.1:5]);
octave> Z=sqrt(25-A.^2-B.^2);
octave> mesh(A,B,Z)
error: octave_base_value::array_value(): wrong type argument `complex matrix'
```

Una solución inmediata consiste en tomar el valor absoluto, con lo que obtenemos una extraña figura:



Otra solución consiste en considerar sólo la parte real de Z :

```
octave> mesh(A,B,real(Z))
```



En este caso se aprecia como defecto la malla plana en $Z = 0$ que corresponde a los valores negativos del radicando (que en rigor no son parte del gráfico), y que siendo imaginarios puros resultan tener una parte real igual a cero.

Podríamos eliminar los valores complejos utilizando el valor especial NaN

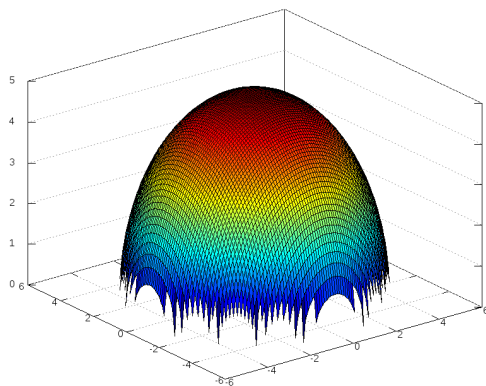
(Not a Number), el cual evitará el gráfico de ciertos puntos de la malla. Encontremos primero los puntos “indeseables”, aquellos con parte imaginaria:

```
octave> INDESEABLE=(imag(Z)~=0);
```

Sus elementos “1” servirán para dehabilitar valores en una copia de nuestra matriz Z :

```
octave> ZLIMPIO=Z;
octave> ZLIMPIO(INDESEABLE)=NaN;
octave> surf(A,B,ZLIMPIO)
```

con esto obtenemos una figura sin malla en $Z = 0$, pero con problemas de continuidad en su base, debido a que la malla se ha roto sin considerar su estructura:



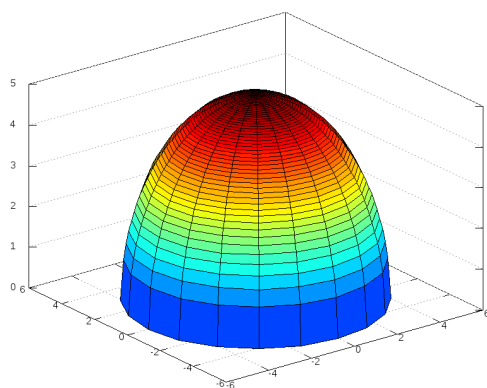
Una solución más sofisticada consiste en generar un dominio no rectangular. Para el presente caso, una representación polar puede ser más conveniente.

Para esto, redefinimos las matrices de dominio del siguiente modo:

```
octave> T=[0:pi/10:2*pi];
octave> R=[0:0.1:5];
octave> [AT,AR]=meshgrid(T,R);
octave> X=AR.*cos(AT);
octave> Y=AR.*sin(AT);
octave> Z=sqrt(25-X.^2-Y.^2);
octave> surf(X,Y,Z)
error: octave_base_value::array_value(): wrong type argument `complex matrix'
...
```

Por limitaciones de precisión algunos valores han de contener valores complejos, pero han de ser muy cercanos a cero. En tal caso, podemos considerar sólo la parte real:

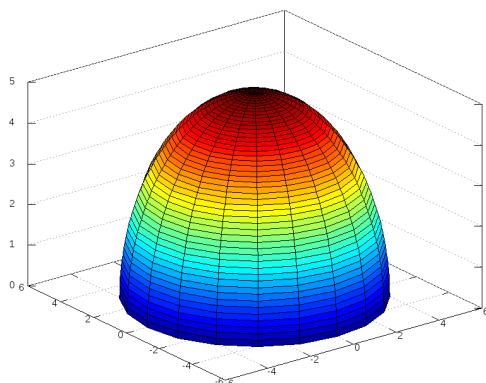
```
octave> surf(X,Y,real(Z))
```



Como se aprecia, la malla está más acorde a la forma de la superficie. Finalmente, podemos mejorar la variación del radio “ R ” a fin de evitar que en la zona superior se acumulen muchas celdas de la malla. En tanto deseamos una malla que ascienda uniformemente, podemos considerar la variación el ángulo PHI que subtiende el eje Z y cada circunferencia de la malla, desde cero a $\pi/2$. El radio R en el plano corresponde al radio de la circunferencia (5) multiplicado por el seno de PHI :

```
octave> PHI=[0:pi/100:pi/2];  
octave> R=5*sin(PHI);  
octave> [AT,AR]=meshgrid(T,R);  
octave> X=AR.*cos(AT);  
octave> Y=AR.*sin(AT);  
octave> Z=sqrt(25-X.^2-Y.^2);  
octave> surf(X,Y,real(Z))
```

con esto, el gráfico resulta mucho más satisfactorio:



Ejercicio: Graficar el clásico “sombrero” dado por la función²⁰

$$z(x, y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}}$$

en el dominio $[-8, 8] * [-8, 8]$.

Solución: El sombrero puede graficarse con:

```
[A,B]=meshgrid([-8:0.1:8],[-8:0.1:8]);
Z = sin (sqrt (A.^2 + B.^2)) ./ (sqrt (A.^2 + B.^2));
surf(A,B,Z);
```

4 Cálculo diferencial

4.1 Integración

En Octave existen diversas funciones que implementan distintos algoritmos con el fin de integrar funciones con ciertas complicaciones (como discontinuidades y singularidades.) Así se tienen `quad`, `quadv`, `quadl`, `quadgk`,

²⁰Como curiosidad, en Octave se proporciona la función “`sombrero()`”.

`quadcc`, y `trapez`;²¹; sin embargo, para máxima compatibilidad con Matlab utilizaremos `integral()`.

Como ejemplo ilustrativo, calculemos la integral:

$$\int_0^{10} \left(4e^{-x/2} + \frac{x^3}{100}\right) dx$$

Para esto definimos la función:

```
function [ ret ] = fun5 (x)
    ret = 4*exp(-x/2) + x.^3 / 100;
end
```

e invocamos a `integral()` con el intervalo $[0, 10]$:

```
octave> integral(@fun5,0,10)
ans = 32.946
```

Por integración exacta, tenemos:

```
% antiderivada de fun5
function [ ret ] = fun5_ad (x)
    ret = -8*exp(-x/2) + x.^4 / 400;
end
```

El cálculo exacto arroja:

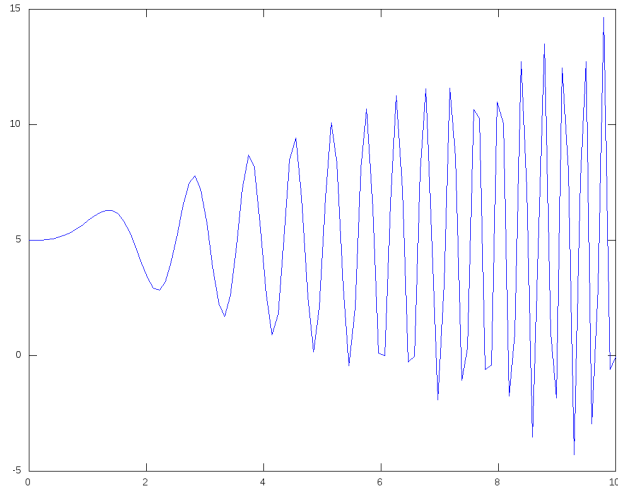
```
octave> fun5_ad(10) - fun5_ad(0)
ans = 32.946
```

Integremos una función más compleja:

```
function [ ret ] = fun6 (x)
    ret = x.*sin(x.*x) + 5;
end
```

Para el dominio $[0, 10]$, gráficamente tenemos:

²¹La documentación de cada función hace referencia al algoritmo implementado; normalmente estos algoritmos operan en forma adaptiva a fin de soportar integrandos no triviales (con discontinuidades y singularidades), así como dominios de integración infinitos.



Aplicando `integral()`:

```
octave> integral(@fun6,0,10)
ans = 50.069
```

De otro lado, encontrando su antiderivada:

```
function [ ret ] = fun6_ad (x)
    ret = -cos(x*x)/2 + 5 * x;
end
```

Obtenemos:

```
octave> fun6_ad(10) - fun6_ad(0)
ans = 50.069
```

4.2 Derivación

La derivada de una función f (diferenciable en x_0) se puede aproximar mediante:

$$f'(x_0) \approx \frac{f(x_0 + \Delta) - f(x_0)}{\Delta}$$

Esta aproximación es más exacta en la medida que Δ se haga más pequeño. Si $x_1 = x_0 + \Delta$ entonces:

$$f'(x_0) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Luego, la derivada se aproxima mediante el cociente de diferencias, para lo que empleamos “`diff()`”. Con `diff` se obtiene un vector con las diferencias relativas entre sus elementos:

$$\text{diff}([x_1; x_2; x_3; x_4 \dots x_n]) = [x_2 - x_1; x_3 - x_2; \dots x_n - x_{n-1}]$$

Por ejemplo:

```
octave> A=[4;5;6;8;10;12;15;19];
octave> diff(A)
ans =
1
1
2
2
2
3
4
```

Ejemplo Hallar y graficar la derivada de $\sin(x^2)$ para $[0, 6]$.

Obtenemos el dominio y la función:

```
octave> x=[0:0.01:6]';
octave> y=sin(x.*x);
```

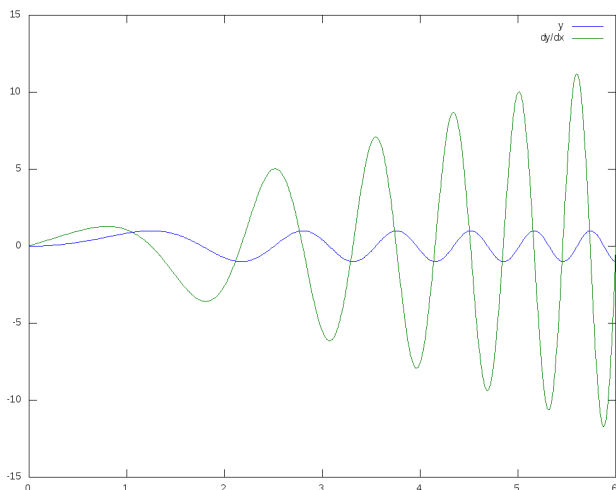
Obtenemos la derivada. Nótese que `dy1` tiene un elemento menos que “ x ” e “ y ”:

```
octave> dy1=diff(y)./diff(x);
octave> size(dy1)
ans =
600 1
octave> size(x)
ans =
601 1
```

Graficamos ambas funciones, extendiendo la derivada `dy1` mediante la repetición de su último valor:

```
octave> plot(x,y,'-',x,[dy1;dy1(end)],'-');
octave> legend('y','dy/dx');
```

El resultado:



Como ejercicio el lector agregar el gráfico de la derivada exacta: $y' = 2x \cos(x^2)$

Ejercicio: Considérese la función $f : \mathbb{R} \rightarrow \mathbb{R}$ definida en $[0, 2)$ del siguiente modo:

para $[0, 1] \rightarrow f(x) = x$

para $[1, 1 + \frac{1}{2}] \rightarrow f(x) = 2(x - 1)$

para $[1 + \frac{1}{2}, 1 + \frac{1}{2} + \frac{1}{4}] \rightarrow f(x) = 4(x - 1 - \frac{1}{2})$

para $[1 + \frac{1}{2} + \frac{1}{4}, 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}] \rightarrow f(x) = 8(x - 1 - \frac{1}{2} - \frac{1}{4})$

etc.

Proporcionar una regla explícita, graficar e integrar en el rango $[0, 1.95]$.

Solución Primero proporcionaremos una definición explícita para $f(x)$. Lo primero es hallar una fórmula más directa para la regla anterior. Utilizando las progresiones geométricas: $S = a(r^n - 1)/(r - 1)$

$$1 + \frac{1}{2} + \frac{1}{4} + \dots + 1/2^{(n-1)} = 2(1 - 1/[2^n]) = p(n)$$

así, los intervalos son:

$n = 1 \rightarrow [0, p(1)] \rightarrow f(x) = 2^0 \cdot (x)$

$n = 2 \rightarrow [p(1), p(2)] \rightarrow f(x) = 2^1 \cdot (x - p(1))$

$n = 3 \rightarrow [p(2), p(3)] \rightarrow f(x) = 2^2 \cdot (x - p(2))$

...

Para $n \rightarrow [p(n-1), p(n)] \rightarrow f(x) = 2^{(n-1)} \cdot (x - p(n-1))$
 De la expresión para $p(n)$ se puede resolver n :

$$n = -\ln(1 - p/2) / \ln(2)$$

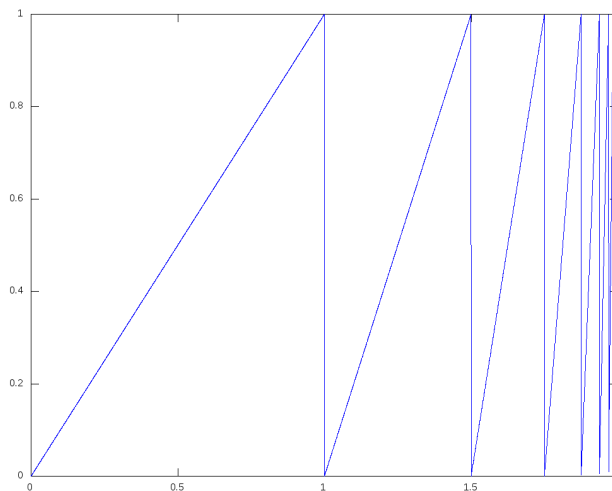
Puesto que los intervalos anteriores tienen la forma $[p(n-1), p(n)]$, para cualquier valor x , es factible hallar el intervalo “ n ” al que pertenece mediante:

```
n = floor(1 - log2(1 - x/2))
```

Esto permite obtener una definición de la función:

```
function [ ret ] = fun7 (x)
    n = floor(1 - log2(1 - x./2));
    n = n - 1;
    p = [2*(1 - 1./(2.^n))];
    ret = 2.^(n) .* (x - p);
end
```

A continuación un gráfico de la función así definida:



Con esto podemos realizar la integración:

```
octave> integral(@fun7,0,1.95)
ans = 0.9713
```

Es interesante el ejercicio de realizar la integración mediante antiderivadas.

Al interior del intervalo n :

$$[p(n-1), p(n)] \rightarrow f(x) = 2^{(n-1)} \cdot (x - p(n-1))$$

Su antiderivada es:

$$F(x) = 2^{(n-1)} \cdot (x^2/2 - p(n-1) \cdot x) + CTE$$

Con lo que la integral en el intervalo $[p(n-1), x]$ resulta:

$$SI(n, x) = 2^{(n-1)} \cdot (x^2/2 - p(n-1) \cdot x) - 2^{(n-1)} \cdot (p(n-1)^2/2 - p(n-1) \cdot p(n-1)) =$$

$$2^{(n-1)} \cdot (x^2/2 - p(n-1) \cdot x + p(n-1)^2/2) = 2^{(n-2)}(x - p(n-1))^2$$

Para calcular el total para un intervalo, es fácil calcular el área de cada triángulo; puesto que la altura es constante, el área es la mitad de la base, es decir, $\frac{1}{2} \cdot (1/(2^{(n-1)})) = 1/(2^n)$; otra forma consiste en usar la fórmula anterior usando $x = p(n)$.

Asimismo, el área total incluyendo los primeros n intervalos es: $1/2 + 1/4 + \dots + 1/(2^n)$. Por la fórmula anteriormente enunciada para progresiones geométricas: $ST(n) = 1 - 1/(2^n)$

Esta fórmula tiende a 1 conforme n tiende a infinito, lo cual coincide con la intuición pues la curva siempre está llenando la mitad del rectángulo del gráfico de $2x1$.

La integral de 0 a x vendrá dada por: $SI(n, x) + ST(n-1) = 2^{(n-2)}(x - p(n-1))^2 + 1 - 1/(2^n) =$

$$2^{(n-2)}(x - 2(1 - 1/[2^{n-1}]))^2 + 1 - 1/(2^{n-1})$$

Para el caso de $x = 1.95$, el intervalo que contiene a x es:

```
octave> n = floor(1-log2(1-1.95/2))
n = 6
```

Reemplazando el valor de n , tenemos:

```
octave> x=1.95;
octave> 2.^(n-2)*(x - 2*(1 - 1./[2.^(n-1)]))^2 + 1 - 1./(2.^(n-1))
ans = 0.9712
```

4.3 Transformación de Fourier

La función `fft()` implementa la transformada discreta de Fourier (DFT) mediante el famoso algoritmo “Fast Fourier Transform”. Para este ejemplo

asumiremos que se dispone de un archivo de audio monocanal en formato wav denominado `train.wav`, el cual se puede investigar con `audioinfo()`²²:

```
octave> audioinfo('train.wav')
ans =

  scalar structure containing the fields:

  Filename = /user/train.wav
  CompressionMethod =
  NumChannels = 1
  SampleRate = 11025
  TotalSamples = 28336
  Duration = 2.5702
  BitsPerSample = 16
```

Para este archivo la frecuencia de muestreo es de 11025 Hz, lo que quiere decir que hay 11025 muestras por cada segundo. El número de muestras es de 28336, por lo que se concluye su duración igual a $28336/11025 = 2.5702$ segundos.

A continuación procedemos a leerlo con la función `audioread()`, la que retorna las muestras de audio así como la frecuencia de muestreo (variable `FS`.)

```
octave> [Y,FS]=audioread('train.wav');
octave> size(Y,1)
ans = 28336
```

La variable `Y` contiene las 28336 muestras en una matriz columna. Al obtener su DFT obtenemos un vector con el mismo número de elementos:

```
octave> FREQ=fft(Y);
octave> SZ=size(FREQ,1)
SZ = 28336
```

La función `fft()` proporciona las frecuencias correspondientes a los rangos $[0, \frac{FS}{2}] \cup [-\frac{FS}{2}, 0)$, lo que hace confusa su gráfica directa. A fin de obtener una distribución correspondiente al rango $(-\frac{FS}{2}, \frac{FS}{2}]$ (con la frecuencia cero centrada) se emplea `fftshift()`:

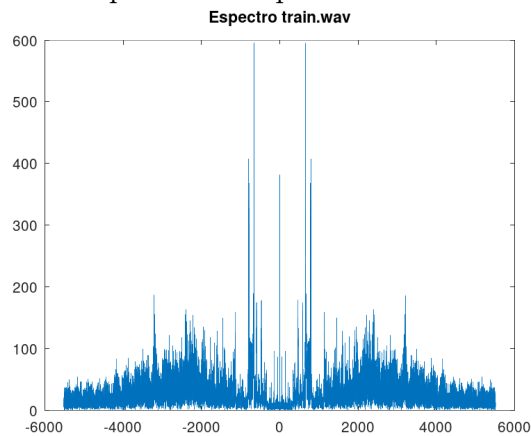
```
octave> FREQ=fftshift(fft(Y));
octave> SZ=size(FREQ,1)
SZ = 28336
```

²²El archivo de prueba que estamos utilizando proviene de la galería de sonidos de LibreOffice; en Linux Debian 12 está ubicado en la ruta `"/usr/lib/libreoffice/share/gallery/sounds/train.wav"`.

Para el rango de frecuencias $(-\frac{FS}{2}, \frac{FS}{2}]$ crearemos un vector usando `linspace()`; esto nos permitirá realizar un ploteo de la DFT considerando la magnitud (`abs`) de los valores complejos:

```
octave> FX=linspace(-FS/2, FS/2, SZ)';
octave> plot(FX, abs(FREQ))
octave> title('Espectro train.wav');
```

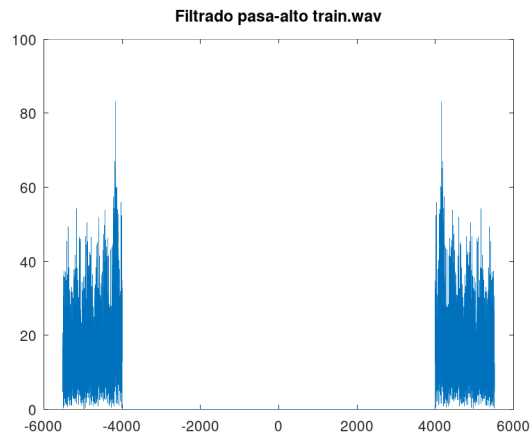
Es sabido que la DFT aplicada a valores reales resulta una función simétrica:



Filtro Pasa Alto A modo de prueba eliminaremos las frecuencias inferiores a 4 KHz; es decir, sólo aquellos valores del espectro mayores a 4000 Hz se conservarán. Estos valores se pueden obtener usando la matriz auxiliar `FXH` generada a partir de la condicional $FX > 4000$ ó $FX < -4000$; esta matriz contiene unos para los valores que satisfacen la condición y ceros en caso contrario; luego, simplemente multiplicamos el espectro `FREQ` por `FXH`:

```
octave> CUT_FREQ=4000;
octave> FXH = (FX > CUT_FREQ) | (FX < -CUT_FREQ);
octave> FREQH=FREQ.*FXH;
octave> figure;
octave> plot(FX, abs(FREQH))
octave> title('Filtrado pasa-alto train.wav');
```

Visualmente conservamos la simetría:



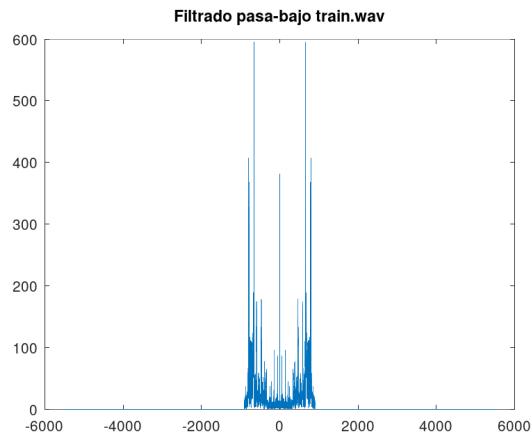
A partir de estos datos filtrados en el “dominio de la frecuencia” podemos obtener una versión en el “dominio del tiempo” aplicando la DFT inversa - previa inversión del `fftshift()` mediante `ifftshift()`; el resultado incluso puede ser usado para generar un nuevo archivo de audio:

```
octave> YH=ifft(ifftshift(FREQH));
octave> audiowrite('train-h.wav', real(YH), FS)
```

Filtro Pasa Bajo El complemento corresponde a filtrar el contenido de alta frecuencia, permitiendo que sobrevivan los componentes de baja frecuencia. A continuación el filtrado para limitar las frecuencias hasta 900 Hz:

```
octave> CUT_FREQ=900;
octave> FXL = (FX < CUT_FREQ) & (FX > -CUT_FREQ);
octave> FREQL=FREQ.*FXL;
octave> figure;
octave> plot(FX, abs(FREQL));
octave> title('Filtrado pasa-bajo train.wav');
octave> YL=ifft(ifftshift(FREQL));
octave> audiowrite('train-l.wav', real(YL), FS)
```

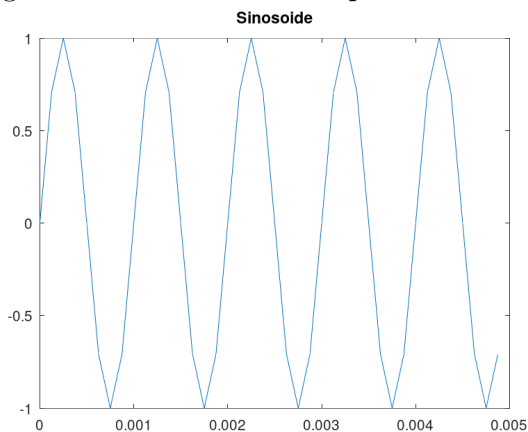
Las frecuencias se concentran ahora cerca del cero:



Sinusoide La función sinusoidal genera corresponde a una frecuencia única. Generaremos una sinusoide de 1000 Hz (cada 1/1000 segundos se repite la onda completa); es interesante oírla a partir de un archivo wav (se puede comparar con ejemplos de sonidos de 1000Hz disponibles en Internet):

