INFORMÁTICA

CURSO 2023-2024

MANUEL FERNÁNDEZ DELGADO

Centro Singular de Investigación
en Tecnoloxías Intelixentes da USC (CiTIUS)
Despacho 207

Grupo de clases expositivas **CLE1**
Grupos de clases interactivas: **CLI1, CLI2, CLI3**

INFORMÁTICA
CURSO 2023-2024

EVA CERNADAS GARCÍA

Coordinadora da materia

Centro Singular de Investigación
en Tecnoloxías Intelixentes da USC (CiTIUS)
Despacho 207

Grupo de clases expositivas **CLE2**
Grupos de clases interactivas: **CLI5, CLI6** e **CLI7**

Ubicación: Centro Singular de Investigación en Tecnoloxías Intelixentes da USC (CiTIUS)

Facultade de Matemáticas

CiTIUS, 2ª planta
Despacho 207

Contidos

Ferramentas informáticas básicas en Matemáticas:

- Software de **cálculo simbólico** e de **cálculo numérico** en problemas matemáticos sinxelos**.**
- Dominar unha **linguaxe de programación estructurada**.
- Analizar, deseñar, programar e implementar algoritmos de resolución de problemas matemáticos sinxelos en distintos campos.

Obxectivos da asignatura

- **MAPLE**: programa de **cálculo simbólico** para realizar operacións matemáticas que xa coñeces (límites e derivación, integración, polinomios, etc).
- **FORTRAN**: linguaxe de **programación estructurada** para desenvolver programas de cálculo numérico.
- **MATLAB/OCTAVE**: linguaxes que permiten desenvolver programas e executar comandos para o **cálculo numérico** e a representación gráfica.

Material da asignatura

- Todo o material da asignatura atópase na páxina web (presentacións, exercicios propostos e resoltos, solucións de exames):

  http://bit.ly/1w3CChz

  http://persoal.citius.usc.es/manuel.fernandez.delgado/informatica/

- Os apuntes poden descargarse en PDF dende o enlace Apuntes da web

- https://servizosdixitais.fundacionusc.gal/lista-productos-csd/

(filtra por Profesores y selecciona Manuel Fernández/Eva Cernadas)

FUNDACIÓN USC

- Utilizaremos os foros do campus virtual da USC: **https://cv.usc.es**

*Maple 12 - /home/delgado/docencia/informatica/examenes/curso_2009_2010/control_maple/sol_cli5.mw - [Server 1]

File Edit View Insert Format Table Drawing Plot Spreadsheet Tools Window Help

Favorites | Accents | Layout | Roman Extended Lower ... | Roman Extended Upper ... | Diacritical Marks | Cyrillic | Handwriting | Expression | Units (SI) | Units (FPS) | Common Symbols

Text | Math | Drawing | Plot | Animation | Hide

2D Output | Lucida Bright | 12

$$f := x \to piecewise\left(x < -1, \sin(x), -1 \le x \text{ and } x < 0, \cos(x), 0 < x \text{ and } x \le 1, x^2 + x - 1, 1 < x \text{ and } x < 2, \ln(x), \frac{1}{x}\right) \int \sin(x)^2 \, dx = \frac{1}{2} x - \frac{1}{4} \sin(2x)$$

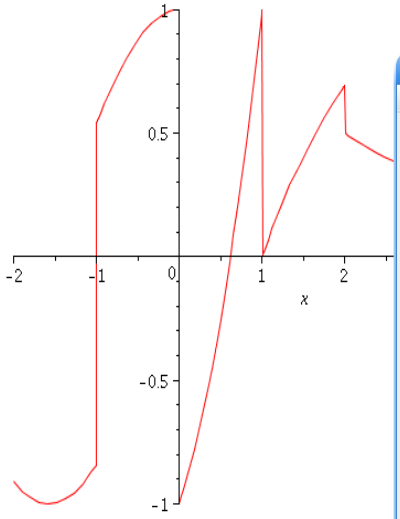
$$x \to piecewise\left(x < -1, \sin(x), -1 \le x \text{ and } x < 0, \cos(x), 0 < x \text{ and } x \le 1, x^2 + x - 1, 1 < x \text{ and } x < 2, \ln(x), \frac{1}{x}\right) \quad (1)$$

$f(x)$

$$\begin{cases} \sin(x) & x < -1 \\ \cos(x) & -1 \le x \text{ and } x < 0 \\ x^2 + x - 1 & 0 < x \text{ and } x \le 1 \\ \ln(x) & 1 < x \text{ and } x < 2 \\ \frac{1}{x} & otherwise \end{cases} \quad (2)$$

Maple

$plot(f(x), x = -2..3)$



$f := a \cdot x^2 + b \cdot y^2 + c \cdot t$

$a x^2 + b y^2 + c t$

$g := unapply(f, x, y, t)$

$(x, y, t) \to a x^2 + b y^2 + c t$

$g(0, 1, 1)$

$b + c$

$int\left(x^2 + y^2 + 2 \cdot x \cdot y, [y = -x..x, x = 0..1]\right)$

Calculus 1 - Integration Methods

File Edit Rule Definition Apply Rule Understood Rules Help

Enter a function

Function sin(x)^2 Variable x from to Start

$$\int \sin^2 x \, dx$$
$$= \int \left(\frac{1}{2} - \frac{1}{2}\cos(2x)\right) dx$$
$$= \int \frac{1}{2} dx + \int -\frac{1}{2}\cos(2x) \, dx$$
$$= \frac{1}{2} x + \int -\frac{1}{2}\cos(2x) \, dx$$
$$= \frac{1}{2} x - \frac{1}{2} \int \cos(2x) \, dx$$
$$= \frac{1}{2} x - \frac{1}{2} \int \frac{1}{2}\cos(u) \, du$$
$$= \frac{1}{2} x - \frac{1}{4} \int \cos(u) \, du$$
$$= \frac{1}{2} x - \frac{1}{4} \sin(u)$$
$$= \frac{1}{2} x - \frac{1}{4} \sin(2x)$$

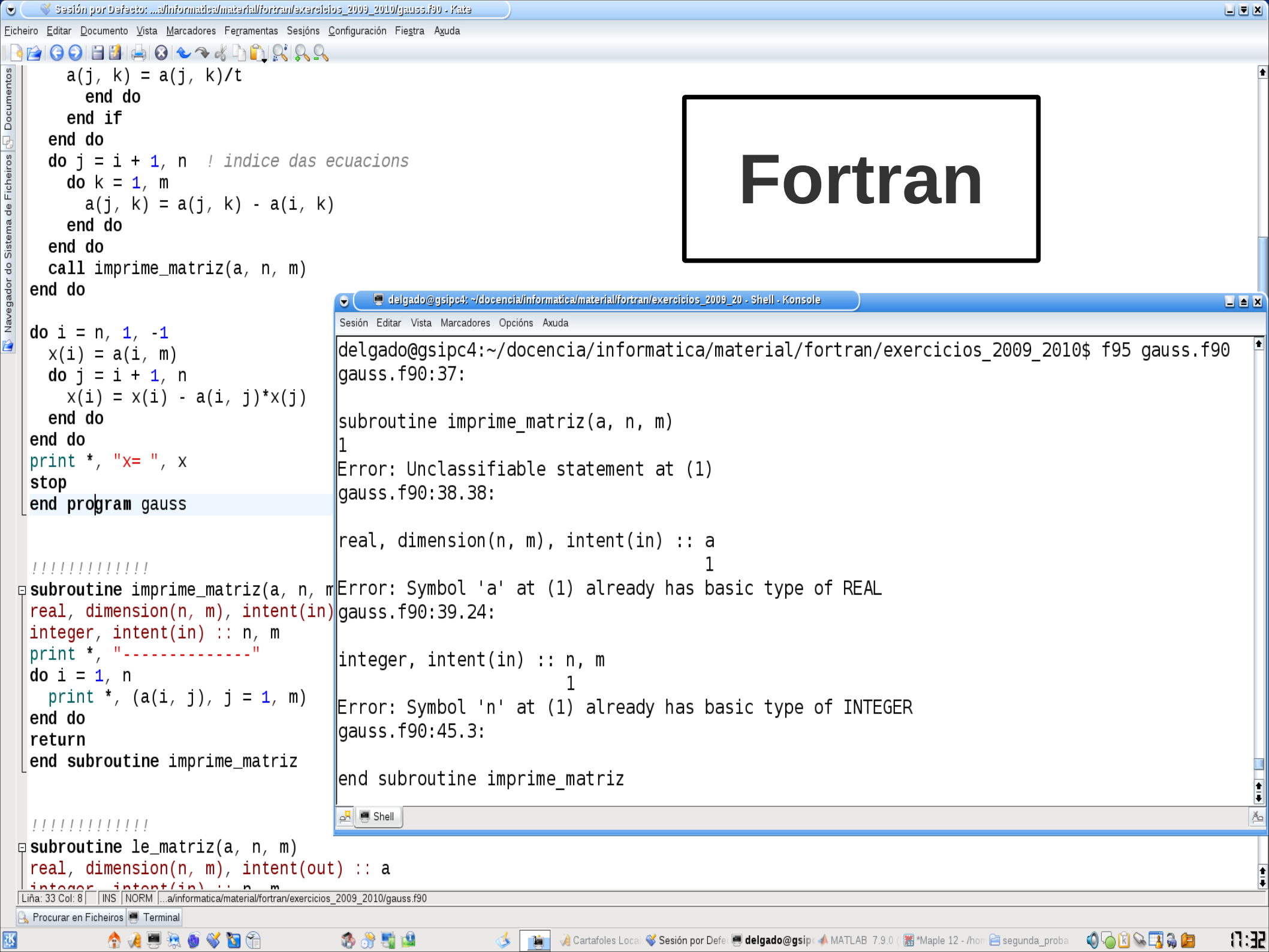
Show Hints | Get Hint

Constant | Identity
Constant Multiple | Sum
Difference | Power
Parts | Partial Fractions
Change | Revert
Solve | Rewrite
Exponential | Natural Logarithm (3)
<trig> | <hyperbolic>
<arctrig> | <archyperbolic> (4)
Flip | Join | Split (5)

Undo | Next Step | All Steps | Close

Evaluating... | Memory: 9.68M | Time: 2.24s | Math Mode

Cartafoles Local | Sesión por Defe | delgado@gsipc4 | MATLAB 7.9.0 | *Maple 12 - /ho | ftp://www-gsi.dec | 17:30


Sesión por Defecto: ...a/informatica/material/fortran/exercicios_2009_2010/gauss.f90 - Kate
Ficheiro Editar Documento Vista Marcadores Ferramentas Sesións Configuración Fiestra Axuda
a(j, k) = a(j, k)/t
end do
end if
end do
do j = i + 1, n ! indice das ecuacions
do k = 1, m
a(j, k) = a(j, k) - a(i, k)
end do
end do
call imprime_matriz(a, n, m)
end do
do i = n, 1, -1
x(i) = a(i, m)
do j = i + 1, n
x(i) = x(i) - a(i, j)*x(j)
end do
end do
print *, "x= ", x
stop
end program gauss
!!!!!!!!!!!!
subroutine imprime_matriz(a, n, m
real, dimension(n, m), intent(in)
integer, intent(in) :: n, m
print *, "-------------"
do i = 1, n
print *, (a(i, j), j = 1, m)
end do
return
end subroutine imprime_matriz
!!!!!!!!!!!!
subroutine le_matriz(a, n, m)
real, dimension(n, m), intent(out) :: a
Fortran
delgado@gsipc4: ~/docencia/informatica/material/fortran/exercicios_2009_20 - Shell - Konsole
Sesión Editar Vista Marcadores Opcións Axuda
delgado@gsipc4:~/docencia/informatica/material/fortran/exercicios_2009_2010$ f95 gauss.f90
gauss.f90:37:
subroutine imprime_matriz(a, n, m)
1
Error: Unclassifiable statement at (1)
gauss.f90:38.38:
real, dimension(n, m), intent(in) :: a
1
Error: Symbol 'a' at (1) already has basic type of REAL
gauss.f90:39.24:
integer, intent(in) :: n, m
1
Error: Symbol 'n' at (1) already has basic type of INTEGER
gauss.f90:45.3:
end subroutine imprime_matriz
Shell
Liña: 33 Col: 8 INS NORM ...a/informatica/material/fortran/exercicios_2009_2010/gauss.f90
Procurar en Ficheiros Terminal
Documentos
Navegador do Sistema de Ficheiros

MATLAB 7.9.0 (R2009b)

File Edit Debug Desktop Window Help

Current Folder: /home/delgado/docencia/informatica/material/matlab/exercicios_2009_2010

Shortcuts How to Add What's New

Current Folder

« exercicios_...

Name

- cli1
- cli2
- cli4
- cli5
- gauss.m
- gauss.m~
- matriz.dat
- matriz2.dat

gauss.m (M-file)

verifica que os coeficientes da diagonal sexan todos

Editor - /home/delgado/docencia/informatica/material/matlab/exercicios_2009_2010/g...

```
1 -   clear all
2 -   n = 3; m = 4;
3 -   x = zeros(1, n);
4 -   a = load('matriz2.dat');
5
6     % verifica que os coeficientes da diagonal sexan todos
7 -   for i = 1:n
8 -       if 0 == a(i, i)
9 -           fprintf('a incógnita %i ten coef 0 en ec %i\n', i, i);
10 -          for j = 1:n
11 -              if a(j, i) ~= 0
12 -                  break
13 -              end
14 -          end
15 -          if j < n + 1
16 -              fprintf('sumo ec %i e ec %i\n', i, j)
17 -              a(i, :) = a(i, :) + a(j, :);
18 -          else
19 -              fprintf('erro: a incógnita %i ten coe
20 -              return
21 -          end
```

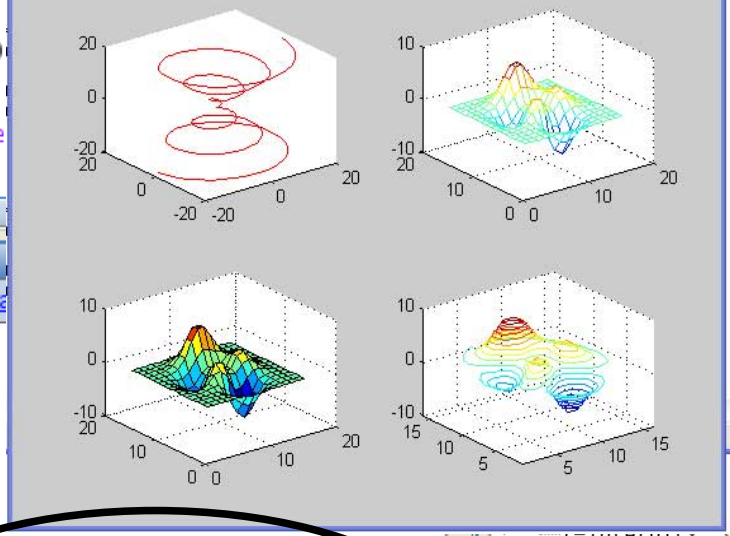
Workspace

Stack:

Name	Value
a	<3x4 double>
i	1
j	3
m	4
n	3
t	1
x	[1,0,-1]

Figure No. 1

File Edit Tools Window Help



Command Window

New to MATLAB? Watch this Video, see Demos, or read Getting Sta

```
         0   -2.0000    1.0000   -1.0000
         0   -1.0000   -0.5000    0.5000

-------
    1.0000    1.0000         0    1.0000
         0    1.0000   -0.5000    0.5000
         0         0    1.0000   -1.0000

-------
    1.0000    1.0000         0    1.0000
         0    1.0000   -0.5000    0.5000
         0         0    1.0000   -1.0000

-------
     1     0    -1

fx >>
```

```
3 AM --%
sultados_n_i
sultados_n_p
7 AM --%

= [-1 -3 -8
plot(x, y, 'o', 0:0.1:5,
syms x y; [x y]=solve((x+
quad('(x - 1)./log(x)',0,
syms x; int((x - 1)/log(x
eval(ans)
%-- 2/3/10  5:18 PM --%
gauss
```

Matlab

Start

OVR

Octave

File Edit Debug Window Help News

Current Directory: /home/delgado

File Browser

/home/delgado

Nome

- analise_exploratoria
- artigos
- asistencia_tribunais_tese
- bibliografia
- burocracia
- ccia
- citius
- Descargas
- dislin
- docencia
- Documentos
- Escritorio
- eva

Workspace

Filter

Name	Class	Dimension
fx	function_handle	1x1
fy	function_handle	1x1
fz	function_handle	1x1

Command History

Filter

```
21+26+18
# Octave 4.0.0, Wed Jun 13 18:05:37 2018 CEST <de
78*83
# Octave 4.0.0, Tue Jul 03 10:33:52 2018 CEST <delg
1+17+37+27+1
# Octave 4.0.0, Wed Jul 11 18:30:00 2018 CEST <delgado@ctde
help ezmesh
fx = @(s,t) cos (s) .* cos (t);
    fy = @(s,t) sin (s) .* cos (t);
    fz = @(s,t) sin (t);
    ezmesh (fx, fy, fz, [-pi, pi, -pi/2, pi/2], 20);
```

Command Window

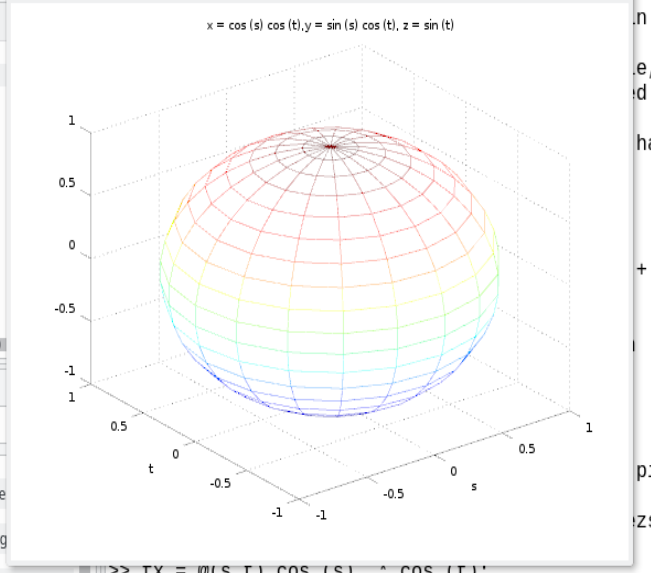
```
-- Function File: ezmesh (..., DOM)
-- Function File: ezmesh (..., N)
-- Function File: ezmesh (..., "circ")
-- Function File: ezmesh (HAX, ...)
-- Function File: H = ezmesh (...)

    Plot the mesh defined by a function.

    F is a string, inline function, or function handle with two
    arguments defining the function.  By default the plot is over
    meshed domain '-2*pi <= X | Y <= 2*pi' with 60 points in each
    dimension.

    If three functions are passed, then plot the parametrically d
    function '[FX (S, T), FY (S, T), FZ (S, T)]'.

    If DOM is a two element vector, it represents the minimum and
    maximum values of both X and Y.  If DOM is a four element vec
    then the minimum and maximum values are '[xmin xmax ymin ymax

                                                s to use in each

                                                function is plotted
                                                n DOM.

                                                e, then plot into t
                                                d by 'gca'.

                                                handle to the creat

                                                + x.^2 + y.^2);

                                                pi/2], 20);

                                                zsurfc, hidden.
>> fx = @(s,t) cos (s) .* cos (t);
>>          fy = @(s,t) sin (s) .* cos (t);
>>          fz = @(s,t) sin (t);
>>          ezmesh (fx, fy, fz, [-pi, pi, -pi/2, pi/2], 20);
>>
```

Command Window | Editor

Figure 1

File Edit Help

Z+ Z- Axes Grid Autoscale



Octave Community News

GNU Octave 4.4.1 Released

Octave Version 4.4.1 has been released and is now available for download. An official Windows binary installer is also available. — *The Octave Developers, Aug 9, 2018*

GNU Octave 4.4.0 Released

GNU Octave version 4.4.0 has been released and is now available for download. An official Windows binary installer is available. For macOS see the installation instructions in the wiki. — *The Octave Developers, Apr 30, 2018*

GNU Octave 4.2.2 Released

Octave Version 4.2.2 has been released and is now available for download. An official Windows binary installer is also available. — *The Octave Developers, Mar 13, 2018*

Octave

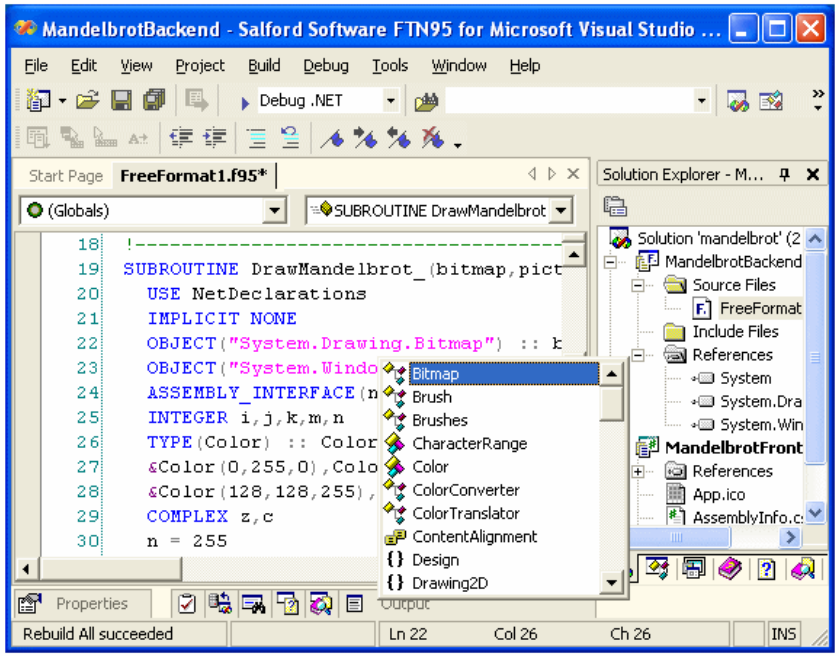
~/docencia Cartafoles pod - Kon Modelos d css : bash presentac Octave Octave Co Figure 1 17:49 03/09/18

Dispoñibilidade de software

- Maple e Matlab son programas de pago: licencias de estudante gratuítas:

  https://www.usc.gal/gl/servizos/atic/software/catalogo

- Fortran: FTN95 (Windows)



- Alternativa libre a Matlab: octave

- Nas clases interactivas usamos o entorno GNU/Linux con gfortran (non FTN95)

- Podes instalar Linux con VirtualBox (ver páxina web), gfortran e octave

Taller de instalación de Linux: mércores 20 de setembro 16,18h

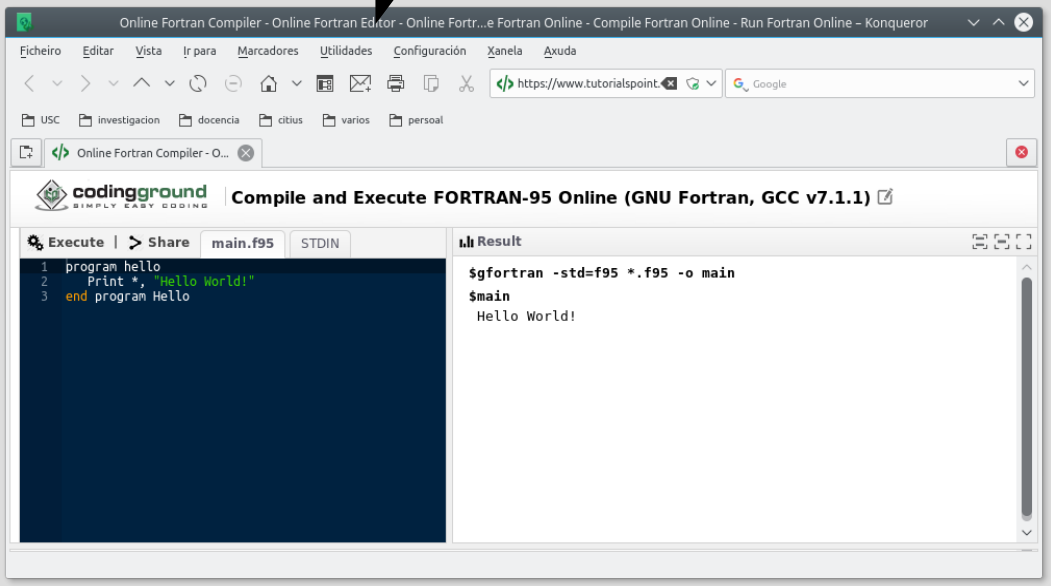
- Impartido pola **Oficina de Software Libre** da CIXUG.
- Instalación de Linux no teu portátil.
- Duración: 2 horas.
- Dúas quendas: 16h e 18h
- Moi recomendable.
- Anótate neste enlace.

