COMPUTACIÓN

Curso 2019

Prof S. A. Grigera

JTP C. Grunfeld

Aux. L. Pili

Programa de la materia

Unidad 1. Introducción a la programación: Octave/Matlab.

1.1. El intėrprete Octave/Matlab. Expresiones aritmėticas. Variables escalares, matriciales y alfanumėricas. Algunas funciones para manipulación de vectores y matrices. Polinomios. Definición de funciones mediante la construcción de función anónima. Gráficas simples de funciones.

1.2. Programas. Entrada y salida por consola. Estructuras de control: secuencia, decisión, iteración. Instrucciones if, while, for. Funciones. Entrada y salida a disco: fopen, fprintf, fclose. Gráficas de datos discretos. Ajuste de datos: polyfit.

1.3. Algoritmos, programas y lenguajes. Datos e instrucciones. CPU, almacenamiento primario (memoria) y secundario (perifėricos). Dispositivos de entrada y salida.

Unidad 2. Representación de la información en sistemas digitales.

2.1. El bit. Almacenamiento de la información como colección de bits. Bytes, palabras y sus múltiplos.

2.2. Representaciones de números enteros. Bases para representación de números enteros positivos: números decimales, binarios, octales y hexadecimales. Enteros negativos: representación con signo y magnitud, inconvenientes. Representaciones en complemento a la base y complemento a la base reducida. Complemento a 2 y complemento a 1. Representación sesgada.

2.3. Representaciones de números reales. Representación de punto fijo. Representación de punto flotante: signo, exponente y mantisa. El standard IEEE 754. Aritmética en punto flotante, excepciones, Inf y NaN. Errores en la representación: error absoluto y relativo. Redondeo y truncamiento. Epsilon de la representación.

2.4. Representación de textos. Códigos de caracteres: BCD, ASCII, Unicode. Texto con formato: representación sobre texto ASCII o Unicode con “markup” (RTF, HTML, LaTeX) o binaria. Archivos de texto vs. binarios.

2.5. Representación de imágenes, sonido y video. Representación matemática de imágenes. Representación del color. Discretización:

Unidad 3. Elementos de arquitectura y organización de computadoras.

3.1. ¿Qué es una computadora? Computadoras analógicas y digitales. Arquitectura vs. organización.

3.2. Expresiones lógicas y álgebra de Boole. Circuitos digitales. Puertas lógicas: conjunto completo de puertas y programación de funciones arbitrarias. Circuitos integrados y PLAs. Circuitos combinacionales: multiplexor, decodificador, sumador. La unidad lógica y aritmética (ULA). Relojes y circuitos secuenciales: latch y flip-flop. Memorias RAM estáticas y dinámicas. Contadores.

3.3. La unidad central de procesamiento (CPU) y la arquitectura de Von Neumann. Almacenamiento primario y secundario: memoria principal, periféricos, dispositivos de entrada, de salida y de entrada/salida. Buses: comunicación con memoria y periféricos. Organización de una CPU: ALU, registros, control y bus interno. El ciclo de ejecución y las interrupciones. Lenguaje de máquina: frontera software/hardware. (Instruction Set Arquitecture). Microcódigo. Instrucciones típicas de un ISA.

3.4. El sistema operativo. Funciones del sistema operativo. Memoria virtual: páginas y fallas de página. Entrada/salida y control de periféricos ("drivers"). Procesos e hilos: contexto, alternancia de procesos, estados de un proceso. El cargador ("bootstrap loader") y el BIOS.

3.5. Lenguaje ensamblador. La traducción o compilación. División de un programa ejecutable en módulos. Reubicación de módulos en

Unidad 4. Programación en lenguaje estructurado

4.1. Introducción. Un primer programa. Proceso de compilación (programa fuente, programa objeto, bibliotecas, ejecutable). La función main() y entrada/salida elemental por consola. Argumentos de la línea de comando. Constantes numéricas, de caracteres y cadenas de caracteres. Expresiones: operadores aritméticos, lógicos, bit a bit, incremento y decremento. Asignación simple y asignación combinada con operación. Variables y tipos de datos. Definición de variables. Arreglos. Conversión entre tipos.

