Uso de OCTAVE y GNUPLOT en Química

Víctor Luaña

(con la colaboración de Alberto Otero y J. M. Recio)

Departamento de Química Física y Analítica, Universidad de Oviedo

Apéndice A. GNUplot y octave

GNUPlot es un programa de dibujo 2D y 3D distribuido gratuitamente como código libre. Produce de modo muy sencillo gráficas de funciones y datos experimentales. El usuario dispone de un control preciso del aspecto del dibujo y de un enorme número de formatos de salida. Proporciona además una herramienta simple y eficiente para realizar un ajuste de mínimos cuadrados de una función no lineal a una colección de datos experimentales. En conjunto se trata de una poderosa herramienta que merece la pena conocer y dominar.

Octave es un lenguaje de programación diseñado para facilitar el acceso a técnicas muy avanzadas de tratamiento de matrices y vectores con un esfuerzo mínimo. Preparado para ser usado interactivamente como si se tratara de una calculadora muy sofisticada, el lenguaje incorpora rutinas para hacer integrales definidas, resolver sistemas lineales, encontrar valores y vectores propios, resolver ecuaciones diferenciales, etc. Con muy poco esfuerzo de aprendizaje el alumno estará rápidamente capacitado para abordar problemas del curso que serían esencialmente intratables sin el uso de un ordenador.

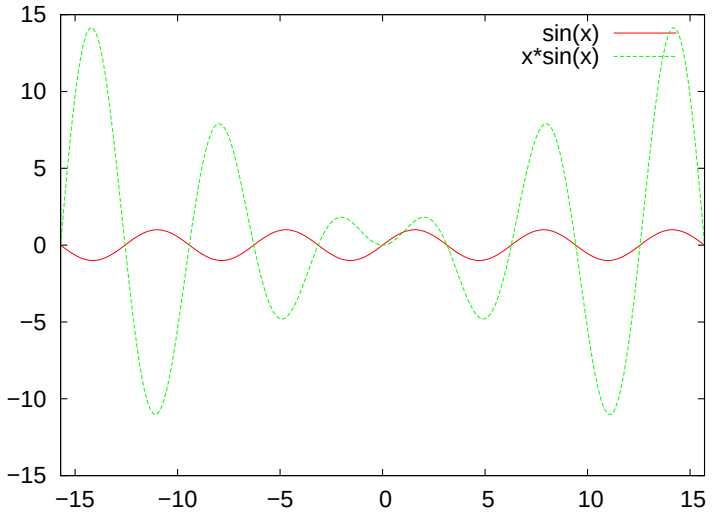
GNUPlot: instalación y primeros dibujos.

Fuente y ejecutables están disponibles en http://www.gnuplot.info.

Uso interactivo y no interactivo:

- Un dibujo sencillo se puede crear interactivamente.
- Las órdenes de dibujo de la sesión interactiva se pueden recuperar y modificar con los cursores.
- Una vez tengamos un dibujo listo, podemos archivar el conjunto completo de instrucciones necesarias para repetirlo con: `save "dibujo.gnu"`.
- Podemos recuperar un archivo anterior en una nueva sesión interactiva: `load "dibujo.gnu"`.
- También podemos hacer que gnuplot *ejecute* el dibujo definido en un archivo:
  - `gnuplot dibujo.gnu` desde una terminal.
  - *abriendo* `dibujo.gnu` con gnuplot en un entorno de escritorio.
- Con un poco de práctica, el archivo de órdenes se crea a mano, y éste llega a ser el modo más eficiente de trabajar cuando se domina el programa o cuando se quiere controlar con gran detalle el aspecto del dibujo.
- El programa permite consultar interactivamente la documentación: `help orden`.
- Hay un extenso manual y un gran número de dibujos de ejemplo en http://www.gnuplot.info.
- La libreta de problemas de química física cuenta con un buen número de gráficos explicados detalladamente: http://www.uniovi.es/qcg/d-qf2/Problemas.pdf.

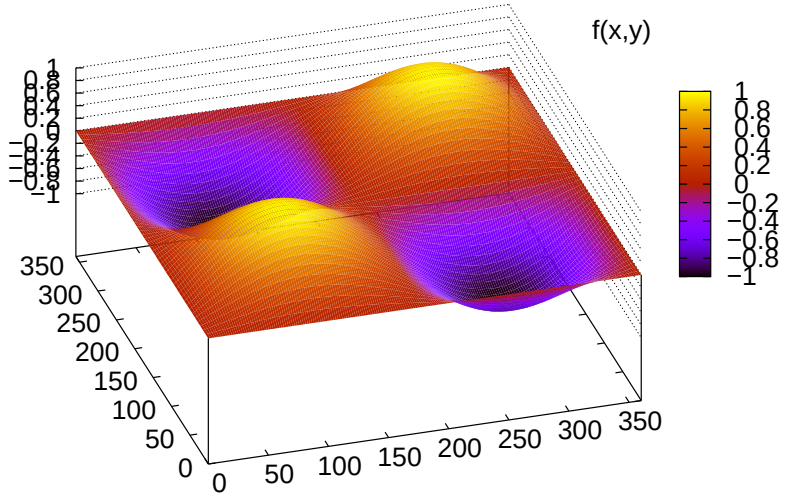
```
1 plot sin(x), x * sin(x)
```




```
1 plot [0:2*pi] sin(x), x*sin(x)
```

```
1 set samples 201
2 set xrange [-5*pi : 5*pi]
3 plot sin(x), x*sin(x)
```

```
1 set isosamples 51,51
2 set grid xtics ytics ztics
3 set view 41,343
4 set xrange [0:360]
5 set yrange [0:360]
6 radian(x) = x * pi / 180
7 f(x,y) = sin(radian(x))*sin(radian(y))
8 splot f(x,y) with pm3d
```



Dibujos de curvas y superficies:

Tipo	orden	dibujo
1D	`plot f(x), ...`	$y = f(x)$
2D	`splot f(x,y), ...`	$z = f(x, y)$

Dibujos paramétricos (se activan con `set parametric` y se desactivan con `unset parametric`):

Tipo	orden	dibujo
1D	`plot x(t),y(t), ...`	$\vec{r}(t) = [x(t), y(t)]$
2D	`splot x(u,v),y(u,v),z(u,v) ...`	$\vec{r}(u, v) = [x(u, v), y(u, v), z(u, v)]$

Una orden `plot` más completa:

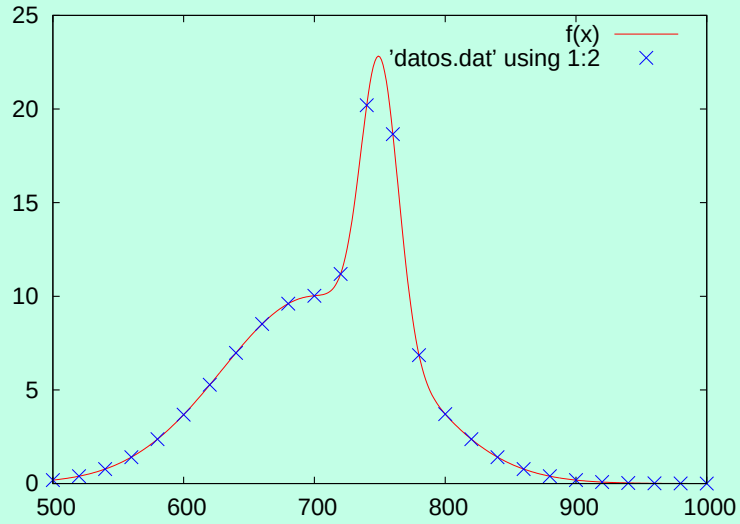
```
plot [<definiciones>,] [<rango>] [<iteracion>]
     {<funcion> | "<archivo.dat>" [modificadores]}
     [axes <ejes>] [<titulo>] [with <estilo>]
     [, ...]
```