CIXUG

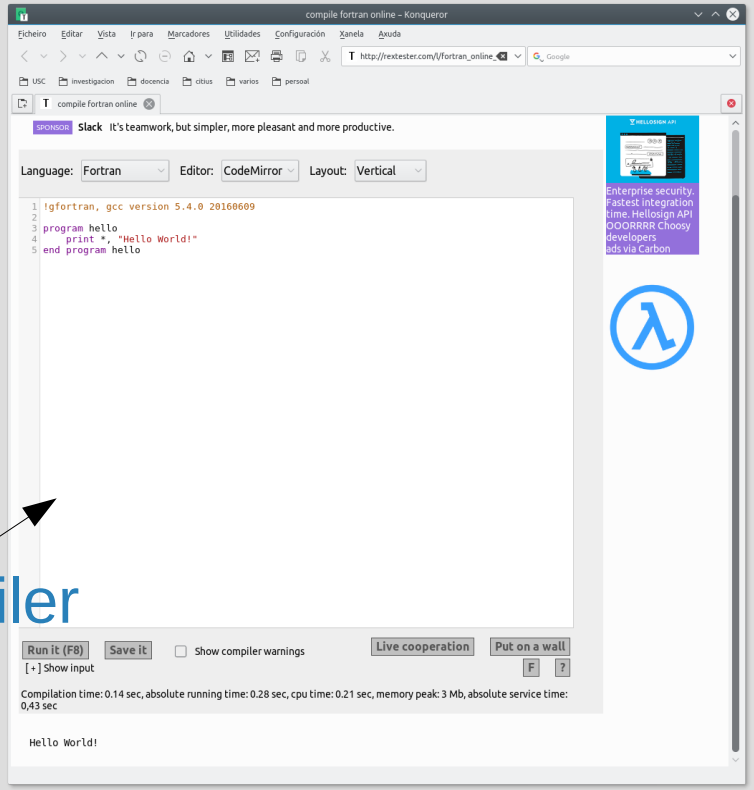
Execución on-line de Fortran

Fortran pódese executar online en dous sitios:

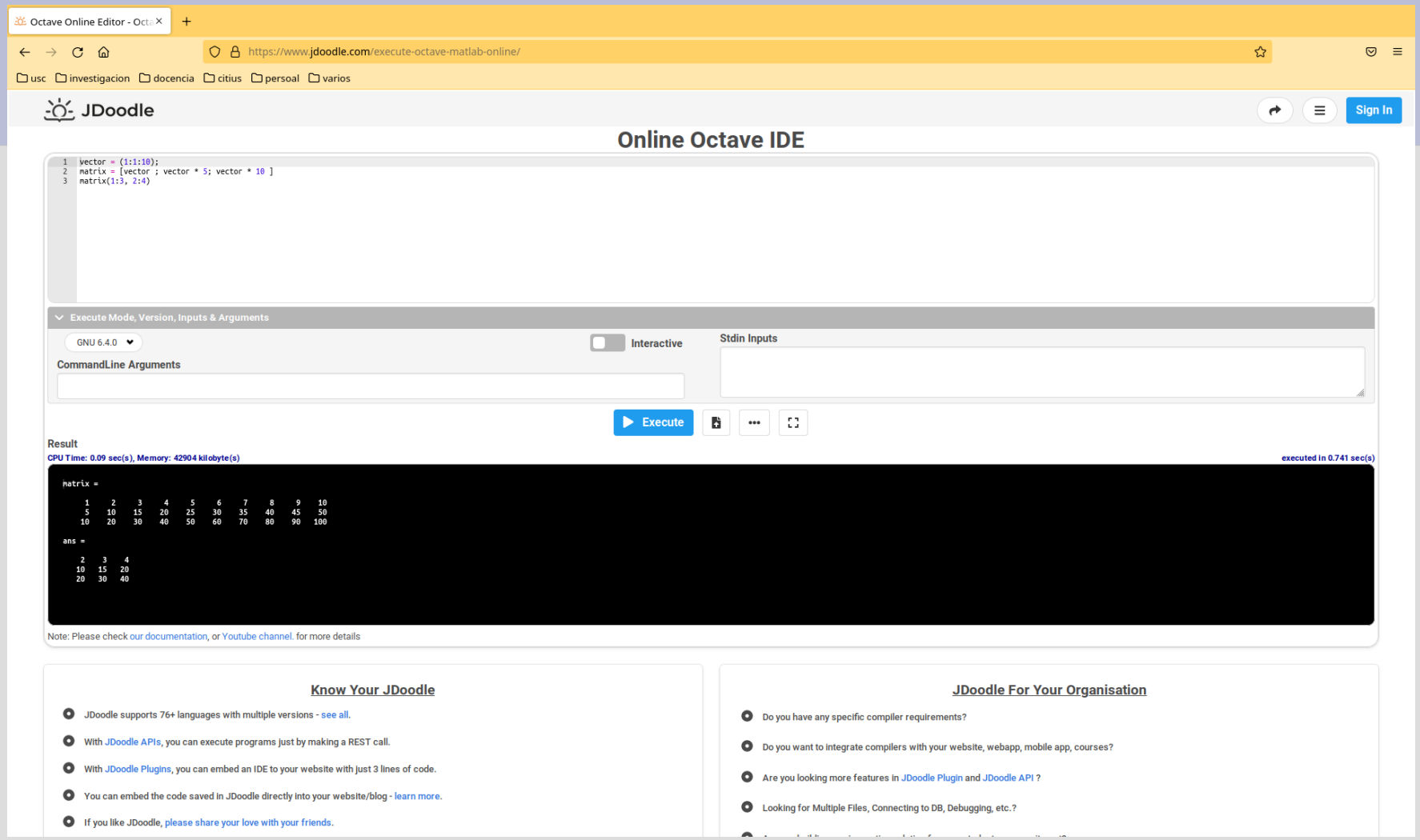
https://www.tutorialspoint.com/compile_fortran_online.php



http://rextester.com/l/fortran_online_compiler



Execución on-line de Octave



https://octave-online.net

https://www.jdoodle.com/execute-octave-matlab-online/

Metodoloxía docente

- **Clases expositivas (Fortran e Matlab)**:
  - Expoñemos os conceptos básicos da programación, escribimos e executamos exemplos representativos no ordenador.
- **Clases interactivas**:
  - **Cálculo simbólico con Maple e Matlab**: executas comandos que realizan operacións matemáticas.
  - **Fortran e Matlab**: escribes, depuras e executas **programas**, resolves incrementalmente problemas de forma planificada e razoada, adoptas decisións de deseño para optimizar a eficiencia (tempo, memoria RAM).

Avaliación

- **Avaliación continua (até 3 puntos)**: realización de exercicios durante as clases interactivas, diante do ordenador, que se entregan para a súa avaliación.
- **Exame final (10 puntos)**: exame diante do ordenador co material do curso en papel ou memoria USB. Contén 3 partes: Maple, Fortran e Matlab, tes que obter como mínimo 1 punto en cada parte.
- **Avaliación final = exame final + avaliación continua**
- Tódolos exames de anos anteriores están resoltos neste enlace

Recomendacións

- **Asistencia a clases expositivas e interactivas.**
- **Realización no ordenador os exercicios propostos por semana e revisar os exames resoltos.**
- Utilización de comandos (Maple e Matlab): **dificultade media.**
- Programación (Fortran e Matlab): **dificultade maior.**
- **Contidos fundamentais en Matlab e Fortran:** manexo de vectores e matrices, sentenzas de selección e iteración e subprogramas con paso de vectores e matrices.

Mulleres na informática

Mulleres na informática - W

https://gl.wikipedia.org/wiki/Mulleres_na_informática

usc investigacion docencia citius persoal varios

Non accedeu ao sistema Conversa Contribucións Crear unha conta Acceder ao sistema

Artigo Conversa | Ler Editar Editar a fonte Ver o historial | Procurar en Wikipedia

WIKIPEDIA
A enciclopedia libre

Portada
Portal da comunidade
A Taberna
Actualidade
Cambios recentes
Artigos de calidade
Páxina aleatoria
Axuda
Doazóns

Ferramentas
Páxinas que ligan con esta
Cambios relacionados
Páxinas especiais
Ligazón permanente
Información da páxina
Citar esta páxina
Elemento de Wikidata

Imprimir/exportar
Crear un libro
Descargar como PDF
Versión para imprimir

Noutros proxectos
Wikimedia Commons

Outras linguas
العربية
Català
Deutsch
English
Español
★ Euskara
Français
Türkçe
中文
4 máis
Editar as ligazóns

En Wiki Loves Monuments agora buscamos completar o que falta: Fotografa un monumento inédito, axuda a Wikipedia e gaña!
Coñece máis

Mulleres na informática

Na Galipedia, a Wikipedia en galego.

As **mulleres na informática** xogaron un papel determinante no seu nacemento e nos seus primeiros pasos. O que nos seus inicios se chamou "computador", tamén coñecido como computadora ou ordenador, debe o seu nome ás chamadas "computers", grupos de mulleres que tanto en Inglaterra como nos EE.UU. traballaban en cálculos matemáticos relacionados coa determinación de traxectorias balísticas durante as guerras.

Non só a primeira programadora da historia foi unha muller, Ada Byron Lovelace (1815-1852), senón que ata os anos 80, coa entrada nos fogares dos primeiros ordeadores persoais, o traballo de programación era considerado un traballo feminino. A partir dese momento as mulleres foron desaparecendo tanto dos estudos como dos traballos relacionados coa informática.

A preocupación mundial sobre o papel actual e futuro das mulleres en tarefas de computación adquiriu máis importancia coa aparición da era da información. Estas preocupacións motivaron a organización de debates públicos sobre a igualdade de xénero ao verse que as aplicacións informáticas exercen unha crecente influencia na sociedade.

Ada Lovelace, a primeira programadora da historia.

Índice [agochar]

Bibliografía

Páxina web da asignatura

- Maple: **Introduction to Maple**, A. Heck, Springer, 2003
- Fortran: **Programación estructurada con Fortran 90/95.** J. Martínez Baena, I. Requena Ramos, N. Marín Ruiz, Editorial Universidad de Granada, 2006
- Matlab: **Matlab©: Una introducción con ejemplos prácticos**. A. Gilat, Editorial Reverté

CÁLCULO SIMBÓLICO

EN MAPLE

Introducción a MAPLE

Qué é MAPLE?

Sistema de **cálculo simbólico** ou alxebraico. Maple mantén e manipula os símbolos é as expresións (non necesita valores numéricos para tódalas variabeis) . Para iniciar MAPLE utilizar o comando **xmaple**.
Internamente estruturase en 3 partes:

- **NUCLEO:** rutinas feitas e compiladas no linguaxe C onde o sistema fai a maior parte dos cálculos básicos.
- **LIBRARÍAS**: os comandos máis importantes ou habituais de MAPLE cárganse en memoria ó executa-lo programa. Os comandos de MAPLE agrúpanse en distintas librarías temáticas (indo o menú "Tools", e logo a "Tasks" e "Browse" pódense consulta-las librarías dispoñibeis en MAPLE e as funcións que contén cada unha). Cando se quere utilizar algunha función ou comando dalgunha libraría (que non se cargou en memoria por defecto ó executa-lo programa) hai que cargala en memoria explicitamente utilizando o comando **readlib(NomeFuncion)**. De todos xeitos, se imos utilizar varias funcións dunha libraría, o máis habitual é carga-la libraría completa utilizando o comando **with(NomeLibraria)**.
- **INTERFACE**: interface gráfica a través da cal nos comunicamos co sistema.

Folla de traballo de MAPLE

Entorno gráfico integrado onde, interactivamente, se resolven problemas (seleccionando no menú "insert" o modo "input maple") e se documenta o traballo (seleccionando no menú "insert" o modo "Text").
Barra de menú
Barra de Ferramentas (contén botóns con tarefas comúns).
Barra de contexto (contén botóns específicos da tarefa que se está a realizar).
A resolución de problemas interactivos redúcese a executa-los comandos axeitados de MAPLE e esperar as suas respostas.

Primeiros exemplos

Antes de nada, pulsa no boón "Gardar" e garda a folla de traballo no directorio **/Z/rai/nome.apelidos** (sendo **nome.apelidos** o correspondente á túa conta da RAI) co nome **maple.mw**

Primeiras operacións aritméticas:
Seleccionase na barra de contexto "Math" e MAPLE espera as nosas intruccións despois do símbolo > (rematadas en ; e INTRO).

```
> 2+2;
```

$$4 \qquad \textbf{(1.3.1)}$$

```
> 5*2;
```

$$10 \qquad \textbf{(1.3.2)}$$

```
> 12/8; 5^2;
```

$$\frac{3}{2}$$

$$25 \qquad \textbf{(1.3.3)}$$

```
> f:=arctan((2*x^2-1)/(2*x^2+1));
```

$$f := \arctan\left(\frac{2x^2-1}{2x^2+1}\right) \qquad \textbf{(1.3.4)}$$

```
> derivada:=diff(f,x); #derivada de f con respecto a x
```

$$derivada := \frac{\frac{4x}{2x^2+1} - \frac{4(2x^2-1)x}{(2x^2+1)^2}}{1+\frac{(2x^2-1)^2}{(2x^2+1)^2}} \qquad \textbf{(1.3.5)}$$

```
> normal(derivada);
```

$$\frac{4x}{4x^4+1} \qquad \textbf{(1.3.6)}$$

```
> valorminimo:=subs(x=0,f); #substitue x=0 en f
```

$$valorminimo := \arctan(-1) \qquad \textbf{(1.3.7)}$$

```
> valorminimo;
```

$$-\frac{1}{4}\pi \qquad \textbf{(1.3.8)}$$

```
> aprox:=evalf(valorminimo);
```

$$aprox := -0.7853981635 \qquad \textbf{(1.3.9)}$$

Tódolos nomes que apareceron (arctan(), diff(), normal(), subs() e evalf() son funcións das librarías de MAPLE (pódese consultar na axuda para que serven poñendo ***?nomefuncion***).

```
> ?diff;
```

MAPLE non evalúa todo o que ven despois do caracter # nunha liña de comandos (este símbolo utilízase para introducir comentarios).

```
> restart; # limpa a memoria interna de MAPLE
> unassign('x')  #borra a variábel x
```

Cálculos con números

Pódese utilizar MAPLE como unha calculadora. Os operadores aritméticos son: suma (+), resta (-), multiplicación (*), exponenciación (^ ou **) e factorial (!). Utiliza as regras de precedencia habituais, que se poden cambiar utilizando parénteses. Cada comando remata en "**;**" (se queremos ver o resultado da evaluación do comando) ou en "**:**" (se queremos evalua-lo comando pero non visualiza-lo resultado).

```
> 5!; 3*5^2;
```

$$120$$

$$75 \qquad \textbf{(1.4.1)}$$

Variabeis e nomes

Os nomes das variabeis son secuencias de letras, numeros e _ (o díxito inicial ten que ser letra ou _). MAPLE distingue entre maiúsculas e minúsculas (non é igual X que x). Hai un conxunto de nomes reservados que non se poden utilizar.
En MAPLE non se necesita declara-lo tipo das variabeis.
Asígnase valores ás variabeis co operador ":=". As variabeis asignadas usanse cando se quere gardar un resultado calculado ou para gardar unha expresión complicada a que se quere facer referencia posteriormente.

```
> restart;
```

```
> x := 7; y := 2*5; z:= x*y;
```

$$x := 7$$
$$y := 10$$
$$z := 70 \qquad \textbf{(1.4.1.1)}$$

```
> anames(user); #devolve as variabeis asignadas polo usuario
```

$$z, y, x \qquad \textbf{(1.4.1.2)}$$

```
> assigned(x); #devolve true se x ten asignado algún valor
  (noutro caso false)
```

$$true \qquad \textbf{(1.4.1.3)}$$

```
> whattype(5.0); # devolve o tipo de dato
```

$$float \qquad \textbf{(1.4.1.4)}$$

Enteiros e números racionais

Para a división de números enteiros simplemente simplifica a fración. MAPLE pode operar con números enteiros moi grandes (2^{19}-9=524279 díxitos).

```
> 50/8;
```

$$\frac{25}{4} \qquad \textbf{(1.4.2.1)}$$

```
>  number:=4^(4^4);
```

$$number := \textbf{(1.4.2.2)}$$
$$13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690\backslash$$
$$0318581864860508537538828119465699464336490060840960$$

```
> length(number); #número de dixitos
```

$$155 \qquad \textbf{(1.4.2.3)}$$

```
> 123456789^987654321;
    # non o executes porque o maple queda colgado
```

```
> isprime(23); #comproba se un número é primo
```

$$true \qquad \textbf{(1.4.2.4)}$$

```
> nextprime(23); #determina o próximo enteiro primo
```

$$29 \qquad \textbf{(1.4.2.5)}$$

```
> ifactor(60); #calcula os factories dun enteiro
```

$$(2)^2 (3) (5) \qquad \textbf{(1.4.2.6)}$$

```
> isqrt(3); # aproximación enteira da raíz cadrada
```

$$2 \qquad (1.4.2.7)$$

```
> a := 1234; b := 56;
```

$$a := 1234$$

$$b := 56 \qquad (1.4.2.8)$$

```
> q := iquo(a, b); #cociente dunha división enteira
```

$$q := 22 \qquad (1.4.2.9)$$

```
> r := irem(a, b); #resto dunha división enteira
```

$$r := 2 \qquad (1.4.2.10)$$

```
> a = q*b+r; #comproba a igualdade
```

$$1234 = 1234 \qquad (1.4.2.11)$$

Números irracionais e números en punto flotante

```
> x:=25^(1/6);
```

$$x := 25^{1/6} \qquad (1.4.3.1)$$

```
> simplify(x);
```

$$5^{1/3} \qquad (1.4.3.2)$$

```
> evalf(x);
```

$$1.709975947 \qquad (1.4.3.3)$$

```
> convert(x,float);
```

$$1.709975947 \qquad (1.4.3.4)$$

```
> y:=25.0^(1/6);
```

$$y := 1.709975947 \qquad (1.4.3.5)$$

```
> y^6;
```

$$25.00000003 \qquad (1.4.3.6)$$

```
> evalf(sqrt(2));
```

$$1.414213562 \qquad (1.4.3.7)$$

```
> Digits; #variábel de Maple que especifica a precisión da
  aritmética en punto flotante (pódese modificar e por
  defecto vale 10)
```

$$10 \qquad (1.4.3.8)$$

```
> Digits:=20: evalf(sqrt(2));
```

$$1.4142135623730950488 \qquad (1.4.3.9)$$

```
> evalf(Pi, 150);  #aproxima Pi con 150 decimais
```

$$3.14159265358979323846264338327950288419716939937510582\backslash \qquad (1.4.3.10)$$
$$09749445923078164062862089986280348253421170679821\backslash$$
$$48086513282306647093844609550582231725359408 13$$

```
> evalf(Pi);
    # Se non se especifica a precisión utiliza a almacenada en Digits.
```

$$3.1415926535897932385 \qquad (1.4.3.11)$$

```
> constants; # visualiza as constantes almacenadas en MAPLE
```

$$false, \gamma, \infty, true, Catalan, FAIL, \pi \qquad (1.4.3.12)$$

Constantes matemáticas en MAPLE		
Constantes matemáticas	**Nomes MAPLE**	**Valor aproximado**
π	Pi	3.141592654
e	exp(1)	2.718281828
O número de Catalan $C = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^2}$	Catalan	0.9159655942
Euler–Mascheroni's $\gamma = \lim_{n\to\infty} \left(\left(\sum_{k=1}^{n} \frac{1}{k} \right) - \ln(n) \right)$	gamma	0.5772156649
Valores lóxicos true, untrue	true, false	
∞	infinity	