4.2. Funciones. La biblioteca standard y la biblioteca matemática. Estructuras de control: if else, switch, while, do, for. Interrupción de lazos y saltos: continue, break, goto.

4.3. Entrada/salida a archivos (fopen, fclose, fprintf, fscanf, fgetc, fputc, fgets). Entrada/salida sin formato (fread y fwrite). Tipo de datos struct (estructuras), typedef y union.

4.4. Declaración vs. definición de funciones: prototipos. Alcance o ámbito (scope) de variables. Variables globales. Módulos: funciones y variables externas. Símbolos públicos y privados: static. Variables automáticas y estáticas y segundo significado de static. Argumentos de funciones: noción de pasaje por valor y por referencia. Recursión.

4.5. Punteros. Punteros y arreglos. Cadenas de caracteres como arreglos de char. Algunas funciones para manipular cadenas.

Unidad 5. Introducción al cálculo numérico.

5.1. Errores numéricos: errores de representación y de discretización. Propagación de errores.

5.2. Métodos básicos de aproximación de funciones. Polinomios. Evaluación numérica de polinomios. Interpolación. Polinomio interpolante de Lagrange. Polinomios osculadores. Interpolación con splines.

5.3. Ajuste de funciones. Ajuste por mínimos cuadrados como ajuste de máxima verosimilitud. Ajuste a una recta, a una función lineal en los parámetros y general

5.4. Derivación numérica: diferencias finitas, interpolación

5.5. Resolución de ecuaciones. Ecuaciones algebraicas de segundo y tercer grado. Raíces de polinomios. Raíces de funciones continuas: métodos de bisección, secante y regula falsi. Criterio de convergencia. Funciones derivables: método de Newton-Raphson. Puntos fijos: método de iteración, condiciones de validez. Sistemas de ecuaciones lineales.

Bibliografía

Borrell i Nogueras, G. (2008), Introducción informal a Matlab y Octave,
Eaton, J. W., Bateman, D. y Hauberg, S. (2008), GNU Octave manual version
Network Theory Ltd., 3 edition.
González, R. C., Woods, R. E. y Eddins, S. L. (2009), Digital image processing
using Matlab, Gatesmark publishing.
Kernighan, B. W. y Ritchie, D. M. (1991), El lenguaje de programación C,
Prentice-Hall Hispanoamericana, Mexico, 2 edition.
Li, Z.-N. y Drew, M. S. (2004), Fundamentals of multimedia, Pearson Educatio
International.
Null, L. y Lobur, J. (2003), Essentials of computer organization and
architecture, Jones and Bartlett, Sudbury, USA.
Press, W. H., Teukolsky, S. A., Vetterling, W. T. y Flannery, B. P. (1992),
Numerical Recipes in C, Cambridge University Press, 2 edition.
Sciutto, S. (2010), Apunte de cátedra, inédito.
Stallings, W. (2003), Computer organization and architecture, Prentice Hall,
New Jersey.
Tanenbaum, A. S. (2006), Structured computer organization, Pearson Prentic
Hall, 5 edition.
Web tutorial (2013), C programming web tutorial,
http://www2.its.strath.ac.uk/courses/c/tableofcontents3 1.html.
Wikipedia (2013), Unicode, http://en.wikipedia.org/wiki/Unicode.

Octave/MatLab

Outline

Help

Para obtener ayuda

```
help cos
```

```
'cos' is a built-in function from the file libinterp/corefcn/mappers.cc

 -- Mapping Function: cos (X)
     Compute the cosine for each element of X in radians.

     See also: acos, cosd, cosh.


Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

Expresiones aritméticas

Operaciones aritméticas y funciones elementales:

```
2+3                          ans =  5
2/3                          ans =  0.66667
4+8*45*(5+2.1+2./3)          ans =  2800
23^(4/5)                     ans =  12.285
sin(pi)                      ans =    1.2246e-16
2*cos(pi/4)                  ans =  1.4142
exp(21)                      ans =    1.3188e+09
```

El punto y coma (;) al final evita la impresión del resultado.
Se pueden usar también números complejos:

```
sqrt(-2)                     ans =  0.00000 + 1.41421i
1+exp(i*pi)                  ans =  0.0000e+00 + 1.2246e-16i
```

Variables

```
a=24;
a^2+2*a+2
texto="Resultado";

ans =  626

a+b  ⟸ Error (b indefinido)

error: 'b' undefined near line 1 column 3
```

Matrices y operaciones matriciales

```
[0,2,0; 0,0,2; 0,0,0]
```

```
ans =
     0     2     0
     0     0     2
     0     0     0
```

```
A=ans;
A^2
```

```
ans =
     0     0     4
     0     0     0
     0     0     0
```

```
A^3
```

```
ans =
     0     0     0
     0     0     0
     0     0     0
```

```
det(A)
```

```
ans = 0
```

Matrices: operadores elemento a elemento

Notar el efecto del punto:

```
A=[0,2,0; 0,0,2; 0,0,0]
```

```
A =
     0     2     0
     0     0     2
     0     0     0
```

```
A.^2
```

```
ans =
     0     4     0
     0     0     4
     0     0     0
```

```
A+1
```

```
ans =
     1     3     1
     1     1     3
     1     1     1
```

```
A+eye(3)
```

```
ans =
     1     2     0
     0     1     2
     0     0     1
```

Más ejemplos de matrices

```
A=[0,2,0; 0,0,2; 0,0,0];
inv(A+eye(3))
```

```
ans =
   1  -2   4
   0   1  -2
   0   0   1
```

```
C = [ 0,3; 0,3]
```

```
C =
   0   3
   0   3
```

```
D = [ 2,2; 2,2]
```

```
D =
   2   2
   2   2
```

```
C*D
```

```
ans =
   6   6
   6   6
```

```
C.*D
```

```
ans =
   0   6
   0   6
```

```
A = [ 1 2 3];
B = [ 2 2 2];
A*B  ⟸ ERROR
A.*B
B'


A*B'
dot(A,B)
```

```
error: operator *: nonconformant arguments (op1 is
ans =
   2   4   6
ans =
   2
   2
   2
ans =  12
ans =  12
```

Subíndices

```
A = [ 1,2,3 ;
      4,5,6;
      7,8,9];
A(1,1)
A(1,3)
A(3,1)
A(:,1)  # columna 1
```

```
ans =  1
ans =  3
ans =  7
ans =
    1
    4
    7
```

Funciones para crear vectores y matrices

```
eye(4)    # Identidad
```

```
ans =
Diagonal Matrix
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

```
ones(2,3) # Unos
```

```
ans =
   1   1   1
   1   1   1
```

```
zeros(3,2) # Ceros
```

```
ans =
   0   0
   0   0
   0   0
```

Secuencias

```
linspace(0,10,10)
linspace(0,10,11)

ans =
 Columns 1 through 7:
    0.00000    1.11111    2.22222    3.33333    4.44444    5.55556    6.66667
 Columns 8 through 10:
    7.77778    8.88889   10.00000
ans =
    0    1    2    3    4    5    6    7    8    9   10

logspace(0,10,11)
linspace(10,0,11)

ans =
 Columns 1 through 6:
   1.0000e+00   1.0000e+01   1.0000e+02   1.0000e+03   1.0000e+04   1.0000e+05
 Columns 7 through 11:
   1.0000e+06   1.0000e+07   1.0000e+08   1.0000e+09   1.0000e+10
ans =
   10    9    8    7    6    5    4    3    2    1    0
```

```
n=linspace(0,5,6);
factorial(n)
2.^n
sum(n)

ans =
     1     1     2     6    24   120
ans =
     1     2     4     8    16    32
ans =  15
```

Polinomios

Los elementos de un vector pueden interpretarse como coeficientes de un polinomio

```
p = [2 1 20];
polyval(p,0)
roots(p)

p-[0,0,22]

roots(p-22)
```

```
ans =  20
ans =
  -0.2500 + 3.1524i
  -0.2500 - 3.1524i
ans =
   2   1  -2
ans =
  -0.94408
  -0.10592
```

Funciones anónimas

```
f = @(x) x^2;
f(2)
f(4)
```

```
ans =  4
ans =  16
```

Conviene definir funciones de modo que trabajen correctamente con matrices y/o vectores.