[opcional]
{obligatorio}
uno | u otro

Ajustes de mínimos cuadrados empleando gnuplot:

Archivo `datos.dat`:

500	0.1831564
520	0.3916390
...	...
1000	0.0012341




```
1 # Funcion de ajuste: suma de dos gausianas
2 f(x) = A1 * exp(-((x-m1)/s1)**2) + A2 * exp(-((x-m2)/s2)**2)
3 # Ajuste no lineal de minimos cuadrados
4 m1 = 650; m2 = 750; s1 = 20; s2 = 5; A1 = 10; A2 = 20;
5 fit f(x) 'datos.dat' using 1:2 via m1,s1,A1,m2,s2,A2
6 # Dibujo de los datos y la funcion, para comprobar
7 set style line 2 lt 3 lw 1 pt 2 ps 2.0
8 plot f(x), 'datos.dat' using 1:2 ls 2
```

Octave: instalación y primeros pasos.

Fuente y ejecutables están disponibles en `http://www.octave.org`. Un extenso número de librerías adicionales están disponibles en `http://octave.sourceforge.net`.

Octave sirve como una calculadora científica avanzada que sabe trabajar con vectores, matrices, números complejos, resolver sistemas lineales, ecuaciones diferenciales, etc. En los cálculos más sencillos usaremos octave interactivamente, escribiendo órdenes y viendo inmediatamente los resultados. Para tareas más complicadas es mejor programar previamente el trabajo y editar un archivo con las órdenes para octave. Los archivos con programas octave llevan el apellido `.m`.

- **Ayuda integrada:** Octave incorpora su propia documentación: `help orden`.
- **La asignación** es la instrucción básica de octave:

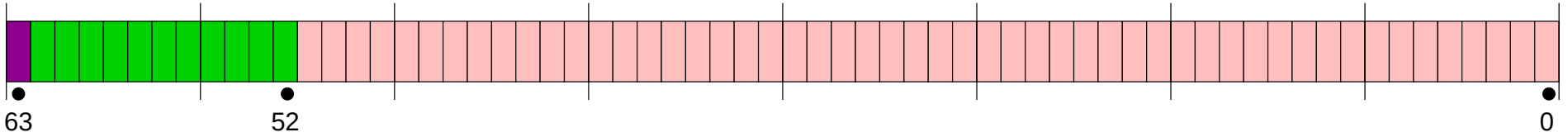
```
1    variable = operacion
```

  - Se realiza la operación y el resultado se almacena en la variable para poder ser usado posteriormente. No debe entenderse como una igualdad matemática. La asignación `x = x+1` es perfectamente válida (aumenta en 1 el valor anterior de `x`).
  - Los nombres de variable comienzan por una letra, seguida de letras, dígitos decimales, y algunos símbolos como '_'. Ej.: `pepe`, `x_123`. Mayúsculas y minúsculas son símbolos diferentes.
  - Una operación puede contener: números, variables, operaciones aritméticas, funciones, ...

- **Números en formato de coma flotante:**

Ejemplo:	+1.234e-73	123	123.	-.12	1e10	.14d+23
Equivale:	$+1{,}234 \cdot 10^{-73}$	123	123,0	$-0{,}12$	$1 \cdot 10^{10}$	$0{,}14 \cdot 10^{+23}$

IEEE 754 Doble precisión (64 bits)



Signo: 1 bit

Exponente: 11 bits

Mantisa: 52 bits

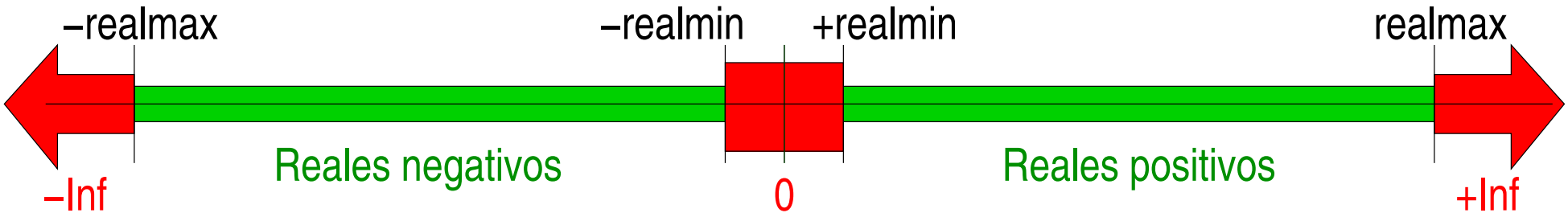
realmax: $1.797... \; 10^{308}$

realmin: $2.225 ... \; 10^{-308}$

precisión: unas 16 cifras decimales

$(1 + eps) - 1 \rightarrow 0$

eps: $2.220 ... \; 10^{-16}$



• Operaciones aritméticas (escalares):

Operaciones: `+` (suma), `-` (resta), `*` (multiplicación), `/` (división), `**` ó `^` (potencia).

No hay producto implícito: es `2*x`, no `2x`. Ejemplos: `2*pi/3`, `3/2 + 12 * sin(pi/3/2) + 5`.

Precedencia:

- (de mayor a menor) `( )` ⟶ signo ⟶ `**` ⟶ `*` y `/` ⟶ `+` y `-`.
- De izquierda a derecha (⟶) en caso de igual precedencia.
- En caso de duda, usar paréntesis: `( )`.

Ejemplos: `2**3**4` ($((2^3)^4)$), `pi/2/3` ($\pi/6$), `pi/(2/3)` ($3\pi/2$).

• Algunas funciones intrínsecas:

`sin(x)`, `cos(x)`, `tan(x)`	x en radianes
`asin(x)`, ...	
`atan2(x,y)`	$\arctan(x/y)$
`log(x)`, `log10(x)`	Ojo: $\ln x$ y $\log[10]x$
`abs(x)`, `exp(x)`, `sqrt(x)`	$\|x\|$, e^x, $\sqrt{x}$

`ceil(z)`	menor entero no inferior
`floor(z)`	mayor entero no superior
`fix(z)`	parte entera
`round(z)`	entero más próximo
`rem(x,y)`	resto de la división entera

• Definir vectores y matrices:

- **Rango:** `inicio:fin` (de 1 en 1) ó `inicio:salto:fin`.
  Ejemplo: `1:10`, `0:pi/10:pi`, `3:-0.1:0`.
  También: `linspace(inicio, fin, num)`, `logspace(inicio, fin, num)`.
- **Vector fila:** `[e1, e2, e3, ...]`.
  Ejemplo: `[12.3, pi/3, 12**2, sin(pi/6)]`.
- **Vector columna:** `[e1; e2; e3; ...]`.
  Ejemplo: `[1; 2; 3]`.
- **Matriz:** `[e11, e12, e13, ...; e21, e22, e23, ...; ...]`.
  Ejemplo: `[1, 2, 3; 4, 5, 6; 7, 8, 9]` produce $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.
- Las funciones intrínsecas (sin, cos, exp, ...) pueden recibir un vector o matriz como argumento y devolver un vector o matriz como resultado. Ejemplo: `y = sin(0 : pi/12 : 2*pi)`.
- Transponer filas en columnas: `x'` ó `transpose(x)`.
- Podemos componer una matriz juntando vectores fila o columna: Si `c1=[1;2;3]` y `c2=[4;5;6]` examina `[c1,c2]` y `[c1';c2']` (prueba otras posibilidades).

