Fundamentos de Computación

Bloque III.

Entrada y Salida de Datos. Cadenas de Control. Trabajando con Ficheros

Gráficas con Octave / Matlab. Definición de Funciones

Sixto Herrera García

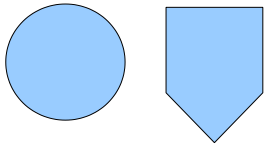
Departamento de Matemática Aplicada y Ciencias de la Computación

Fundamentos de Computación

Entrada y Salida de Datos
Cadenas de Control
Trabajando con Ficheros

Diagramas de Flujo: Elementos (ISO 5807)

Comienzo/Fin



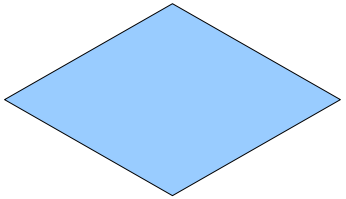
Enlaces

Calculo basico

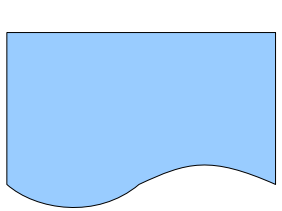
Calculo predefinido

Preparacion

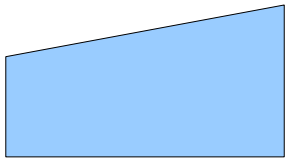
Entrada/Salida (IO) generica



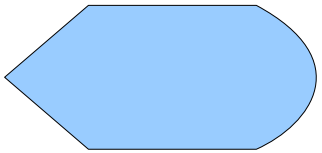
Decision (2 salidas!)



IO Fichero



Entrada manual

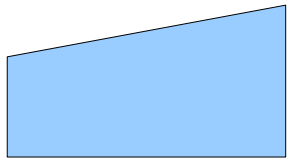


Salida a pantalla

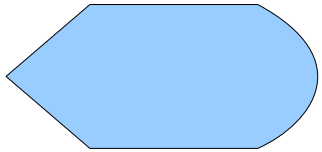
Entrada|Salida de datos

La entrada-salida interactiva de datos, como hemos visto, se puede realizar a través de las funciones ***input()***, ***sprintf()*** y ***disp()***.

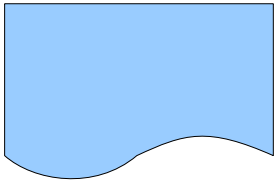
```
%OCTAVE/MATLAB
Nombre = input('Escribe tu nombre: ');
Text = sprintf('Escribe la edad de %s:\n', Nombre);
Edad = input(Text);
Text = sprintf('%s tiene %d anios.\n', Nombre, Edad);
disp(Text)
```



Entrada manual



Salida a pantalla



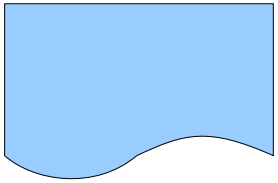
IO Fichero

Entrada|Salida de datos

En ***Octave/Matlab*** existen diferentes opciones para leer y escribir datos en ficheros de ***texto plano*** o ***binarios***.

- ***Formato .mat***: Es el formato propio de ***Octave/Matlab*** que guarda y lee los datos en formato ***binario comprimido*** a través de las funciones ***save()*** y ***load()***.

```
%OCTAVE/MATLAB
A = randn(10,10);
Nombre = 'pepe';
save('datosEjemplo.mat','A','Nombre');
clear A Nombre
load('datosEjemplo.mat');
whos
clear A Nombre
load('datosEjemplo.mat','A');
whos
```



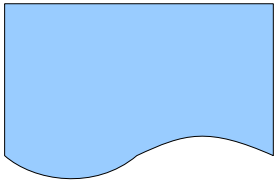
IO Fichero

Entrada|Salida de datos

En ***Octave/Matlab*** existen diferentes opciones para leer y escribir datos en ficheros de ***texto plano*** o ***binarios***.

- ***Formato ASCII***: Es un formato de ***texto plano formateado*** a través de un separador. Las funciones ***save()*** y ***load()*** permiten escribir y leer datos en este formato:

```
%OCTAVE/MATLAB
A = randn(10,10);
save('datosEjemplo.txt','-ASCII','A');
clear A
load('datosEjemplo.txt');
whos
```



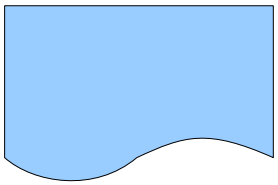
Entrada|Salida de datos

En ***Octave/Matlab*** existen diferentes opciones para leer y escribir datos en ficheros de ***texto plano*** o ***binarios***.

- ***Formato ASCII***: Es un formato de ***texto plano formateado*** a través de un separador. Las funciones ***save()*** y ***load()*** permiten escribir y leer datos en este formato:

```
%OCTAVE/MATLAB
A = randn(10,10);
save('datosEjemplo.txt','-ASCII','A');
clear A
load('datosEjemplo.txt');
whos
```

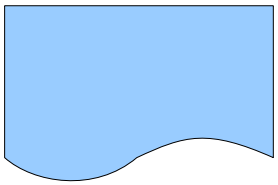
- ***Ficheros de texto plano***: En general, un fichero de texto plano correctamente formateado (campos separados por un separador, definidos por un ancho fijo, etc.) puede leerse y escribirse con diversas funciones: ***csvread(), dlmread(), textread(), textscan(), csvwrite()*** ó ***dlmwrite()***.



Entrada|Salida de datos

Las funciones ***fprintf***, ***sprintf***, ***fscanf*** y ***scanf*** nos permiten escribir y leer formatos tanto binarios como de texto plano con formatos más complejos.

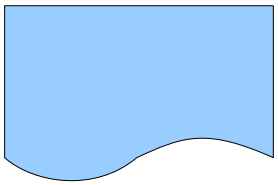
```
A = randn(10,10);
fid=fopen('datosEjemplo.csv','w');
fprintf(fid,'Fila Nº');
for i=1:10
   fprintf(fid,', Dato %d',i);
end
fprintf(fid,'\r\n');
for n=1:10
   fprintf(fid,'fila %d',n);
   for i=1:10
      fprintf(fid,', %f',A(n,i));
   end
   fprintf(fid,'\r\n');
end
fclose(fid)
```



Entrada|Salida de datos

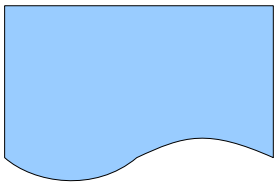
Las funciones ***fprintf***, ***sprintf***, ***fscanf*** y ***scanf*** nos permiten escribir y leer formatos tanto binarios como de texto plano con formatos más complejos.

```
A = randn(10,10);
fid=fopen('datosEjemplo.txt','w');
fprintf(fid,'FilaN');
for i=1:10
   fprintf(fid,' Dato%d',i);
end
fprintf(fid,'\r\n');
for n=1:10
   fprintf(fid,'fila%d',n);
   for i=1:10
      fprintf(fid,' %f',A(n,i));
   end
   fprintf(fid,'\r\n');
end
fclose(fid)
```



Entrada|Salida de datos

```
fid=fopen('datosEjemplo.txt','r');
A=[];
headerLines=fgetl(fid);
while ~feof(fid)
   line=fgetl(fid);
   [VAL,COUNT,ERRMSG,POS]=sscanf(line,'%s',1);
   B=[];
   while POS<length(line)
      [VAL,COUNT,ERRMSG,POS1]=sscanf(line(POS:end),'%f',1);
      B=[B VAL];
      POS = POS + POS1;
   end
   A=[A;B];
end
fclose(fid);
```



Entrada|Salida de datos

```
fid=fopen('datosEjemplo.csv','r');
A=[];
headerLines=fgetl(fid);
while ~feof(fid)
   line=fgetl(fid);
   [B1,B2,B3,B4,B5,B6,B7,B8,B9,B10]=strread(line,'%*s%f%f%f%f%f
%f%f%f%f%f','delimiter',',');
   A=[A;[B1,B2,B3,B4,B5,B6,B7,B8,B9,B10]];
end
fclose(fid);

[rowNames,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10]=textread('datosEjemplo
.csv','%s%f%f%f%f%f%f%f%f%f%f','delimiter',',','headerlines',1);
A=[B1,B2,B3,B4,B5,B6,B7,B8,B9,B10];
whos rowNames % Tipo de dato celda.
```

Cadenas de Control

En el caso anterior es sencillo pero, ¿qué ocurre cuando tengo que mezclar muchas variables de tipos diferentes?:

```
%OCTAVE/MATLAB
Edad=24;
Nombre='Pedro Piqueras';
DNI='77.777.777-Z';
Direccion='Benito Vercimuelles';
Numero=45;
Piso=3;
Letra='C';
```

¿Cómo construyo el mensaje: "***Pedro Piqueras, de 24 años de edad, DNI: 77.777.777-Z y con domicilio en la Calle Benito Vercimuelles Nº 45, 3ºC, nació en el año 1992***"?

Cadenas de Control

```
Edad=24;
Nombre='Pedro Piqueras';
DNI='77.777.777-Z';
Direccion='C/Benito Vercimuelles';
Numero=45;
Piso=3;
Letra='C';
```

¿Cómo construyo el mensaje: "***Pedro Piqueras, de 24 años de edad, DNI: 77.777.777-Z y con domicilio en la Calle Benito Vercimuelles Nº 45, 3ºC, nació en el año 1992***"?

```
% Con la función disp:
disp([Nombre ', de ' num2str(Edad) ' años de edad, DNI: '
DNI ' y con domicilio en la Calle ' Direccion ' Nº '
num2str(Numero) ', ' num2str(Piso) 'º' Letra ', nació en
el año ' num2str(2016-Edad)]);
```

¿Hay algún modo más fácil de componer mensajes con variables de tipos diferentes?