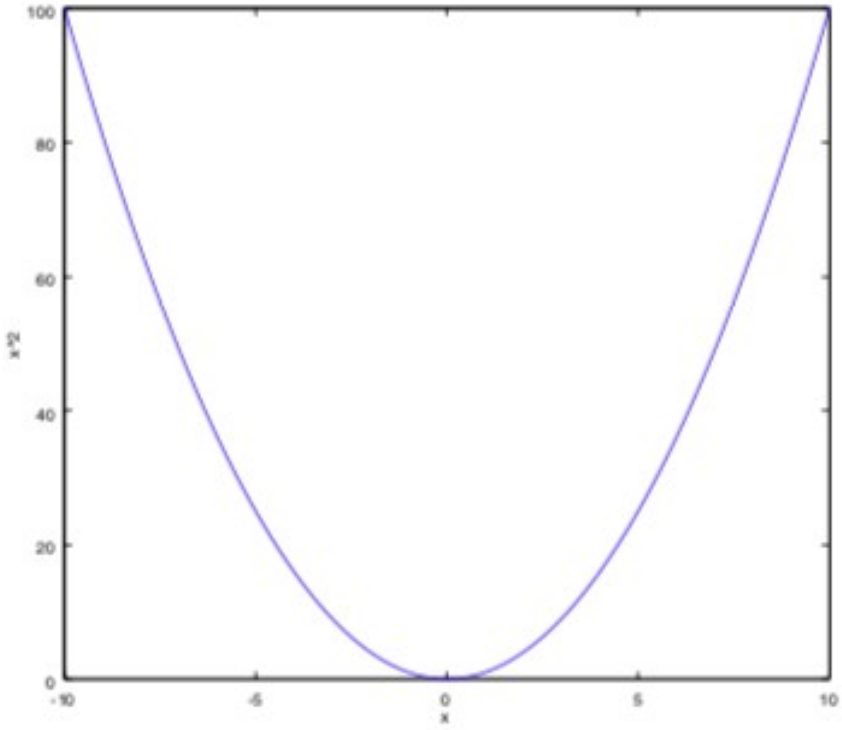
```
f = @(x) x^2;
g = @(x) x.^2;
f([2 3; 4 5])
g([1 2 3 4]) # Con f seria un error
```

```
ans =
   16   21
   28   37
ans =
    1    4    9   16
```

Gráficas de funciones

```
plot(vector de x,vector de y)
```

```
x=linspace(-10,10,100);
plot(x,x.^2);
xlabel("x");
ylabel("x^2");
```



Programas

En este caso (interpretado), también llamados *scripts*. Basta crear archivo de texto.

area.m

```
b=18;
h=21;
b*h/2
```

```
area

ans =  189
```

Entrada/salida elemental

area.m

```
b=input('Ingrese la base: ');
h=input('Ingrese la altura: ');
disp("Area:");
disp(b*h/2)
```

```
octave> area
Ingrese la base: 3
Ingrese la altura: 4
Area:
 6
```

area2.m

```
b=input('Ingrese la base: ');
h=input('Ingrese la altura: ');
printf("Area: %g\n",b*h/2);
```

```
octave> area2
Ingrese la base: 12.4
Ingrese la altura: 22
Area: 136.4
```

E/S a un archivo (disco)

Elemental

Para guardar y leer todas las variables definidas:

```
save archivo
load archivo
```

Para guardar y leer variables específicas:

```
save archivo a b c ...
load archivo a b c ...
```

Estructuras de control

Secuencia

```
instrucción 1;
instrucción 2;
instrucción 3;
...;
```

Iteración condicional

```
while (condición)
   instrucción 1;
   instrucción 2;
   ...;
end
```

Decisión

```
if (condición)
   instrucción 1;
   ...;
elseif (condición)
   instrucción 2;
   ...;
elseif ...;
else
   instrucción 3;
   ...;
end
```

Expresiones lógicas

Cómo expresar las condiciones

Espresión lógica o Booleana: una cuyo resultado es V/F

```
x>y
x==1
x>=1 && x^2<8
```