• Controlar el resultado de una operación:

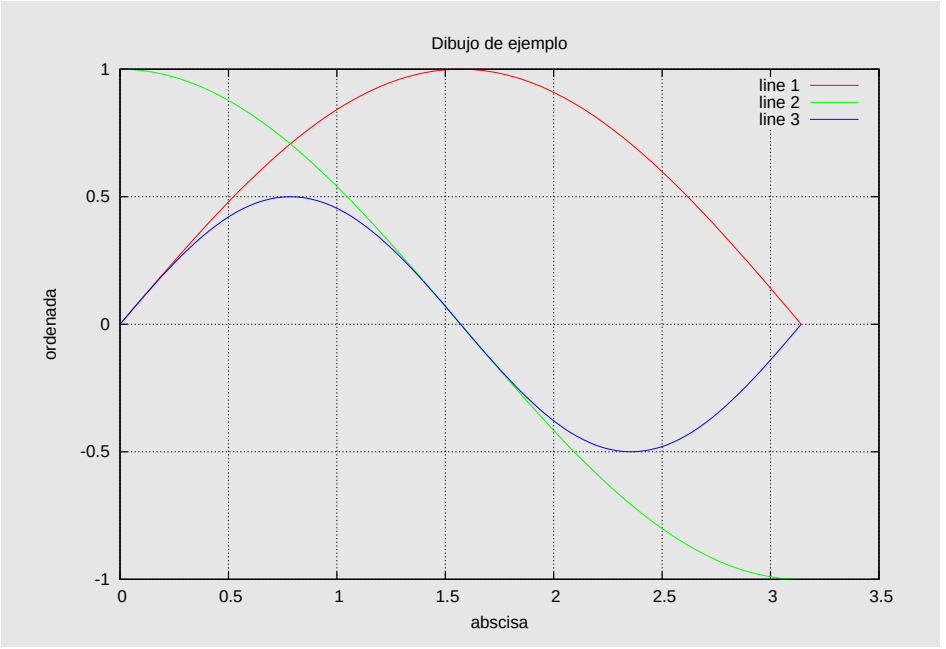
- Si una operación finaliza con punto y coma (`operacion;`) no se imprime su resultado.
- Varias asignaciones pueden escribirse como una sóla orden separamos por punto y coma.
  Ejemplo: `x = 0:pi/6:2*pi; y = sin(x); z = log(y);`
- `format <formato>` permite controlar el aspecto de los resultados de una operación. *Esto no afecta a la precisión interna de los cálculos*.
  `format short; pi` produce `3.1416`.
  `format long; pi` produce `3.14159265358979`.
  Consultar otras opciones: `help format`.
- El resultado más reciente de una operación que no haya sido asignado explícitamente a una variable se puede recuperar de la variable por defecto `ans` .
  Ejemplo: `pi/3; pi/6; pi/8` produce `ans = 0.39270`.
- `diary on` comienza a guardar las órdenes y resultados que se produzcan en un fichero llamado "diary" en el directorio de trabajo.
  `diary off` finaliza el archivo de órdenes y resultados.
  `diary 'fichero'` establece 'fichero'como cuaderno de bitácora y lo activa.
- `disp(X)` imprime en la pantalla el valor de `X`.

• Tablas de valores de una o varias funciones:

1. Rejilla de valores de la variable independiente:
   - A mano: `x = [0, pi/6, pi/4, pi/3, pi];`
   - Un rango: `x = xini : dx : xfin;`
   - `x = linspace(xini, xfin, n);`
2. Evaluamos las funciones: `y = sin(x); z = cos(x);` ... Pueden ser funciones propias, pero deben estar preparadas para recibir y devolver un vector.
3. `disp([x; y; z])` produce
$$\begin{array}{cccc} x_1 & x_2 & x_3 & \cdots \\ y_1 & y_2 & y_3 & \cdots \\ z_1 & z_2 & z_3 & \cdots \end{array}$$
4. `disp([x', y', z', t'])` produce
$$\begin{array}{cccc} x_1 & y_1 & z_1 & t_1 \\ x_2 & y_2 & z_2 & t_2 \\ x_3 & y_3 & z_3 & t_3 \\ & \cdots & & \end{array}$$
5. `printf('%8.3f %12.6f %12.6f\n', [x; y1; y2])` permite controlar el formato de cada columna.

• Dibujo de una o varias funciones de una variable:

```
1  x = linspace(0, pi, 101);
2  y1 = sin(x);
3  y2 = cos(x);
4  y3 = sin(x) .* cos(x);
5  xlabel('abscisa')
6  ylabel('ordenada')
7  title('Dibujo de ejemplo')
8  grid
9  plot(x,y1, x,y2, x,y3)
10 print('dibujo1.eps', '-depsc2')
```



También produce el mismo dibujo: `plot(x, [y1; y2; y3])`.

Otros estilos de dibujo 2D: semilogx, semilogy, loglog, polar, bar, pie, hist, stem, stairs, ...

Dibujos 3D: plot3, meshgrid, contour, ...

Mirar también: axis, legend, text, replot, subplot, figure.

Agregar dibujos sobre uno previo: `hold on`, `hold off`. Varios dibujos en una página: `subplot(rows,cols,index)`. Volcar un dibujo a un fichero: `print("archivo", "estilo")`.

• Operaciones con complejos:

- Un complejo en forma cartesiana: `z = 2 + 3i` .
  `i,j,I,J` están inicialmente asociados a `(0+1i)`.
  También: `z = x + i * y` , siendo `x` e `y` las partes real e imaginaria del número.

- Un complejo en forma polar: `z = r * exp(i*theta)` .
  De la forma cartesiana a la polar: $r = |z| = \sqrt{x^2 + y^2}$, $\theta = \arctan(y/x)$.
  De la forma polar a la cartesiana: $x = r\cos\theta$, $y = r\,\mathrm{sen}\,\theta$.

- Fórmula de Euler: $e^{i\theta} = \cos\theta + i\,\mathrm{sen}\,\theta$.

- Funciones como `sin()`, `cos()`, `sqrt()`, etc, operan sobre complejos o sobre reales.

- Funciones específicas para manipular complejos:
  `abs(z)` : módulo de `z`.
  `angle(z)` **ó** `arg(z)` : ángulo de fase.
  `conj(z)` : conjugado complejo (z^*).
  `real(z)` : parte real de `z`.
  `imag(z)` : parte imaginaria de `z`.