Cadenas de Control

```
Edad=24;
Nombre='Pedro Piqueras';
DNI='77.777.777-Z';
Direccion='C/Benito Vercimuelles';
Numero=45;
Piso=3;
Letra='C';
% Con la función sprintf y cadenas de control:
Text = sprintf('%s, de %d años de edad, DNI: %s y con
domicilio en la Calle %s Nº %d, %dº%s, nació en el año
%d', Nombre, Edad, DNI, Direccion, Numero, Piso, Letra,
2016-Edad);
disp(Text)
fprintf(fid,'%s, de %d años de edad, DNI: %s y con
domicilio en la Calle %s Nº %d, %dº%s, nació en el año
%d', Nombre, Edad, DNI, Direccion, Numero, Piso, Letra,
2016-Edad);% fprintf requiere el puntero al fichero
```

Fundamentos de Computación

Gráficas con Octave/Matlab

Gráfico Simple

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
```

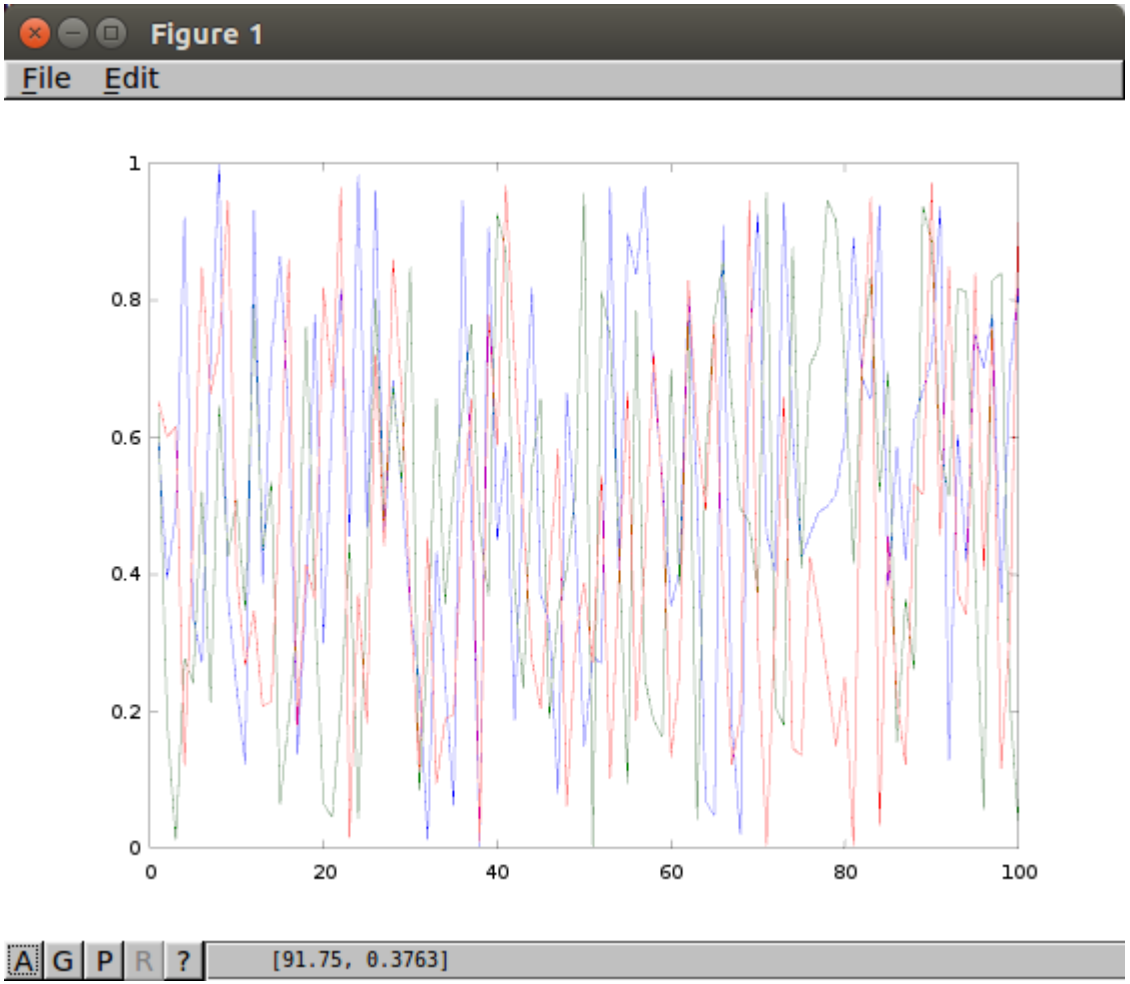


Gráfico Simple: Modificadores

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
title('Series Aleatorias') % Titulo
xlabel('Time') % Etiqueta del Eje X
ylabel('Value') % Etiqueta del Eje Y
```