Operaciones lógicas:

- y: &&
- o: ||
- no: !

```
if (x>=0) y=sqrt(x);
else disp("ERROR: la varianza debe ser positiva");
end
r = b^2-4*a*c;
if (r>0)
   disp("Dos raices reales");
elseif (r==0)
   disp("Raices coincidentes");
else
   disp("Raices complejas conjugadas");
end
```

Iteraciones

```
i=1;
while (i<=10)
  disp(i);
  i=i+1;
end
```

```
while (yes_or_no("Sigo? "))
  ...;
end
```

Formas alternativas de iteración (pueden resolverse siempre con `while`):

do ... while

```
do
  ...;
while (yes_or_no("Sigo? "))
```

También: `break` para interrupción prematura de un bucle

for

```
for i = [ 2 , 4, 8]
  disp(i);
end
for i = 2 : 2 : 8
  disp(i);
end
for i = 1 : 10
  disp(i);
end
```

Ejemplo: búsqueda binaria

Buscar a que intervalo del vector v pertenece x.

```
i=1;
j=length(v);
while (j!=i+1)
  im = i + fix( (j-i)/2);
  if (v(im)>x) j=im;
  else i=im;
  end
end
```

(ahora j==i+1 y x está entre i y j)

Pero ¿qué pasa si x *no está* en ningún intervalo?

```
i=1;
j=length(v);
if (x<v(i) || x>v(j)) i=-1 end
while (i!=-1 && j!=i+1)
  im = i + fix( (j-i)/2);
  if (v(im)>x) j=im;
  else i=im;
  end
end
```

(ahora j==i+1 y x está entre i y j)

Funciones

```
function var = nombre(arg1,arg2,...)
  ...;
  ...;
  var=...;
end
```

Por ejemplo:

```
function i=buscar(v,x)
  i=1;
  j=length(v);
  if (x<v(i) || x>v(j)) i=-1; return end
  while (i!=-1 && j!=i+1)
    im = i + fix( (j-i)/2);
    if (v(im)>x) j=im;
    else i=im;
    end
  end
end
```

Notar el uso de `return` para terminación anticipada de la función.

Ámbito o alcance: variables locales y globales

locales Son accesibles sólo dentro de una función. El mismo nombre fuera de la función se refiere a *otra* variable.

globales Son compartidas entre todas las funciones y fuera de ellas.

Las variables *locales* a menos que se indique lo contrario:

```
global a=2;
x=1;
function f ()
  x=44;
  global a;
  a=70;
end
f();
a
x
```

```
a =  70
x =  1
```

Tiempo de vida: variables automáticas y persistentes

automáticas Variables locales que se crean cada vez que la ejecución entra en el ámbito y se destruyen al finalizar. Pierden su valor entre dos llamadas a una función.

persistentes Una vez que son creadas, existen hasta la finalización del programa (aún cuando son inaccesibles fuera de su ámbito). También llamadas *estáticas*.

```
function f()
  y=0;
  persistent x=1;
  x=x+1;
  y=y+1;
  printf("x = %d, y = %d\n",x,y);
end
f()
f()
f()
```

```
x = 2, y= 1
x = 3, y= 1
x = 4, y= 1
```

Argumentos

Los *argumentos* de una función son variables locales que toman su valor inicial de los argumentos concretos provistos en la llamada.
En Octave, los argumentos se pasan *por valor*, es decir son *copias* de los argumentos provistos:

```
function f(x)
  x=77;
end
x=1;
f(x);
x
```

```
x =  1
```

Otros lenguajes (p. ej. en FORTRAN) pasan *por referencia*: los argumentos son otros nombres para la misma variable del argumento provisto.

Argumentos como interfase

Definir una función implica definir una interfase:

- datos de entrada (en cierto orden y formato)
- datos de salida

El lenguaje permite una declaración *sintáctica* (¿cuántos argumentos hay? ¿son numéricos? etc.), pero la función define también una *semántica*, es decir que interpreta los argumentos de cierta manera y no de otra (ej: el primero es un ángulo, el segundo una longitud).
Ambos aspectos son parte de la interfase. Una buena interfase será

- clara, no ambigua
- intuitiva
- estable en el tiempo

Entrada/salida sin formato

Archivos como cajas negras

```
save [opt] file var1 var2 ...;
load [opt] file var1 var2 ...;
```

Opcion -append (para no sobreescribir) (hay muchas otras).