• Operaciones con vectores y matrices:

Notación: $\underline{\underline{A}}, \underline{\underline{B}}, \cdots$ matrices; $\underline{x}, \underline{y}, \cdots$ vectores fila; $m, n, \cdots$ dimensiones; $i, j, \cdots$ índices.

operación	de matrices		elemento a elemento	
suma	`C = A+B`	$C_{ij} = A_{ij} + B_{ij}$	`C = A.+B`	lo mismo
resta	`C = A-B`	$C_{ij} = A_{ij} - B_{ij}$	`C = A.-B`	lo mismo
producto	`C = A*B`	$C_{ij} = \sum_k A_{ik} B_{kj}$	`C = A.*B`	$C_{ij} = A_{ij} B_{ij}$
potencia	`C = A**n`	$\underline{\underline{C}} = \underline{\underline{A}}\,\underline{\underline{A}} \cdots \underline{\underline{A}}$	`C = A.**n`	$C_{ij} = A_{ij}^n$
división por la derecha	`C = A/B`	$\underline{\underline{C}} = [(\underline{\underline{B}}^{\mathsf{T}})^{-1} \underline{\underline{A}}^{\mathsf{T}}]^{\mathsf{T}}$	`C = A./B`	$C_{ij} = A_{ij} / B_{ij}$
división por la izquierda	`C = A\B`	$\underline{\underline{C}} = \underline{\underline{A}}^{-1} \underline{\underline{B}}$	`C = A.\B`	$C_{ij} = B_{ij} / A_{ij}$
transpuesta conjugada	`C = A'`	$C_{ij} = A^*_{ji}$		
transpuesta	`C = A.'`	$C_{ij} = A_{ji}$		

Producto de vectores:

Escalar	$s = \vec{x} \cdot \vec{y}$	`x*y'` ó `dot(x,y)`
Vectorial	$\vec{z} = \vec{x} \times \vec{y}$	`cross(x,y)`
Directo o cartesiano	$\underline{\underline{A}} = \vec{x} \otimes \vec{y}$	`x'*y`

Propiedades de vectores y matrices:

`det(A)`	determinante de una matriz cuadrada
`trace(A)`	traza de la matriz ($\sum_i A_{ii}$)
`rank(A)`	rango (dimensión del mayor determinante no nulo)
`conj(A)`	matriz conjugada
`inv(A)`	matriz inversa
`inverse(A)`	matriz inversa
`[vec,val] = eig(A)`	vectores y valores propios
`[R,K] = rref(A)`	forma escalonada reducida por filas
`sum(A,dim)`	suma de elementos según la dimensión `dim` (1 por defecto)
`sum(sum(A))`	suma de todos los elementos de la matriz
`sumsq(A,dim)`	suma de cuadrados
`diff(x,k)`	diferencias finitas de orden k del vector $\vec{x}$
`[m,n] = size(A)`	dimensiones ($m \times n$) de la matriz
`length(A)`	dimensión máxima de la matriz

• Matrices especiales:

`eye(n)`	Devuelve la matriz unidad $n \times n$.
`ones(m,n)`	Matriz $m \times n$ formada por unos. La forma `ones(n)` equivale a `ones(n,n)`.
`zeros(m,n)`	Matriz $m \times n$ formada por ceros. La forma `zeros(n)` equivale a `zeros(n,n)`.
`repmat(A,m,n)`	Matrix $m \times n$ que tiene `A` en cada bloque.
`rand(m,n)`	Matriz $m \times n$ formada por números aleatorios uniformemente distribuidos en $[0, 1)$. Es válida la variante `rand(n)` (matriz $n \times n$) y `rand` (un sólo número). Con `rand("seed")` se puede consultar el valor que tiene la semilla inicial del generador, y con `rand("seed",valor)` se puede establecer una nueva semilla.
`randn(m,n)`	Matriz de números aleatorios normalmente distribuidos, con centro en 0 y desviación típica unidad. Acepta las mismas variantes que `rand()`.
`rande(m,n)`	Matriz de números aleatorios que siguen una distribución exponencial.
`randp(m,n)`	Matriz de números aleatorios que siguen una distribución de Poisson.
`randperm(n)`	Permutación aleatoria de los enteros $1 \ldots n$.
`diag(v,k)`	Crea una matriz con el vector `v` ocupando la diagonal `k`.
`vander(v)`	Crea una matriz de Vandermonde cuya penúltima columna es el vector `v`.

• Seleccionar elementos y bloques de un vector o matriz:

`A`	Matriz o vector completo
`A(i,j)`	Elemento A_{ij}
`x(i)`	Elemento x_i
`A(i,:)`	Fila i-ésima de la matriz (vector fila)
`A(:,j)`	Columna j-ésima de la matriz (vector columna)
`A(1:3,:)`	Las tres primeras filas de $\underline{\underline{A}}$
`A([1,7,5],[2,8])`	Bloque 3×2 de la matriz, resultado de la intersección de las filas 1, 7 y 5 y de las columnas 2 y 8
`A(:)`	Matriz convertida en un vector columna y ordenada por columnas
`i = find(A>0.9)`	Indice de los elementos que cumplen $A_{ij} > 0.9$
`A(i)`	Elementos correspondientes al índice `i`
`A(find(A<=0.5))`	Elementos que cumplen $A_{ij} \leq 0.5$

- `A(:,[1,2]) = A(:,[2,1]);` # intercambio de dos columnas de una matriz.
- `A = 3*ones(5); A(2:4,2:4) = 2; A(3,3) = 1;` # produce $\begin{pmatrix} 3 & 3 & 3 & 3 & 3 \\ 3 & 2 & 2 & 2 & 3 \\ 3 & 2 & 1 & 2 & 3 \\ 3 & 2 & 2 & 2 & 3 \\ 3 & 3 & 3 & 3 & 3 \end{pmatrix}$

• Resolver un sistema lineal de ecuaciones:

Si hay tantas ecuaciones como incógnitas y la matriz de coeficientes no es singular, la solución es única:

$$\left.\begin{array}{rcrcrcr} 2x & + & 3y & - & z & = & 5 \\ 6x & - & 2y & + & z & = & 33 \\ x & + & y & + & z & = & -2 \end{array}\right\} \Longrightarrow \begin{bmatrix} 2 & 3 & -1 \\ 6 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 33 \\ -2 \end{bmatrix} \Longrightarrow \underline{\underline{A}}\,\underline{x} = \underline{b} \Longrightarrow \boxed{x = A\backslash b}.$$

En forma más general:

- M: número de variables. Determina la dimensión del espacio de trabajo ($\mathbb{R}^M$).
- N: número de ecuaciones. De momento, sólo examinaremos el caso $M \geq N$.
- R: rango de la matriz de coeficientes (`R = rank(A)`). Número de ecuaciones linealmente independientes.
- $L = M - R$: grados de libertad. Número de variables que son independientes entre sí y que no están especificadas por el sistema.
- La solución del sistema es un subespacio $\mathbb{R}^L$ en el espacio $\mathbb{R}^M$ del problema.
  $L = 0$: Solución única (un punto).
  $L = 1$: la solución es una recta.
  $L = 2$: la solución es un plano.

Las soluciones de un sistema lineal no se modifican si se somete a los siguientes cambios:

- una ecuación se intercambia por otra.
- una ecuación se multiplica por una constante no nula.
- una ecuación se sustituye por una suma de sí misma y un múltiplo de otra ecuación.
- dos variables se intercambian entre sí en todas las ecuaciones (intercambio de columnas), aunque hay que tener en cuenta el cambio para interpretar la solución final.

Mediante estos cambios, todo sistema lineal se puede convertir a una forma escalonada por filas, o forma de Gauss-Jordan. Ejemplo:

$$\left.\begin{array}{rcrcrcl} x & + & y & + & z & + t & = 1 \\ 2x & + & 3y & - & 5z & + t & = 2 \\ 3x & + & 2y & + & 5z & - t & = 3 \end{array}\right\} \Rightarrow \boxed{\underline{\underline{A}}\,\underline{x} = \underline{b}} \Rightarrow \left\{\begin{array}{rcrcrcl} x & + & 0 & + & 0 & - 6t & = 1 \\ 0 & + & y & + & 0 & + 6t & = 0 \\ 0 & + & 0 & + & z & + t & = 0 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} x = 1 + 6t \\ y = -6t \\ z = -t \end{array}\right.$$

Es muy cómodo trabajar con la matriz de coeficientes ampliada con la columna de términos independientes:

$\boxed{\underline{\underline{B}} = (\underline{\underline{A}}|\underline{b}).}$ En el ejemplo: $\left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 1 \\ 2 & 3 & -5 & 1 & 2 \\ 3 & 2 & 5 & -1 & 3 \end{array}\right) \Rightarrow \left(\begin{array}{cccc|c} 1 & 0 & 0 & -6 & 1 \\ 0 & 1 & 0 & 6 & 0 \\ 0 & 0 & 1 & 6 & 0 \end{array}\right)$

Si el rango de la matriz ampliada supera al rango de la matriz de coeficientes, el sistema es imposible.