Funcións habituais en MAPLE:

Nome en MAPLE	Función matemática
exp	Función exponencial
ln, log	Logaritmo natural
log10	Logaritmo en base 10
sqrt	función raíz cadrada
abs	valor absoluto
sin, cos, tan, csc, sec, cot	funcións trigonométricas
arcsin, arccos, arctan	funcións trigonométricas inversas
sinh, cosh, tanh, csch, sech, coth	funcións hiperbólicas
arcsinh, arccosh, arctanh	funcións hiperbólicas inversas

Números complexos

O número complexo $i = \sqrt{-1}$ representase en MAPLE por I. MAPLE realiza automáticamente aritmética con números complexos.

```
> x:=(2+3*I)*(4+5*I);
```

$$x := -7 + 22\,I \qquad \textbf{(1.4.4.1)}$$

```
> Re(x); Im(x); conjugate(x); |x|
```

$$-7$$

$$22$$

$$-7 - 22\,I$$

$$\sqrt{533} \qquad \textbf{(1.4.4.2)}$$

```
> argument(x);
```

$$-\arctan\left(\frac{22}{7}\right) + \pi \qquad \textbf{(1.4.4.3)}$$

```
> polar(x);
```

$$\mathrm{polar}\left(\sqrt{533},\ -\arctan\left(\frac{22}{7}\right) + \pi\right) \qquad \textbf{(1.4.4.4)}$$

```
> 1/x;
```

$$-\frac{7}{533} - \frac{22}{533}\,I \qquad \textbf{(1.4.4.5)}$$

```
> cos(I);
```

$$\cosh(1) \qquad \textbf{(1.4.4.6)}$$

```
> sqrt(-8);
```

$$2\,I\sqrt{2} \qquad \textbf{(1.4.4.7)}$$

```
> z:=1/(2+3*I);
```

$$z := \frac{2}{13} - \frac{3}{13}\,I \qquad \textbf{(1.4.4.8)}$$

Para poñer unha expresión complexa en forma cartesiana (parte real + I * parte imaxinaria), hai que usar a función **evalc(...)**

```
> sqrt(3+2*I)
```

$$\sqrt{3 + 2\,I} \qquad \textbf{(1.4.4.9)}$$

```
> evalc(sqrt(3+2*I))
```

$$\frac{1}{2}\sqrt{6 + 2\sqrt{13}} + \frac{1}{2}\,I\sqrt{-6 + 2\sqrt{13}} \qquad \textbf{(1.4.4.10)}$$

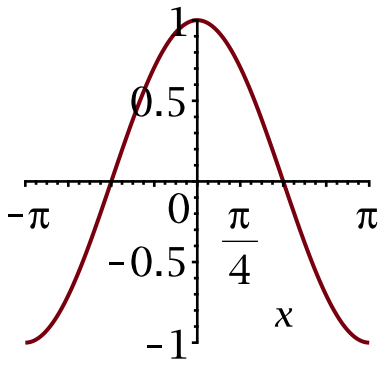
```
> evalf(%)
```

$$1.817354021 + 0.5502505225\,I \qquad \textbf{(1.4.4.11)}$$

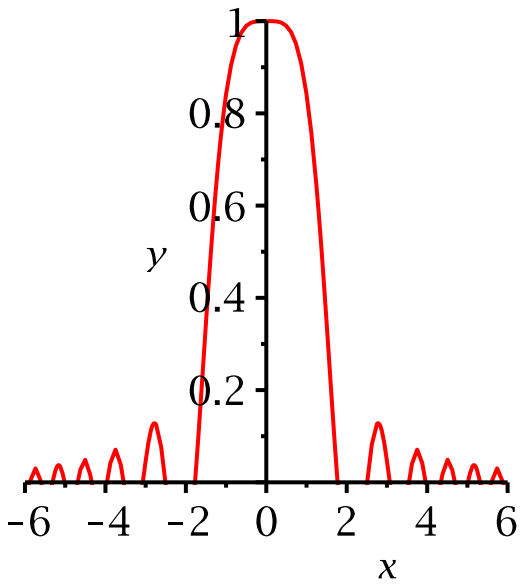
Representación gráfica básica

Plot permite representar unha función ***f*** no intervalo **(a,b)** segundo a sintaxe ***plot(f(x), x=a..b, options);*** onde *options* pódese seleccionar o tipo de liña, cor, escalas, títulos, etc. Tamén se van poder fixar posteriormente sobre a gráfica interactivamente.

```
> restart;plot(cos(x), x=-Pi..Pi, style=line); # representa a
  función cos(x) no intervalo (-Pi, Pi)
```

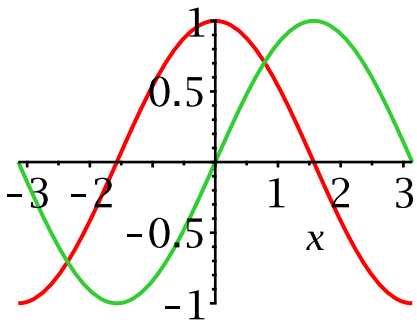



```
> plot(sin(x^2)/x^2, x=-6..6, y=0..1); # fixa a escala de y ó
  intervalo (0,1)
```



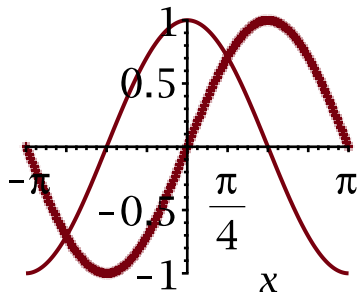
Se se quere representar máis dunha función na mesma gráfica:

```
> plot([cos(x),sin(x)],x=-Pi..Pi);
```




```
> with(plots): F:=plot(cos(x), x = -Pi .. Pi):G:=plot(sin(x),
  x = -Pi .. Pi, style=point):display({F, G}, title = `Funcións
  coseno e seno`);  # alternativa: coa función display(...)
```

Funcións coseno e seno



Vectores e matrices

MAPLE ten definidos internamente varios tipos de datos compostos: secuencias (ou secesións), conxuntos, listas, vectores, matrices e táboas.

Definición de vectores

Para definir un vector en MAPLE, utilízase o comando

Vector[o](n, init, ro, sym, sh, st, dt, f, a, o)

- **o** (opcional) especifica a orientación do vector ([row] horizontal e [column] vertical. Por defecto devolve un vector columna
- **n** (opcional) é un enteiro ou rango de enteiro especificando o número de elementos. Por defecto rechea o vector con 0.
- **init** (opcional) un procedemento MAPLE para especifica-los elementos iniciais do vector. Este procedemento pode ser a través dun tipo de dato table, array, list, Array, Matrix, Vector, conxunto de ecuacións ou expresións de tipo alxebráico.
- **ro** (opcional) Booleano. Especifica se o vector se pode modificar (false) ou non (true).
- **sym** (opcional) *symbol*=nome especifica o nome a utilizar nos elementos do vector cando non se inicializa o vector.
- **sh** (opcional) *shape*=nome ou *shape*=lista de nomes. Os posibeis nomes son: *constant[x]* (pon tódolos elementos ao valor *x*), *scalar[i, x]* (pon o elemento *i* ao valor *x*), *unit[i]* (pon a 0 tódolos elementos agás o *i*-ésimo, que pon a 1), *zero*.
- **dt** (opcional) *datatype*=nome especifica o tipo de dato almacenado no vector. Os posibeis nomes son: integer, float, complex.
- **f** (opcional) *fill*=valor, que é o valor de recheo do vector para os elementos restantes (por defecto 0).

```
> Vector(2); #vector con 2 elementos. Por defecto inicializado a 0.
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \textbf{(1.1.1)}$$

```
> Vector(1..3,5); #vector con 3 elementos inicializado a 5.
```

$$\begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} \quad \textbf{(1.1.2)}$$

```
> Vector[row]([1, x^2+y, sqrt(2)]); #vector fila inicializado con 3
  elementos
```

$$\begin{bmatrix} 1 & x^2+y & \sqrt{2} \end{bmatrix} \quad \textbf{(1.1.3)}$$

```
> f:=(j)->x^(j-1): Vector[row](6,f); # vector fila inicializado cun
  procedemento ou función MAPLE
```

$$\begin{bmatrix} 1 & x & x^2 & x^3 & x^4 & x^5 \end{bmatrix} \quad \textbf{(1.1.4)}$$

```
> s:={1=0, 2=1}: Vector(2, s);
```

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (1.1.5)$$

```
> Vector[row](5, shape = constant[x])
```

$$\begin{bmatrix} x & x & x & x & x \end{bmatrix} \quad (1.1.6)$$

```
> Vector[row](3, shape = scalar[2, infinity])
```

$$\begin{bmatrix} 0 & \infty & 0 \end{bmatrix} \quad (1.1.7)$$

```
> Vector[row](5, shape = unit[3])
```

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (1.1.8)$$

Definición de matrices

Para construir unha matriz en MAPLE utilízase o comando

Matrix(r, c, init, ro, sym, sc, sh, st, o, dt, f, a)

onde os argumentos son:

r - (opcional) filas da matriz

c - (opcional) columnas da matriz

O resto dos valores son semellantes os do comando Vector. Os valores para *shape* son:

- *identity* : pon tódolos elementos a 0 e a diagonal a 1: *Matrix(3,5,shape= identity)*
- *scalar[x]*: pon tódolos elementos da diagonal a x, o resto a 0. Ex: *Matrix(3,3, shape=scalar[4.2])*
- *diagonal*: inicializa a diagonal co vector especificado en *init*. Ex: *Matrix(3,3, Vector([3,2,1]),shape=diagonal)* ou *f:=i->x^i:Matrix(4,f,shape=diagonal)*
- *triangular[upper/lower]*. Crea unha matriz triangular co valor especificado. Ex: *Matrix(3,3,5,shape=triangular[upper])*
- *symmetric* : Forza a que a matriz sexa simétrica: *Matrix(3,3,shape= symmetric); M[1,2]:=5* (fai tamén $M_{21}=5$)
- *antisymmetric* : Igual que *symmetric*, pero forza a que $M_{ij} = -M_{ji}$

Hai moitas outras máis, correspondentes con matrices especiais.

NOTA: Podemos convertir unha matriz en vector co comando *convert*:

```
> M := Matrix(2, 3, 5);
```

$$M := \begin{bmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \end{bmatrix} \quad (1.2.1)$$

```
> convert(M, Vector[row])
```

$$\begin{bmatrix} 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix} \quad (1.2.2)$$

```
> Matrix(2); Matrix(2,3); # por defecto, rechease con 0
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (1.2.3)$$

```
> Matrix(3, shape=identity); # matriz identidade
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.2.4)$$

```
> m1:=Matrix(1..2, 1..3, 5); #definición da matriz especificando
  rango de índices (os rangos sempre empezan por 1) e recheada co
  número 5
```

$$m1 := \begin{bmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \end{bmatrix} \quad (1.2.5)$$

```
> m2 := Matrix([[1, 2, 3], [4, 5, 6]]); #inicialización dunha
  matriz con datos de tipo list
```

$$m2 := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad (1.2.6)$$

```
> Matrix(m1+m2);
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix} \quad (1.2.7)$$

```
> m3 := Matrix(3, 2, [1, 2, 3, 4, 5]); #outra inicialización da
  matriz con listas. O elemento (3,2) inicialízase por defecto a 0.
```

$$m3 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 0 \end{bmatrix} \quad (1.2.8)$$

```
> Matrix(4, 3, m3, fill = 9); # agora os elementos restantes
  inicialízanse por defecto a 9
```

$$\begin{bmatrix} 1 & 2 & 9 \\ 3 & 4 & 9 \\ 5 & 0 & 9 \\ 9 & 9 & 9 \end{bmatrix} \quad (1.2.9)$$

```
> f:=(i,j)->x^(i+j-1); Matrix(2,f); #inicialización cunha función
```

$$f := (i, j) \rightarrow x^{i+j-1}$$

(1.2.10)

$$\begin{bmatrix} x & x^2 \\ x^2 & x^3 \end{bmatrix} \quad (1.2.10)$$

```
> s:={(1,1)=0, (1,2)=1}; Matrix(1,3,s); # inicialización dos
  elementos da matriz utilizando índices
```

$$s := \{(1, 1) = 0, (1, 2) = 1\}$$

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \quad (1.2.11)$$

```
> Matrix(2, 3, symbol = m); # inicialización utilizando un símbolo
```

$$\begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \end{bmatrix} \quad (1.2.12)$$

Tamén podemos crear unha matriz diagonal coa función **DIagonalMatrix** (require o módulo **LinearAlgebra**):

```
> v := Vector[row](3, [1, 2, 3]) : with(LinearAlgebra) : DiagonalMatrix(v)
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \quad (1.2.13)$$

Operacións con matrices

A transposición de matrices faise coa función Transpose con **with (LinearAlgebra)** incluido:

```
> with(LinearAlgebra):A:=Matrix(2,3,[1,2,3,4,5,6]);Transpose(A);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad (1.3.1)$$

A **suma** e a **resta** de matrices pode realizarse como a + e - de escalares.

```
> restart;
> A:=Matrix([[23,123,7],[22,17,18],[1,2,6]]);
```

$$A := \begin{bmatrix} 23 & 123 & 7 \\ 22 & 17 & 18 \\ 1 & 2 & 6 \end{bmatrix} \quad (1.3.2)$$

```
> B:=Matrix([[1, 10,5], [0,5,1], [1, 2,3]]);
```

(1.3.3)

$$B := \begin{bmatrix} 1 & 10 & 5 \\ 0 & 5 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad (1.3.3)$$

```
> A+B;
```

$$\begin{bmatrix} 24 & 133 & 12 \\ 22 & 22 & 19 \\ 2 & 4 & 9 \end{bmatrix} \quad (1.3.4)$$

```
> A-B;
```

$$\begin{bmatrix} 22 & 113 & 2 \\ 22 & 12 & 17 \\ 0 & 0 & 3 \end{bmatrix} \quad (1.3.5)$$

```
> A.B;
```

$$\begin{bmatrix} 30 & 859 & 259 \\ 40 & 341 & 181 \\ 7 & 32 & 25 \end{bmatrix} \quad (1.3.6)$$

```
> A^2; # cálculo da podencia dunha matriz
```

$$\begin{bmatrix} 3242 & 4934 & 2417 \\ 898 & 3031 & 568 \\ 73 & 169 & 79 \end{bmatrix} \quad (1.3.7)$$

```
> A^(-1); # cálculo da matriz inversa
```

$$\begin{bmatrix} -\frac{22}{4105} & \frac{724}{12315} & -\frac{419}{2463} \\ \frac{38}{4105} & -\frac{131}{12315} & \frac{52}{2463} \\ -\frac{9}{4105} & -\frac{77}{12315} & \frac{463}{2463} \end{bmatrix} \quad (1.3.8)$$

```
> 1/A;  # forma alternativa de calcular a matriz inversa
```

$$\begin{bmatrix} -\frac{22}{4105} & \frac{724}{12315} & -\frac{419}{2463} \\ \frac{38}{4105} & -\frac{131}{12315} & \frac{52}{2463} \\ -\frac{9}{4105} & -\frac{77}{12315} & \frac{463}{2463} \end{bmatrix} \quad (1.3.9)$$

Podemos borrar unha fila ou columna coa función **DeleteRow/DeleteColumn** na

libraría **LinearAlgebra**:

```
> with(LinearAlgebra) : DeleteRow(A, 2)
```

$$\begin{bmatrix} 23 & 123 & 7 \\ 1 & 2 & 6 \end{bmatrix} \quad \textbf{(1.3.10)}$$

Podemos ver as dimensións dunha matriz coa función **Dimension**:

```
> A; Dimension(A)
```

$$\begin{bmatrix} 23 & 123 & 7 \\ 22 & 17 & 18 \\ 1 & 2 & 6 \end{bmatrix}$$

$$3, 3 \quad \textbf{(1.3.11)}$$

Para ver a diagonal da matriz como vector columna, a función **Diagonal**:

```
> Diagonal(A)
```

$$\begin{bmatrix} 23 \\ 17 \\ 6 \end{bmatrix} \quad \textbf{(1.3.12)}$$

Para calcular a traza:

```
> Trace(A)
```

$$46 \quad \textbf{(1.3.13)}$$

A función **copy** serve para crear un duplicado de algo polo que nos permite realizar copias reais de vectores e matrices.

```
> C:=B; #non se realiza unha copia real senón que só se nomea a
  matriz B co nome C
```

$$C := \begin{bmatrix} 1 & 10 & 5 \\ 0 & 5 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad \textbf{(1.3.14)}$$

```
> C[1,1]:=70: B; # se modificamos o primeiro elemento de C e
  visualizamos B comprobamos que tamén se modificou
```

$$\begin{bmatrix} 70 & 10 & 5 \\ 0 & 5 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad \textbf{(1.3.15)}$$

```
> C1:=copy(B);
```

$$C1 := \begin{bmatrix} 70 & 10 & 5 \\ 0 & 5 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad \textbf{(1.3.16)}$$

```
> C1[1,1]:=a; C1;
```

$$C1_{1,1} := a$$

$$\begin{bmatrix} a & 10 & 5 \\ 0 & 5 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad (1.3.17)$$

```
> B; #B non se modificou, so C1
```

$$\begin{bmatrix} 70 & 10 & 5 \\ 0 & 5 & 1 \\ 1 & 2 & 3 \end{bmatrix} \quad (1.3.18)$$

Funcións básicas de álxebra linear

Neste apartado descríbense algunhas das funcións máis importantes da libraría **Linear Algebra**. Para poder utiliza-las funcións da libraría hai que primeiro cargalas na memoria do ordenador mediante o comando **with(libraría).**

```
> with(LinearAlgebra): #finalizando con : carga a libraría en
  memoria, se puxésemos ; cargaría a libraría en memoria e máis
  visualizaríamos tódalas funcións da libraría.
```

Determinante dunha matriz

A función ***Determinant(A, m)*** calcula o determinante da matriz A (m é un parámetro opcional que índica o método utilizado para calcula-lo determinante). A matriz pode estar definida de xeito numérico ou simbólico.

```
> B:=Matrix([[2,23,1],[2,4,6],[1,7,3]]);
```

$$B := \begin{bmatrix} 2 & 23 & 1 \\ 2 & 4 & 6 \\ 1 & 7 & 3 \end{bmatrix} \quad (1.4.1.1)$$

```
> d:=Determinant(B);
```

$$d := -50 \quad (1.4.1.2)$$

Matriz característica e polinomio característico

A función **CharacteristicMatrix(A, lambda, outopts)** constrúe a matriz característica da matriz A (é dicir, lambda*I-A, sendo I a matriz identidade), onde **A** é unha matriz cadrada e **lambda** a variabel que se usa.

```
> A := Matrix([[1, 2, 3], [1, 2, 3],[1, 5, 6]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 5 & 6 \end{bmatrix} \quad (1.4.2.1)$$

```
> CharacteristicMatrix(A, lambda);
```

(1.4.2.2)

$$\begin{bmatrix} \lambda - 1 & -2 & -3 \\ -1 & \lambda - 2 & -3 \\ -1 & -5 & \lambda - 6 \end{bmatrix} \quad \textbf{(1.4.2.2)}$$

A función **CharacteristicPolynomial(A, lambda)** cálcula o polinomio característico da matriz cadrada A (é dicir, **(-1)^n * det(A-lambda*I**), onde I é a matriz identidade e n a dimensión da matriz A).

```
> CharacteristicPolynomial(A, lambda);
```

$$\lambda^3 - 9\lambda^2 \quad \textbf{(1.4.2.3)}$$

Cálculo dos valores e vectores propios dunha matriz cadrada

A función **Eigenvalues** cálcula os valores propios e a función **Eigenvectors** cálcula os vectores propios dunha matriz cadrada. A sintase das funcións é: **Eigenvalues(A, C, imp, o, outopts)** e **Eigenvectors(A, C, imp, o, outopts)** onde **A** é a matriz a calcular os valores propios. Os demáis argumentos son opcionais e representan: **C** e a matriz para o caso xeneralizado, **imp** é unha variabel lóxica que nos dí se vai devolve-los valores como raíz dunha ecuación(RootOf) ou como radicais, **o** é o obxeto no que queremos que se devolva o resultado (Vector, Vector [row], Vector[column] ou list) e **outopts** fai referencias as opcións de construcción do obxeto de saída.