Entrada/salida standard

"C i/o"

Tres etapas para E/S:

Apertura Asociar el archivo a un identificador interno (*handle*) y prepararlo para E/S. Aquí se identifica el archivo, el dispositivo en el cual se encuentra y sus características (lectura, escritura, acceso secuencial o directo, etc).

Operación Operaciones de lectura/escritura, a través del "*handle*" provisto por el sistema.

Cierre Indica que han finalizado las operaciones de e/s y que el archivo debe dejarse en un estado consistente, listo para la próxima apertura. Un archivo incorrectamente cerrado puede quedar en un estado que lo haga ilegible en el futuro.

Apertura

```
[fid, err] = fopen("archivo.dat","modo");
```

`fid` identificador / handle (-1 indica error)

`err` eventual mensaje de error

`modo` puede ser

r	sólo lectura (debe existir)
w	sólo escritura (sobreescribe si existe)
a	sólo escritura (escribe al final)
r+	lectura y escritura, debe existir
w+	lectura y escritura, sobreescribe lo existente
a+	lectura y escritura, comienza al final

Recordar modo de acceso secuencial vs. directo.

Cierre

```
fclose(fid);
```

Escritura con formato

Similar a `printf`:

```
fprintf(fid,FORMATO,var1,var2,...);
```

Formato "Texto y campos %x o \n", donde %x es

%d	Entero
%f	Real
%e	Real en notación científica
%g	El más corto de %e y %f
%s	Texto

\n indica fin de línea (hay más códigos \x)

```
a=2/3;
printf("Dos␣tercios␣es␣%f,␣%g,␣%e\n",a,a,a);
b=39283943/3;
printf("Y␣b␣es␣%f,␣%g,␣%e\n",b,b,b);
```

```
Dos tercios es 0.666667, 0.666667, 6.666667e-01
Y b es 13094647.666667, 1.30946e+07, 1.309465e+07
```

Escritura con formato

Ancho del campo (para formatear columnas):

```
printf("%4d␣␣%5.2f␣%5.2f\n",2,2/3,10.1);
printf("%4d␣␣%5.2f␣%5.2f\n",20,20/3,13.4);
```

```
 2    0.67 10.10
20    6.67 13.40
```

Para imprimir matrices, iterar los elementos o bien

```
A = [1 2 3 4 5 6 7];
printf("%d␣%d\n",A);
```

```
1 2
3 4
5 6
7
```

Lectura

```
[v, c, err] = fscanf(fid,"%g",count);
```

`v`	Vector/matrix en que se almacenan los datos leídos
`c`	Número de datos efectivamente leídos
`err`	Eventual mensaje de error

`count` puede ser:

`Inf`	Lee todo lo posible, devuelve vector columna
`NF`	Lee hasta `NF` filas, devuleve vector columna
`[NF,Inf]`	Lee todo lo posible y lo organiza en una matriz con `NF` filas
`[NF,NC]`	Lee hasta `NF*NC` elementos y lo organiza en una matriz con `NF` filas

Fin del archivo

```
feof(fid)
```

Devuelve verdadero si se ha intentado leer más allá del final del archivo.

Ejemplo de lectura y feof

Leer datos de a tres

```
[f,e] = fopen(arch,"r");
if f==-1 printf("Error␣de␣lectura:␣%s\n",e);
else
  l=0;
  while (1)
    [v,c] = fscanf(f,"%g",3);
    if feof(f) break; end
    l=l+1;
    x(l)=v(1);
    y(l)=v(2);
    z(l)=v(3);
  end
end
# Ahora l es el numero de ternas leidas y almacenadas en los vectores x,y,z
```