```
octave> t=linspace(0, 30, 30*FS)';
octave> Y=sin(2*pi*1000*t);
octave> plot(t(1:40), Y(1:40))
octave> title('Sinusoide ');
octave> audiowrite('sine-1000hz.wav', real(Y), FS)
```

El gráfico sólo considera los primeros 40 valores:



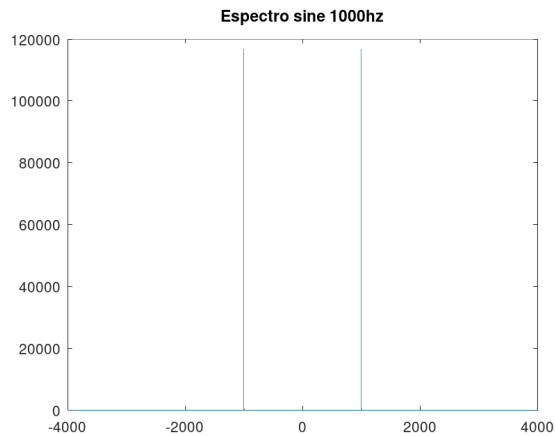
Esta función genera un pico de en el diagrama de frecuencias (en teoría, una función “delta de Dirac”):

```
octave> FREQ=fftshift(fft(Y));
```

```

octave> SZ=size(FREQ,1);
octave> FX=linspace(-FS/2, FS/2, SZ)';
octave> f=figure;
octave> plot(FX, abs(FREQ))
octave> title('Espectro sine 1000hz');

```



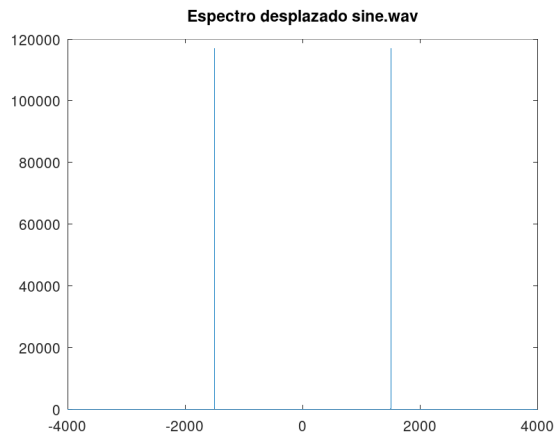
Haciendo un desplazamiento de los valores de 500 Hz obtenemos una senoide a 1500Hz. El archivo wav tiene un sonido más agudo:

```

octave> DELTA_HZ = 500;
octave> MID=SZ/2 + 1;
octave> OFFSET=round(SZ * DELTA_HZ / FS);
octave> FREQ2=zeros(SZ,1);
octave> FREQ2(1 : MID - OFFSET) = FREQ(OFFSET : MID - 1);
octave> FREQ2(MID + OFFSET : SZ) = FREQ(MID : SZ - OFFSET);
octave> f=figure
octave> plot(FX, abs(FREQ2))
octave> title('Espectro desplazado sine.wav');
octave> Y2=ifft(fftshift(FREQ2));
octave> audiowrite('sine-1500hz.wav', real(Y2), FS)

```

Compárese con el gráfico anterior:



5 Polinomios

Se representan como una matriz de coeficientes que corresponden a potencias decrecientes. Por ejemplo:

$$f(x) = x^2 - 5x + 6$$

Se representa por:

```
octave> P = [1 , -5 , 6];
```

Para evaluar $f(7)$:

```
octave> polyval(P,7)
ans = 20
```

Sus raíces:

```
octave> roots(P)
ans =
 3
 2
```

También se pueden obtener las raíces complejas:

```
octave> roots([1,1,1])
ans =
-0.50000 + 0.86603i
-0.50000 - 0.86603i
octave> roots([1,1,1,1,1])
ans =
0.30902 + 0.95106i
0.30902 - 0.95106i
```

```
-0.80902 + 0.58779i
-0.80902 - 0.58779i
```

Producto de dos polinomios²³:

```
octave> conv([1,1,1],[2,0,1])
ans =
2 2 3 1 1
```

Para encontrar el máximo común divisor se utiliza `polygcd()`. A modo de ejemplo, simplifiquemos la expresión:

$$E(x) = \frac{x^6 + 4x^5 - 5x^4 - 3x^3 + 5x^2 - 6x + 18}{2x^6 + 4x^5 - 10x^4 - 7x^3 + 19x^2 + 2x - 3}$$

Usando Octave:

```
octave> R=[1 , 4 , -5 , -3 , 5 , -6 , 18];
octave> S=[2 , 4 , -10 , -7 , 19 , 2 , -3];
octave> G=polygcd(R,S)
G =
1 -1 -2 3
```

Matlab no proporciona la función `polygcd()`²⁴, por lo cual hemos adaptado el código de Octave para hacerla ejecutable en Matlab. Para evitar confusiones, la hemos denominado `xpolygcd()`:

```
function x = xpolygcd (b, a, tol)
% adaptado de polygcd.m de la libreria de GNU Octave
if (nargin == 2)
    if (isa (a, "single") || isa (b, "single"))
        tol = sqrt (eps ("single"));
    else
        tol = sqrt (eps);
    end
end
if (length (a) == 1 || length (b) == 1)
    if (a == 0)
        x = b;
    elseif (b == 0)
        x = a;
    else
        x = 1;
    end
end
```

²³En realidad esta función corresponde a la operación de convolución de funciones (cuyos valores vienen dados en dos matrices.) La convolución numérica coincide con la operación sobre los coeficientes del producto de polinomios.

²⁴Sin embargo, la función `gcd()` permite obtener el MCD de polinomios expresados simbólicamente.

```

else
  a = a / a(1);
  while (1)
    [d, r] = deconv (b, a);
    nz = find (abs (r) > tol);
    if (isempty (nz))
      x = a;
      break;
    else
      r = r(nz(1):length (r));
    end
    b = a;
    a = r / r(1);
  end
end
end
end

```

Para dividir polinomios utilizamos `deconv()`²⁵, que retorna el cociente y el residuo de los polinomios:

```

octave> [Qr,Rr]=deconv(R,G)
Qr =
  1   5   2   6
Rr =
  0   0   0   0   0   0   0
octave> [Qs,Rs]=deconv(S,G)
Qs =
  2   6   0  -1
Rs =
  0   0   0   0   0   0   0

```

Los cocientes del numerador y denominador permiten obtener la fracción simplificada:

$$E_{simp}(x) = \frac{x^3 + 5x^2 + 2x + 6}{2x^3 + 6x^2 - 1}$$

5.1 Derivada e integral de polinomios:

Se utilizan las funciones `polyder()` y `polyint()`:

```

octave> polyder([1,5,0,0])
ans =
  3 10 0

```

²⁵También puede emplearse la función “`residue()`” que está disponible en Octave y Matlab pero con ligeras diferencias en su comportamiento. Esta función está orientada al análisis de los polos complejos de una función racional.