```
> R := Matrix([[611, 196, -192, 407, -8, -52, -49, 29], [899, 113,
  -192, -71, -43, -8, -44], [899, 196, 61, 49, 8, 52], [611, 8, 44,
  59, -23], [411, -599, 208, 208], [411, 208, 208], [99, -911],
  [99]], shape = symmetric, scan = triangular[upper]);
```

$$R := \begin{bmatrix} 611 & 196 & -192 & 407 & -8 & -52 & -49 & 29 \\ 196 & 899 & 113 & -192 & -71 & -43 & -8 & -44 \\ -192 & 113 & 899 & 196 & 61 & 49 & 8 & 52 \\ 407 & -192 & 196 & 611 & 8 & 44 & 59 & -23 \\ -8 & -71 & 61 & 8 & 411 & -599 & 208 & 208 \\ -52 & -43 & 49 & 44 & -599 & 411 & 208 & 208 \\ -49 & -8 & 8 & 59 & 208 & 208 & 99 & -911 \\ 29 & -44 & 52 & -23 & 208 & 208 & -911 & 99 \end{bmatrix} \quad \textbf{(1.4.3.1)}$$

```
> Eigenvalues(R);
```

(1.4.3.2)

$$\begin{bmatrix} 0 \\ 1020 \\ 510+100\sqrt{26} \\ 510-100\sqrt{26} \\ 10\sqrt{10405} \\ -10\sqrt{10405} \\ 1000 \\ 1000 \end{bmatrix} \quad \textbf{(1.4.3.2)}$$

```
> A := Matrix([[-1, -3, -6],[3, 5, 6], [-3, -3, -4]]);
```

$$A := \begin{bmatrix} -1 & -3 & -6 \\ 3 & 5 & 6 \\ -3 & -3 & -4 \end{bmatrix} \quad \textbf{(1.4.3.3)}$$

```
> v, e := Eigenvectors(A); #o vector v son os valores propios e a
  matriz e son os vectores propios
```

$$v, e := \begin{bmatrix} 2 \\ 2 \\ -4 \end{bmatrix}, \begin{bmatrix} -2 & -1 & 1 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{bmatrix} \quad \textbf{(1.4.3.4)}$$

Triangulación dunha matriz

A función **GaussianElimination(A, m, outopts)** realiza a triangulación dunha matriz cadradautilizando eliminación gausiana. O resultado é unha matriz triangular superior das mesmas dimensións que a matriz A. Os argumentos son: **A** a matriz a triangulizar, os outros dous argumentos **m** e **outopts** son opcionais e especifican o método utilizado e outras opcións avanzadas.

```
> A :=Matrix([[8, 3, -1, -5], [4, -5, 0, -2], [-5, 8, 3, -1],[-5,
  5, -4, -9]]);
```

$$A := \begin{bmatrix} 8 & 3 & -1 & -5 \\ 4 & -5 & 0 & -2 \\ -5 & 8 & 3 & -1 \\ -5 & 5 & -4 & -9 \end{bmatrix} \quad \textbf{(1.4.4.1)}$$

```
> GaussianElimination(A);
```

(1.4.4.2)

$$\begin{bmatrix} 8 & 3 & -1 & -5 \\ 0 & -\frac{13}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{163}{52} & -\frac{175}{52} \\ 0 & 0 & 0 & -\frac{2607}{163} \end{bmatrix} \quad \textbf{(1.4.4.2)}$$

Resolución dun sistema de ecuacións lineares

A función **LinearSolve(A, B, m, t, c, ip, outopts, methopts)** resolve un sistema lineal de ecuacións do tipo **Ax=B** (devolve o vector x que satisface o sistema). O argumento **A** e unha matriz ou lista. Os demáis argumentos son opcionais: **B** é unha matriz ou vector columna, **m** especifica o método de resolución, **t** especifica as variabeis libres en solucións parametrizadas e as demáis son opcións avanzadas.

```
> M := <<(1, 1, 1, 4)>|<(1, 1, -2, 1)>|<(3, 1, 1, 8)>|<(-1, 1, -1,
  -1)>|<(0, 1, 1, 0)>>;
```

$$M := \begin{bmatrix} 1 & 1 & 3 & -1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -2 & 1 & -1 & 1 \\ 4 & 1 & 8 & -1 & 0 \end{bmatrix} \quad \textbf{(1.4.5.1)}$$

```
> LinearSolve(M);
```

$$\begin{bmatrix} \frac{25}{6} \\ \frac{4}{3} \\ -\frac{5}{2} \\ -2 \end{bmatrix} \quad \textbf{(1.4.5.2)}$$

Límites, diferenciación e integración

Definición dunha función. Diferencia cunha expresión

En MAPLE as relacións funcionais pódense definir de dous xeitos:
1. Mediante unha expresión ou fórmula
2. Cunha función matemática propiamente dita

Se temos unha expresión (caso 1):

```
> f:=x^3+1;
```

$$f := x^3 + 1 \qquad \textbf{(1.1.1)}$$

e a queremos evaluar nun punto x=3, necesítase utiliza-lo comando **subs**

```
> subs(x=3, f); # substitúe x=3 na expresión f
```

$$28 \qquad \textbf{(1.1.2)}$$

Para definir unha función propiamente dita (caso 2) utilizase o **operador frecha (->)**

```
> f:=x->x^3+1;f(3); #evaluamos f en x=3
```

$$f := x \to x^3 + 1$$

$$28 \qquad \textbf{(1.1.3)}$$

A función tamén pode ter varias variábeis:

```
> f := (x, y)->x + y
```

$$f := (x, y) \mapsto x + y \qquad \textbf{(1.1.4)}$$

```
> f(x, y)
```

$$x + y \qquad \textbf{(1.1.5)}$$

Tamén podemos definir unha función que dea como imaxe un vector de 2 ou máis valores. Por exemplo: $f := \mathbb{R}^2 \to \mathbb{R}^3$ definida por :

```
> f:=(x,y)->(x^2,x-1,exp(-x));
```

$$f := (x, y) \to (x^2, x - 1, e^{-x}) \qquad \textbf{(1.1.6)}$$

```
> f(x,y);
```

$$x^2, x - 1, e^{-x} \qquad \textbf{(1.1.7)}$$

```
> f(1,2);
```

$$1, 0, e^{-1} \qquad \textbf{(1.1.8)}$$

Conversión de expresións en funcións

Para convertir unha expresión nunha función utilízase o comando:

unapply(expr, x, y, ..)

onde:

expr - expresión
x,y,.. - nomes de variabeis

```
> expresion:=(a^2*x^3+b*exp(t)+c^3*sin(x))/(a*x^2+c*t);
```

$$expresion := \frac{a^2 x^3 + b\,e^t + c^3 \sin(x)}{a x^2 + c t} \quad (1.2.1)$$

```
> f:=unapply(expresion, x, t); #crease unha función nas
  variabeis (x,t)
```

$$f := (x, t) \to \frac{a^2 x^3 + b\,e^t + c^3 \sin(x)}{a x^2 + c t} \quad (1.2.2)$$

```
> f(0,1);
```

$$\frac{b\,e}{c} \quad (1.2.3)$$

```
> restart; # limpa a memoria interna de MAPLE
```

Operacións sobre funcións

Con MAPLE pódese realizar as operacións de suma, multiplicación e composición de funcións da seguinte forma:

```
> f:=x->ln(x)+1; g:=y->exp(y)-1;
```

$$f := x \to \ln(x) + 1$$

$$g := y \to e^y - 1 \quad (1.3.1)$$

```
> h:=f+g; h(z);
```

$$h := f + g$$

$$\ln(z) + e^z \quad (1.3.2)$$

```
> h:=f*g; h(z);
```

$$h := f\,g$$

$$(\ln(z) + 1)\,(e^z - 1) \quad (1.3.3)$$

```
> h:=f@g; # composición de función co operador @
```

$$h := f@g \quad (1.3.4)$$

```
> h(z);
```

$$\ln(e^z - 1) + 1 \quad (1.3.5)$$

```
> h:=g@f; h(z);
```

$$h := g@f$$

$$e^{\ln(z) + 1} - 1 \quad (1.3.6)$$

```
> simplify(%); #simplifica a función anterior
```

$$z\,e - 1 \quad (1.3.7)$$

```
> (f@@4)(z); # equivalente a f(f(f(f(z))));
```

$$\ln(\ln(\ln(\ln(z) + 1) + 1) + 1) + 1 \quad (1.3.8)$$

Na composición temos que fixarnos que a segunda función poda aplicarse sobre a imaxen da primeira función aplicada. Por exemplo, que o nº de saídas da 1ª función (se é unha función vectorial) coincida co nº de entradas que espera a 2ª función).

Definición de funcións por intervalos

Hai que usar o comando:

piecewise(cond_1, f_1, cond_2, f_2, ..., cond_n, f_n, f_otherwise)

onde os parámetros son:

f_i - expresión

cond_i - intervalo de definición (relación ou combinación booleana de inecuacións)

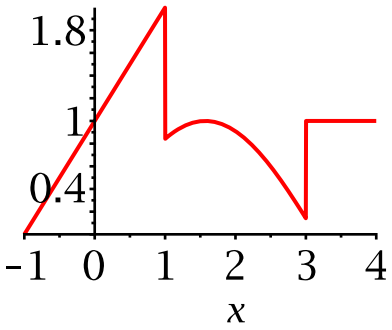
f_otherwise - (opcional) expresión por defecto

```
> f:=x->piecewise(x<=1, x+1, 1<x and x<3, sin(x), 1);f(x);
```

$$f := x \to piecewise(x \le 1, x+1, 1 < x \text{ and } x < 3, \sin(x), 1)$$

$$\begin{cases} x+1 & x \le 1 \\ \sin(x) & 1 < x \text{ and } x < 3 \\ 1 & otherwise \end{cases} \qquad \textbf{(1.4.1)}$$

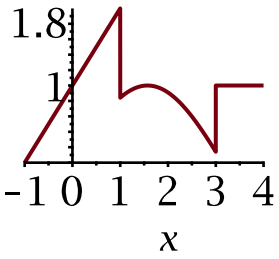
```
> plot(f(x), x=-1..4);
```



Pódese facer exactamente igual cunha expresión:

```
> f:=piecewise(x<=1, x+1, 1<x and x<3, sin(x), 1);plot(f,x=-1.
  .4);
```

$$f := \begin{cases} x+1 & x \le 1 \\ \sin(x) & 1 < x \text{ and } x < 3 \\ 1 & otherwise \end{cases}$$



Definición e representación gráfica de funcións de dúas variabeis

```
> f:=(x,y)->x^3-3*x*y^2;
```

(1.5.1)

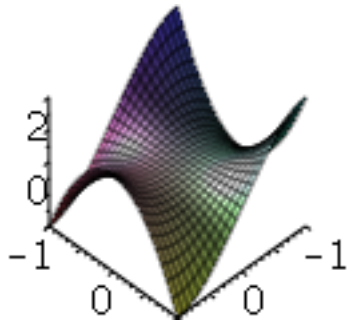
$$f := (x, y) \to x^3 - 3\,x\,y^2 \qquad \textbf{(1.5.1)}$$

```
> f(3,2); #evaluamos a función no punto (3,2)
```

$$-9 \qquad \textbf{(1.5.2)}$$

Representamos a función de 2 variábeis co comando **plot3d**:

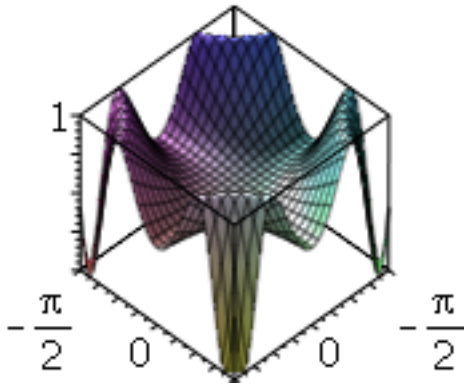
```
> plot3d(f,-1..1,-1..1, axes=FRAME, style=PATCH);
```



O mesmo podemos facer con expresións:

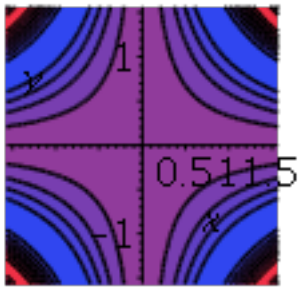
```
> f:=sin(x^2*y^2);plot3d(f,x=-Pi/2..Pi/2,y=-Pi/2..Pi/2);
```

$$f := \sin(x^2\,y^2)$$



Outro tipo de gráfica que permite representar en 2D funcións de dúas variábeis, é o **mapa de calor**, que mostra en vermello (resp. azul) as rexións do plano XY con valores elevados (resp. baixos) da función. Hai que usar o comando **contourplot**, no módulo **plots**:

```
> with(plots):contourplot(sin(x^2*y^2),x=-Pi/2..Pi/2,y=-Pi/2..
  Pi/2,filledregions=true);
```



Cálculo de límites

MAPLE pode calcular límites de expresións ou de funcións utilizando o comando

limit(f, x= a, dir) $\lim_{x \to a} f$

Limit(f, x= a, dir) (non evalúa o límite senon que so o deixa indicado)

onde os argumentos son:

f	-	expresión alxebráica
x	-	nome da variábel
a	-	punto do límite (pode ser infinity, ou -infinity)
dir	-	(opcional) refírese a dirección na que se calcula o límite, aproximando pola dereira (right) ou pola esquerda (left). Por defecto aproximase polos dous lados.

```
> limit( cos(x)/x , x=Pi/2);
```

$$0 \qquad \textbf{(1.6.1)}$$

```
> limit( (-x^2+x+1)/(x+4) , x=infinity);
```

$$-\infty \qquad \textbf{(1.6.2)}$$

```
> limit( tan(x) , x=Pi/2);
```

$$\textit{undefined} \qquad \textbf{(1.6.3)}$$

```
> limit( tan(x) , x=Pi/2 , right);
```

$$-\infty \qquad \textbf{(1.6.4)}$$

```
> limit( tan(x) , x=Pi/2, left);
```

$$\infty \qquad \textbf{(1.6.5)}$$

```
> Limit( cos(x)/x , x=Pi/2);
```

$$\lim_{x \to \frac{1}{2}\pi} \frac{\cos(x)}{x} \qquad \textbf{(1.6.6)}$$

```
> value(%); # evalúase o límite anterior
```

$$0 \qquad \textbf{(1.6.7)}$$

```
> Limit( tan(x) , x=Pi/2, left)=limit(tan(x),x=Pi/2,left);
```

$$\lim_{x \to \frac{1}{2}\pi^{-}} \tan(x) = \infty \qquad \textbf{(1.6.8)}$$

```
> value(%);
```

$$\infty \qquad \textbf{(1.6.9)}$$

Cálculo de derivadas

MAPLE pode calcula-la derivada dunha expresión con respecto a unha variábel dada utilizando o comando **Diff** ou **diff** ou o operador **D** (que opera sobre funcións). Comando **diff**:

diff(f, x1, ..., xj)	$\frac{d^j}{dx_j \ldots dx_1} f$
diff(f, [x1$n])	$\frac{d^n}{dx_1^n} f$
diff(f, x1$n, [x2$n, x3], ..., xj, [xk$m])	$\frac{d^r}{dx_k^m dx_j \ldots dx_3 dx_2^n dx_1^n} f$

onde:

f	-	expresión que se quere derivar
x1, x2, ..., xj	-	variabeis respecto as cales se calcula a derivada
n	-	orde de derivación

Tamén se pode utiliza-lo comando Diff co mesmo efecto que no caso do limite.

```
> f:=exp(-2*x);
```

$$f := e^{-2x} \quad \textbf{(1.7.1)}$$

```
> derivada:=diff(f,x); # devolve unha expresión
```

$$derivada := -2 e^{-2x} \quad \textbf{(1.7.2)}$$

```
> f_prima:=unapply(derivada, x); # agora f_prima é unha
  función
```

$$f\_prima := x \rightarrow -2 e^{-2x} \quad \textbf{(1.7.3)}$$

```
> f_prima(3); #evalúa a función nun punto
```

$$-2 e^{-6} \quad \textbf{(1.7.4)}$$

```
> Diff(x^3+2*x , x);
```

$$\frac{d}{dx}(x^3 + 2x) \quad \textbf{(1.7.5)}$$

```
> value(%); # evalúa a expresión anterior
```

$$3x^2 + 2 \quad \textbf{(1.7.6)}$$

```
> Diff(x^3+2*x , x$2); #derivada segunda
```

$$\frac{d^2}{dx^2}(x^3 + 2x) \quad \textbf{(1.7.7)}$$

```
> value(%);
```

$$6x \quad \textbf{(1.7.8)}$$

O **diff** serve tamén para derivar funcións con respecto a varias variábeis ao mesmo tempo

```
> Diff(sin(x*y),x$2,y)=diff(sin(x*y),x$2,y);
```

$$\frac{\partial^3}{\partial y \partial x^2} \sin(xy) = -\cos(xy)\, x y^2 - 2 \sin(xy)\, y \quad \textbf{(1.7.9)}$$

Se queres calcular o valor da derivada nun punto, tes que empregar o **diff**

```
> subs(x=Pi,y=-Pi,diff(sin(x*y),x,y));
```

$$\sin(-\pi^2)\,\pi^2+\cos(-\pi^2) \qquad \textbf{(1.7.10)}$$

Con funcións (definidas con ->) hai que empregar o operador **D**:

D(f) D[i](f) D[i](f)(x, y, ...)

onde os argumentos son:

f - función a derivar

i - enteiro ou enteiros, que indican a orde da variábel a respeito da cal se deriva (p.ex. se temos *f(x, y, z)*, entón **D[3,2,1](f)** é a derivada a respeito de *z*, *y*, *x*, sucesivamente

x, y, ... - punto de aplicación

O operador **D** é máis xeral que **diff** xa que pode evaluar a derivada nun punto.
Co comando **convert** pódese convertir un formato no outro.

```
> restart;
> D[1,1,2](g)(x,y);
```

$$D[1,\,1,\,2](g)(x,\,y) \qquad \textbf{(1.7.11)}$$

```
> convert(%, diff);
```

$$\frac{\partial^3}{\partial y\,\partial x^2}\,g(x,\,y) \qquad \textbf{(1.7.12)}$$

```
> Diff(x^3+2*x , x);
```

$$\frac{d}{dx}\,(x^3+2\,x) \qquad \textbf{(1.7.13)}$$

```
> convert(%, D);
```

$$3\,x^2+2 \qquad \textbf{(1.7.14)}$$

```
> restart;
> f:=x->ln(x)+sin(x); # definición dunha función
```

$$f:=x\to\ln(x)+\sin(x) \qquad \textbf{(1.7.15)}$$

```
> f_prima:=D(f); #devolve unha función
```

$$f\_prima:=x\to\frac{1}{x}+\cos(x) \qquad \textbf{(1.7.16)}$$

```
> g:=(x, y, z)->exp(x*y)+sin(x)*cos(z)+x*y*z; #función de
  varias variabeis
```

$$g:=(x,\,y,\,z)\to e^{xy}+\sin(x)\,\cos(z)+x\,y\,z \qquad \textbf{(1.7.17)}$$

```
> der:=D[2](g); #derivada respecto a y por ocupar-la segunda
  posición
```

$$der:=(x,\,y,\,z)\to x\,e^{xy}+x\,z \qquad \textbf{(1.7.18)}$$

Integración definida e indefinida

MAPLE realiza a integración definida e indefinida co comando **int**

int(expression,x)

	$\int expression\, dx$
int(expression,x=a..b)	$\int_a^b expression\, dx$
int(expression, [x = a..b, y = c..d, ...])	$\int_c^d \int_a^b expression\, dx dy$

onde os parámetros son:

expression	-	expresión a integrar
x, y	-	variabeis de integración
a, b, c, d	-	intervalo de integración no caso de integral indefinida

```
> restart;
> expresion:=2*x*exp(x^2);
```

$$expresion := 2\,x\,e^{x^2} \quad \textbf{(1.8.1)}$$

```
> int(expresion, x); # integral indefinida
```

$$e^{x^2} \quad \textbf{(1.8.2)}$$

```
> diff(%, x); # derivamos o resultado da integral e da
  expresion
```

$$2\,x\,e^{x^2} \quad \textbf{(1.8.3)}$$

```
> int(sin(y)*cos(y),y);
```

$$-\frac{1}{2}\cos(y)^2 \quad \textbf{(1.8.4)}$$

```
> int(1/exp(x^2)+x,x);
```

$$\frac{1}{2}\sqrt{\pi}\,\mathrm{erf}(x) + \frac{1}{2}\,x^2 \quad \textbf{(1.8.5)}$$

```
> int(1/x, x=2..4); # integral definida
```

$$\ln(2) \quad \textbf{(1.8.6)}$$

```
> expresion2:=1/(1+x^2);
```

$$expresion2 := \frac{1}{1+x^2} \quad \textbf{(1.8.7)}$$

```
> int(expresion2, x=0..infinity);
```

$$\frac{1}{2}\,\pi \quad \textbf{(1.8.8)}$$

O comando **Int** é interesante para visualizar a integral que estamos a realizar pero non avalía a expresión.