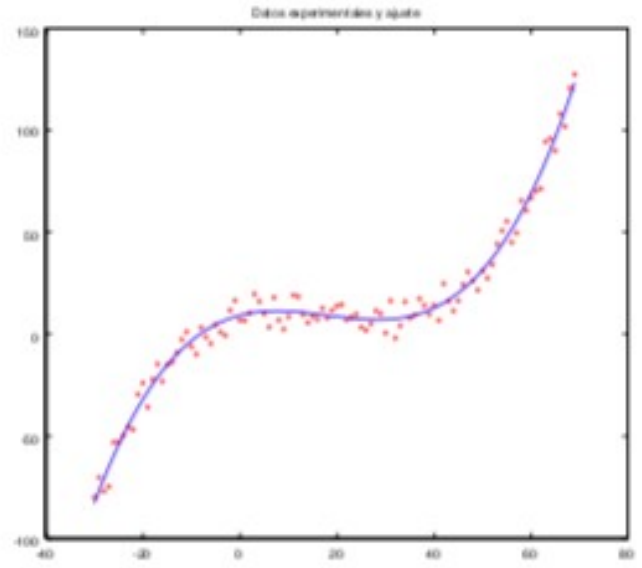
Otra solución

```
[f,e] = fopen("ar","r");
if (f==-1) printf("Error de lectura: %s\n",e);
else
  m = fscanf(f,"%g",[3,Inf]);
  x=m(1,:);
  y=m(2,:);
  z=m(3,:);
end
l = length(x)
# Ahora l es el numero de ternas leidas y almacenadas en los vectores x,y,z
```

Ajustes de polinomios

La función `polyfit` permite ajustar datos experimentales a un polinomio.

```
[x,y] = leer("DatosAjPol.dat");
fit = polyfit(x,y,3);
xmin = min(x);
xmax = max(x);
pts = linspace(xmin,xmax,150);
plot(x,y,".r",pts,polyval(fit,pts),"-b");
title("Datos␣experimentales␣y␣ajuste");
```



2. COMPUTACIÓN

7 horas semanales, 15 semanas, 105 horas

1. Introducción a los sistemas de computadoras. Primeras definiciones. Historia de las computadoras. Organización y arquitectura de las computadoras modernas. Principales componentes. El rol de los sistemas operativos y los programas de aplicación. La PC y los sistemas de cómputo. Componentes de la computadora, funciones, estructura e interconexión. El sistema de entrada/salida. Almacenamiento de la información. Periféricos. Organización de la memoria. Interrupciones. Procesamiento de instrucciones. Clasificación de las computadoras. La PC y los grandes sistemas de cómputo. Redes de computadoras e Internet. Conceptos generales.

2. Representación de la información en sistemas digitales. Sistemas de numeración binaria, octal y hexadecimal. Representación de números enteros. Magnitud y signo, complemento a 1, complemento a 2 y representación en punto fijo. Aritmética con enteros. Representación en punto flotante. El standard IEEE 754. Aritmética en punto flotante. Representación binaria de información no numérica. Códigos de caracteres: BCD y ASCII. Gráficos.

3. El sistema operativo. Objetivos y funciones. Tipos de sistemas operativos. Herramientas de programación de alto nivel: Editor, compilador, librerías, linker, make y debugger. IDE (entornos integrados de desarrollo). Prácticas comunes en la escritura de código: estilo, formato, claridad, simplicidad.

4. Programación de alto nivel. Conceptos generales. Modelización de problemas. Tipos de datos, arreglos y estructuras de datos. Operadores y expresiones. Estructuras de control y algoritmos. Procedimientos, funciones y estructura de un programa. Punteros, arreglos y estructuras. Entrada/salida. Los diferentes lenguajes de programación. Lenguajes compilados versus interpretados. La programación en lenguaje C.

5. Entornos avanzados de programación científica. Introducción al uso de entornos inteligentes y amigables para la ejecución de aplicaciones matemáticas y/o científicas. Cálculo simbólico, cálculo numérico, aplicaciones estadísticas, simulaciones Monte Carlo, aplicaciones orientadas a la Física Médica. Generación de gráficos. Generación de imágenes sintéticas. Amplia ejercitación utilizando entornos de uso difundido en la comunidad científica, como (lista sólo ejemplificativa) Mathematica, Matlab, MuPad, Opera, SciLab y/o programas similares.