```
octave> polyint([3,10,0])
ans =
1 5 0 0
```

Una representación más “estándar” de los polinomios para efectos de visualización se obtienen con “polyout()”:

```
octave> polyout([1, 6, -6, -5])
1*s^3 + 6*s^2 - 6*s^1 - 5
octave> polyout([1, 6, -6, -5], 'x')
1*x^3 + 6*x^2 - 6*x^1 - 5
```

5.2 Ajuste de datos por interpolación de polinomios

Empezaremos “fabricando” un ejemplo de problema a interpolar. Partimos de un polinomio arbitrario el cual supuestamente no se conoce, pero que describe el comportamiento de cierto fenómeno²⁶:

$$P(x) = 1.1x^2 - 5x + 6.5$$

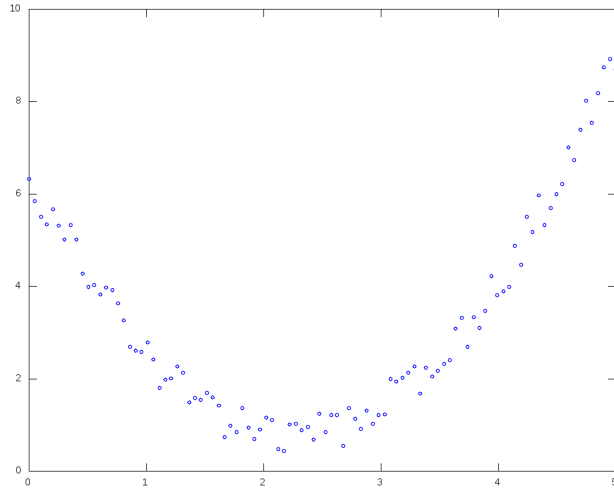
```
octave> P=[1.1, -5, 6.5]
P =
1.1000 -5.0000 6.5000
```

y generamos un conjunto de valores a lo largo del eje x , a los cuales les agregaremos un “ruido” R correspondiente a números aleatorios en el intervalo $(-0.5, 0.5)$:

```
octave> x=linspace(0,5,100);
octave> R=(rand(100,1)-0.5)';
octave> DATA=polyval(P,x) + R;
octave> plot(x,DATA,'o')
```

Estos puntos permiten apreciar la forma de la gráfica del polinomio:

²⁶En la práctica, esta función podría no ser un polinomio; sin embargo, las técnicas de interpolación asociadas a este tópico intentan obtener polinomios que la aproximen en forma óptima desde el punto de vista de la minimización de la suma del cuadrado de los errores. No abordamos aquí el (complejo) problema de buscar la función más conveniente para aproximar una muestra (comparación de modelos estadísticos.)

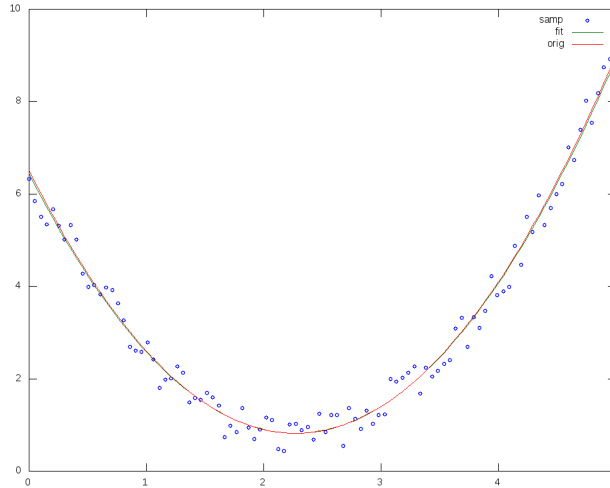


A continuación realizamos un ajuste a un polinomio de grado 2:

```
octave> PFIT=polyfit(x,DATA,2)
PFIT =
1.0868 -4.9362 6.4287
```

Se puede apreciar la gran similitud de PFIT con P. El gráfico presentado a continuación ilustra los puntos de la “muestra”, el polinomio ajustado, así como el polinomio que sirvió como base para crear la muestra:

```
octave> plot(x,DATA,'o', x,polyval(PFIT,x),'-', x,polyval(P,x),'-')
octave> legend('samp','fit','orig')
```



6 Ecuaciones diferenciales

Octave proporciona un conjunto de funciones que implementan diversos algoritmos para la solución de ecuaciones diferenciales ordinarias. En esta sección ilustramos la función `ode45()` que implementa el algoritmo conocido como Dormand-Prince de orden 4 y es compatible con Matlab.

6.1 Ecuaciones diferenciales ordinarias de primer orden

Como ejemplo ilustrativo, se requiere obtener $y(10)$ dada la ecuación diferencial:

$$\frac{dy}{dt} = t + \cos(t)$$

con condición inicial $y(0) = 1$.

Este ejemplo es trivial. Por integración directa es evidente que $y(t) = t^2/2 + \sin(t) + 1 \rightarrow y(10) = 50,456$

Mediante Octave, debemos especificar una función tal que retorne $dy/dt = E(t, y)$:

```
function [ r ] = xfun ( t,y)
    r = t + cos(t);
end
```

Proporcionaremos el intervalo $[0 - 10]$ y la condición inicial $f(0) = 1$; si la ecuación es de orden superior se deberá proporcionar una matriz conteniendo el valor inicial de la función y de las derivadas de ésta, como es usual en un problema de Cauchy. Finalmente resolvemos la ecuación usando `ode45()`:²⁷:

```
octave> [T, Y] = ode45(@xfun, [0,10], 1)
```

Los valores vienen dados en las matrices T e Y:

```
octave> [T,Y]
ans =

      0      1.0000
  0.0681      1.0704
  0.1703      1.1840
  0.3236      1.3704
  0.5535      1.6789
  0.8985      2.1860
  1.4158      2.9903
  2.1918      4.2154
  3.1918      6.0437
  4.1918      8.9182
  5.1918     13.5904
  6.1918     20.0783
  7.1918     27.6500
  8.1918     35.4966
  9.1918     43.4758
 10.0000     50.4560
```

Es decir, mediante aproximaciones hallamos que $f(10) = 50.4560$, coincidiendo con la solución analítica.

Nota: También se podría haber definido la función “en línea”, dado que su definición es muy sencilla:

```
[T, Y] = ode45(@(t,y) t+cos(t), [0,10], 1);
```

6.2 Ecuaciones diferenciales ordinarias de orden superior

Para la solución de estos problemas consiste en realizar un cambio de variables que transforma la ecuación diferencial de orden superior en un sistema

²⁷En una versión anterior de este documento utilizamos `lsode()` que no está disponible en Matlab; por este motivo la hemos reemplazado con `ode45()`.

de ecuaciones de primer orden. Ilustraremos este procedimiento con dos ejemplos ilustrativos.

Ejemplo Sea la ecuación diferencial para la función $x(t)$:

$$100x'' + x = t^2 - 7t + 204$$

con condiciones iniciales $x(0) = 4$; $x'(0) = -6.9$

Previamente, analicemos su solución analítica (que supuestamente desconocemos):

$$x(t) = t^2 + \sin(t/10) - 7t + 4$$

En el rango $[0, 10]$ podemos obtener algunos valores representativos:

```
function [ ret ] = xfun3_sol (t)
    ret = t.*t + sin(t/10) - 7 * t + 4;
end

octave> [[0:10] ', xfun3_sol([0:10] ')]
ans =
  0.00000    4.00000
  1.00000   -1.90017
  2.00000   -5.80133
  3.00000   -7.70448
  4.00000   -7.61058
  5.00000   -5.52057
  6.00000   -1.43536
  7.00000    4.64422
  8.00000   12.71736
  9.00000   22.78333
 10.00000   34.84147
```

Haciendo $x \rightarrow x_1$ y $x' = x'_1 = x_2$, reescribimos la ecuación:

$$100x'_2 + x_1 = t^2 - 7t + 204$$

lo que conforma el sistema:

$$\begin{cases} x'_2 = (t^2 - 7t + 204 - x_1)/100 \\ x'_1 = x_2 \end{cases}$$

Así, tenemos dos ecuaciones de primer orden de la forma $dy/dt = E(t, y)$. Este sistema de ecuaciones se procesa del mismo modo que el caso anterior²⁸:

```
function [ ret ] = xfun3 (t, x)
    ret(1,1) = x(2);
    ret(2,1) = (t * t - 7 * t + 204 - x(1))/100;
end
octave> [T, Y] = ode45(@xfun3, [0,10],[4,-6.9])
```

Visualizamos los valores de $[T, Y]$ correspondientes a t , x_1 y x_2 ; no olvidar que $x(t)$ corresponde a x_1 , por lo que la solución obtenida es $x(10) = 34.8415$.