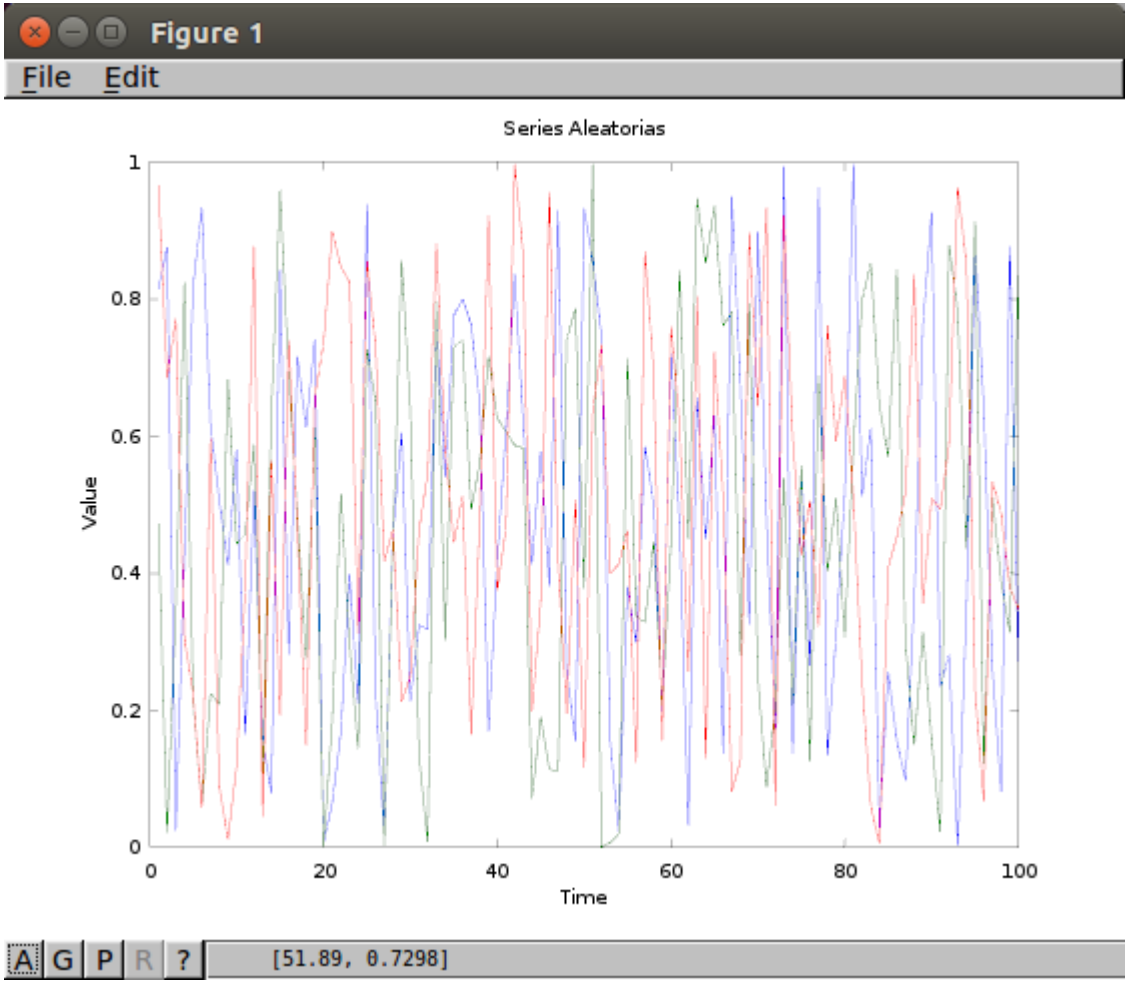


Gráfico Simple: Modificadores

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
title('Series Aleatorias') % Titulo
xlabel('Time') % Etiqueta del Eje X
ylabel('Value') % Etiqueta del Eje Y
% Incluimos un Texto en el gráfico:
text(50,0.5,'Punto Medio')
```

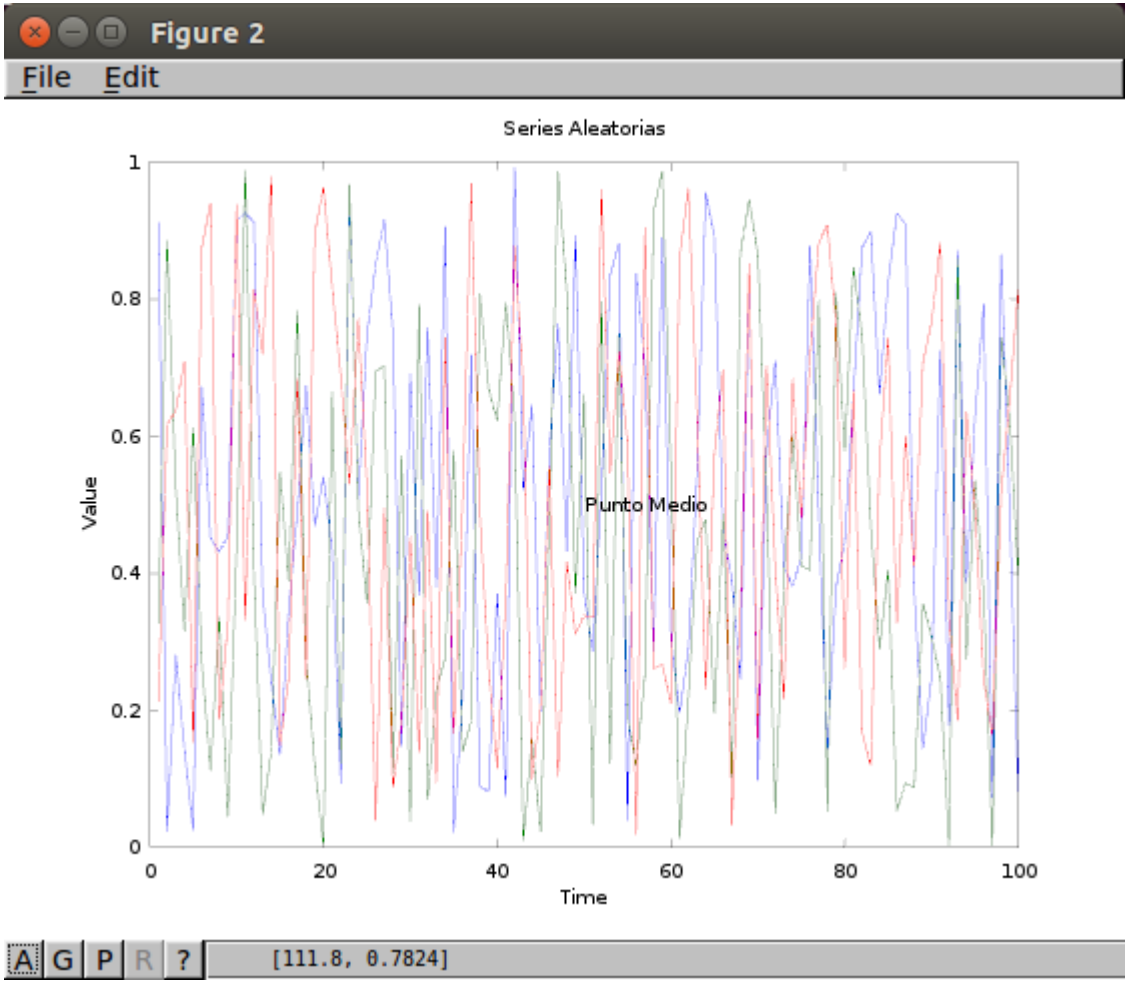


Gráfico Simple: Modificadores

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
title('Series Aleatorias') % Titulo
xlabel('Time') % Etiqueta del Eje X
ylabel('Value') % Etiqueta del Eje Y
text(50,0.5,'Punto Medio') % Texto
% Leyenda
legend('Serie 1','Serie 2','Serie 3')
% Rejilla de referencia
grid on
% Podemos incluir texto con el ratón
gtext('Punto con raton')
```

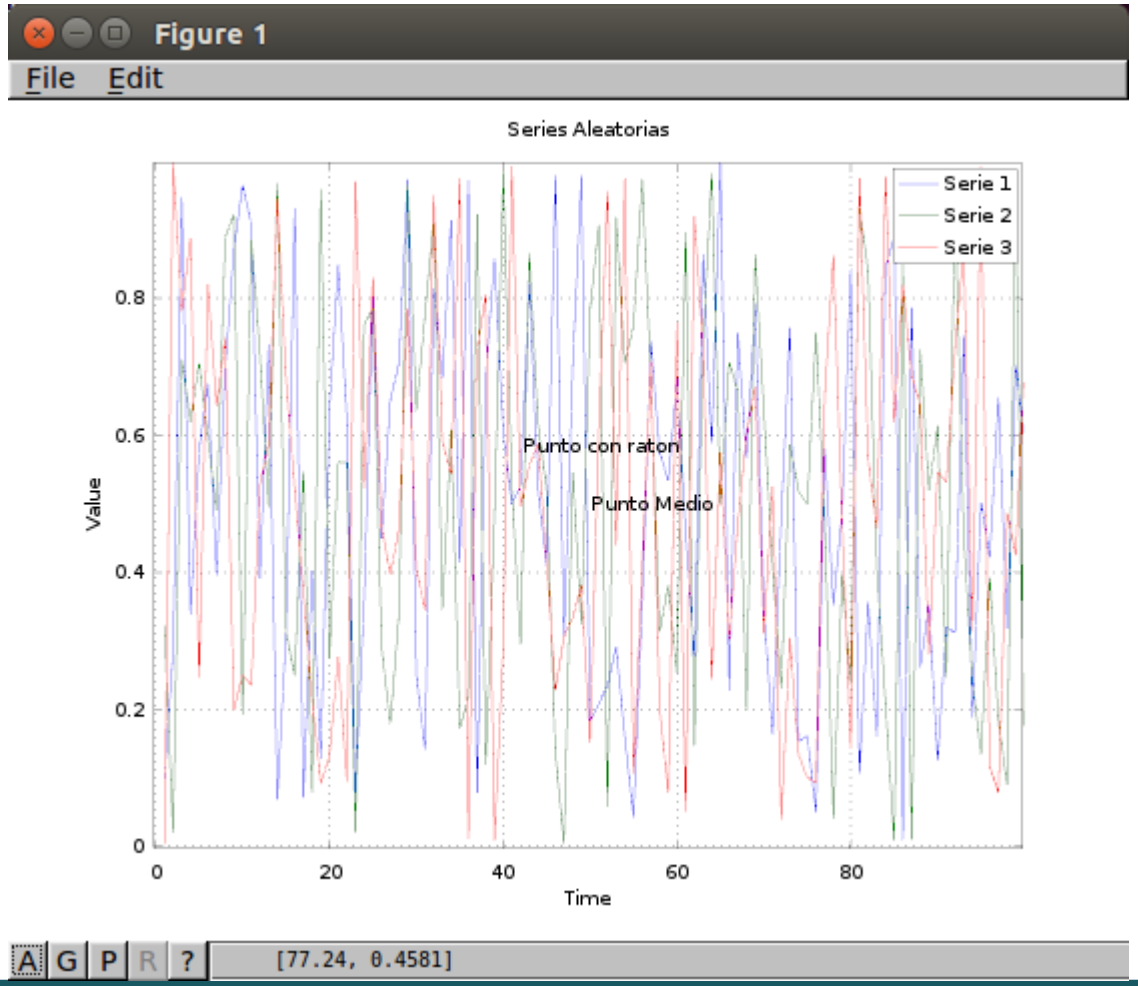


Gráfico Simple: Extrayendo Propiedades

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
title('Series Aleatorias') % Titulo
xlabel('Time') % Etiqueta del Eje X
ylabel('Value') % Etiqueta del Eje Y
text(50,0.5,'Punto Medio') % Texto
% Leyenda
legend('Serie 1','Serie 2','Serie 3')
% Rejilla de referencia
grid on
% Podemos extraer propiedades
get(gca)% Get Current Axis
```

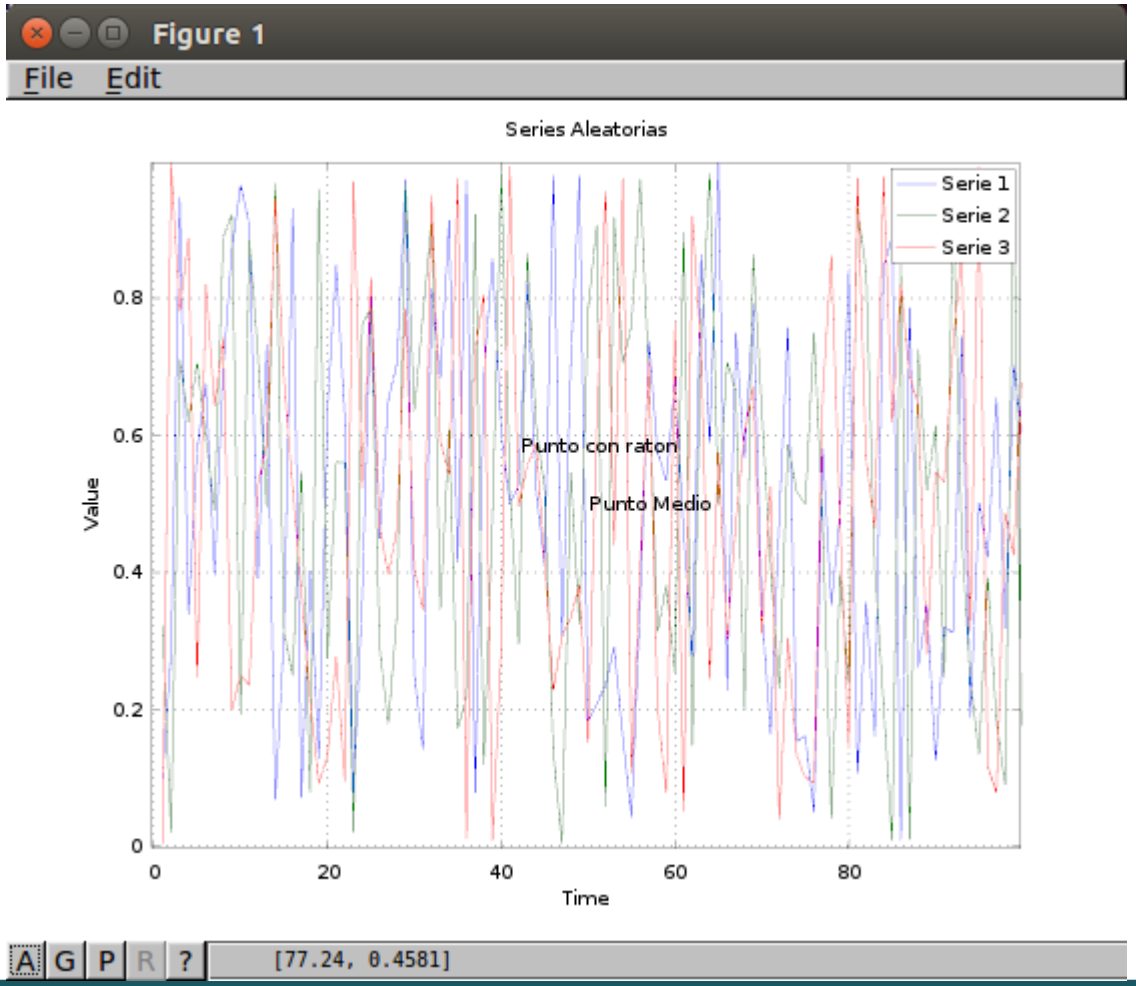


Gráfico Simple: Extrayendo Propiedades

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
title('Series Aleatorias') % Titulo
xlabel('Time') % Etiqueta del Eje X
ylabel('Value') % Etiqueta del Eje Y
text(50,0.5,'Punto Medio') % Texto
% Leyenda
legend('Serie 1','Serie 2','Serie 3')
% Rejilla de referencia
grid on
% Podemos extraer propiedades
get(gca)% Get Current Axis
% Podemos extraer una propiedad
get(gca,'xlim') % Limites del Eje x.
```