Usando octave, ingresaríamos la matriz de coeficientes y la de términos independientes del ejemplo mediante:

```
A = [1, 1, 1, 1;  2, 3, -5, 1;  3, 2, 5, -1];
b = [1; 2; 3];
```

Podemos construir la matriz ampliada y ver los rangos de $\underline{\underline{A}}$ y $\underline{\underline{B}}$ (3 y 3 en este caso) con

```
B = [A, b];
rank(A)
rank(B)
```

El sistema se puede reducir a la forma escalonada empleando la función `rref(B)`:

```
[R, K] = rref(B)
R =  1.00000   0.00000   0.00000  -6.00000   1.00000
     0.00000   1.00000   0.00000   6.00000   0.00000
     0.00000   0.00000   1.00000   1.00000   0.00000
K =  1  2  3
```

En este resultado, `R` contiene la forma escalonada, y `K` indica cuáles son las variables que han sido reducidas. La rutina `rref()` forma parte de la librería octave forge.

• Valores y vectores propios de una matriz cuadrada:

Sea $\underline{\underline{A}}$ una matriz $n \times n$, $\vec{x}_i$ un vector propio (columna) asociado al valor propio λ_i. Se cumplen las ecuaciones

$$\underline{\underline{A}}\,\underline{x}_i = \lambda_i \underline{x}_i \quad \Leftrightarrow \quad \underline{\underline{A}}\,\underline{\underline{X}} = \underline{\underline{X}}\,\underline{\underline{\Lambda}} \quad \Leftrightarrow \quad \underline{\underline{\Lambda}} = \underline{\underline{X}}^{-1}\,\underline{\underline{A}}\,\underline{\underline{X}}$$

donde $\underline{\underline{X}}$ es la matriz que tiene a los vectores propios por columnas, es decir $[\underline{x}_1 \underline{x}_2 \cdots \underline{x}_n]$, y $\underline{\underline{\Lambda}}$ es la matriz diagonal de los vectores propios. Se dice que $\underline{\underline{X}}$ diagonaliza la matriz $\underline{\underline{A}}$ mediante la transformación de semejanza $\underline{\underline{X}}^{-1}\underline{\underline{A}}\underline{\underline{X}}$.

Otros nombres: autovectores, *eigenvectors*, autovalores, *eigenvalues*.

Dos casos nos interesan particularmente:

Si $\underline{\underline{A}}$ es simétrica $(\underline{\underline{A}}^{\mathsf{T}} = \underline{\underline{A}})$ $\underline{\underline{X}}$ es ortogonal $(\underline{\underline{X}}^{\mathsf{T}} = \underline{\underline{X}}^{-1})$ y los valores propios son números reales.

Si $\underline{\underline{A}}$ es hermítica $(\underline{\underline{A}}^{\mathsf{H}} = \underline{\underline{A}})$ $\underline{\underline{X}}$ es unitaria $(\underline{\underline{X}}^{\mathsf{H}} = \underline{\underline{X}}^{-1})$ y los valores propios son números reales.

(Nota: recuerda que $\underline{\underline{A}}^{\mathsf{H}} = (\underline{\underline{A}}^{*})^{\mathsf{T}}$.)

Empleando octave:

```
1    [vectores, valores] = eig(A);
```

• Polinomios:

Un polinomio de grado n se define por su matriz de coeficientes:

$$P_n(x) = c_1 x^n + c_2 x^{n-1} + \cdots + c_n x + c_{n+1}$$

Las principales funciones de octave en el trabajo con polinomios son:

- `y = polyval(c,x)` Evalúa el polinomio de coeficientes `c` en los puntos que indica el vector `x`. El resultado es un vector de valores.
- `c = polyfit(x,y,n)` Ajusta, en el sentido de mínimos cuadrados, un polinomio de grado n a los datos contenidos en los vectores `x` e `y`. El resultado es el vector de coeficientes.
- `r = roots(c)` Encuentra las raices del polinomio de coeficientes `c`.
- `polyout(c, VAR)` Escribe el polinomio de coeficientes `c` empleando `VAR` como nombre de la variable (por defecto se llama `'s'`).
- `y = conv(a,b)` Si `a` y `b` son los coeficientes de sendos polinomios, `conv(a,b)` devuelve los coeficientes del polinomio producto. En general, devuelve la *convolución* de a y b.
- `[b,r] = deconv(y,a)` Encuentra cociente y resto de la división de polinomios, de modo que `y=conv(a,b)+r`.

Otras funciones relacionadas con polinomios son: `compan`, `filter`, `poly`, `polyder`, `polyderiv`, `polyinteg`, `polyreduce`, `polyvalm`, `residue`.

• Definir nuevas funciones:

```
function argumentos_salida = nombre(argumentos_entrada)
   instrucciones de la funci n
endfunction
```

Ejemplo:

```
function y = mifun(x)
   y = 3*x**2 + 5*sin(x/2);
endfunction
mifun(3*pi/8)
for x = 0:pi/10:pi
   disp([x, mifun(x)])
endfor
```

Función vectorizada: es una función preparada para recibir un escalar/vector/matriz de datos y devolver, a su vez, un escalar/vector/matriz de resultados. Muchas rutinas de integración, minimización, etc, requieren que se les pasen funciones vectorizadas. Como receta sencilla, las operaciones internas deben ser las variantes *elemento a elemento*: `.*`, `./`, `.**` ...

Ejemplo:

```
function y = mifun(x)
   y = 3.*x.**2 .+ 5.*sin(x./2);
endfunction
x = 0:pi/10:pi; y = mifun(x); plot(x,y)
```

Argumentos de entrada:

- En la definición de la función se trata de una lista de variables separadas por comas. Cada variable puede ser un escalar, vector, matriz, ...
- Al llamar una función, cada argumento puede ser una expresión: *los argumentos se pasan por valor*.
- Los argumentos en la llamada y en la definición se asocian por orden y no por nombre.

Argumentos de salida:

- En la definición de la función se trata de una variable, o de una colección de variables encerradas entre corchetes y separadas por comas. Se supone que los argumentos de salida reciben un valor dentro de la función, y este es el resultado que se devuelve al programa de llamada.

Las funciones constituyen el mecanismo de octave para facilitar la programación en módulos independientes. Cada función es un módulo, con sus propias variables locales. Los argumentos de entrada y salida (o una instrucción `global`) son los únicos mecanismos por los que las funciones se transmiten información.

Características avanzadas:

- El número de argumentos de entrada y salida puede ser variable. Las funciones `nargin()` y `nargout()` permiten conocer el número de argumentos que se han usado en la llamada y que se esperan como resultado de la función, respectivamente.

- Las funciones de octave admiten recursión directa e indirecta.
- Una variable puede ser declarada `global` para ser compartida en varias funciones. Todas las funciones que la necesiten deben declararla. Esto permite una cadena de llamadas: A llama a B, B llama a C, en la que A y C pero no B compartan una variable global.
- Por defecto, cuando acaba una rutina sus variables desaparecen y pierden su valor. Esto puede evitarse declarando una variable como `persistente`.
- La instrucción `return` puede usarse para interrumpir una rutina y devolver, de inmediato, el control al programa de llamada.
- Una función llamada 'func'y almacenada en el fichero 'func.m'será visible para octave siempre que el fichero se encuentre en un directorio definido en el árbol de llamadas (véase `path` y `addpath`). También puede invocarse explícitamente con `source('func.m')`.