```
octave:69> [T, Y]
ans =

    0.0000    4.0000   -6.9000
    0.0681    3.5346   -6.7637
    0.1703    2.8538   -6.5594
    0.3236    1.8718   -6.2528
    0.5535    0.4869   -5.7931
    0.8985   -1.3922   -5.1035
    1.4158   -3.7650   -4.0694
    2.1918   -6.3213   -2.5187
    3.1918   -7.8412   -0.5214
    4.1918   -7.3643    1.4750
    5.1918   -4.8915    3.4705
    6.1918   -0.4236    5.4651
    7.1918    6.0385    7.4589
    8.1918   14.4940    9.4520
    9.1918   24.9422   11.4443
   10.0000   34.8415   13.0540
```

Ejemplo Resolver en $y(t)$ para $t \in [0, 2]$:

$$y''' + y'' - 6y' = 2 + 2t - 6t^2$$

condiciones iniciales: $y(0) = -4$, $y'(0) = -10$, $y''(0) = -20$.

Para resolverlo consideramos los cambios de variable $y = y_1$; $y' = y_2$; $y'' = y_3$, y reescribimos:

$$y''' = 2 + 2t - 6t^2 - y'' + 6y' = 2 + 2t - 6t^2 - y_3 + 6y_2$$

Así, la función con las derivadas $[y', y'', y''']$ es:

²⁸Notar que la función retorna un vector columna (variable **ret**); esto es obligatorio en Matlab.

```
function [ ret ] = xfun4 (y,t)
    ret(1,1) = y(2);
    ret(2,1) = y(3);
    ret(3,1) = 2 + 2 * t - 6 * (t.^2) + 6 * y(2) - y(3);
end
```

```
[T, Y] = ode45(@xfun4, [0,2], [-4, -10, -20]);
```

La solución para (t, y_1, y_2, y_3) resulta:

```
octave:81> [T,Y]
ans =

      0 -4.0000e+00 -1.0000e+01 -2.0000e+01
 6.8129e-02 -4.7298e+00 -1.1455e+01 -2.2783e+01
 1.7032e-01 -6.0276e+00 -1.4030e+01 -2.7776e+01
 3.2361e-01 -8.5399e+00 -1.8998e+01 -3.7558e+01
 5.2361e-01 -1.3201e+01 -2.8223e+01 -5.5948e+01
 7.2361e-01 -2.0130e+01 -4.1990e+01 -8.3579e+01
 9.2361e-01 -3.0448e+01 -6.2569e+01 -1.2500e+02
 1.1236e+00 -4.5835e+01 -9.3352e+01 -1.8698e+02
 1.3236e+00 -6.8801e+01 -1.3940e+02 -2.7965e+02
 1.5236e+00 -1.0311e+02 -2.0825e+02 -4.1809e+02
 1.7236e+00 -1.5436e+02 -3.1116e+02 -6.2482e+02
 1.9236e+00 -2.3094e+02 -4.6493e+02 -9.3342e+02
 2.0000e+00 -2.6933e+02 -5.4198e+02 -1.0880e+03
```

En la segunda columna observamos el valor final $y(2) = y_1(2) = -269.33$.

La solución analítica de la ecuación diferencial corresponde a $y(t) = \frac{t^3}{3} - 5e^{2t} + 1$, lo que permite contrastar la aproximación obtenida:

```
octave> 2^3/3-5*exp(2*2)+1
ans = -269.32
```

7 Otras facilidades del lenguaje

7.1 Textos como arrays de caracteres

En el momento que escribimos este documento, Octave y Matlab difieren ligeramente en el tratamiento de los textos. En Octave un “string” es un vector fila conteniendo caracteres, y se puede construir utilizando comillas simples o dobles:

```
octave> class('something')
ans = char
octave> class("something")
ans = char
```

```

octave> isequal(['s','o','m','e'], "some")
ans = 1
octave> isequal(['s','o','m','e'], 'some')
ans = 1
octave> isequal("some", 'some')
ans = 1
octave> size("something")
ans =
     1     9
octave> size('something')
ans =
     1     9

```

Matlab proporciona un tratamiento diferenciado para cadenas de texto “string” y arrays de caracteres “char”:

```

MATLAB>> class('something')
ans =
     'char'
MATLAB>> class("something")
ans =
     'string'
MATLAB>> size("something")
ans =
     1     1
MATLAB>> size('something')
ans =
     1     9

```

Con cierto cuidado es posible desarrollar programas capaces de procesar portablemente indistintamente cadenas de texto y arrays de caracteres.

La función “dec2bin()” permite convertir un número decimal a base 2; sin embargo, este valor binario se proporciona en un String. Por ejemplo, la siguiente instrucción permite obtener las entradas de una “tabla de verdad” para tres variables:

```

octave> dec2bin(0:2^3-1)
ans =
000
001
010
011
100
101
110
111

```

7.1.1 La función arrayfun()

A fin de convertir estos dígitos a una matriz de ceros y unos, debemos iterar en estos caracteres, convirtiéndolos a valores numéricos. La iteración se puede efectuar con un bucle `for`, pero para casos simples suele ser suficiente con `arrayfun()`:

```
octave> T=arrayfun(@(s) str2num(s), dec2bin(0:7])
T =
    0    0    0
    0    0    1
    0    1    0
    0    1    1
    1    0    0
    1    0    1
    1    1    0
    1    1    1
```

Aprovechando esta tabla, suponiendo que se desea construir una tabla de verdad para las expresiones lógicas “ $X \oplus Y \oplus Z$ ” y “ $X \wedge (Y \vee Z)$ ” podemos proceder como sigue²⁹:

```
octave> X=T(:,1);
octave> Y=T(:,2);
octave> Z=T(:,3);
octave> [X, Y, Z, xor(X,Y,Z), and(X, or(Y,Z))]
ans =
    0    0    0    0    0
    0    0    1    1    0
    0    1    0    1    0
    0    1    1    0    0
    1    0    0    1    0
    1    0    1    0    1
    1    1    0    0    1
    1    1    1    1    1
```

Donde las columnas corresponden a X , Y , Z y las dos expresiones lógicas, respectivamente.

7.2 Estructuras y celdas

El operador “.” permite definir miembros de estructuras:

```
octave> pox.a=33;
octave> pox.b=55;
```

²⁹En Matlab las funciones lógicas no aceptan más de dos argumentos por lo que el ejemplo debe modificarse así: `[X, Y, Z, xor(X,xor(Y,Z)), and(X, or(Y,Z))]`.


```

octave> pox
pox =
{
  a = 33
  b = 55
}
octave> pox.a
ans = 33
octave> class(pox)
ans = struct

```

De otro lado, mientras que las matrices permiten almacenar listas de valores del mismo tipo, los “cell” permiten heterogeneidad:

```

octave> e={'a','b',44; 'j',12,'k'};
octave> e
e =
{
  [1,1] = a
  [2,1] = j
  [1,2] = b
  [2,2] = 12
  [1,3] = 44
  [2,3] = k
}

octave> e{2,1}
ans = j
octave> e{2,2}
ans = 12

```

7.3 Interacción con el sistema operativo

El siguiente “script” se ha almacenado en un archivo llamado `sistema_de_archivos.m`, e ilustra un conjunto de funciones que interactúan con el sistema operativo:

```

while true
  disp('Sistema de Archivos');
  disp('-----');
  disp('');
  p = pwd;
  fprintf("Directorio actual: %s\n\n", p);
  o = menu('Opciones disponibles', ...
    'Cambiar de directorio', ...
    'Listar directorio', ...
    'Mostrar archivo', ...
    'Crear archivo', ...

```