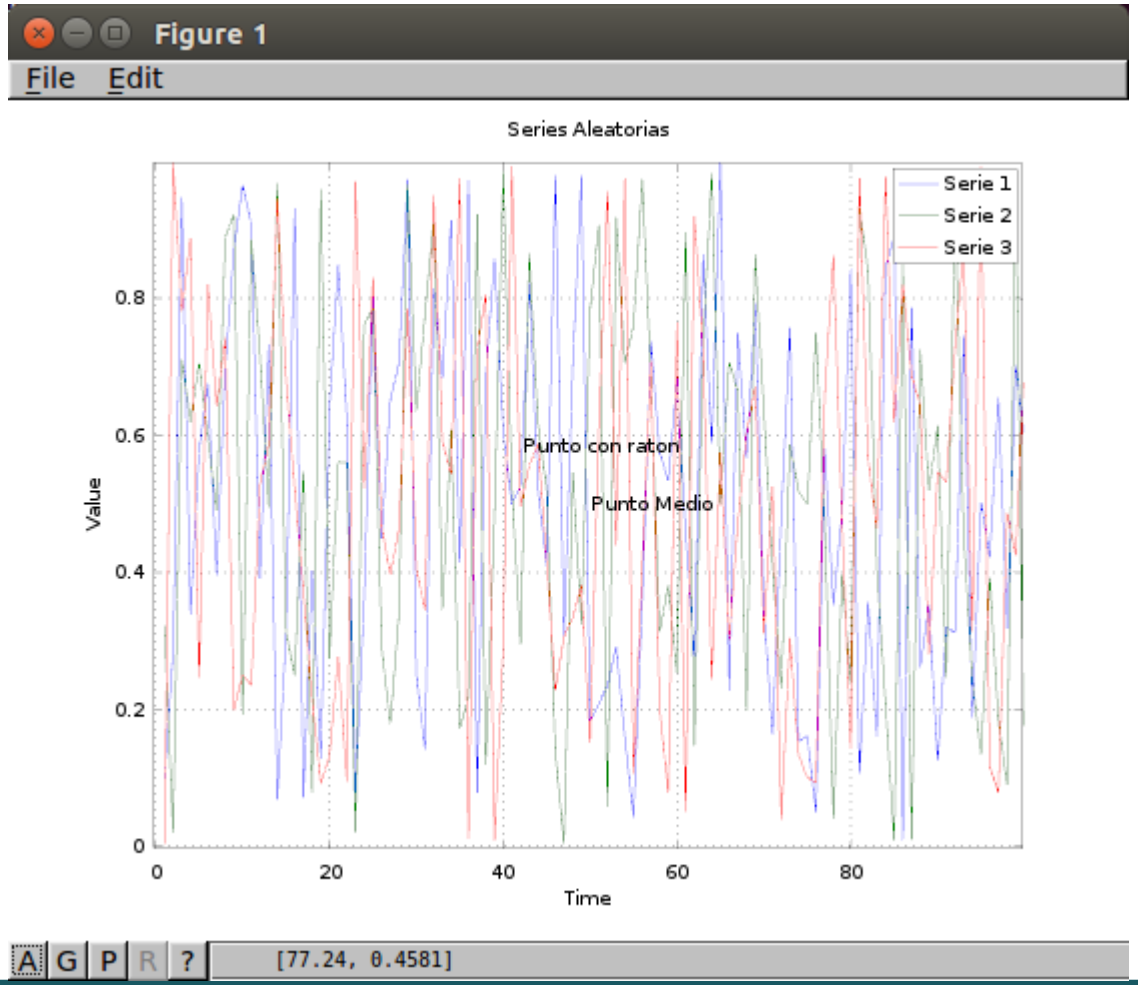


Gráfico Simple: Modificando Propiedades

```
data=rand(100,3); % Matriz aleatoria
figure, % Abrimos un objeto grafico
plot([1:100],data) % dibujamos los datos
title('Series Aleatorias') % Titulo
xlabel('Time') % Etiqueta del Eje X
ylabel('Value') % Etiqueta del Eje Y
text(50,0.5,'Punto Medio') % Texto
% Leyenda
legend('Serie 1','Serie 2','Serie 3')
% Rejilla de referencia
grid on
% Podemos extraer propiedades
get(gca)% Get Current Axis
% Podemos extraer una propiedad
get(gca,'xlim') % Limites del Eje x.
% Podemos modificar una propiedad
set(gca,'ylim',[-1 1]) % Limites del Eje y.
```