```
> Int(expresion2, x=0..infinity);
```

(1.8.9)

$$\int_0^{\infty} \frac{1}{1+x^2}\,dx \qquad (1.8.9)$$

```
> value(%);
```

$$\frac{1}{2}\,\pi \qquad (1.8.10)$$

Nota que a integral infinita é o límite da integral finita cando o intervalo de integración tende a infinito:

$$\int_a^{\infty} f(x)\,dx = \lim_{x\to\infty} \int_a^{x} f(t)\,dt$$

Por exemplo:

$$\int_1^{\infty} \frac{1}{x^2}\,dx = \lim_{x\to\infty} \int_1^{x} \frac{1}{t^2}\,dt$$

```
> int(1/t^2, t=1..x) assuming x::positive
```

$$\frac{x-1}{x} \qquad (1.8.11)$$

```
> limit(%, x=infinity);
```

$$1 \qquad (1.8.12)$$

```
> int(1/x^2, x=1..infinity);
```

$$1 \qquad (1.8.13)$$

No caso de integrais dobres (é dicir, de funcións de dúas variábeis):

$$\int_0^a \int_{-x}^{x} (x^2y + xy^2)\,dy\,dx = \int_0^a \left[\int_{-x}^{x} (x^2y + xy^2)\,dy \right] dx$$

```
> Int(x^2*y+x*y^2, [y=-x..x,x=0..a])=int(int(x^2*y+x*y^2, y=-
  x..x),x=0..a);
```

$$\int_0^a \int_{-x}^{x} (x^2 y + x y^2)\,dy\,dx = \frac{2}{15}\,a^5 \qquad (1.8.14)$$

```
> Int(x^2*y+x*y^2, [y=-x..x,x=0..a])=int(x^2*y+x*y^2, [y=-x..
  x,x=0..a]);
```

$$\int_0^a \int_{-x}^{x} (x^2 y + x y^2)\,dy\,dx = \frac{2}{15}\,a^5 \qquad (1.8.15)$$

Dado que a integral debe ser un número que non pode depender de *x* nin de *y*, está claro que os límites variábeis (neste caso, - *x* e *x*) corresponden á variábel *y*, e os límites constantes (neste caso, *0* e *a*) corresponden a *x*. Sempre hai que integrar primeiro con respeito á variábel que ten os límites de integración variábeis (neste caso, *y*).

Desenvolvemento en serie de funcións. Series numéricas

Aproximación dunha función mediante unha serie de potencias

O conxunto de monomios $\{x^n\}_{-\infty}^{\infty}$ constitúe unha base no espazo vectorial das funcións. Isto significa que toda función *f(x)* pódese aproximar en torno a un punto *x=a* como unha serie de potencias (tamén chamada Polinomio de Taylor) de orde *n*:

$$f(x) \approx \sum_{i=1}^{n} a_i (x-a)^i, \quad x \in [a-\varepsilon, a+\varepsilon]$$

Isto é un polinomio en *(x-a)*. A aproximación terá un erro denotado por *O* $[(x-a)^{n+1}]$ que se le "erro de orde *n*". Este erro que será menor canto maior sexa *n* (porque o polinomio ten máis coeficientes e, polo tanto, máis graos de liberdade para aproximar a función *f*) e canto menor sexa |*x-a*| (canto máis perto esteamos de *x=a*).

$$f(x) = \sum_{i=1}^{n} a_i (x-a)^i + O((x-a)^{n+1}) \quad x \in [a-\varepsilon, a+\varepsilon]$$

A serie de Taylor é unha serie particular, que toma a forma

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n$$

onde $f^{(n)}(a)$ é á *n*-ésima derivada de *f* en *x=a* : é dicir, $a_n = f^{(n)}(a)/n!$.

Utilízanse as función **series(expr, x=a)** ou **series(expr, x=a, n)** para a aproximación dunha expresión nunha serie e a función **taylor(expr, x=a, n)** para a descomposición nunha **serie ou polinomio de Taylor**. Ambas realizan un desenvolvemento nunha serie de orde **n** (de Taylor no caso da función **taylor**) da expresión **expr** no punto **x=a**. Os argumentos son: **expr** a expresión a desenvolver, **x** o nome da variabel independente, **a** (x=0 se non se especifica o punto) o punto de descomposición e **n** a orde de descomposición.

```
> series(x/(1-x-x^2), x = 0);
```

$$x + x^2 + 2x^3 + 3x^4 + 5x^5 + O(x^6) \qquad \textbf{(1.1.1)}$$

```
> series(x/(1-x-x^2), x); # se non se especifica o punto
  considérase x=0
```

$$x + x^2 + 2x^3 + 3x^4 + 5x^5 + O(x^6) \qquad \textbf{(1.1.2)}$$

```
> s1:=series(x/(1-x-x^2), x, 10);
```

$$s1 := x + x^2 + 2x^3 + 3x^4 + 5x^5 + 8x^6 + 13x^7 + 21x^8 + 34x^9 + O(x^{10}) \quad \textbf{(1.1.3)}$$

```
> type(s1, 'series'); #comproba se s1 é de tipo serie
```

true **(1.1.4)**

```
> type(s1, 'polynom'); # comproba se s1 é de tipo polinomio
```

false **(1.1.5)**

```
> p1:=convert(s1, 'polynom'); # convirte a serie nun polinomio
```

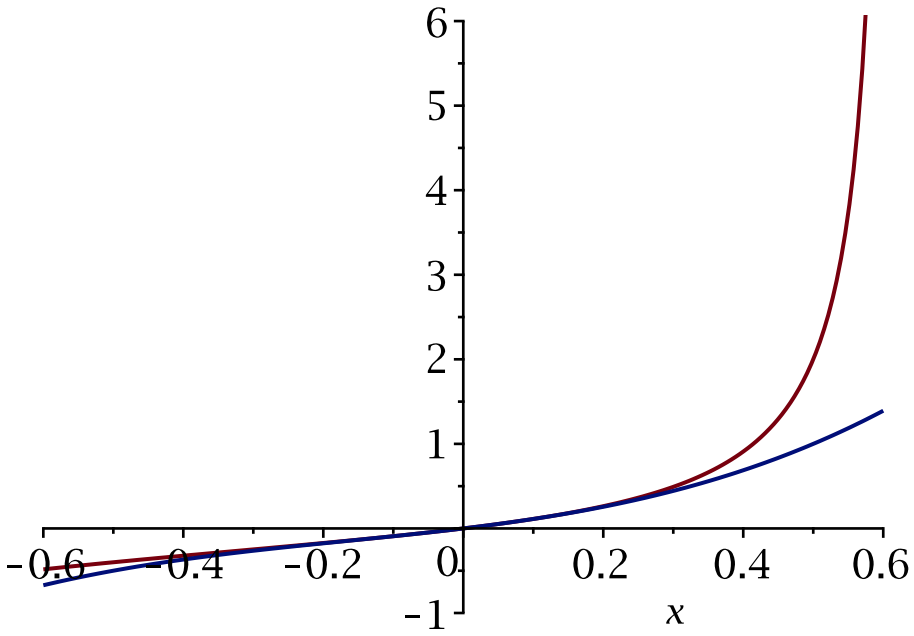
$$p1 := x + x^2 + 2x^3 + 3x^4 + 5x^5 + 8x^6 + 13x^7 + 21x^8 + 34x^9 \quad \textbf{(1.1.6)}$$

```
> type(p1, 'polynom');
```

true **(1.1.7)**

Esta serie ou polinomio de Taylor aproxima á función (neste caso *f(x)=x/(1-x-x²)*) nun entorno de *x=0*: a aproximación é mellor cando máis perto esteamos de *x=0* e canto meirande sexa a orde *n* do polinomio. Podemos comprobar esta aproximación co comando:

```
> f:=x/(1-x-x^2):plot([f, convert(series(f,x=0,4),polynom)],x=
  -0.6..0.6);
```



Aumentando a orde da serie (no caso anterior, 4), acádase unha mellor aproximación. Nota que a diferencia aumenta cando nos alonxamos de *x=0*.

```
> series(x^3/(x^4+4*x-5), x = infinity);
```

$$\frac{1}{x} - \frac{4}{x^4} + \frac{5}{x^5} + O\left(\frac{1}{x^7}\right) \quad \textbf{(1.1.8)}$$

```
> taylor(exp(x), x = 0, 6); #serie de Taylor
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6) \quad \textbf{(1.1.9)}$$

```
> type(%, 'series');
```

true **(1.1.10)**

```
> taylor(sin(x), x = Pi);
```

$$-(x-\pi)+\frac{1}{6}(x-\pi)^3-\frac{1}{120}(x-\pi)^5+O\left((x-\pi)^6\right) \qquad \textbf{(1.1.11)}$$

Podemos calcular o coeficiente *i* coa función **coeff(p, x^i)**:

```
> coeff(p1,x^2);
```

$$1 \qquad \textbf{(1.1.12)}$$

onde *p1* é o polinomio de Taylor.

Tamén se pode calcula-lo desenvolvemento en serie de Taylor de funcións de varias variábeis (p.ex. *f(x,y)*) con **mtaylor()**. Por exemplo, *f(x, y)* = $e^{x^2+y^2}$ pódese desenvolver usando:

```
> mtaylor(exp(x^2+y^2),[x=0,y=0],8);
```

$$1+x^2+y^2+\frac{1}{2}x^4+y^2x^2+\frac{1}{2}y^4+\frac{1}{6}x^6+\frac{1}{2}y^2x^4+\frac{1}{2}y^4x^2+\frac{1}{6}y^6 \qquad \textbf{(1.1.13)}$$

Suma de series numéricas

As series finitas numéricas e infinitas son da forma:

$$\sum_{k=1}^{n} a_k \qquad \sum_{n=1}^{\infty} a_n = \lim_{k\to\infty} \sum_{n=1}^{k} a_n$$

Utilízase o comando **sum** ou **Sum** (non evalúa a expresión) combinado con **value**.

sum(f, k= m..n)

$$\sum_{k=m}^{n} f$$

onde:

f	-	expresión
k	-	nome do índice do sumatorio
m, n	-	extremos do sumatorio (enteiros ou expresións)

A función *sum( )* calcula a expresión simbólica dunha suma (finita ou infinita). Para calcular unha suma finita dun número constante de sumandos (non *k*, senón p.ex. 5 sumandos), mellor usa-lo comando *add(f, i = m..n)*. Exemplo: $add(i^2, i=1..5)$

```
> Sum(1/k!, k=0..infinity); value(%);
```

$$\sum_{k=0}^{\infty} \frac{1}{k!}$$

$$e \qquad \textbf{(1.2.1)}$$

```
> Sum(1/n^2,n=1..infinity)=sum(1/n^2,n=1..infinity);
```

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{6}\pi^2 \qquad \textbf{(1.2.2)}$$

Productos de series

Os produtos finitos están definidos por

$$\prod_{i=1}^{n} a_i = a_1 a_2 \ldots a_n$$

E os produtos infinitos están definidos por

$$\prod_{n=1}^{\infty} a_n = \lim_{k\to\infty} \prod_{n=1}^{k} a_n$$

En Maple, estos productos calcúlanse coa función **product(f, k=1..n)**. A función **Product(f,k=1..n)** fai o mesmo pero só mostra a expresión. Por exemplo, para calcular:

$$\prod_{n=0}^{\infty}\left[1 - \frac{4}{(2\cdot n+1)^4}\right]$$

```
> Product(1-4/(2*n+1)^4,n=0..infinity)=product(1-4/(2*n+1)^4,
  n=0..infinity);
```

$$\prod_{n=0}^{\infty}\left(1 - \frac{4}{(2\,n+1)^4}\right) = \sin\left(\pi\left(\frac{1}{2} + \frac{1}{2}\,I\sqrt{2}\right)\right)\sin\left(\pi\left(-\frac{1}{2}\sqrt{2} + \frac{1}{2}\right)\right) \qquad \textbf{(1.3.1)}$$

Representación gráfica

A visualización dos resultados pode ser unha ferramenta interesante a hora de interpreta-los resultados. MAPLE permite realizar gráficos en 2 e 3 dimensións. MAPLE dispón dunha gran variedade de comandos para a representación gráfica de expresións que se poden invocar en liña de comandos ou a través da interface gráfica interactiva indo ó menú "Tools", "Assistants", "Plot builder".

¿Qué se pode representar? Pequeno catálogo do máis básico:

- Curvas planas en forma explícita, paramétrica, implícita ou polar.
- Animacións dunha función *f(x)* que varía co tempo ou con outra variabel.
- Sistemas de inecuacións lineares en 2 variabeis.
- Curvas en 3D dadas en forma paramétrica.
- Superficies en 3D dadas en forma explícita, paramétrica, implícita ou en coordenadas esféricas/cilíndricas.
- Animacións dunha función *f(x,y)*, correspondente a unha superficie en 3D, que varía co tempo.
- Conxuntos ou listas de puntos.

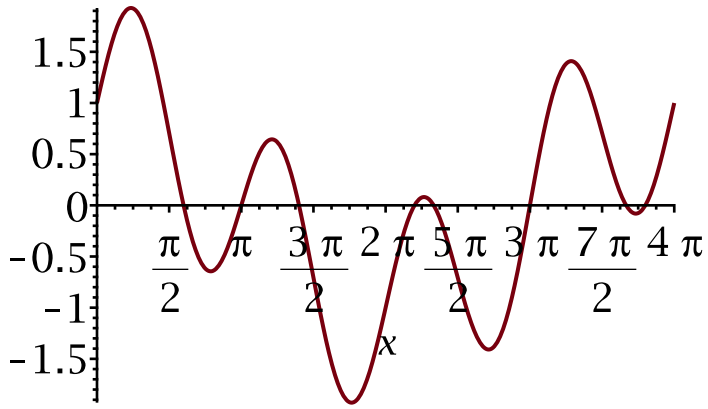
¿Qué opcións se poden variar no gráfico?

- Modifica-lo tamaño e cor de tódolos elementos do gráfico (eixos, títulos, gráficos, etc).
- Modifica-las escalas dos eixos.
- Fixa-lo intervalo de representación da curva, superficie, etc.
- Poñer lendas os eixos e título ó gráfico
- Nos gráficos en 2D: elexir se queremos ve-los puntos unidos por liñas, só liñas ou só puntos. Dependendo do caso seleccionado, pódese fixa-lo tipo e cor da liña e do punto.
- Graba-lo gráfico en formatos de gráficos habituais (JPEG, GIF e EPS).
- Copialo e pegalo a un documento noutra aplicación (Word)
- Etc.

A continuación veñen algúns exemplos en liña de comandos que presentan as posibilidades de representación gráfica de MAPLE. Os gráficos que construimos co programa quedan insertados no propio documento e, con posterioridade, poderíanse modifica-las opcións do gráfico. En primeiro lugar, cargaremos en memoria a libraría plots porque máis tarde ímola necesitar:

Gráficos en 2D

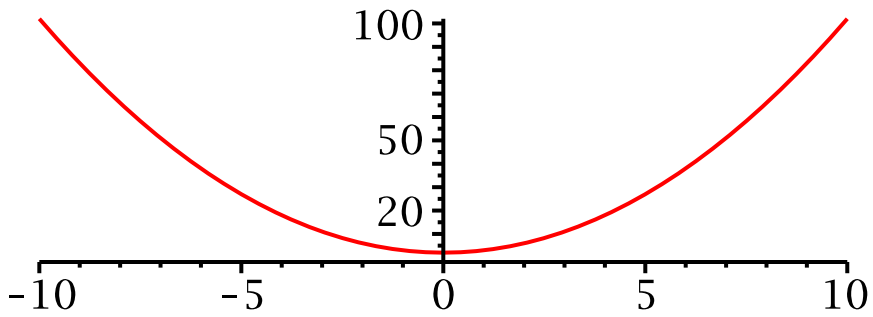
```
> with(plots):plot(cos((1/2)*x)+sin(2*x), x = 0 .. 4*Pi); #
  expresión especificando o rango no eixo x
```




```
> f:=x->x^2+2; # defino unha función
```

$$f := x \to x^2 + 2 \qquad \textbf{(1.1.1)}$$

```
> plot(f); # as escalas no eixo x e y establécense por
  defecto.
```



Cando se pulsa co rato no gráfico, actívanse os menús **Plot** e **Drawing**, que proporcionan botóns para configura-lo gráfico (grid, escalas do eixo, cores e tamanos de liñas, puntos e texto, títulos de gráfico e eixos, lendas, ...) e para engadir debuxos (menú **Drawing**).

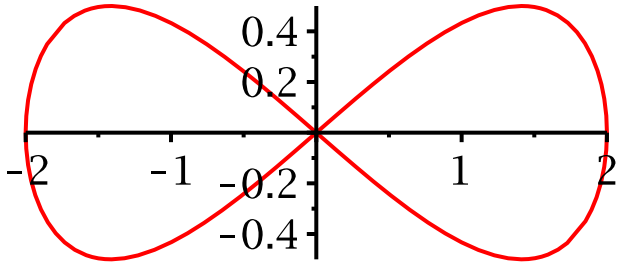
Unha **curva paramétrica** ven dada por ecuacións en función dun parámetro, por exemplo, nun movemento en 2D uniforme en X (con velocidade inicial v_{0x}) e uniformemente acelerado en Y (con velocidade inicial v_{0y} e aceleración a_y) a posición $(x(t),y(t))$ está dada por:

$$x(t)=x_0+ v_{0x}t$$

$$y(t)=y_0+ v_{0y}t +\frac{1}{2} a_y t^2$$

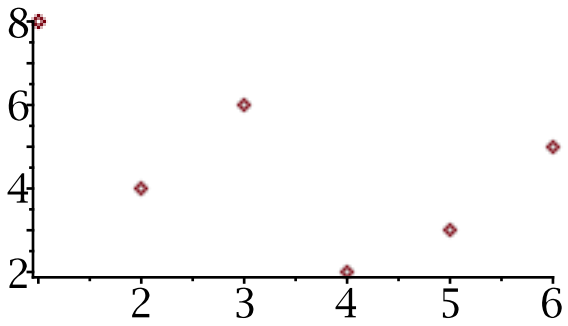
Neste caso, o parámetro é t (o tempo). Para representar unha curva paramétrica:

```
> plot([2*sin(t), sin(t)*cos(t), t=0..2*Pi]); # visualización
  dunha curva en forma paramétrica
```



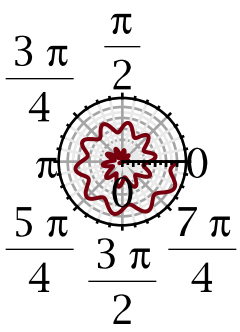
A curva paramétrica defínese como unha lista na que se especificas tanto as expresións como o rango de valores. Para non ver distorsionada a curva especifícase a opción *scaling=constrained* (que fixa o eixo x e y a mesma escala). Tamén se pode especificar premendo no botón dereito do rato (unha vez estea seleccionado o gráfico).

```
> plot([1, 2, 3, 4, 5, 6],[8, 4, 6, 2, 3, 5], style=point,
  symbolsize=20); #para visualizar un vector de puntos
  respecto a outro
```

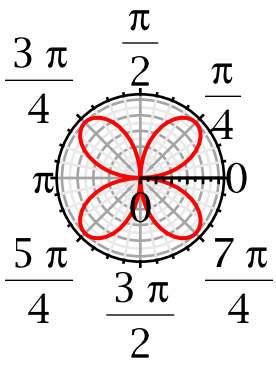


Unha **curva en coordenadas polares** está dada polas coordenadas ρ (radio) e θ (ángulo) de cada punto, de forma $\rho = \rho(\theta)$. As ecuacións de cambio de coordenadas cartesianas a polares son $\rho=\sqrt{x^2+y^2}$, $\theta = \arctan\left(\frac{y}{x}\right)$. As ecuacións de cambio de polares a cartesianas son $x=\rho \cdot\cos(\theta)$, $y=\rho\cdot\mathrm{sen}(\theta)$. Unha curva en coordenadas polares está dada por unha ecuación da forma $\rho=\rho(\theta)$. Por exemplo $\rho(\theta) = 4\theta$ (espiral de Arquímedes)

```
> polarplot(4*theta+5*sin(10*theta), theta=0..4*Pi);
```

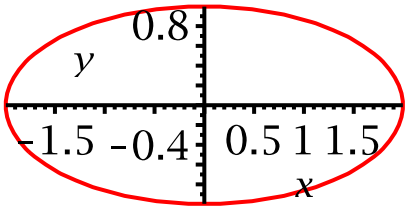



```
> polarplot(sin(2*theta));
```

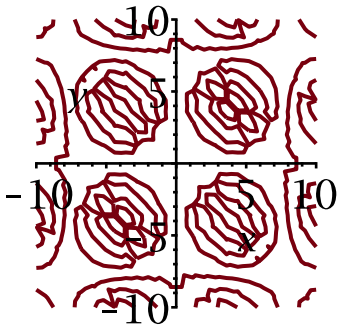


Unha **curva** (en 2D) **en forma implícita** ten a forma *f(x, y) = 0*. Por exemplo, $x^2 + y^2 - 1 = 0$ é a ecuación implícita dunha circunferencia de radio 1 e centro (0, 0). Para representar unha curva implícita:

```
> implicitplot((x^2)/4+y^2=1, x=-4..4, y=-1..1, scaling=
  constrained); #debuxa curvas en duas dimensións dadas de
  forma implícita
```

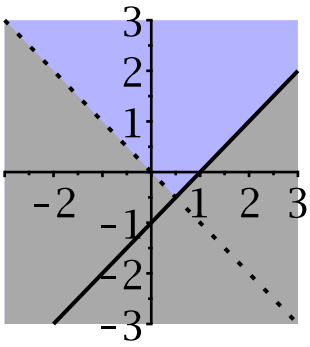



```
> implicitplot(x·y·sin(x·y) + 1, x = -10..10, y = -10..10)
```