```

        'Salir');
switch o
case 1
    nd = input('A que directorio?', 's');
    cd(nd);
case 2
    fls = dir('.');
    for t = 1:size(fls)
        if fls(t).isdir == 1
            fprintf('[%s]\n' , fls(t).name)
        else
            disp(fls(t).name)
        end
    end
case 3
    nom = input('Nombre?', 's');
    eval(strcat("type ",nnnn));
case 4
    nom = input('Nombre?', 's');
    eval(strcat("edit ",nnnn));
case 5
    return;
otherwise
    error ("Opcion incorrecta");
end
end
end

```

Se ilustra la función “`input()`”, que permite solicitar un valor del teclado; `input` tiene una sintaxis inusual: cuando se desea obtener un texto literal debe agregarse un argumento ‘`s`’, de lo contrario asumirá que el texto introducido es una expresión matemática a ser evaluada. De otro lado, `menu()` permite presentar un menú de opciones, y `eval()` la evaluación de expresiones arbitrarias, lo que es necesario para ciertos comandos.

El ejemplo también ilustra un conjunto de funciones relacionadas al sistema de archivos inspiradas en los comandos usuales de línea de comandos (shell.) Por ejemplo, la función `cd()` permite cambiar de directorio de trabajo; el directorio actual se obtiene de la variable especial `pwd`. Asimismo se ilustra la función `dir(directorio)`³⁰ la cual permite obtener información acerca del contenido de un directorio; este caso es interesante, pues `dir()` retorna un array de estructuras cuyos elementos son `cells`. Así tenemos:

```

octave> T=dir('/tmp ');
octave> T

```

³⁰Notar que `dir()` sin argumento de salida simplemente imprime el directorio actual en la pantalla. De otro lado, Octave también se proporciona un comando equivalente `ls()`.

```

T =
{
  26x1 struct array containing the fields:
    name
    date
    bytes
    isdir
    datenum
    statinfo
}

```

Podemos obtener un elemento así:

```

octave> T(11)
ans =
{
  name = ch6.pdf
  date = 07-Apr-2011 11:41:26
  bytes = 228752
  isdir = 0
  datenum = 7.3457e+05
  statinfo =
  {
    dev = 2053
    ino = 134432
    mode = 33024
    modestr = -r-----
    nlink = 1
    uid = 1000
    gid = 1000
    rdev = 0
    size = 228752
    atime = 1.3022e+09
    mtime = 1.3022e+09
    ctime = 1.3022e+09
    blksize = 4096
    blocks = 448
  }
}

```

y el nombre de un archivo:

```

octave> T(11).name
ans = ch6.pdf

```

Para almacenar la lista de nombres debemos emplear un cell array. El “cell array” puede considerarse una matriz con capacidad de almacenar elementos de distinto tipo y tamaño. En nuestro caso, los elementos son strings de distinto tamaño.

```

octave> NOMBRES={T.name};

```

```

octave> NOMBRES
N =
{
  [1,1] = .
  [1,2] = ..
  [1,3] = .ICE-unix
  [1,4] = .X0-lock
  ...

```

Análogamente a las matrices, podemos acceder a sus elementos con {fila,columna}:

```

octave> NOMBRES{1,12}
ans = eclipse

```

También a rangos de elementos:

```

octave> NOMBRES{1,12:14}
ans = eclipse
ans = hsperfdata_diego
ans = jna319.tmp

```

Y recorrerlos en un loop for...end:

```

octave> for i = NOMBRES ; fprintf("Archivo : %s\n", i{1,1}) ; end
Archivo : .
Archivo : ..
Archivo : .ICE-unix
Archivo : .X0-lock
Archivo : .X11-unix
...

```

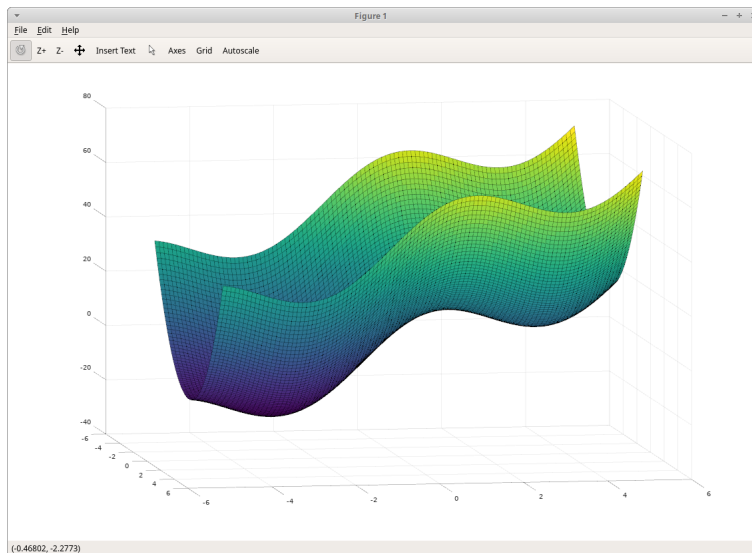
8 Optimización de funciones

Ahora considérese la siguiente función con dominio en $[-5, 5] * [-5, 5]$:

```

octave> [X,Y] = meshgrid([-5:0.1:5],[-5:0.1:5]);
octave> Z = 2*X.^2 + 10*sin(Y+1) + 5*Y;
octave> surf(X, Y, real(Z))

```



Esta función tiene claramente un mínimo local alrededor de $(0, 3)$ y un mínimo global alrededor de $(0, -4)$. Busquemos estos mínimos con Octave: la función de dos variables se registrará en `tominv.m`:

```
function [ ret ] = tominv(v)
    x=v(1);
    y=v(2);
    ret = 2*x.^2+10*sin(y+1)+5*y;
end
```

ahora emplearemos `fminunc()`; esta función recibe como argumento una función a minimizar. Notar que se proporciona el “punto inicial” $[0, 3]$ de la búsqueda:

```
octave> [x, fval, info, output, grad, hess] = fminunc(@tominv,[0;3])
x =
```

```
0.00000
3.18879
```

```
fval = 7.2837
info = 3
output =
```

scalar structure containing the fields:

```
iterations = 5
successful = 4
funcCount = 20
```

```

grad =

    0.0000e+00
    3.1055e-06

hess =

    1.00000    0.00000
    0.00000    8.65978

```

Este ejemplo muestra la información retornada por la función; el valor `info=3` significa que las iteraciones han llegado a un punto en que la mejora entre evaluaciones es menor que el atributo “`TolFun`” (no empleado en este ejemplo en forma explícita) que por omisión es $1e - 7$.

El otro mínimo (el global) se puede encontrar partiendo desde otro punto:

```

octave> [x, fval, info, output, grad, hess] = fminunc(@tominv,[0;-4])
x =

    0.00000
   -3.09439

fval = -24.132
info = 3
...

```

Si no se especifican los argumentos de salida, sólo se retorna el punto mínimo:

```

octave> fminunc(@tominv,[0;-4])
ans =

    0.00000
   -3.09439

```

8.1 Optimización cuadrática con restricciones

El mismo resultado puede ser obtenido con `sqp()`³¹, la cual permite agregar restricciones a la optimización; observar que el punto inicial es el primer argumento:

```

octave> [x, obj, info, iter, nf, lambda] = sqp([0;3],@tominv)
x =

```

³¹En Matlab no existe esta función; sin embargo, existen diversas alternativas como `quadprog()`.