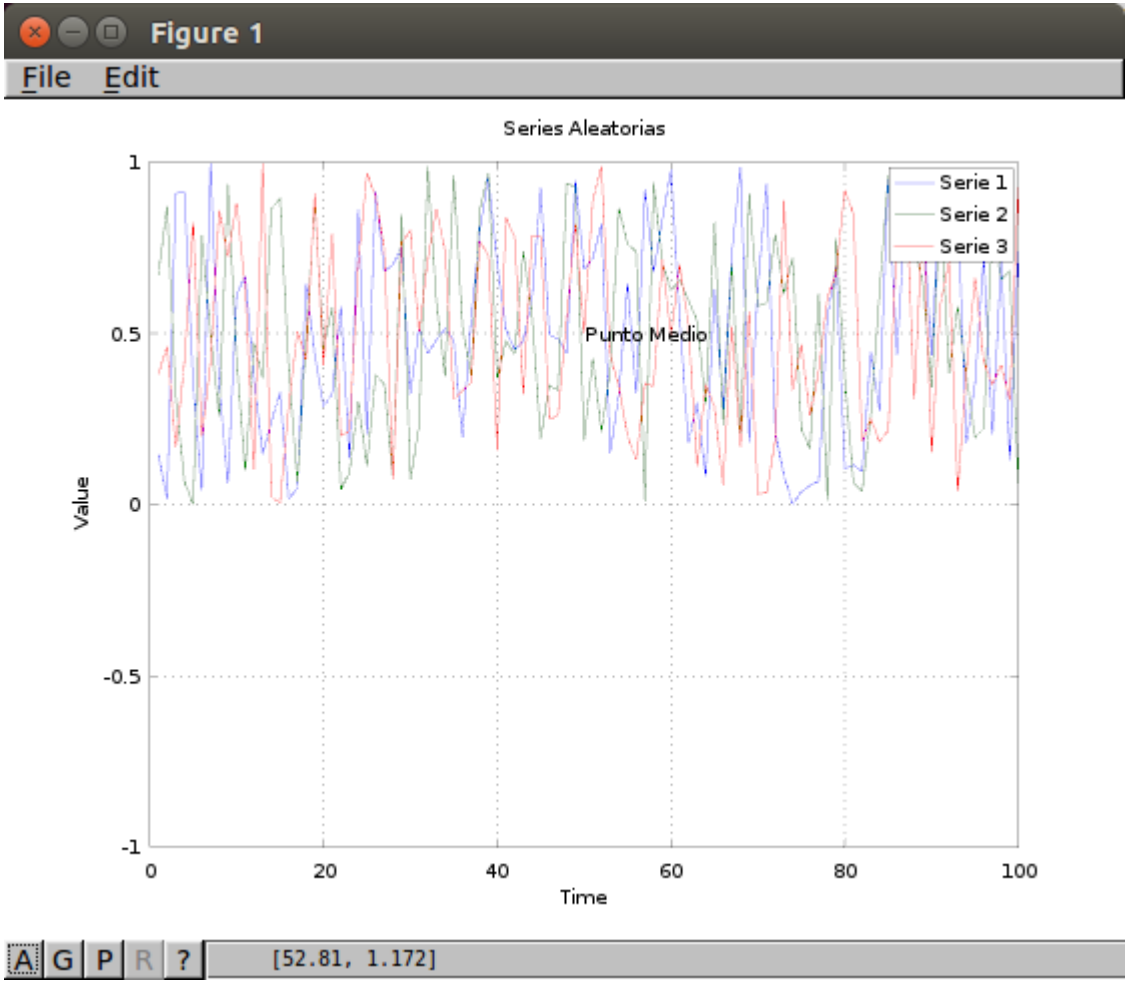
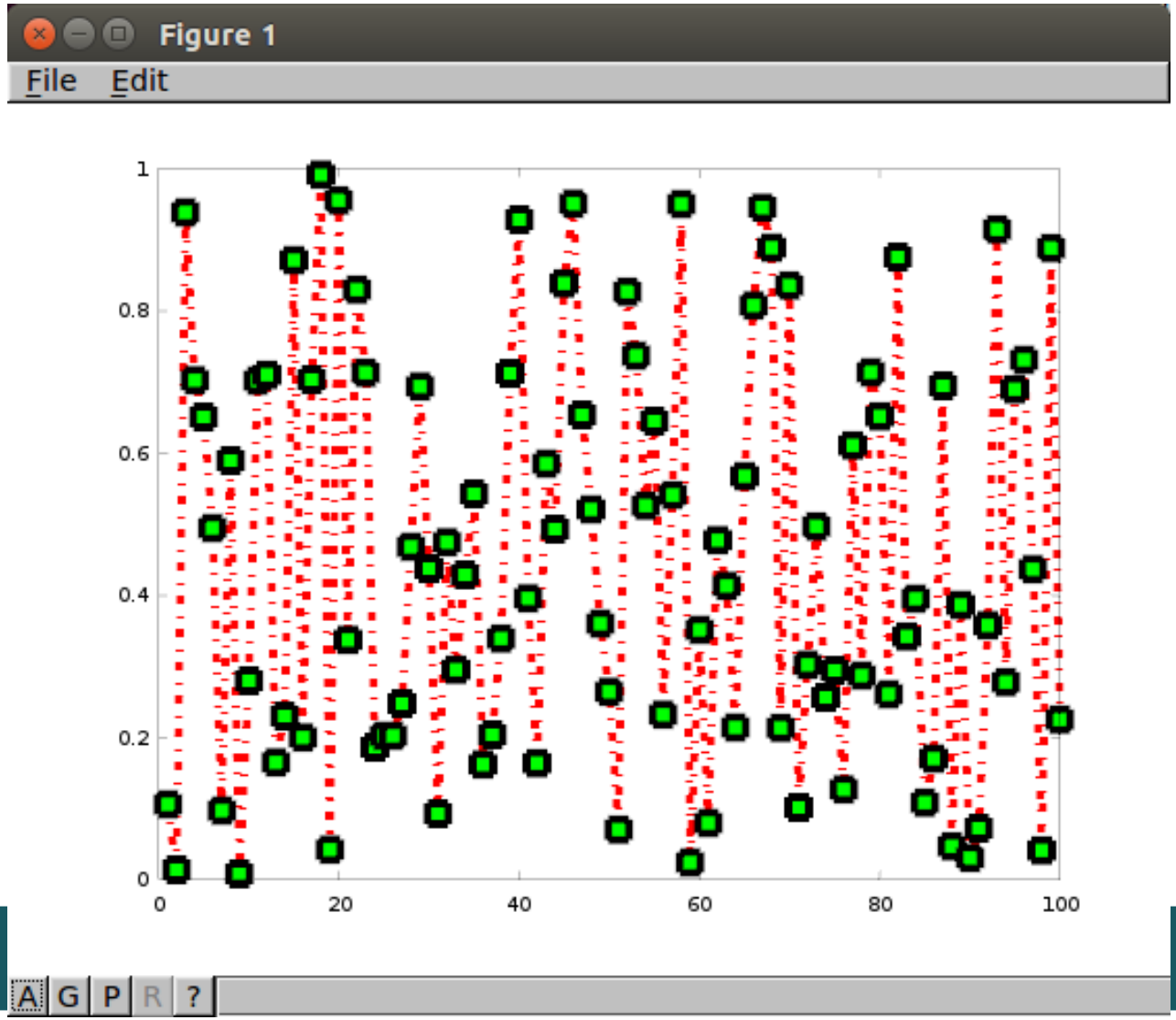


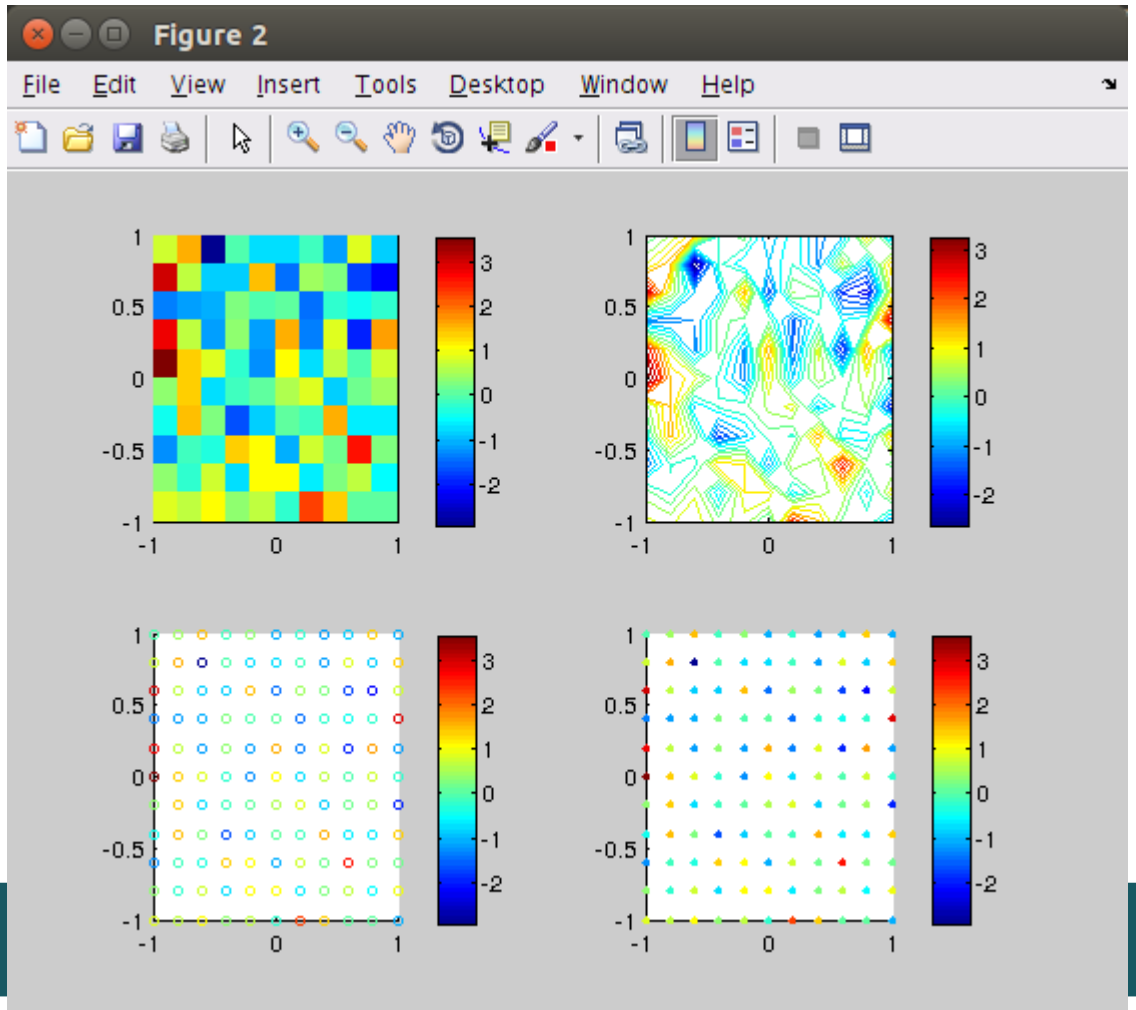
Gráfico Simple: Otros Modificadores

```
figure,
X=[1:100];% Valores del Eje X
data=rand(100,1); % Matriz aleatoria
% Incluimos diferentes modificadores en el estilo de la gráfica.
plot(X,data,'-.rs', 'LineWidth',4, 'MarkerEdgeColor','k', 'MarkerFaceColor', 'g','MarkerSize',10)
```