• Raices de ecuaciones $\vec{f}(\vec{x}) = \vec{0}$:

Llamada general: [1]

```
[x, info, msg] = fsolve(fcn, x0)
```

fcn: Nombre de la función que calcula las $\vec{f}(\vec{x})$.

x0: Vector que contiene el valor inicial de las variables. Si el sistema tiene múltiples soluciones será necesario realizar una búsqueda cuidadosa comenzando en diferentes puntos.

x: Solución encontrada.

info: Condición de terminación opcional. Puede consultarse su significado con `perror('fsolve',info)`.

msg: Mensaje de terminación opcional.

Ejemplo:[1]

```
[x, info, msg] = fsolve('(1-2*x)*cos(x)', 0.1)
```

Ejemplo: Resolver el siguiente sistema no lineal partiendo del punto inicial $(x, y) = (1, 2)$.

$$f_1(x, y) = -2x^2 + 3xy + 4\,\mathrm{sen}\, y - 6 = 0, \qquad f_2(x, y) = 3x^2 - 2xy^2 + 3\cos x + 4 = 0.$$

```
function y = f(x)
   y(1) = -2*x(1)**2 + 3*x(1)*x(2) + 4*sin(x(2)) - 6;
   y(2) = 3*x(1)**2 - 2*x(1)*x(2)**2 + 3*cos(x(1)) + 4;
endfunction
[x, info] = fsolve("f", [1;2])
```

```
   ==> x: 0.57983 2.54621
   ==> info: 1
perror('fsolve', info)
   ==> solution converged to requested tolerance
```

En el caso de sistemas multidimensionales, `fsolve`, puede utilizar la información del Jacobiano del sistema, es decir, de las derivadas parciales $J_{ij} = \partial f_i / \partial x_j$. Para ello, debe haber una función que proporcione las componentes de la matriz J_{ij}, y el argumento de llamada `fcn` debe consistir en un vector con los nombres de la función que evalúa $\vec{f}(\vec{x})$ y de la que lo hace con J_{ij}. Es el caso del ejemplo siguiente

```
function y = fun(x)
   y(1) = -2*x(1)**2 + 3*x(1)*x(2) + 4*sin(x(2)) - 6;
   y(2) = 3*x(1)**2 - 2*x(1)*x(2)**2 + 3*cos(x(1)) + 4;
endfunction
function J = jac(x)
   J(1,1) = -4*x(1) + 3*x(2);
   J(1,2) = +3*x(1) + 4*cos(x(2));
   J(2,1) = 6*x(1) - 2*x(2)**2 - 3*sin(x(1));
   J(2,2) = -4*x(1)*x(2);
endfunction
[x, info] = fsolve(["fun","jac"], [1;2])
```

• Integrales definidas:

Octave dispone de la función `quad()` para evaluar numéricamente integrales definidas. Un ejemplo sencillo sería

```
1 K = quad('sin', 0, pi/3)
```

$$K = \int_0^{\pi/3} \operatorname{sen} x \, dx$$

Una forma más completa de la orden es:

```
1    [valor, ier, nfun, err] = quad(f, a, b, TOL, SING)
```

Los argumentos de entrada son:

- `f`: nombre de la función que calcula en integrando. Normalmente será una función de usuario, y debe estar preparada para que `quad()` la llame como `y=f(x)`.
- `a` y `b`: límites de integración. Pueden ser `+Inf` y `-Inf`.
- `TOL`: tolerancia o precisión requerida de la integral. Si se trata de un vector, el primer elemento indica una tolerancia absoluta y el segundo elemento una relativa.
- `SING`: si el integrando presenta singularidades, la rutina de integración tratará con particular cuidado los puntos que se indiquen en el vector `SING`.

Los argumentos de salida son:

- `valor`: resultado de la integración.
- `ier`: posible condición de error. El valor de `ier` será no nulo en el caso de que haya habido algún problema o no se haya alcanzado la precisión requerida.
- `nfun`: número de evaluaciones del integrando que se han realizado.
- `err`: estimación del error de la solución.

Un ejemplo más complicado: $K = \int_0^3 x \operatorname{sen}(1/x)\sqrt{|1-x|}\, dx.$

```
1  function y = integrando(x)
2     y = x .* sin(1./x) .* sqrt(abs(1.-x));
3  endfunction
4  [K, ier, nfun, err] = quad("integrando", 0, 3)
5     => 1.9819
6     => 1
7     => 5061
8     => 1.1522e-07
```

Una representación del integrando permite apreciar las complejidades de esta integral. Pese a la bandera de error (`ier=1`) el resultado es bastante preciso.

El paquete `integration` del conjunto de Octave-forge incluye rutinas de integración en dos o más dimensiones. Pueden consultarse: `gquadnd`, `gquad2dgen`, `quad2dcgen`, `quad2dggen`, `quadndg`, ...

• Ecuaciones diferenciales ordinarias:

La rutina `lsode()` resuelve sistemas de ecuaciones diferenciales de primer orden, $\dot{\vec{r}} = \vec{f}(\vec{r}, t)$, con condiciones iniciales $\vec{r}(t_0) = \vec{r}_0$:

```
1 [x, info,msg] = lsode(fcn, x0, t, tcrit)
```

fcn: Nombre de la función que calcula las $\vec{f}(\vec{r}, t)$. En algunos problemas es útil crear una segunda rutina que calcule la matriz jacobiana, $J_{ij} = \partial f_i / \partial x_j$. En ese caso `fcn` es `['fun';'jac']` (string array) ó `{'fun';'jac'}` (cell array). Las llamadas a estas rutinas serán `fun(x,t)` y `jac(x,t)`, donde `x` es un vector y `t` un escalar.

x0: Condiciones iniciales $\vec{r}(t_0) = \vec{r}_0$.

t: Vector con los valores de t en los que debe evaluarse la función integrada. Generalmente se crea superponiendo una o varias llamadas a `linspace()` o `logspace()`.

tcrit: Puntos singulares que deben ser evitados (opcional).

x: Matriz de resultados.

info: Condición de terminación opcional (2 si todo ha ido bien).

msg: Mensaje de terminación opcional.

La rutina `lsode_options()` puede utilizarse para modificar el funcionamiento de `lsode()`.

Ejemplo: Integrar $\dot{x} = x \cos t$ con $x(0) = 1$ en $t \in [0, 2\pi]$ (Solución exacta: $x = e^{\text{sen}\, t}$).

```
t = 0:pi/51:2*pi;
[x,info,msg] = lsode('x*cos(t)', 1, t);
plot(t,x)
```

Una ecuación diferencial de orden n puede convertirse en n ecuaciones acopladas de primer orden y ser resuelta mediante `lsode()`.

Ejemplo: Integrar $\ddot{y} + \dot{y} + 2y = 0$ con $y(0) = 0$ y $\dot{y}(0) = 1$ en $t \in [0, 1]$ (Solución exacta: $y(t) = e^{-t} - e^{-2t}$). Definimos: $x_1 = y$, $x_2 = \dot{y}$, de modo que $\dot{x}_1 = x_2$ y $\dot{x}_2 = -3x_2 - 2x_1$ con $[x_1(0), x_2(0)] = [0, 1]$. De aquí:

```
function xdot = fcn(x,t)
   xdot(1) = x(2);
   xdot(2) = -3*x(2)-2*x(1);
endfunction
t = linspace(0,1,101); x0 = [0,1];
[x,info,msg] = lsode('fcn', x0, t);
plot(t,x)
```

Existen otras rutinas para el problema similar de resolver el sistema $\vec{f}(\vec{r}, \dot{\vec{r}}, t) = 0$ con las condiciones $\vec{r}(t_0) = \vec{r}_0$ y $\dot{\vec{r}}(t_0) = \dot{\vec{r}}_0$. Son: `daspk`, `dassl`, y `dasrt`.