```

0.00000
3.18879

obj = 7.2837
info = 104
iter = 5
nf = 9
lambda = [](0x1)

octave> [x, obj, info, iter, nf, lambda] = sqp([0;-4],@tominv)
x =

0.00000
-3.09440

obj = -24.132
info = 101
iter = 6
nf = 9
lambda = [](0x1)

```

Para apreciar el poder de `sqp()`, obsérvese lo que ocurre con otro punto inicial $(0, 0)$:

```

octave> [x, obj, info, iter, nf, lambda] = sqp([0;0],@tominv)
x =

-6.3853e-08
-9.3776e+00

obj = -55.548
info = 104
iter = 10
nf = 14
lambda = [](0x1)

```

El mínimo hallado corresponde (aproximadamente) a $(0, -9.37)$, el cual está fuera del dominio que hemos definido inicialmente. La función `sqp()` permite especificar valores máximos y mínimos para las coordenadas del mínimo:

```

octave> [x, obj, info, iter, nf, lambda] = sqp([0;0],
      @tominv, [], [], [-5;-5], [5;5])
x =

0.00000
-3.09440
obj = -24.132
info = 101
iter = 9
nf = 12

```

```
lambda =
  0
  0
  0
  0
```

Observar que hemos pasado matrices vacías en los argumentos tercero y cuarto; los valores mínimos van en el quinto argumento, y los máximos en el sexto.

Una forma más general consiste en emplear el cuarto argumento, proporcionándole una función que retorna un vector cuyos valores deben ser mayores que cero (restricciones para el espacio de búsqueda); como nuestro dominio es cuadrado, este equivale a las desigualdades $-5 < x < 5$, es decir $|x| < 5$, equivalente a $5 - |x| > 0$ (análogamente para la coordenada “ y ”); esto se puede escribir en una función del siguiente modo:

```
octave> constr=@(v) [5-abs(v(1));5-abs(v(2))];
```

y se proporciona como cuarto argumento a Octave (notar que ya no empleamos ni el quinto ni el sexto):

```
octave> [x, obj, info, iter, nf, lambda] = sqp([0;0],
      @tominv, [], constr)
x =
  1.9378e-08
 -3.0944e+00
obj = -24.132
info = 104
iter = 12
nf = 22
lambda =
  0
  0
```

9 Cálculo Simbólico

Octave proporciona la extensión “Symbolic Package” para ejecutar operaciones de forma simbólica; esto puede requerir de un proceso de instalación complementario³².

³²En Linux Debian 12 viene dado por el paquete `octave-symbolic`; notar que el año 2021 este paquete tuvo problemas de compilación con sus dependencias (ver <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=980707>), lo que afectó a distribuciones dependientes; por ejemplo Ubuntu 22.04.

Una vez instalado el paquete “symbolic”, éste debe cargarse:

```
octave> pkg load symbolic
```

Las variables empleadas para cálculo simbólico deben especificarse en forma explícita. Por ejemplo:

```
octave> syms x a
```

Una ecuación tal como

$$2x - 5a = 0$$

puede resolverse simbólicamente mediante:

```
octave> solve(2*x-5*a==0,x)
ans = (sym)
  5.a
  ---
  2
```

9.1 Derivación e Integración

A partir de variables simbólicas, es posible definir funciones en forma simbólica. Mediante `diff()` e `int()` es posible calcular derivadas e integrales:

```
octave> f=3*log(5*x+1)
f = (sym) 3.log(5.x + 1)
octave> g=diff(f)
g = (sym)
```

$$\frac{15}{5.x + 1}$$

```
octave> h=sin(x)^3+cos(4*x)
h = (sym)
```

$$\sin^3(x) + \cos(4.x)$$

```
octave> int(h)
ans = (sym)
```

$$\frac{\sin(4.x)}{4} + \frac{\cos^3(x)}{3} - \cos(x)$$

Es posible realizar integraciones impropias. Por ejemplo, la transformada de Laplace:

$$L\{t\} = \int_0^{\infty} te^{-st} dt = \frac{1}{s^2}$$

Esto es válido sujeto a $Re(s) > 0$ ³³. Empleando Octave, esto puede ser calculado así:

```
octave> syms s t
octave> int(t*exp(-s*t), t, 0, inf)
      1      pi
      -      for |arg(s)| < -
      2      2
      s
...

```

Notar que Octave anota el dominio admisible para s , así como la imposibilidad de integrar fuera de este dominio³⁴. A continuación otro ejemplo que introduce una constante adicional:

$$L\{t^n\} = \int_0^{\infty} t^n e^{-st} dt = \frac{\Gamma(n+1)}{s^{n+1}}$$

Válido para $Re(s) > 0$ y $n > -1$. Usando el mismo procedimiento se obtiene el resultado esperado, así como una especificación del dominio admisible para s . Usualmente sólo se requiere el cálculo formal de la antiderivada, lo cual suele simplificarse si restringimos el dominio a los reales positivos:

```
octave> syms s t n
octave> assume(s, 'positive');
octave> assume(n, 'positive');
octave> int(t^n*exp(-s*t), t, 0, inf)
      -n
      s  .Gamma(n + 1)
      -----
      s

```

Como siguiente ejemplo considérese:

$$L\{e^{-at} \sin(bt)\} = \int_0^{\infty} e^{-at} \sin(bt) e^{-st} dt = \frac{b}{(s+a)^2 + b^2}$$

³³https://en.wikipedia.org/wiki/Laplace_transform#Table_of_selected_Laplace_transforms

³⁴La función `children()` permite extraer los componentes de esta respuesta como elementos de una matriz.

Usando las mismas consideraciones que en el caso anterior obtenemos la antiderivada correcta, salvo que además requiere ser simplificada:

```
octave> syms s t a b
octave> assume(s, 'positive');
octave> assume(a, 'positive');
octave> assume(b, 'positive');
octave> simplify(int(exp(-a*t)*sin(b*t)*exp(-s*t), t, 0, inf))
      b
-----
      2      2
b  + (a + s)
```

Para nuestro ejemplo final, calculemos las transformadas de Laplace de la función escalón unitario (`heaviside()`) e impulso unitario (`dirac()`) centradas en el punto “w”:

```
octave> syms s t w
octave> assume(s, 'positive');
octave> assume(w, 'positive');
octave> int(heaviside(t-w)*exp(-s*t),t,0,inf)
      -sw
      e
      -----
      s
octave> int(dirac(t-w)*exp(-s*t),t,0,inf)
      -s.w
      e
```

9.2 Matrices simbólicas

La operación con matrices también se puede efectuar simbólicamente. El siguiente ejemplo define una matriz de rotación en el plano y se calcula su inversa. Esta inversa es luego explícitamente simplificada:

```
octave> syms theta
octave> rot=[cos(theta),-sin(theta);sin(theta),cos(theta)]
rot = (sym 2x2 matrix)

      | cos(theta)  -sin(theta) |
      |              |
      | sin(theta)   cos(theta) |

octave> rotinv=inv(rot)
rotinv = (sym 2x2 matrix)

      |              |
      |      2      |
      | 1 - sin (theta)  |
      |              |
```

```

|----- sin(theta)|
|  cos(theta)      |
|                  |
| -sin(theta)     cos(theta)|

octave> rotinv=simplify(rotinv)
rotinv = (sym 2x2 matrix)

|cos(theta)  sin(theta)|
|           |
|-sin(theta) cos(theta)|

```

9.3 Substitución de Valores

La función `subs()` permite efectuar la substitución de valores en forma efectiva:

```

octave> subs(rot,theta,sym(pi)/6)
ans = (sym 2x2 matrix)

```

```

|\3      |
|--  -1/2|
|2       |
|        |
|      \3 |
|1/2  --- |
|      2  |

```

```

octave> subs(rot,theta,sym(pi)/7)
ans = (sym 2x2 matrix)

```

```

|  pi      pi |
|cos(--) -sin(--)|
|   7      7  |
|           |
|  pi      pi |
|sin(--) cos(--)|
|   7      7  |

```

Este ejemplo ilustra el uso de `sym(.)` para definir constantes. Es importante notar que debe emplearse `sym(pi)/7` y no `sym(pi/7)`, pues en el último caso se efectuará la división (en punto flotante) la cual podría perder su forma “simbólica”.

El siguiente ejemplo ilustra que la substitución puede ser mediante nuevas expresiones simbólicas más complejas:

```

octave> k=subs(rot,theta,x+theta)

```

```

k = (sym 2x2 matrix)

|cos(theta + x)  -sin(theta + x)|
|                |
|sin(theta + x)   cos(theta + x)|

octave> kinv=subs(rotinv,theta,x+theta)
kinv = (sym 2x2 matrix)

|cos(theta + x)   sin(theta + x)|
|                |
|-sin(theta + x)  cos(theta + x)|

octave> k*kinv
ans = (sym 2x2 matrix)

|      2          2          |
|sin (theta + x) + cos (theta + x)      0      |
|                |
|                2          2          |
|                0          sin (theta + x) + cos (theta + x)|

octave> simplify(k*kinv)
ans = (sym 2x2 matrix)

|1  0|
|  |
|0  1|

```