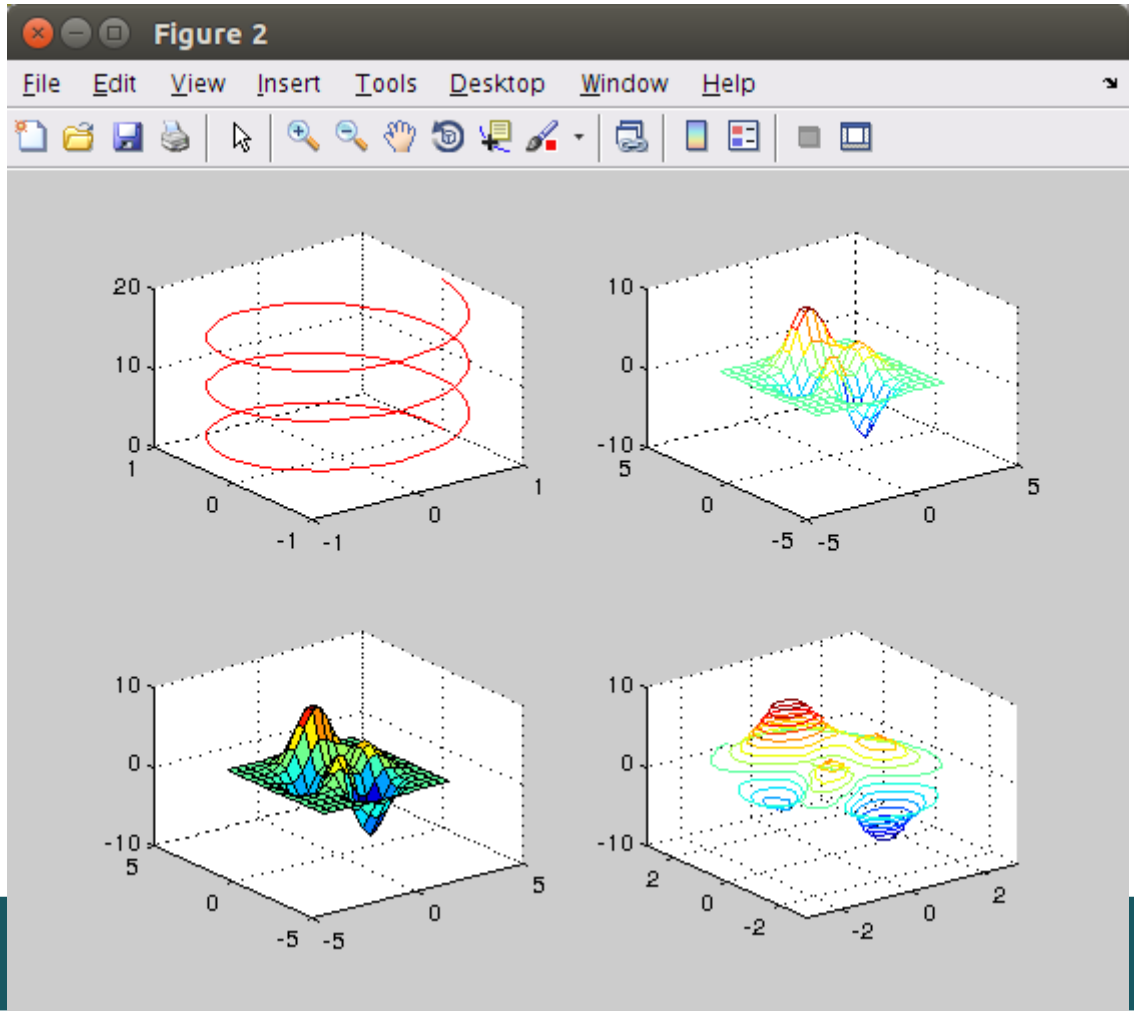
Otros Gráficos: Gráficos 2D

```
x=linspace(-1,1,11);% Valores del Eje X
y=linspace(-1,1,11);% Valores del Eje Y
[X,Y]=meshgrid(x,y);% Coord. del Plano X-Y
z=randn(length(y),length(x));% Datos
figure,
subplot(2,2,1),% Divimos la figura en 4
pcolor(x,y,z),% Proyección en un plano
shading flat,% Elimina los bordes
colorbar% Escala de colores
subplot(2,2,2),
contour(x,y,z,20),% Curvas de nivel
colorbar
subplot(2,2,3),
scatter(X(:),Y(:),20,z(:)),% Puntos
colorbar
subplot(2,2,4),
scatter(X(:),Y(:),20,z(:),'filled'),% Rellenos
colorbar
```



Otros Gráficos: Gráficos 3D

```
x=[-3:0.4:3]; % Valores del Eje X
y=[-3:0.4:3]; % Valores del Eje X
[X,Y]=meshgrid(x,y); % Puntos en el plano X-Y
% Valores del Eje Z
Z =  3*(1-X).^2.*exp(-(X.^2) - (Y+1).^2) - 10*(X/5 - X.^3 - Y.^5).*exp(-X.^2-Y.^2) - 1/3*exp(-(X+1).^2 - Y.^2);
figure
subplot(2,2,1)
% Valores del Eje X en radianes
fi=[0:pi/20:6*pi];
plot3(cos(fi),sin(fi),fi,'r')
grid
subplot(2,2,2)
mesh(X,Y,Z) % Malla/rejilla
subplot(2,2,3)
surf(X,Y,Z) % Superficie
subplot(2,2,4)
contour3(X,Y,Z,16) % Lineas de nivel
```



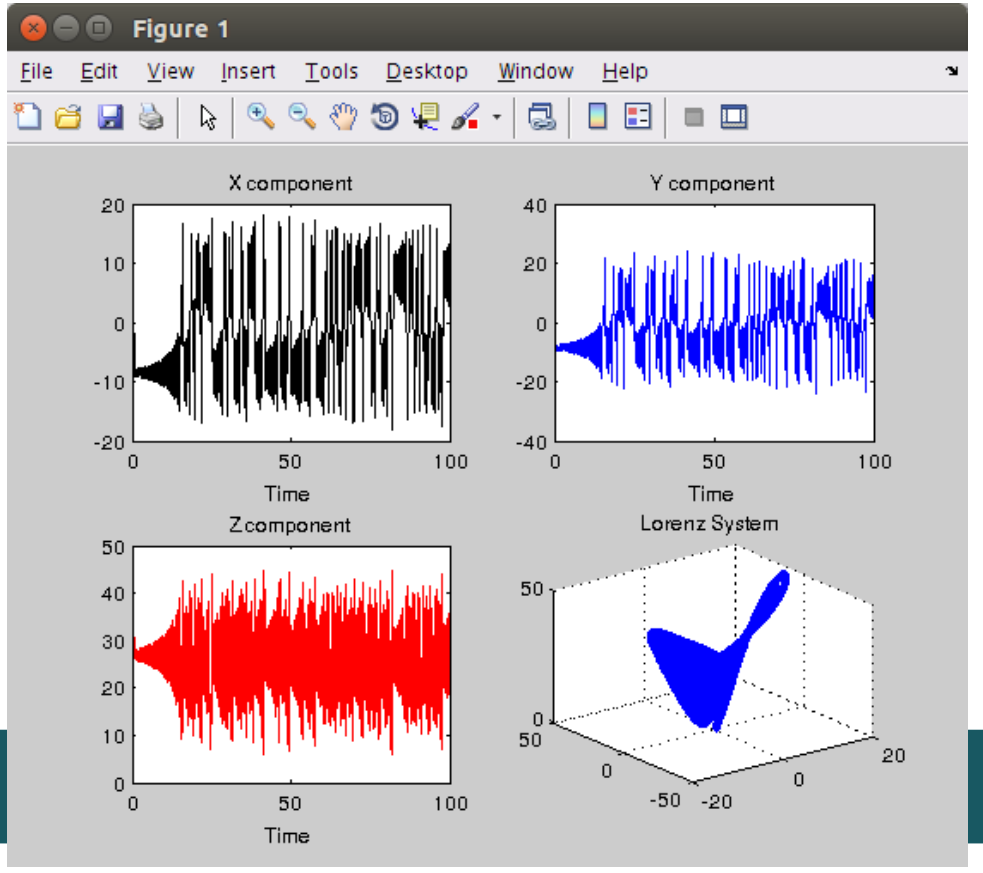
Ejercicios

El Sistema de Lorenz es un sistema dinámico caótico cuya derivada viene definida por las expresiones: $\frac{dx}{dt} = a(y-x)$ $\frac{dy}{dt} = x(b-z)-y$ $\frac{dz}{dt} = xy - cz$

Siendo a=10, b=28 y c=8/3. Dada la expresión de las derivadas y un paso de tiempo dt=0.001, se puede obtener el estado del sistema en un tiempo T a partir del instante anterior T-dt sin más que usar el método de Euler:

[x(T) y(T) z(T)]=[x(T-dt) y(T-dt) z(T-dt)] + dt*[dx/dt dy/dt dz/dt];

Considerando una condición inicial, X0=rand(1,3);, haced un programa (***lorenzSystem.m***) que reproduzca la siguiente figura:

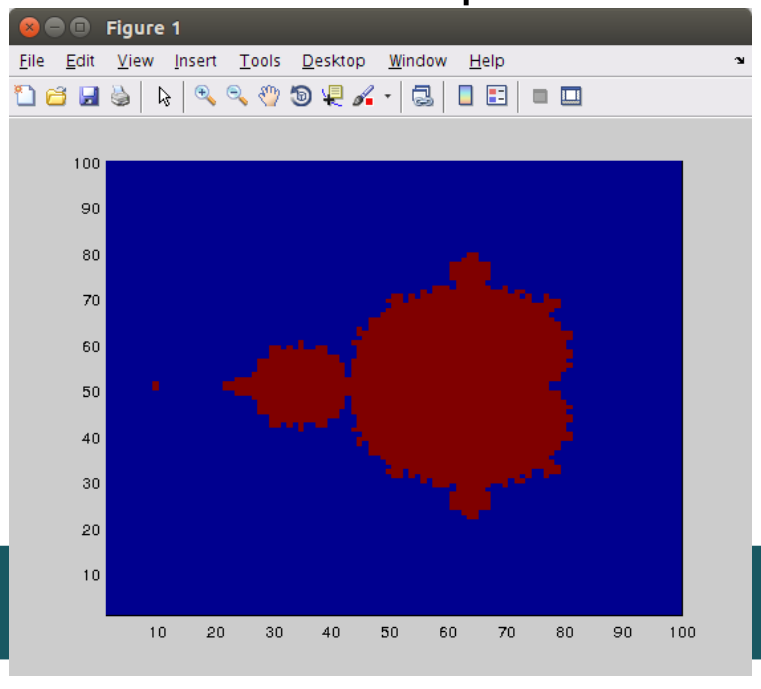


Ejercicios

El conjunto de Mandelbrot en el rectángulo [-2 1]x[-1.5 1.5] se construye a través de dos parámetros, el número máximo de repeticiones (Nmax) y un umbral de corte (threshold), según la iteración z = z^2 + c, donde z comienza tomando el valor 0 y se calcula como el número complejo x+y*i, siendo x e y los puntos del plano X-Y e "i" el número complejo.

Al término de las iteraciones, el conjunto de Mandelbrot serán los puntos del plano X-Y cumpliendo: abs(z)<threshold.