• Programación y estructuras de control

Un programa es una colección estructurada de instrucciones, *algoritmos*, que actúan sobre una colección de datos organizados:

$$\text{Algoritmos} + \text{Estructuras de datos} = \text{Programas}$$

Teorema de la estructura: Cualquier programa se puede construir utilizando tan sólo tres tipos de estructuras de control:

Secuencial. Por defecto, las instrucciones de un programa se ejecutan secuencialmente, siguiendo el orden en el que se han escrito o ingresado.

Condicional. Permite bifurcar la marcha del programa en dos o más rutas diferentes dependiendo de alguna condición que cumplan los datos.

Iterativa o bucle. Repite un fragmento del programa. Puede tratarse de:

- **Bucle con contador:** El fragmento se repite un número predeterminado de veces.
- **Bucle con condición inicial:** El fragmento se repite mientras se cumpla una cierta condición. Posiblemente nunca llegue a realizarse si la condición es falsa desde el principio.
- **Bucle con condición final:** El fragmento se repite hasta que se cumpla una cierta condición. En cualquier caso, el bucle se realiza al menos en una ocasión.

Una estructura puede contener otra, siempre y cuando la segunda se encuentre totalmente contenida dentro de la primera: *estructuras anidadas*.

• Estructura condicional: bloques if

```
if (condición)
    bloque-sí
else
    bloque-no
endif
```

```
if (condición)
    bloque-sí
endif
```

```
if (condición-1)
    bloque-sí-1
elseif (condición-2)
    bloque-sí-2
else
    bloque-no
endif
```

El bloque if puede tener tantas partes elseif como sea necesario.

Cuidado: elseif no debe tener espacios. La aparición de `else if` se interpreta como un segundo bloque if dentro de la parte else del primer bloque.

Ejemplo elemental:

```
1  n = input('Ingresa un numero natural');
2  if (rem(n,2) == 0)
3     disp([str2num(n), ' es par'])
4  elseif (rem(n,2) == 1)
5     disp([str2num(n), ' es impar'])
6  else
7     disp('No juego con tramposos!')
8  endif
```

Ejemplo: Programa capaz de resolver cualquier ecuación de segundo grado $ax^2 + bx + c = 0$.

```
disp ('Resolviendo a*x**2 + b*x + c = 0')
a = input('Ingresa a -> '); b = input('b -> '); c = input('c -> ')
if (a == 0)
  if (b == 0)
    if (c == 0)
      disp('Sistema indeterminado')
    else
      disp('Sistema imposible')
    endif
  else
    disp(['Raiz unica: ', num2str(-c/b)])
  endif
else
  d2 = b*b - 4*a*c;
  if (d2 == 0)
    disp(['Raiz doble: ', num2str(-b/2/a)])
  else
    m = 'Dos raices ';
    if (d2>0) disp([m,'reales:']) else disp([m,'complejas:']) endif
    disp((-b+sqrt(d2))/(2*a)); disp((-b-sqrt(d2))/(2*a))
  endif
endif
```

Prueba con (a, b, c): $(1, 1, 1)$, $(1, 2, 1)$, $(0, 0, 0)$, $(1, 0, 0)$, $(0, 0, 1)$, $(1, 0, 2)$, $(1+3i, 3i, -7+2i)$.

• Estructuras iterativas: bucles for, while y do until

for índice = rango
    interior-del-bucle
endfor

while (condición)
    interior-del-bucle
endwhile

do
    interior-del-bucle
until (condición)

Ejemplo 1: Construye los primeros 20 términos de la sucesión de Fibonacci ($F_1 = 1$, $F_2 = 1$, $F_3 = 2, \ldots, F_k = F_{k-2} + F_{k-1}$).

```
1  fib(1) = 1; fib(2) = 1;
2  for k = 3 : 20
3     fib(k) = fib(k-1) + fib(k-2);
4  endfor
5  disp(fib)
```

Ejemplo 2: Encuentra el menor número $F_n > 10^4$.

```
1  fib(1) = 1; fib(2) = 1;
2  n = 2;
3  do
4     n = n + 1;
5     fib(n) = fib(n-1) + fib(n-2);
6  until (fib(n) > 10000)
7  disp(['fib(', num2str(n), ') = ', num2str(fib(n))])
```

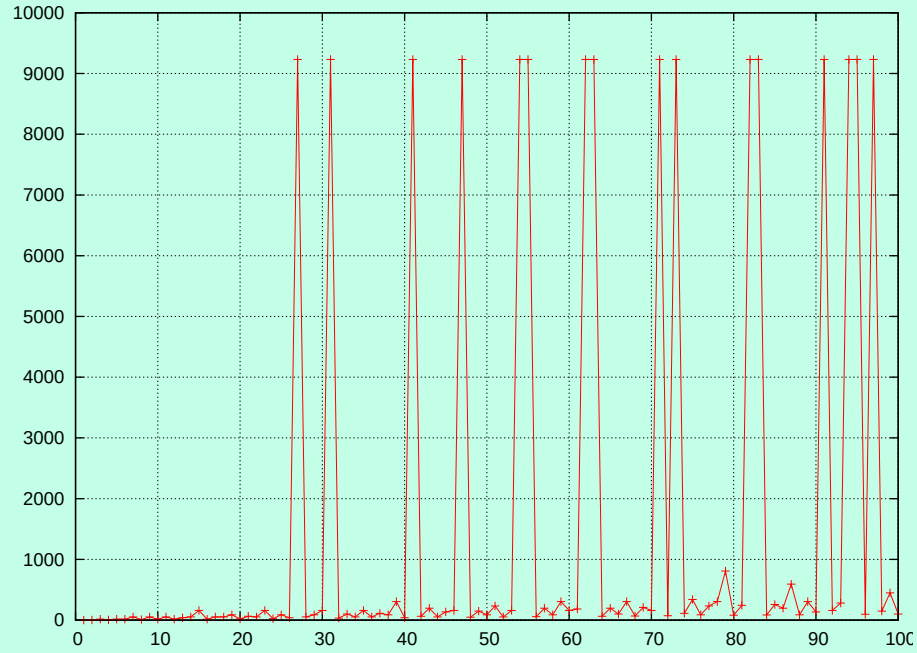
Ejemplo: (http://en.wikipedia.org/Collatz_conjecture) Sea la función entera