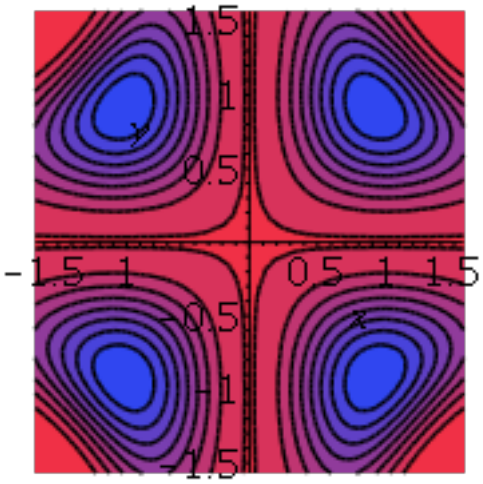
Unha **inecuación** é unha relación dada polos operadores > e < no canto de =. A función **inequal** ofrece a posibilidade de representar **sistemas de inecuacións lineais** en dúas variabeis. A sintase é: **inequal(ineqs, xspec, yspec, options)**, onde **ineqs** son as inecuacións, **xspec** e **yspec** son os rangos nos que se representa e **options** representa o modo de representación (ver axuda).

```
> inequal({x-y <= 1, 0 < x+y}, x = -3 .. 3, y = -3 .. 3);
```



Finalmente, o **mapa de calor**, coa función **contourplot**, representa cun código de cores (vermello= valores baixos, azul= valores altos) os valores dunha función de dúas variábeis no plano:

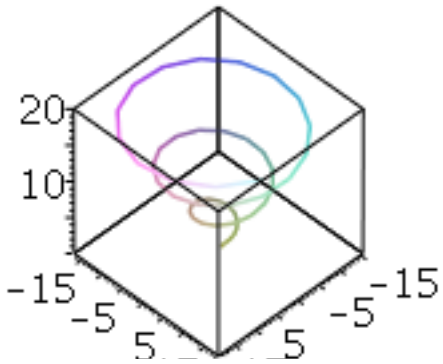
```
> contourplot(exp(-x^2-y^2)*sin(x^2*y^2),x=-Pi/2..Pi/2,y=-
  Pi/2..Pi/2,filledregions=true);
```



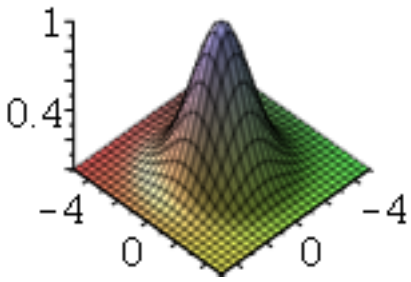
Gráficos en 3D

Os gráficos tridimensionais poden ser de varios tipos. Unha **curva en 3D** exprésase usualmente en **ecuacións paramétricas** *x=x(t), y=y(t), z=z(t)*, en función dun parámetro *t*. Para representala, emprégase a función ***spacecurve([x(t),y(t),z(t)],t=a..b)***. Por exemplo, dada a curva *x(t)=t cos t; y(t)=t sen t; z=t*, para representala executaremos:

```
> with(plots):spacecurve([t*cos(t),t*sin(t),t],t=0..20);
```



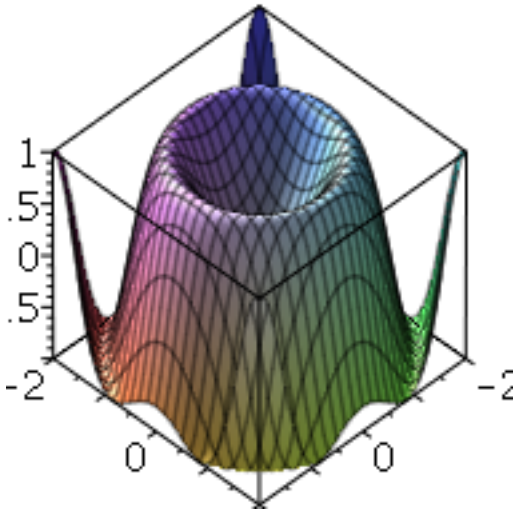
Unha **superficie 3D** ten a forma $z = f(x, y)$: $z = exp\left(- \frac{x^2 + y^2}{a^2} \right)$ é unha gaussiana de ancho a.



plot3d permite representar gráficos tridimensionais definidos por unha expresión de 2 variabeis definindo o rango no que se quere representar ou como unha función de 2 variabeis (poñemos os rangos sen incluir as variabeis).

```
> plot3d((x^2+y^2)*sin(x+y), x=-2..2, y=-2..2,axes=NORMAL);
  #utilizando unha expresión
```

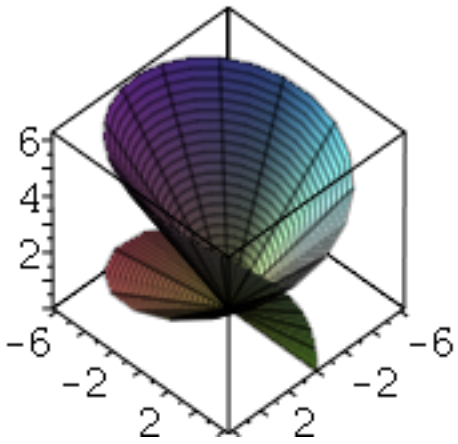

```
> f:=(x,y)->sin(x^2+y^2):plot3d(f, -2..2, -2..2); #utilizando
  unha función
```



Gráficos en 3 dimensións utilizando **funcións paramétricas** dependentes de 2 parámetros u e v, superficies dadas polas ecuacións: $x = x(u, v)$, $y = y(u, v)$, $z = z$

(u, v)

```
> plot3d([u*sin(v^2), u*cos(v^2), u*sin(v)], u=0..2*Pi, v=0..
  Pi); #con expresións
```



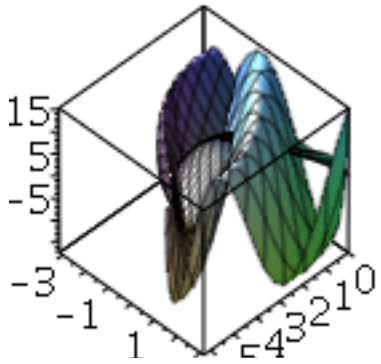
Se pinchas co rato no gráfico actívase o menú **Plot** e podes facer cousas co gráfico (rotalo, cambia-lo recheo, os eixos, entre outros).

```
> f:=(x,y)->x*sin(y); g:=(x,y)->y*cos(x); h:=(x,y)->x*y*sin(x*
  y);plot3d([f,g,h], 0..2*Pi, 0..Pi);
```

$$f := (x, y) \to x \sin(y)$$

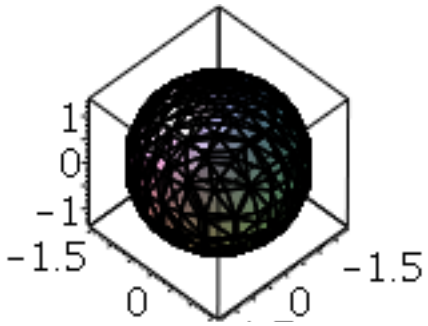
$$g := (x, y) \to y \cos(x)$$

$$h := (x, y) \to x\, y \sin(x\, y)$$



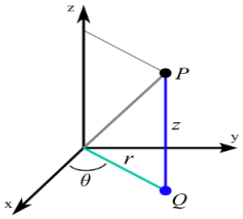
Superficies en 3D en forma implícita:

```
> restart:with(plots):implicitplot3d(2*x^2+3*y^2+4*z^2=8, x=
  -2..2, y=-1.7..1.7, z=-2..2);
```



Xa que $2 \cdot x^2 + 3 \cdot y^2 + 4 \cdot z^2 = 8$ é a ecuación dun elipsoide en 3D

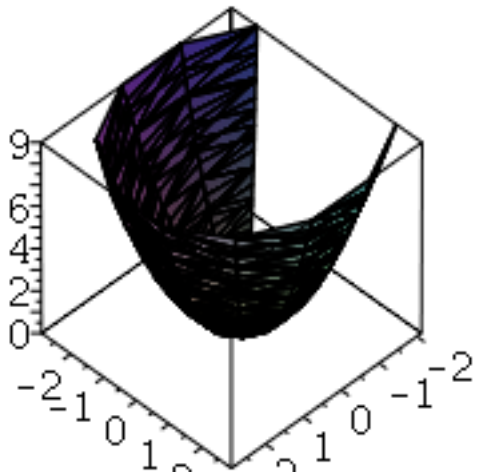
Superficies en coordenadas cilíndricas: As coordenadas cilíndricas (ρ, θ, z) teñen o significado mostrado nesta figura, e as ecuacións de transformación a coord. cartesianas móstranse na dereita:



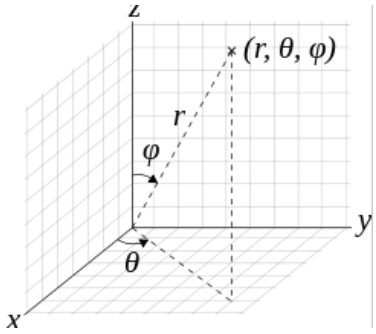
$x = \rho * \cos\theta$
$y = \rho * \text{sen}\,\theta$
$z = z$

$$\rho = \sqrt{x^2 + y^2}$$
$$\theta = arctan\left(\frac{y}{x}\right)$$
$$z = z$$

```
> implicitplot3d(rho^2=z, rho=0..3, theta=-3*Pi/4..3*Pi/4, z=
  0..9, coords=cylindrical);
```



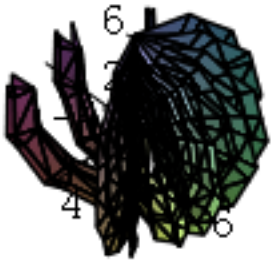
Superficies en coordenadas esféricas *(r , q, f)***.** As coordenadas esféricas teñen o significado mostrado nesta figura, e as ecuacións de transformación a coord. cartesianas móstranse na dereita:



$$x = \rho\,\text{sen}(\phi)\,\cos(\theta)$$
$$y = \rho\,\text{sen}(\phi)\,\text{sen}(\theta)$$
$$z = \rho\cos(\phi)$$

$$\rho = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arctan\left(\frac{\sqrt{x^2 + y^2}}{z}\right)$$
$$\phi = \arctan\left(\frac{y}{x}\right)$$

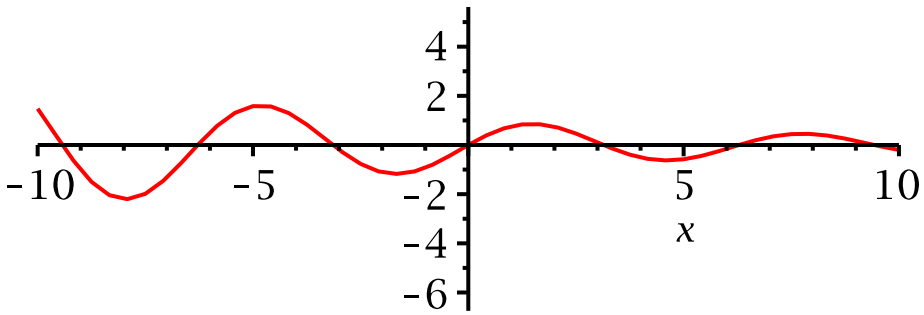
```
> implicitplot3d(rho=sin(10*phi)*2^(theta/2), theta=-2*Pi..2*
  Pi, phi=0..Pi, rho=0..5, coords=spherical,axes=NORMAL);
```



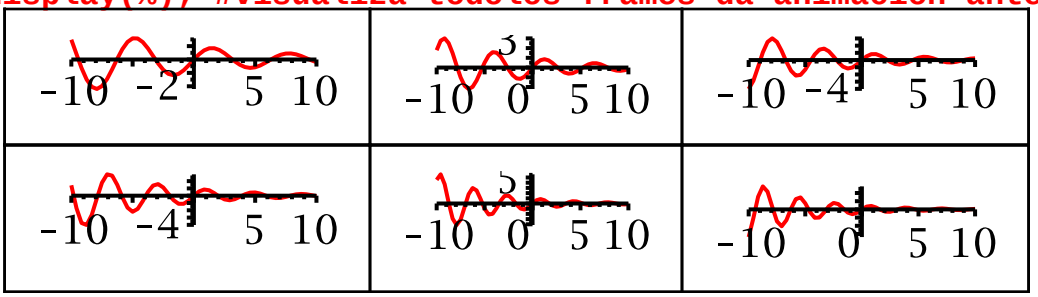
Animacións

MAPLE realiza animacións en 2D e 3D. Unha animación representa unha función que varía no tempo ou con algún parámetro. En 2D a sintase básica é **animate(F, rango_x, rango_t)** onde **F(*x*,*t*)** é a función a animar, **rango_x** o rango da variábel **x** e **rango_t** o rango do parámetro de animación **t** (o tempo). Posteriormente, pódese modifica-las características da animación interactivamente e ve-la animación como se fose un vídeo. Cando se pulsa co rato no gráfico, actívase o menú **Animation** (xunto con **Plot** e **Drawing**) e pódese reproducci-lo vídeo e configura-la reproducción (velocidade, etc.).

```
> animate(exp(-0.1*x*t)*sin(x*t), x=-10..10, t=1..2, frames=6)
  ;
```

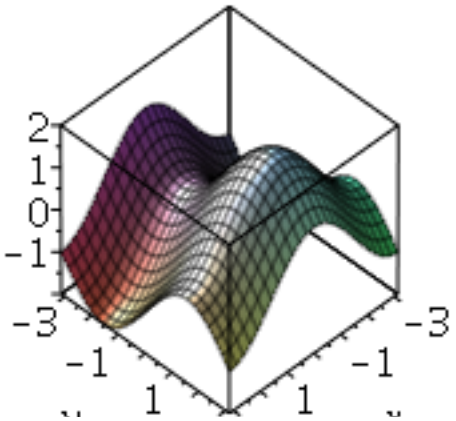



```
> display(%); #visualiza tódolos frames da animación anterior
```



No caso de 3D é análogo utilizando a función **animate3d(F, x, y, t)** pero especificando o rango de dúas variabeis **x** e **y**.

```
> animate3d(cos(t*x)+sin(t*y), x=-Pi..Pi, y=-Pi..Pi, t=1..2,
  frames=4);
```




```
> display(%) # visualiza os 4 frames da animación anterior.
```

Táboa resumo dos gráficos 2D e 3D

Finalmente, aquí tedes esta gráfica para resumir os tipos de gráficos:

Tipo de Gráfico	Ecuación	Función en Maple
Gráficos en 2D		
Curva en forma explícita (función real de variábel real)	*y = f(x)*	*plot(f(x),x=a..b)*
Curva en forma paramétrica	*x = x(t), y = y(t)*	*plot([x(t),y(t),t=a..b])*
Vector de puntos	$(x_1,y_1)...(x_n,y_n)$	*plot(Vector([x_1,...,x_n]), Vector ([y_1, ..., y_n]))*
Curva en coordenadas polares *(r,q)*	$\rho = \rho(\theta)$	*polarplot(r(q), θ=a..b)*
Curva en forma implícita	*f(x, y) = 0*	*implicitplot(f(x, y) = 0, x = a..b, y = a..b)*
Inecuacións	$f_1(x, y) > 0, ..., f_n(x, y) > 0$	*inequal({f_1(x, y) > 0, ..., f_n(x, y) > 0}, x = a..b, y = a. .b)*
Mapa de calor	*z=f(x,y)*	*contourplot(f(x,y),x=a..b,y= a..b,filledregions=true)*
Curva animada	*y = f(x, t)*	*animate(f(x, t), x = a..b, t = a..b, frames = n)*
Gráficos en 3D		

Superficie en forma explícita	*z = f(x, y)*	*plot3d(f(x, y), x = a..b, y = a..b)*
Superficie en forma paramétrica	*x = x(u, v), y = y(u, v), z = z(u, v)*	*plot3d([x(u, v), y(u, v), z(u, v)], u = a..b, v = a..b)*
Curva en forma paramétrica	*x = x(t), y = y(t), z = z(t)*	*spacecurve([x(t), y(t), z(t)], t = a..b, numpoints = n)*
Superficie en forma implícita	*f(x, y, z)=0*	*implicitplot3d(f(x, y, z) = 0, x = a..b, y = a..b, z = a..b)*
Superficie en coordenadas cilíndricas *(r , q, z*)	*f(r , q, z)=0*	*implicitplot3d(f(r ,q,z), r =a..b,q=a..b, z=a..b,coords= cylindrical)*
Superficie en coordenadas esféricas *(r , q, ϕ*)	*f(r , q, ϕ)=0*	*implicitplot3d(f(r ,q, ϕ), r = a..b, q=a..b, ϕ = a..b, coords = spherical)*
Superficie animada	*z = f(x, y, t)*	*animate3d(f(x, y, t), x = a..b, y = a..b, t = a..b, frames = n)*

Solución de ecuacións e sistemas de ecuacións

MAPLE pode resolver ecuacións e inecuacións (con unha ou varias incógnitas) de xeito **simbólico** (obte-la expresión analítica da solución; isto só é posíbel en ecuacións relativamente simples) ou **numérico** (obter valores numéricos aproximados das solucións).

Resolución simbólica

A solución simbólica obtense co comando

solve(equations, variables)

onde

equations - ecuación ou inecuación (pode ser un set ou lista de ecuacións ou inecuacións).

variables - (opcional) nome ou lista de nomes das incógnitas do sistema (se non o introducimos considerará a todas as existentes).

Se MAPLE non é capaz de atopar unha solución devolve NULL.

Se como primeiro argumento introducimos unha expresión, MAPLE interpretao como expresión=0.

```
> solve(x+y=0, x); # os argumentos son unha ecuación e
  resólvese para a incognita x
```

$$-y \qquad \textbf{(1.1.1)}$$

```
> solve(x+y=0, {x,y}); # os argumentos son unha ecuación e
  resólvese para as incognitas x e y (un set).
```

$$\{x = -y,\ y = y\} \qquad \textbf{(1.1.2)}$$

```
> solve(x+y=0); # o mesmo efecto que no caso anterior
```

$$\{x = -y,\ y = y\} \qquad \textbf{(1.1.3)}$$

```
> solve(x+y, x); # os argumentos son unha expresión e
  resólvese para a incognita x
```

$$-y \qquad \textbf{(1.1.4)}$$

```
> solve({x+y=0}, {x}); # os argumentos son sets(conxuntos) de
  ecuacións e incognitas
```

$$\{x = -y\} \qquad \textbf{(1.1.5)}$$

Polo tanto, a función **solve** pode empregarse para despexar unha variábel en función de outra(s):

```
> solve(2*y-(x-1)^2 = 2, y);
```

$$\frac{1}{2}x^2 - x + \frac{3}{2} \qquad \textbf{(1.1.6)}$$

```
> solve({x+(1/4)*y+z = 1,  3.2*x+1.3*y+4.2*z = 5,  8.7*x+19*
  y+11.2*z = 94}, [x, y, z]); #sistema de 3 ecuacións con 3
  incognitas: as ecuacións van entre chaves (é un conxunto de
  ecuacións)
```

(1.1.7)

$$[[x = 0.4969502408,\ y = 5.187800963,\ z = -0.7939004815]] \quad \textbf{(1.1.7)}$$

```
> solve({x^2 = 9,  x+y < 10}, [x, y]); #resolvendo inecuacións
```

$$[[x = 3,\ y < 7],\ [x = -3,\ y < 13]] \quad \textbf{(1.1.8)}$$

```
> solve({a*x^2+b*x+c},{x}); # toma a expresión igualada a 0
```

$$\left\{x = \frac{1}{2}\ \frac{-b+\sqrt{b^2-4\,a\,c}}{a}\right\},\ \left\{x = -\frac{1}{2}\ \frac{b+\sqrt{b^2-4\,a\,c}}{a}\right\} \quad \textbf{(1.1.9)}$$

```
> x;
```

$$x \quad \textbf{(1.1.10)}$$

NOTA IMPORTANTE: se nos da a solución en termos de **RootOf**, podemos executar **allvalues(%)** e ás veces si calcula unha solución explícita (non en termos de **RoofOf**). Tamén podemos usar **evalf(%)** para obter a(s) solución(s) en punto flotante.

```
> s := solve({sin(x+y)-y·exp(x), x^2-y-2}, {x, y});
Warning, solutions may have been lost
```

$$s := \{x = RootOf(e^{-Z}\_Z^2 - \sin(\_Z^2 + \_Z - 2) - 2\,e^{-Z}),\ y = RootOf(e^{-Z}\_Z^2 - \sin(\_Z^2 + \_Z - 2) - 2\,e^{-Z})^2 - 2\} \quad \textbf{(1.1.11)}$$

```
> evalf(s);
```