Haced un programa (***mandelbrotSet.m***) que solicite el número de puntos en ambas dimensiones, el número máximo de repeticiones y el valor umbral, y que construya el conjunto de Mandelbrot con esos parámetros obteniendo una figura similar a la siguiente:

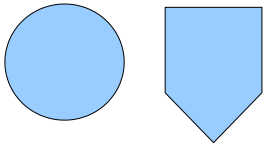


Fundamentos de Computación

Definición de Funciones

Diagramas de Flujo: Elementos (ISO 5807)

Comienzo/Fin



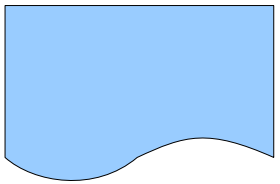
Enlaces

Cálculo básico

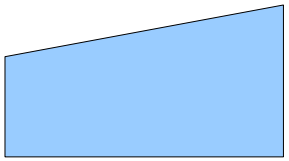
Cálculo predefinido

Preparación

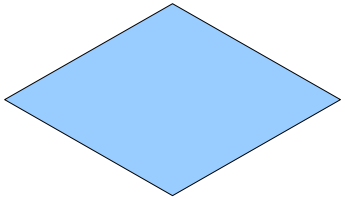
Entrada/Salida (IO) genérica



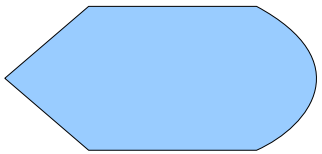
IO Fichero



Entrada manual

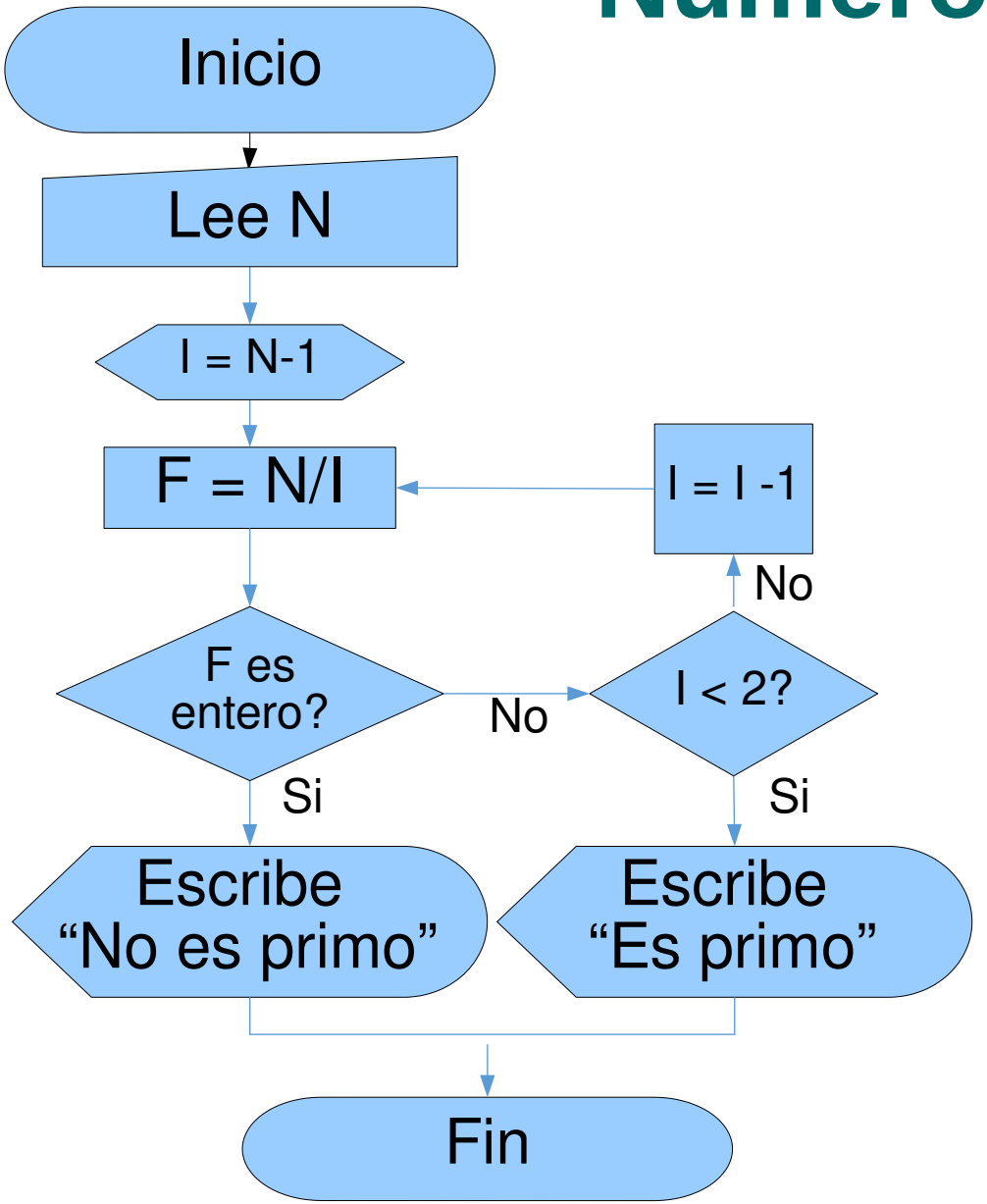


Decisión (2 salidas!)