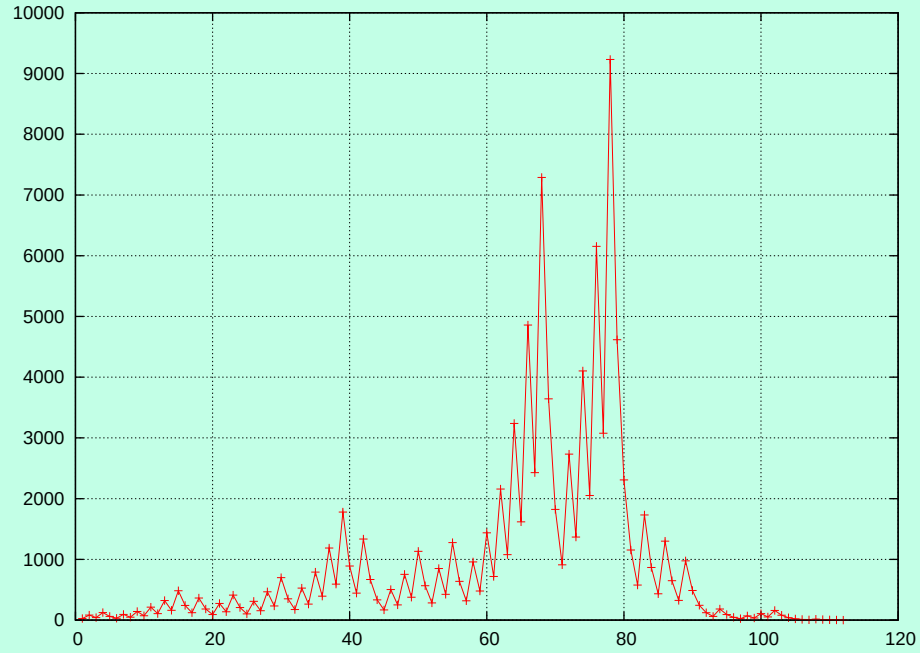
$$F(n) = \begin{cases} 3n+1 & \text{si} \quad n \text{ es impar,} \\ n/2 & \text{si} \quad n \text{ es par.} \end{cases}$$

La sucesión S_n está formada por los números naturales $x_0 = n$, $x_1 = F(x_0)$, ..., $x_n = F(x_{n-1})$, ..., hasta que se alcance un término de valor unidad, en cuyo momento la serie se considera terminada. El cardinal de S_n es el número de términos de la serie, y su cota es el valor más alto que alcanza un término cualquiera. Dibuja los términos de S_{27}. Calcula y dibuja las cotas y cardinales de las series $S_1 ... S_{50}$.

```
1  function Sn = Serie3n1(n)
2  # Calcula la sucesion 3n+1 que comienza en n.
3     k = 1; Sn(k) = n;
4     while (Sn(k) > 1)
5        s = Sn(k);
6        if (rem(s,2) == 0)
7           s = s/2;
8        else
9           s = 3*s+1;
10       endif
11       k = k+1; Sn(k) = s;
12    endwhile
13 endfunction
14
```

```
15 plot(Serie3n1(27));
16 for n = 1 : 100
17    s = Serie3n1(n);
18    cota(n) = max(s);
19    cardinal(n) = length(s);
20 endfor
21 plot(cota);
22 plot(cardinal);
```





• Comparaciones y álgebra lógica:

Operadores de comparación:

==	igualdad	~= ó !=	desigualdad
>	mayor que	<	menor que
>=	mayor o igual que	<=	menor o igual que

Valores lógicos:

`false` ó 0	falso
`true` ó 1	cierto

Cualquier número no nulo equivale a cierto.

Operadores lógicos:

! ó ~	Negación ($\neg$)	|	Disyunción (o inclusivo, $\cup$, $\vee$)
&	Conjunción (y, $\cap$, $\wedge$)	`xor(x,y)`	O exclusivo

`x&&y` y `x||y` detienen la evaluación tan pronto como se puede establecer su valor.

Orden de precedencia: () $\longrightarrow$! ó ~ $\longrightarrow$ & $\longrightarrow$ |

Tablas de verdad:

x	y	~x	x|y	x&y	xor(x,y)
t	t	f	t	t	f
t	f	f	t	f	t
f	t	t	t	f	t
f	f	t	f	f	f

• Entrada/Salida:

Vamos a ver sólo un conjunto breve de las órdenes que permiten a un programa OCTAVE interactuar con el exterior.

• `r = input('Mensaje')` ó `r = input('Mensaje','s')`. Escribe en pantalla el mensaje y recibe la respuesta del usuario. En la primera forma la respuesta puede ser una expresión, cuyo valor se evalúa. La opción `'s'` hace que la respuesta se devuelva como una cadena de caracteres, sin ninguna interpretación.

• `save fichero variables` y `load fichero` sirven para salvar el contenido de una o varias variables en un archivo y recuperarlo posteriormente.

• `fid = fopen(fichero,modo)` y `fclose(fid)`. Antes de leer datos de un fichero o escribir él es necesario *abrirlo* con la orden `fopen`, y la orden `fclose` lo *cierra* una vez acabado el trabajo. El modo de apertura puede ser 'r'(sólo leer), 'w'(sólo escribir), 'r+'(leer y escribir).

• `line = fgetl(fid)` Leer una línea desde el archivo abierto con el identificador fid.

• `printf(formato, lista_de_expresiones)` o `fprintf(fid, formato, lista)` o `txt = sprintf(formato, lista)`. Escritura a la salida estándar, a un archivo, o a una variable. Especificadores de formato: `%d` (entero), `%f` (real en coma fija), `%e` (real en notación exponencial), `%s` (cadena de caracteres), `\n` (salto de línea). El campo y decimales se indican como `%12.6f`. Ejemplo: `printf('Entero:%6d, Real:%14.6e\n', 12, pi)`.

Ej: Escribir, numeradas, las líneas de un archivo externo:

```
fichero = 'datos.txt';
fid = fopen(fichero,'r');
n = 0;
while (! feof(fid))
   linea = fgetl(fid);
   n = n + 1;
   printf('%4d %s\n', n, linea);
endwhile
fclose(fid)
```

Un diálogo tonto:

```
n = input('Dime tu nombre:');
printf('%s, sabias que tu nombre tiene %d letras?\n', n, length(n));
```

- **Sólo ejemplos:** ¿Es un número primo o es divisible?

```
do
   n = input('Ingresa un numero natural -> ');
   ok = (n > 0) && (n == fix(n));
   if (!ok)
      disp ('Sin trampas, por favor!')
      disp (['El numero debe estar en el rango 1:', num2str(intmax)])
   endif
until (ok)
maxdivisor = fix(sqrt(n));
ParecePrimo = true;
k = 1
while (k < maxdivisor & ParecePrimo)
   k = k + 1;
   ParecePrimo = (rem(n,k) != 0);
endwhile
if (ParecePrimo)
   disp ([num2str(n), ' es primo'])
else
   disp ([num2str(n), ' es divisible por ', num2str(k)])
endif
```

Sumamos una serie: $S = \sum_{k=1}^{20} \frac{3k^2 + 2k}{56 - k}$

```
S = 0
for k = 1:20
   S = S + (3*k**2+2*k)/(56-k);
endfor
disp(S)
```

Otra forma de sumar la misma serie:

```
k = 1:20;
sk = (3.*k.**2.+2.*k)./(56.-k);
S = sum(sk)
```

La serie de McLaurin: $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$

```
function y = mclaurin(x, TOL)
   if (nargin < 2) tol = 2*eps; else tol = TOL; endif
   y = 1 + x; term = x; k = 1;
   do
      k = k + 1;
      term = term .* x / k;
      y = y + term;
   until (max(max(abs(term))) <= tol)
endfunction
x = -3:0.1:3;
y = mclaurin(x);
plot(x,y)
```

Adivina mi número:

```
m = 100;
disp(["Piensa un entero entre 1 y ", num2str(m)])
menor = 1; mayor = 100; intento = 0;
do
  intento = intento + 1;
  p = fix((mayor+menor)/2);
  do
    s=input(["Dime si ",num2str(p)," es > < = que tu numero: "], "s");
  until (s == ">" | s == "<" | s == "=")
  if (s == "=")
    disp(["Lo he adivinado en el intento ", num2str(intento)]);
    exit()
  elseif (s == ">")
    mayor = p;
  elseif (s == "<")
    menor = p;
  endif
until ((mayor-menor) <= 0)
disp("Has hecho trampa!")
```