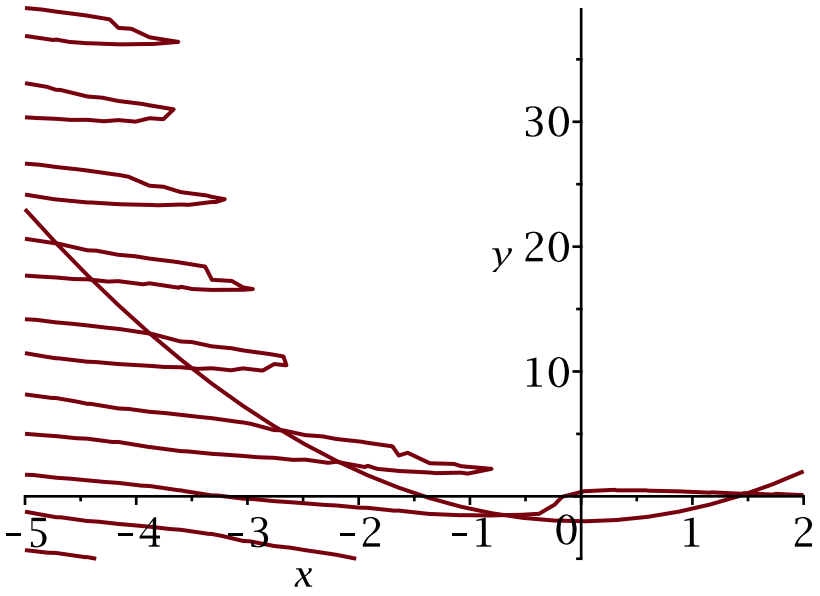
$$\{x = 1.490927732 + 0.\,I,\ y = 0.222865502 + 0.\,I\} \quad \textbf{(1.1.12)}$$

```
> evalf(allvalues(s));
```

$$\{x = -0.6687012050,\ y = -1.552838698\},\ \{x = -2.742322948,\ y = 5.520335151\},\ \{x = 1.490927732,\ y = 0.222865502\},\ \{x = 2.518820998 - 0.8147435813\,I,\ y = 3.680652117 - 4.104386481\,I\},\ \{x = 3.461326200 - 0.8260768945\,I,\ y = 9.29837603 - 5.718643196\,I\},\ \{x = 0.6078432441 - 1.104304339\,I,\ y = -2.850014664 - 1.342487864\,I\},\ \{x = 2.518820998 + 0.8147435813\,I,\ y = 3.680652117 + 4.104386481\,I\},\ \{x = 3.461326200 + 0.8260768945\,I,\ y = 9.29837603 + 5.718643196\,I\},\ \{x = 0.6078432441 + 1.104304339\,I,\ y = -2.850014664 + 1.342487864\,I\},\ \{x = -2.096487462,\ y = 2.395259678\},\ \{x = -0.9101370208 - 1.831561999\,I,\ y = -4.526269960 + 3.333944762\,I\},\ \{x = -0.9101370208 + 1.831561999\,I,\ y = -4.526269960 - 3.333944762\,I\} \quad \textbf{(1.1.13)}$$

Podemos comprobar as solucións representando gráficamente con **implicitplot** o sistema de ecuacións:

```
> with(plots) : implicitplot([sin(x+y)-y·exp(x), x^2-y-2], x=-5
    ..2, y=-5..40);
```



Maple, por defecto, non asigna as solucións ás variabeis. Podemos utiliza-lo comando **assign**.

```
> restart;
> res:=solve({cos(x)+y=9}, {x});
```

$$res := \{x = \pi - \arccos(y-9)\} \quad \textbf{(1.1.14)}$$

```
> x;
```

$$x \quad \textbf{(1.1.15)}$$

```
> assign(res): x;
```

$$\pi - \arccos(y-9) \quad \textbf{(1.1.16)}$$

Podes introducir hipóteses sobre as solucións ou sobre os parámetros (constantes) que aparecen na ecuación co comando **assuming**:

```
> solve(x^2-a, x) assuming a::negative;
```

$$I\sqrt{-a},\ -I\sqrt{-a} \quad \textbf{(1.1.17)}$$

```
> solve(x^2-a, x)
```

$$\sqrt{a},\ -\sqrt{a} \quad \textbf{(1.1.18)}$$

Pódese comproba-las solucións substituindo as solucións nas ecuacións orixinais co comando **eval** que neste contexto realiza a substitución

eval(e, x, = a) $e\Big|_{x=a}$

ou tamén co comando **subs**: **subs(x= a,expr)**

```
> restart;
> ecs:={x+2*y=3, y+1/x=1}; # Set de ecucións
```

$$ecs := \left\{x + 2y = 3,\ y + \frac{1}{x} = 1\right\} \quad \textbf{(1.1.19)}$$

```
> sols:=solve(ecs, {x, y});
```

(1.1.20)

$$sols := \{x = -1, y = 2\}, \left\{x = 2, y = \frac{1}{2}\right\} \quad \textbf{(1.1.20)}$$

```
> sols[1]; # unha solución
```

$$\{x = -1, y = 2\} \quad \textbf{(1.1.21)}$$

```
> sols[2]; # outra solución
```

$$\left\{x = 2, y = \frac{1}{2}\right\} \quad \textbf{(1.1.22)}$$

```
> eval(ecs, sols[1]);
```

$$\{1 = 1, 3 = 3\} \quad \textbf{(1.1.23)}$$

```
> eval(ecs, sols[2]);
```

$$\{1 = 1, 3 = 3\} \quad \textbf{(1.1.24)}$$

```
> subs(sols[1], ecs);
```

$$\{1 = 1, 3 = 3\} \quad \textbf{(1.1.25)}$$

Resolución numérica

Utilízase o comando **fsolve** que resolve únicamente ecuacións sen símbolos (só pode ter números):

fsolve(equations, complex)

onde

equations - ecuación, lista ou set de ecuacións

complex - (opcional) nome literal que hai que poñer cando se queren atopar solucións complexas.

A declaración do comando con tódalas opcións é:

fsolve(equations, variables, complex, fulldigits, interval, starting_values, options)

máis adiante mencionanse algunhas das opcións máis relevantes.

```
> restart;
> pol := 2*x^5-11*x^4-7*x^3+12*x^2-4*x = 0;
```

$$pol := 2x^5 - 11x^4 - 7x^3 + 12x^2 - 4x = 0 \quad \textbf{(1.2.1)}$$

```
> fsolve(pol);
```

$$-1.334383488,\ 0.,\ 5.929222024 \quad \textbf{(1.2.2)}$$

```
> pol2:=3*x^4-16*x^3-3*x^2+13*x+16;
```

$$pol2 := 3x^4 - 16x^3 - 3x^2 + 13x + 16 \quad \textbf{(1.2.3)}$$

```
> fsolve(pol2, x); # só mostra solucións reais se o polinomio
  ten coeficientes reais (con coeficientes complexos calcula
  tódalas solucións).
```

$$1.324717957,\ 5.333333333 \quad \textbf{(1.2.4)}$$

```
> fsolve(pol2, x, complex); # para mostra-las solucións
  complexas
```

$$-0.662358978622373 - 0.562279512062301\,I,\ -0.662358978622373 + 0.562279512062301\,I,\ 1.32471795724475,\ 5.33333333333333 \quad \textbf{(1.2.5)}$$

Podes obter resultados similares usando solve combinado con evalf:

```
> evalf(solve(pol2));
```

$$5.333333333,\ 1.324717958,\ -0.6623589786+0.5622795125\,I,\ -0.6623589786-0.5622795125\,I \qquad \textbf{(1.2.6)}$$

Para limita-lo número de solucións utilízase a opción **maxsols**.

```
> fsolve(pol2, {x}, maxsols=1); # mostra só unha solución
```

$$\{x=1.324717957\} \qquad \textbf{(1.2.7)}$$

```
> fsolve(sin(x)=0, {x}); # especifíacase o argumento variabeis
```

$$\{x=0.\} \qquad \textbf{(1.2.8)}$$

Se o programa só nos proporciona unha solución que non nos interesa, podemos forzalo a que proporcione outras solucións coa opción **avoid.**

```
> fsolve(sin(x)=0, {x}, avoid={x=0});
```

$$\{x=-3.141592654\} \qquad \textbf{(1.2.9)}$$

Tamén se pode especificar un intervalo de búsqueda.

```
> fsolve(pol2, {x}, -Pi..Pi);
```

$$\{x=1.324717957\} \qquad \textbf{(1.2.10)}$$

```
> f:=sin(x+y)-exp(x)*y=0;
```

$$f:=\sin(x+y)-e^{x}y=0 \qquad \textbf{(1.2.11)}$$

```
> g:=x^2-y=2;
```

$$g:=x^{2}-y=2 \qquad \textbf{(1.2.12)}$$

```
> fsolve({f,g}, {x,y},{x=-1..1, y=-2..0});
```

$$\{x=-0.6687012050,\ y=-1.552838698\} \qquad \textbf{(1.2.13)}$$

E, finalmente, tamén podemos definir un punto de comezo (opción starting_values):

```
> fsolve(sin(x), x=3.25);
```

$$3.141592654 \qquad \textbf{(1.2.14)}$$

Resolución de ecuacións recurrentes

As ecuacións recurrentes son da forma:

$$f(n)=F(\{f(n-k),\ k=1,\ \ldots,\ K\}),\ f(n_1)=F_1,\ldots,\ f(n_{1+K})=F_K$$

É dicir, coñecemos o termo *n* en función de *K* termos anteriores, e coñecemos *K* termos iniciais $F_1, \ldots, F_K$. Pódense resolver co comando **rsolve({expresión recurrente, valores iniciais}, incógnitas)**, onde a expresión e os valores inicialis son os anteriores, e a incógnita é o valor a obter. Por exemplo, os números de Fibonacci están definidos por $f(1)=f(2)=1,\ f(n)=f(n-1)+f(n-2),\ n>2$. Neste caso, podemos resolver esta ecuación recurrente (é dicir, obter *f* como función de *n*) usando:

```
> rsolve({f(n)=f(n-1)+f(n-2),f(1)=1,f(2)=1},f(n));
```

$$\frac{1}{5}\sqrt{5}\left(\frac{1}{2}\sqrt{5}+\frac{1}{2}\right)^{n}-\frac{1}{5}\sqrt{5}\left(-\frac{1}{2}\sqrt{5}+\frac{1}{2}\right)^{n} \qquad \textbf{(1.3.1)}$$

```
> evalf(%)
```

$$0.4472135954\,1.618033988^{n} - 0.4472135954\,(-0.6180339880)^{n} \qquad \textbf{(1.3.2)}$$

Solucións enteiras de ecuacións

A función **isolve(ecuacións, vars)** permite resolver ecuacións obtendo só solucións enteiras. Se as solución están parametrizadas (é dicir, se 3 é solución, $3k\ \forall k \in \mathbb{Z}$ tamén é solución), **vars** son os parámetros destas solucións. Por exemplo, dada a ecuación (ten 2 incógnitas, e polo tanto só podemos poñer unha incógnita como función da outra) $3x-5y=7$, podemos resolvela con:

```
> restart;isolve(3*x-5*y=7);
```

$$\{x = 4 + 5\,\_Z1,\ y = 1 + 3\,\_Z1\} \qquad \textbf{(1.4.1)}$$

```
> isolve(3*x-5*y=7,k);
```

$$\{x = 4 + 5\,k,\ y = 1 + 3\,k\} \qquad \textbf{(1.4.2)}$$

Vemos polo tanto que k é o valor que parametriza o conxunto de solucións. Dándolle un valor a k obtemos os valores de x e y:

```
> subs(k=1, %);
```

$$\{x = 9,\ y = 4\} \qquad \textbf{(1.4.3)}$$

Manipulación de polinomios e funcións racionais

Polinomios dunha variabel

Chámase forma canónica dun polinomio a:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

onde n é o grao do polinomio, a_n o primeiro coeficiente e a_0 o último. A definición dun polinomio farase mediante o operador de asignación

```
> p1:=-3*x + 7*x^2 - 3*x^3 + 7*x^4;
```

$$p1 := 7 x^4 - 3 x^3 + 7 x^2 - 3 x \qquad \textbf{(1.1.1)}$$

Para avaliar $p_1(x_0=1)$:

```
> eval(p1,x=1);
```

$$8 \qquad \textbf{(1.1.2)}$$

```
> type(p1, 'polynom'); # devolve true se p1 é un polinomio
```

$$true \qquad \textbf{(1.1.3)}$$

```
> lcoeff(p1); # devolve o coeficiente do término de maior grado
```

$$7 \qquad \textbf{(1.1.4)}$$

```
> degree(p1); #devolve o grao do polinomio
```

$$4 \qquad \textbf{(1.1.5)}$$

```
> p2:=5*x^5 + 3*x^3 + x^2 -2*x + 1; # definición do polinomio p2
```

$$p2 := 5 x^5 + 3 x^3 + x^2 - 2 x + 1 \qquad \textbf{(1.1.6)}$$

```
> 2*p1 + 4*p2 +3;
```

$$20 x^5 + 14 x^4 + 6 x^3 + 18 x^2 - 14 x + 7 \qquad \textbf{(1.1.7)}$$

```
> p1*p2;
```

$$(7 x^4 - 3 x^3 + 7 x^2 - 3 x)\,(5 x^5 + 3 x^3 + x^2 - 2 x + 1) \qquad \textbf{(1.1.8)}$$

```
> expand(%); # calcula a expresión anterior
```

$$35 x^9 - 15 x^8 + 56 x^7 - 17 x^6 + 4 x^5 + 11 x^4 - 20 x^3 + 13 x^2 - 3 x \qquad \textbf{(1.1.9)}$$

```
> sort(%); # para obter o polinomio en forma canónica
```

$$35 x^9 - 15 x^8 + 56 x^7 - 17 x^6 + 4 x^5 + 11 x^4 - 20 x^3 + 13 x^2 - 3 x \qquad \textbf{(1.1.10)}$$

```
> coeff(p1,x^3); #devolve o coeficiente de x^3 do polinomio p1
```

$$-3 \qquad \textbf{(1.1.11)}$$

Algúns comandos para a manipulación de polinomios que xa vimos son: **coeff( )** e **degree( )** (que só poden operar sobre polinomios en forma agrupada).
En MAPLE pódese calcula-lo cociente e o resto dunha división de polinomios coas funcións **quo** e **rem**:

rem(a, b, x) e **rem(a, b, x, 'q')** devolve o resto de dividir a entre b
quo(a, b, x) e **quo(a, b, x, 'r')** devolve o cociente de dividir a entre b

a e **b** son polinomios na variabel **x** e **'q'** e **'r'** son opcionais e representan os nomes das variabeis as que se asignará o resultado das respectivas operacións (resto para o comando quo e cociente para o comando rem).
O resto **r** e o cociente **q** cumpren: **a = b*q + r** onde **degree(r, x) < degree(b, x)**

```
> q1:=quo(p2, p1, x, 'r1'); # devolve o cociente e en r1
  almacena o resto
```

$$q1 := \frac{5}{7}\,x + \frac{15}{49} \qquad \textbf{(1.1.12)}$$

```
> r1;
```

$$-\frac{53}{49}\,x^3 + x^2 - \frac{53}{49}\,x + 1 \qquad \textbf{(1.1.13)}$$

```
> testeq(p2=expand(q1*p1+r1)); #comprobación do resultado
  anterior
```

true **(1.1.14)**

```
> r2:=rem(p2, p1, x, 'q2'); # devolve o resto e almacena o
  cociente en q2
```

$$r2 := -\frac{53}{49}\,x^3 + x^2 - \frac{53}{49}\,x + 1 \qquad \textbf{(1.1.15)}$$

```
> q2;
```

$$\frac{5}{7}\,x + \frac{15}{49} \qquad \textbf{(1.1.16)}$$

```
> testeq(p2=expand(q2*p1+r2));
```

true **(1.1.17)**

A función **divide** devolve verdadeiro (*true*) cando a división entre dous polinomios é exacta (resto cero), i.e., *p1* divide a *p2*, e senón devolve falso (*false*).

```
> divide(p2, p1);
```

false **(1.1.18)**

Para calcula-lo máximo común divisor de dous polinomios utilízase a función **gcd** e para o mínimo común múltiplo a función **lcm**

gcd(a, b) **lcm(a, b,...)**

onde **a** e **b** son os polinomios.

```
> gcd(p2, p1);
```

$$x^2 + 1 \qquad \textbf{(1.1.19)}$$

```
> lcm(p2,p1);
```

$$(5\,x^3 - 2\,x + 1)\,(7\,x^4 - 3\,x^3 + 7\,x^2 - 3\,x) \qquad \textbf{(1.1.20)}$$

```
> expand(%); # calcula a expresión anterior
```

$$35\,x^7 - 15\,x^6 + 21\,x^5 - 2\,x^4 - 17\,x^3 + 13\,x^2 - 3\,x \qquad \textbf{(1.1.21)}$$

```
> lcm(6, -8, 3, 4, 12); # lcm pódese aplicar a máis de dous
  factores
```

24 **(1.1.22)**

Tamén se pode calcular as raíces dun polinomio coa función **roots**. Só da as **raíces exactas** (enteiras, racionais, certas raíces irracionais, e raíces complexas

con partes real/imaxinaria enteira, racional ou certas irracionais).

roots(a, x, K)

e devolve unha lista de pares da forma **[[r1, m1], ..., [rn, mn]]** onde **ri** e a raíz do polinomio **a** con multiplicidade **mi**, é dicir, **(x - ri)^mi** divide ao polinomio **a**.O argumento **K** indica o conxunto no que se buscan as raíces (por defecto, Z ou <; se se especifica **{sqrt(2)}**, búscanse nos irracionais que se poden poñer en termos de $\sqrt{2}$ (ídem se se pon **sqrt(3)**, etc); se se especifica **I**, busca nos números complexos con partes real e imaxinaria enteiras, racionais ou sqrt(2), etc.

```
> p3:=expand(p1*p2);
```

$$p3 := 35\,x^9 - 15\,x^8 + 56\,x^7 - 17\,x^6 + 4\,x^5 + 11\,x^4 - 20\,x^3 + 13\,x^2 - 3\,x \quad \textbf{(1.1.23)}$$

```
> roots(p3);
```

$$\left[[0, 1], \left[\frac{3}{7}, 1\right]\right] \quad \textbf{(1.1.24)}$$

Devolve só dúas raíces cando o polinomio é de grao 9. Esto débese a que só devolve as raíces racionais. Se queremos outras raíces hai que especifica-lo campo (distinto do campo dos racionais)

```
> roots(x^4-4, x); #ningunha raíz exacta racional
```

$$[\,] \quad \textbf{(1.1.25)}$$

```
> roots(x^4-4, sqrt(2)); # raices reais mútiplos de sqrt(2)
```

$$[[-\sqrt{2}, 1], [\sqrt{2}, 1]] \quad \textbf{(1.1.26)}$$

```
> roots(x^4-4, {sqrt(2), I}); # raices reais e mútiplos de
  sqrt(2)
```

$$[[-I\sqrt{2}, 1], [I\sqrt{2}, 1], [-\sqrt{2}, 1], [\sqrt{2}, 1]] \quad \textbf{(1.1.27)}$$

Pódese factorizar un polinomio (escribir un polinomio como producto de factores irreducibeis con coeficientes racionais) coa función **factor**:

factor(a) **factor(a, K)**

a é o polinomio e K un argumento opcional igual que en roots.

```
> factor(p3);
```

$$x\,(7\,x-3)\,(5\,x^3-2\,x+1)\,(x^2+1)^2 \quad \textbf{(1.1.28)}$$

Polinomios de varias variabeis

MAPLE permite definir polinomios de varias variabeis e proporciona algunhas funcións para a súa manipulación:

```
> restart;
> poli:=6*x*y^5+12*y^4+14*x^3*y^3-15*x^2*y^3+9*x^3*y^2-30*x*
  y^2-35*x^4*y+18*y*x^2+21*x^5;
```

$$poli := 14\,x^3\,y^3 + 6\,x\,y^5 + 21\,x^5 - 35\,x^4\,y + 9\,x^3\,y^2 - 15\,x^2\,y^3 + 12\,y^4 + 18\,x^2\,y - 30\,x\,y^2 \quad \textbf{(1.2.1)}$$

```
> sort(poli); #por defecto ordea por graos totais (suma de
  potencia de x + potencia de y) decrecentes
```