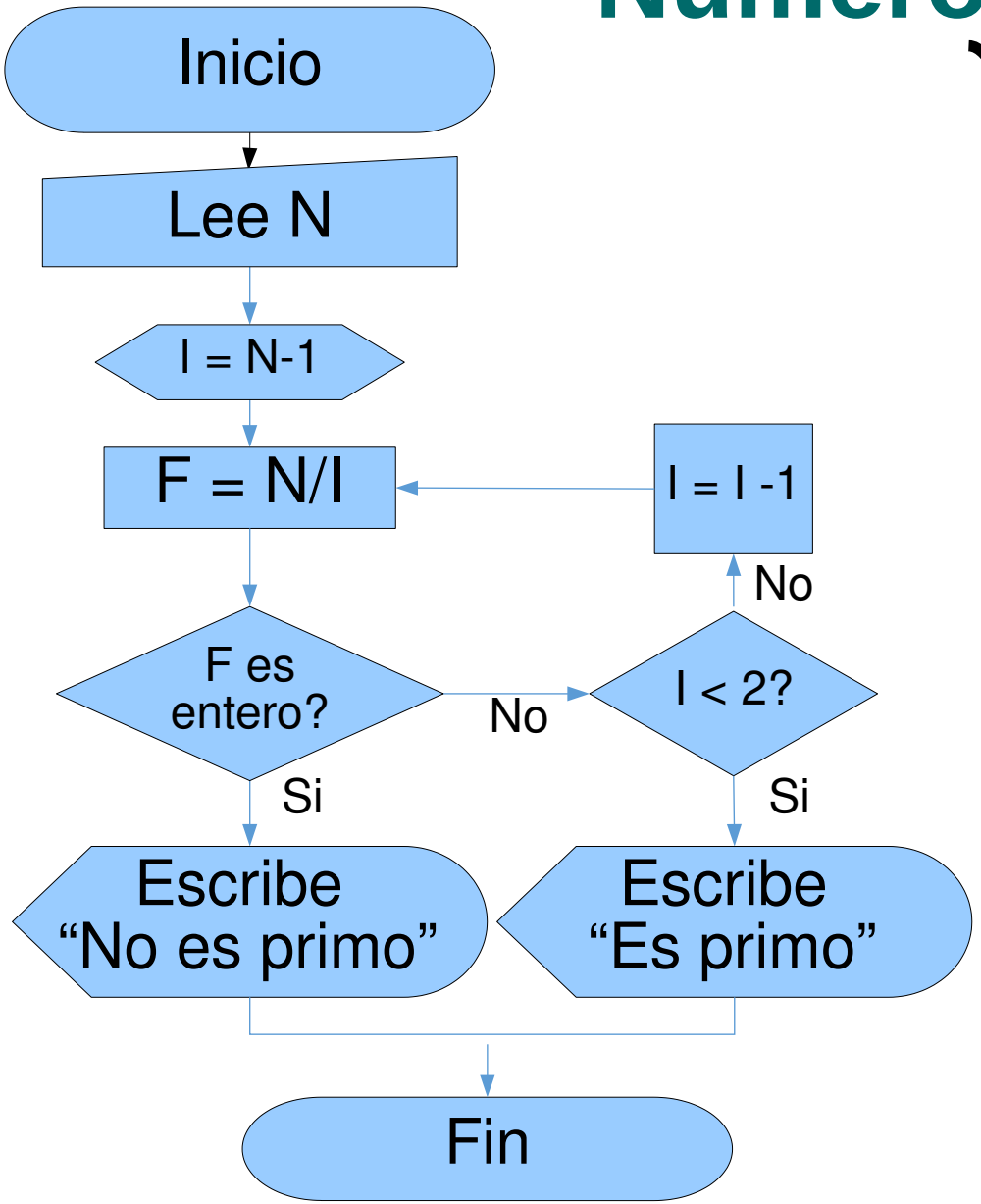


Salida a pantalla

Números Primos



Números Primos

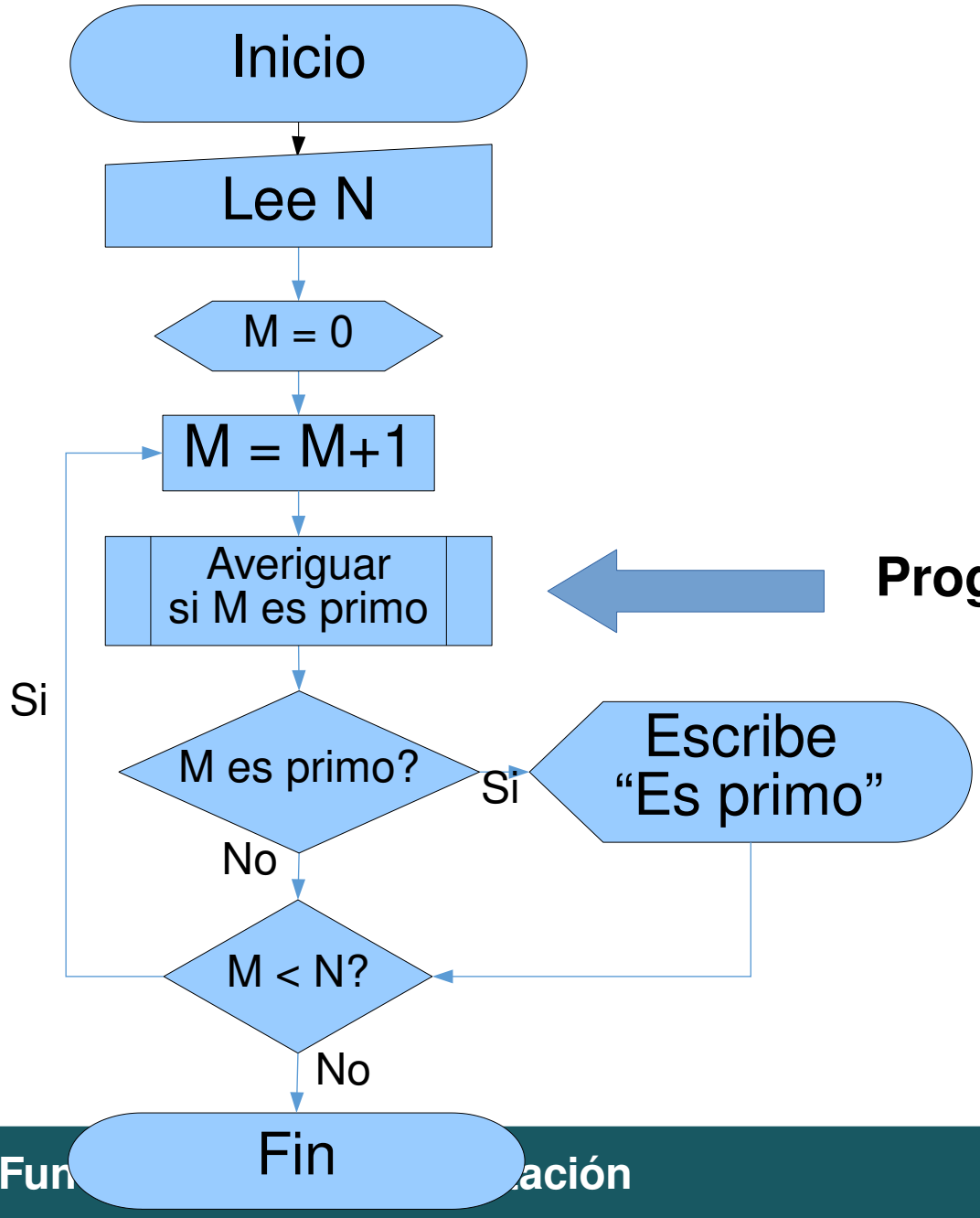


Cálculo predefinido

Averiguar si M es primo

Programación modular

Mostrar los primos menores que N



Programación modular

Funciones

Hemos visto que Octave nos permite llamar a funciones para llevar a cabo determinadas tareas o cálculos.

```
a=input('Introduce un valor: ');
b=sqrt(a);
disp('su raiz cuadrada vale: ');
disp(b);
```

Del mismo modo, Octave permite definir funciones propias para llevar a cabo otras tareas o cálculos.

```
function [a]=modulo(x,y)
    a=sqrt(x^2+y^2);
[...]
disp('Su modulo vale: ');
disp(modulo(x,y));
```

El uso de funciones definidas por el usuario permite:

- Modularizar los programas, simplificando el código.
- Simplificar la depuración del código.

Funciones: Modularización

En muchos programas es bastante común querer ejecutar un conjunto de instrucciones en varias partes de un mismo programa de forma repetitiva.
Este conjunto de instrucciones puede ser incluido dentro de una función pudiéndose *acceder* a ella en cualquier momento.

Por tanto, el uso de funciones ***evita la necesidad de programar de forma redundante*** las mismas instrucciones.

Debido a esta descomposición, los programas poseen una ***lógica más clara*** facilitando su escritura, lectura y depuración de errores.

Esto es especialmente cierto para programas complicados y largos, y es por ello que normalmente cualquier programa en Octave es modularizado siendo una ***buena costumbre de programación.***

Funciones: Definición

Una función es un sub-programa definido por el programador que es escrito usando instrucciones estándar de Octave en un fichero ***.m***, y usado como cualquier otra instrucción de Octave en cualquier otro fichero ***.m***.

Como cualquier otra instrucción de Octave, las funciones tienen un número y posición de argumentos de entrada y salida.

Una función puede ser accesible desde cualquier punto de un programa y ***una vez que la función ha llevado a cabo su tarea, el control será devuelto al punto desde donde se llamó***.

Generalmente una función procesará la información que se le pase desde el punto de llamada y devolverá unos valores. A esta información se la denomina ***argumentos o parámetros de entrada*** *y* a los valores devueltos se les denomina ***argumentos o parámetros de salida***.

Funciones: Definición

Una función tiene dos componentes principales:

- La declaración de la función con sus argumentos
- Y el cuerpo de la función

```
function [as1, as2, ... asM]=nombreFuncion(ae1, ae2, ... aeN)
Cuerpo de la función
```

- Para declarar una función el comando es **function**
- No existe límite en el número de argumentos de entrada o salida.
- Los argumentos de salida se engloban con [ ] y los de entrada con ( )

A la hora de nombrar las funciones hay que seguir los mismos criterios que para las variables.

Ejemplo:

```
function [a]=modulo(x,y)
    a=sqrt(x^2+y^2);
```

Funciones: Definición

- Las funciones tienen su propio fichero **.m**.
- Lo primero para crear una función es crear un nuevo fichero **.m**.
- La primera linea del fichero ha de ser la instrucción **function**
- **IMPORTANTE**: asegurarse de que el nombre de la función es algo fácil de recordar.
- Después de crear la función hay que salvar el fichero .m
- **IMPORTANTE**: El nombre del fichero tiene que ser el mismo que el nombre de la función
- Las lineas de código que se escriben a continuación de la instrucción **function** se denominan ***cuerpo de la función*** y contienen otras instrucciones Octave u otras funciones. Es aquí donde todos los operaciones, o lógica de la función son escritos.

Ejemplo:

```
function [a]=modulo(x,y)
    a=sqrt(x^2+y^2);
```

Funciones: Definición (en la asignatura)

Cuando se escribe una función es importante determinar los argumentos/parámetros de entrada y salida. Por tanto, es una buena idea desarrollar un diagrama de flujo o un pseudo-código del proceso antes de desarrollar la función

Ejemplo del fichero `modulo.m`:

```
function [a]=modulo(x,y)
%a=modulo(x,y) : función que calcula el modulo de un vector
%   a partir de sus coordenadas x e y.
%
%   Argumentos de entrada:
%       x: coordenada x del vector
%       y: coordenada y del vector
%
%   Argumentos de salida:
%       m: modulo del vector
a=sqrt(x^2+y^2);
```

Realizar las siguientes funciones:

- ***notaMedia.m***: que reciba un vector de notas y devuelva las notas en formato texto (suspenso: x<5; aprobado: 5<=x<7; notable: 7<=x<8.5 y sobresaliente: x>=8.5) y numérico.
- ***ecCuadratica.m***: que reciba un vector con los coeficientes de una ecuación cuadrática (a*x^2+b*x+c=0), la clasifique en función del discriminante y devuelva las soluciones y la clasificación. Las soluciones complejas se darán en función de su parte real e imaginaria. La función debe comprobar que el vector tiene 3 elementos, uno para cada coeficiente de la ecuación, y salir con un error en caso contrario.
- ***horaDia.m***: que reciba un número entre 0 y 24 con la hora y devuelva si es por la mañana (6<hora<=12), por la tarde (12<hora<=20) o de noche(20<hora<=6). La función debe comprobar que el valor introducido está en el rango adecuado y salir con un error en caso contrario.
- ***miFactorial.m***: que reciba un número natural positivo y devuelva su factorial. La función debe comprobar que se cumplen ambas condiciones y salir con un error en caso contrario.
- ***esPrimo.m***: que reciba un número natural positivo y devuelva un 0/1 en el caso de que no sea/sea primo. La función debe comprobar que se cumplen ambas condiciones y salir con un error en caso contrario.
- ***bisectriz.m***: que reciba un vector definiendo un intervalo y un número real positivo definiendo una tolerancia y devuelva la raíz del polinomio x^3-3x^2-2x+6 en ese intervalo con dicha tolerancia. En el caso de no existir raiz en el intervalo que devuelva un NaN.

Ejercicios

- Escribir una función, llamada `miTriangulo` a la cual le pasemos la longitud de los 3 lados y devuelva el área y perímetro del triangulo usando la formula de Heron

- Un vector puede venir dado por sus coordenadas (x,y) en el plano. Escribe un programa que pida al usuario las coordenadas de un vector y que muestre su modulo y dirección.

- Crea la función *binomial(i,j)* definida por:

$$binomial(i,j) = \begin{cases} \dfrac{i!}{j!(i-j)!} & \text{si } 0 \le j \le i \\ 0 & \text{en otro caso} \end{cases}$$

- Crear una función que calcule la exponencial de un número utilizando la suma de los *m* primeros términos de la serie de Maclaurin.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

  Los argumentos de entrada de esta función son el número x y el número de términos de la serie. Realizar un script que pida ambos variables, calcule la aproximación y muestre el error cometido comparándolo con la función *exp()*