(1.2.2)

$$14x^3y^3+6xy^5+21x^5-35x^4y+9x^3y^2-15x^2y^3+12y^4+18x^2y-30xy^2 \quad (1.2.2)$$

```
> sort(poli, [x,y], plex,descending); # ordea os termos de
  forma alfabética (en inglés). Ve-la axuda do comando sort
  para ver tódalas posibilidades: plex=purely lexicographic
```

$$21x^5-35x^4y+14x^3y^3+9x^3y^2-15x^2y^3+18x^2y+6xy^5-30xy^2+12y^4 \quad (1.2.3)$$

```
> sort(poli, [x, y], plex, ascending);
    #` `ordea os monomios por potencias crecentes de x (con
    empates, por potencias crecentes de y)
```

$$12y^4-30xy^2+6xy^5+18x^2y-15x^2y^3+9x^3y^2+14x^3y^3-35x^4y+21x^5 \quad (1.2.4)$$

```
> sort(poli, [x, y], tdeg, descending);
    # tdeg=total degree: ordea os monomios por sumas de
    potencias de x e y decrecentes
```

$$14x^3y^3+6xy^5+21x^5-35x^4y+9x^3y^2-15x^2y^3+12y^4+18x^2y-30xy^2 \quad (1.2.5)$$

```
> sort(poli, [x, y], tdeg, ascending);
    # ordea os monomios por sumas de potencias de x e y
    crecentes
```

$$-30xy^2+18x^2y+12y^4-15x^2y^3+9x^3y^2-35x^4y+21x^5+6xy^5+14x^3y^3 \quad (1.2.6)$$

```
> sort(poli, [y, x], plex, descending);
```

$$6y^5x+12y^4+14y^3x^3-15y^3x^2+9y^2x^3-30y^2x-35yx^4+18yx^2+21x^5 \quad (1.2.7)$$

```
> sort(poli, [y, x], plex, ascending);
```

$$21x^5+18yx^2-35yx^4-30y^2x+9y^2x^3-15y^3x^2+14y^3x^3+12y^4+6y^5x \quad (1.2.8)$$

```
> sort(poli, [y, x], tdeg, descending);
```

$$6y^5x+14y^3x^3-15y^3x^2+9y^2x^3-35yx^4+21x^5+12y^4-30y^2x+18yx^2 \quad (1.2.9)$$

```
> sort(poli, [y, x], tdeg, ascending);
```

$$18yx^2-30y^2x+12y^4+21x^5-35yx^4+9y^2x^3-15y^3x^2+14y^3x^3+6y^5x \quad (1.2.10)$$

```
> collect(poli, x); # ordea o polinomio segundo as potencias
  de x
```

$$21x^5-35x^4y+(14y^3+9y^2)x^3+(18y-15y^3)x^2+(-30y^2+6y^5)x+12y^4 \quad (1.2.11)$$

```
> collect(poli, y);  # ídem de y
```

$$6\,x\,y^5+12\,y^4+(-15\,x^2+14\,x^3)\,y^3+(9\,x^3-30\,x)\,y^2+(-35\,x^4+18\,x^2)\,y+21\,x^5 \quad \textbf{(1.2.12)}$$

```
> coeff(poli, x^3); # coeficientes de x^3
```

$$14\,y^3+9\,y^2 \quad \textbf{(1.2.13)}$$

Funcións racionais

As funcións racionais exprésanse como cocientes de dous polinomios, sendo o denominador distinto de cero.

```
> restart;
> f:=x^2+3*x+2; g:=x^2+5*x+6; h:=f/g;
```

$$f := x^2+3\,x+2$$

$$g := x^2+5\,x+6$$

$$h := \frac{x^2+3\,x+2}{x^2+5\,x+6} \quad \textbf{(1.3.1)}$$

```
> numer(%); # comando que devolve o numerador da expresión
  anterior
```

$$x^2+3\,x+2 \quad \textbf{(1.3.2)}$$

```
> denom(h); #comando de devolve o denominador da función
  racional h
```

$$x^2+5\,x+6 \quad \textbf{(1.3.3)}$$

Por defecto, MAPLE non simplifica as funcións racionais. As simplificacións só se realizan cando o programa recoñece factores comúns.

```
> ff:=(x-1)*f;
```

$$ff := (x-1)\,(x^2+3\,x+2) \quad \textbf{(1.3.4)}$$

```
> gg:=(x-1)^2*g;
```

$$gg := (x-1)^2\,(x^2+5\,x+6) \quad \textbf{(1.3.5)}$$

```
> ff/gg;
```

$$\frac{x^2+3\,x+2}{(x-1)\,(x^2+5\,x+6)} \quad \textbf{(1.3.6)}$$

Para forzar unha simplificación utilízase a función **normal**:

```
> normal(f/g);
```

$$\frac{x+1}{x+3} \quad \textbf{(1.3.7)}$$

```
> normal(ff/gg);
```

$$\frac{x+1}{(x+3)\,(x-1)} \quad \textbf{(1.3.8)}$$

Tamén se poden definir funcións racionais en varias variabeis.

```
> restart;
> f:=161*y^3+ 333*x*y^2+184*y^2+162*x^2*y+144*x*y+77*y+99*
```

```
x+88;
```

$$f := 161\,y^3 + 333\,x\,y^2 + 184\,y^2 + 162\,x^2\,y + 144\,x\,y + 77\,y + 99\,x + 88 \quad \textbf{(1.3.9)}$$

```
> g:=49*y^2+28*x^2*y+63*x*y+147*y+36*x^3+32*x^2+177*x+104;
```

$$g := 49\,y^2 + 28\,x^2\,y + 63\,x\,y + 147\,y + 36\,x^3 + 32\,x^2 + 177\,x + 104 \quad \textbf{(1.3.10)}$$

```
> f/g;
```

$$\frac{161\,y^3 + 333\,x\,y^2 + 184\,y^2 + 162\,x^2\,y + 144\,x\,y + 77\,y + 99\,x + 88}{49\,y^2 + 28\,x^2\,y + 63\,x\,y + 147\,y + 36\,x^3 + 32\,x^2 + 177\,x + 104} \quad \textbf{(1.3.11)}$$

```
> normal(%);
```

$$\frac{161\,y^3 + 333\,x\,y^2 + 184\,y^2 + 162\,x^2\,y + 144\,x\,y + 77\,y + 99\,x + 88}{49\,y^2 + 28\,x^2\,y + 63\,x\,y + 147\,y + 36\,x^3 + 32\,x^2 + 177\,x + 104} \quad \textbf{(1.3.12)}$$

Pódese realiza-la **descomposición en fraccións parciais** utilizando o comando **convert** e especificando a opción **'parfrac'**

```
> restart;
> f:=(x^5+1)/(x^4-x^2);
```

$$f := \frac{x^5 + 1}{x^4 - x^2} \quad \textbf{(1.3.13)}$$

```
> convert(f, 'parfrac', x);
```

$$x - \frac{1}{x^2} + \frac{1}{x - 1} \quad \textbf{(1.3.14)}$$

Operacións con expresións

Maple dispón de moitas ferramentas para manipular expresións matemáticas. Algunhas das funcións máis utilizadas son:

Función expand:
intenta simplificar unha expresión en forma de sumas de productos de funcións máis sinxelas.

expand(expr, expr1, expr2, ..., exprn)

onde **expr** é a expresión matemática que se quere simplificar e **expr1, expr2, ..., exprn** son argumentos opcionais que indican as subexpresións que non queremos que se simplifiquen. Algúns exemplos:

```
> expand((x+1)*(x+2));
```

$$x^2 + 3\,x + 2 \tag{1.1}$$

```
> expand(((x+1)*(x+3)*x)/(x+2)); # só simplifica o numerador da
  expresión
```

$$\frac{x^3}{x+2} + \frac{4\,x^2}{x+2} + \frac{3\,x}{x+2} \tag{1.2}$$

```
> expand(sin(x+y));
```

$$\sin(x)\cos(y) + \cos(x)\sin(y) \tag{1.3}$$

```
> expand(cos(2*x));
```

$$2\cos(x)^2 - 1 \tag{1.4}$$

```
> expand(exp(a+ln(b)));
```

$$e^a\,b \tag{1.5}$$

```
> expand((x+1)*(y+z));
```

$$x\,y + x\,z + y + z \tag{1.6}$$

```
> expand(ln(x/(1-x)^2)); # só se expanden as expresións cando é
  posíbel
```

$$\ln\left(\frac{x}{(1-x)^2}\right) \tag{1.7}$$

```
> expand((x+1)*(y+z), x+1); # o segundo argumento (x+1)
  especifica a subexpresión que desexamos que non se simplifique.
```

$$(x+1)\,y + (x+1)\,z \tag{1.8}$$

Función combine: realiza a tarefa inversa a **expand**. Combina varias expresións para conseguir unha expresión máis compacta ou reducida.

combine(f) ou **combine(f, n)**

onde **f** é unha expresión, lista ou set de expresións a combinar, **n** é o nome, lista ou set de nomes que indican o tipo de elementos que contén a expresión para que MAPLE sepa que tipo de regras ten que utilizar. Valores usuais son: **trig, exp, ln, power, etc**. As regras de combinación en cada caso son:

Para **trig**

sin(a)*sin(b) ==> 1/2*cos(a-b) - 1/2*cos(a+b)

sin(a)*cos(b) ==> 1/2*sin(a-b) + 1/2*sin(a+b)
cos(a)*cos(b) ==> 1/2*cos(a-b) + 1/2*cos(a+b)

Para **exp**

exp(x)*exp(y) ==> exp(x+y)
exp(x)^y ==> exp(x*y)
exp(x+n*ln(y)) ==> y^n*exp(x) where n is an integer

Para **ln**

a*ln(x) ==> ln(x^a) (provided a*argument(x)=argument(x^a))
ln(x)+ln(y) ==> ln(x*y) (provided argument(x*y) = argument(x) + argument(y))

```
> combine(sin(x)*cos(y)+cos(x)*sin(y), trig); #especificamos que use as regras trigonométricas
```

$$\sin(x+y) \tag{1.9}$$

```
> exp(x+3*ln(y));
```

$$e^{x+3\ln(y)} \tag{1.10}$$

```
> combine(%, exp);
```

$$y^3 e^x \tag{1.11}$$

```
> exp(sin(a)*cos(b))*exp(cos(a)*sin(b));
```

$$e^{\sin(a)\cos(b)} e^{\cos(a)\sin(b)} \tag{1.12}$$

```
> combine(%, [trig, exp]); #especificamos que utilice un lista de regras de combinación
```

$$e^{\sin(a+b)} \tag{1.13}$$

Función simplify: é o comando xeral de simplificación en MAPLE. Ó igual que **combine** podemos especifica-lo tipo de simplificación que desexamos (trig, exp, ln,...). Se non se especifica nada MAPLE intenta aplica-lo maior número de regras de simplificación posíbel. Nalgúns casos devolve o mesmo resultado que **expand.**

```
> expresion:=sin(x)^2+ln(2*x)+cos(x)^2;
```

$$expresion := \sin(x)^2 + \ln(2x) + \cos(x)^2 \tag{1.14}$$

```
> simplify(expresion);
```

$$1 + \ln(2) + \ln(x) \tag{1.15}$$

```
> simplify(expresion, trig); # aqui so se aplicas as regras de simplificación trigonométricas. Obsérvese a diferenza co exemplo anterior.
```

$$1 + \ln(2x) \tag{1.16}$$

Función normal: resulta útil para aplicar a expresións alxebraicas que conteñan **fraccións**. A función normal convirte a fracción na forma normal factorizada, é dicir, fracción da forma numerador/denominador onde numerador e denominador son polinomios primos (indivisibeis) con coeficientes enteiros.

```
> restart;
> expresion:=(x^2-y^2)/(x-y)^3;
```

$$expresion := \frac{x^2 - y^2}{(x-y)^3} \tag{1.17}$$

```
> normal(expresion);
```

$$\frac{x+y}{(x-y)^2} \quad (1.18)$$

```
> normal((x^2-1)/(x-1));
```

$$x+1 \quad (1.19)$$

```
> normal(sin(x*(x+1)-x));
```

$$\sin(x^2) \quad (1.20)$$

Se desexamos que normal expanda no seu resultado tanto o numerador como denominador hai que proporcionarlle como segundo argumento *expanded*:

```
> normal(expresion, expanded);
```

$$\frac{x+y}{x^2-2xy+y^2} \quad (1.21)$$

Función factor: permite descompoñer un polinomio en factores (para descompoñer un número utilízase o comando **ifactor**).

```
> factor(6*x^2+18*x-24);
```

$$6(x+4)(x-1) \quad (1.22)$$

```
> factor(6);
```

$$6 \quad (1.23)$$

```
> ifactor(6);
```

$$(2)(3) \quad (1.24)$$

```
> expr1:=1/(x^2-1)+1/(x^2+3*x+2);
```

$$expr1 := \frac{1}{x^2-1} + \frac{1}{x^2+3x+2} \quad (1.25)$$

```
> factor(expr1);
```

$$\frac{2x+1}{(x+2)(x+1)(x-1)} \quad (1.26)$$

```
> ifactor(902/24);
```

$$\frac{(11)(41)}{(2)^2(3)} \quad (1.27)$$

Función convert: descompón unha fracción alxebraica en fraccións simples

convert(expr, form, arg3, ...)

onde **expr** é a fracción a descompoñer, **form** indica o tipo de descomposición, **arg3** indica a variabel respecto da cal se realiza a descomposición (opcional se non hai máis dunha variabel). A lista de valores que pode toma-lo argumento **form** pódese consultar na axuda.

```
> restart;
> f := (x^3+x)/(x^2-1);
```

$$f := \frac{x^3+x}{x^2-1} \quad (1.28)$$

```
> convert(f, parfrac, x);
```

$$x + \frac{1}{x-1} + \frac{1}{x+1} \quad (1.29)$$

Titoriais e Asistentes de Maple

Pódense acceder dende os menús Tools → Assistants, Tools→Tutors e Tools→Tasks →Browse

Standard Functions

Tools->Precalculus

Curve-fitting

Permite definir un conxunto de puntos e interpolar entre eles, é dicir, calcular funcións que se axustan (fit) a estos puntos
. Menú Tools -> Assistants-> Curve Fitting

Plot builder

Menú Tools -> Assistants-> Plot Builder

Antiderivatives

Permite calcular e representar integrais indefinidas de funcións. Menú Tools -> Tutors->Calculus Single-Variable->Antiderivatives

Approximate Integration

Menú Tools -> Tutors->Calculus Single-Variable -> Approximate Integration

Derivatives

Menú Tools -> Tutors->Calculus Single-Variable -> Derivatives

Limit methods

Menú Tools->Tutors->Calculus Single-Variable->Limit methods

Volume / surface of revolution

Menú Tools→Tutors→Calculus Single − Variable

Método de Newton

Permite usar o método de Newton para resolver ecuacións non lineares dunha variábel. Menú Tools→Tutors→Calculus Single − Variable.

Linear systems plot

Menú Tutors -> Linear Algebra

Inverse Matrix

Menú Tutors->Linear Algebra

Linear system solving

Menú Tutors->Linear Algebra

Eigenvectors plot

Menú Tutors->Linear Algebra

Eigenvalues

Menú Tutors->Linear Algebra

Function composition

Menú Tutorts -> Precalculus

Limits

Menú Tutors->Precalculus

Polinomials and roots

Menú Tutors->Precalculus

Rational functions

Menú Tutors->Precalculus

Browse Tasks

Menú Tools->Tasks->Browse Tasks: Outras utilidades

Boletín de MAPLE

Semana 1

Traballo a desenvolver pol@ alumn@

1. **Cálculo con números**

   a) Determina se son primos os seguintes números: 7, 41, 143, 239. Calcular o seguinte enteiro máis próximo que sexa primo.

   b) Calcula os factores primos dos seguintes números enteiros: 160, 2000, 4562, 54879.

   c) Dados os seguintes números racionais, calcula a división enteira, o resto, o número racional simplificado e o resultado en punto flotante con 4 díxitos decimais:
   $\frac{20}{8}, \frac{748}{36}, \frac{1245}{40}, \frac{84}{738}, \frac{48}{72}, \frac{546}{67}, \frac{5436}{840}, \frac{345}{125}$.

   d) Asigna o valor 20 a variabel x e o valor -3.4 a variabel y.

   e) Visualiza as constantes que tes almacenadas no entorno de traballo de MAPLE.

   f) Avalía as seguintes operacións utilizando aritmética en punto flotante con 20 díxitos decimais:
   $\sqrt{3\pi/2}$, $\exp(7\pi/3)$, $8\pi^{50}$, $\sqrt{40\pi^5 + 6\pi^{457}e^{7\pi^{565}}}$.

2. **Vectores e matrices**

   a) Define os seguintes vectores:

   1) Un vector columna de 5 elementos inicializado a 0.
   2) Un vector fila de 4 elementos inicializado a 4.
   3) Un vector columna inicializado cos elementos: $5x$, $\sqrt{yx^2}$, 0, 3.
   4) Un vector fila de 6 elementos inicializado cos valores x^n, $n = 1 \ldots 6$.
   5) O vector $\mathbf{x} = (1, 4, 2, 0, 3, 3)$.

   b) Define as seguintes matrices:

   1) Unha matriz cadrada **M1**de 2×2 inicializada con 0.
   2) Unha matriz **M2** de 2 filas e 3 columnas inicializada co número 3.
   3) Unha matriz **M3** identidade de tamaño 4.
   4) Unha matriz **M4** de 2 filas e 3 columnas inicializada co vector do apartado 4 do exercicio anterior.
   5) Unha matriz **M5** de 2 filas e 6 columnas inicializada cos vectores dos apartados 4 e 5 do exercicio anterior.
   6) Unha matriz **M6** de 3×3 inicializada co vector do apartado 4 do exercicio anterior. Os elementos restantes inicialízanse a 0.
   7) Unha matriz **M7** de 3×3 inicializada co vector do apartado 4 do exercicio anterior. Os elementos restantes inicialízanse a 6.
   8) Unha matriz **M8** que sexa **M2+M4**
   9) Unha matriz **M8** que sexa **M2** $\cdot$ **M3**.
   10) Unha matriz **M9** de tamaño 5×5 onde os elementos da matriz se calculen mediante a función $f(i, j) = \exp(3i - j)$, onde i representa filas e j representa columnas.

   c) Dadas as matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

   calcula as matrices: A+B, A+C, AB, BA, BC, AC+BA, 3A, 5B, 8C, 3A-5B, A^2, B^3.

d) Calcula o determinante e a inversa das seguintes matrices:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 1 & 2 \\ 1 & 1 & 2 \end{bmatrix}$$

e) Dadas a seguintes matrices:

$$\mathbf{MI} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

calcular:

1) A matriz inversa de $MI - A$
2) $(MI - A)^4$
3) A matriz inversa de $MI + A$
4) $\dfrac{MI + A}{MI - A}$

f) Calcula o determinante e matriz inversa das seguintes matrices:

$$\begin{bmatrix} 3 & 1 \\ -2 & 4 \end{bmatrix} \qquad \begin{bmatrix} 2 & 5 \\ 3 & 7 \end{bmatrix} \qquad \begin{bmatrix} 2 & -4 \\ -3 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 5 & 2 \\ 1 & -1 & -1 \\ 2 & 3 & 4 \end{bmatrix} \qquad \begin{bmatrix} 2 & 5 & -2 \\ 3 & -1 & 4 \\ 5 & 4 & 2 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4 & 1 & 0 \\ 1 & -2 & 5 & 2 \\ 2 & -1 & 1 & -1 \\ 3 & 3 & 4 & 0 \end{bmatrix}$$

g) Resolve o seguinte sistema de ecuacións:

$$\begin{aligned} x_1 + x_2 - x_3 &= 1 \\ x_1 + 2x_2 + x_3 &= 2 \\ 2x_1 + x_2 - x_3 &= 3 \\ -3x_1 + x_2 + 2x_3 &= -5 \end{aligned}$$

h) Resolve o seguinte sistema de ecuacións:

$$\begin{aligned} 2x + 3y + z &= 4 \\ x - 2y + z &= -2 \\ 8x + 5y + 5z &= 1 \end{aligned}$$

i) Resolve o seguinte sistema de ecuacións:

$$\begin{aligned} 2x + 3y + z - t &= 1 \\ x - 3y - 2z + t &= 6 \\ 3x + 2y - 3z + 2t &= 9 \\ x - y - z + 2t &= 10 \end{aligned}$$

Semana 3

Traballo a desenvolver pol@ alumn@

1. **Funcións e límites**

a) Define a seguinte función por intervalos e represéntaa gráficamente:

$$f(x) = \begin{cases} 1+x & x \leq 0 \\ x & 0 < x < 1 \\ 2-x & 1 \leq x \leq 2 \\ 3x - x^2 & x > 2 \end{cases}$$

b) Define a seguinte función definida por intervalos e representala gráficamente:

$$f(x) = \begin{cases} -x & x < 0 \\ x & 0 \leq x < \dfrac{1}{2} \\ 1-x & \dfrac{1}{2} \leq x < 1 \\ x-1 & resto \end{cases}$$

c) Dadas as funcións $f(x,y) = x^2 + y^2$, $h(x,y) = ye^x$ e $k(x,y) = e^x \operatorname{sen} y$, realiza as seguintes operacións con funcións: $f - h$, $f + h$, $f \times h$, f/h e $f \circ (h,k)$, onde $\circ$ designa ao operación de composición, e (h,k) indica o par ordeado resultante de aplica-las funcións $h(x,y)$ e $k(x,y)$, e calcula o valor das mesmas no punto $(1,1)$.

d) Representa gráficamente as funcións e calcula os limites seguintes:

1) $\lim\limits_{x\to 0} \dfrac{\operatorname{sen} x}{x}$
2) $\lim\limits_{x\to 0} (\operatorname{sen} x)^{1/x}$
3) $\lim\limits_{x\to 0} \dfrac{1-\cos x}{x}$
4) $\lim\limits_{x\to \infty} \left(1 + \dfrac{\pi}{x}\right)^x$
5) $\lim\limits_{x\to 0} x^{\operatorname{sen} x}$
6) $\lim\limits_{x\to \infty} (2^x + 3^x)^{1/x}$
7) $\lim\limits_{x\to \infty} \dfrac{\ln x}{x}$
8) $\lim\limits_{x\to \infty} \dfrac{\ln x}{e^x}$
9) $\lim\limits_{x\to \infty} \dfrac{x^2 + \operatorname{sen} x}{2x^2 + \cos(4x)}$
10) $\lim\limits_{x\to \infty} \operatorname{senh}(\tanh x) - \tanh(\operatorname{senh} x)$
11) $\lim\limits_{x\to 2} \dfrac{x^2 + x - 6}{x^2 - 4}$
12) $\lim\limits_{x\to 0} e^{|x|/x}$
13) $\lim\limits_{x\to \infty} \dfrac{2x - sen x}{cos 2x}$
14) $\lim\limits_{x\to 0} \dfrac{x \arctan(x/2)}{cos x (sen 2x)^2}$
15) $\lim\limits_{x\to \infty} x log\left(\dfrac{x+1}{x-1}\right)$
16) $\lim\limits_{x\to 1} \dfrac{log x}{x - \sqrt{x}}$
17) $\lim\limits_{x\to 1} \dfrac{x-1}{ArgSh(x-1)}$
18) $\lim\limits_{x\to 1} \dfrac{x^{1/3} - 1}{x - 1}$

19) $\lim_{x\to 0}\left(\dfrac{1}{\operatorname{sen}^2 x}-\dfrac{1}{x^2}\right)$

2. **Derivadas**

a) Calcula tódalas derivadas parciais de orden 2 da función $f(x,y)=\dfrac{\ln(1+x^4+y^4)}{\sqrt{x^2+y^2}}$

b) Calcular a seguinte derivada $\dfrac{\partial^2}{\partial x\partial y}(\operatorname{sen}(xy)+\cos y)$

c) Calcula e representa na mesma gráfica as seguintes funcións e as súas derivadas de orden 1 e 2:

1) x^2
2) $8x^3$
3) x^2-2x+1
4) x^3-x^2-8x+1
5) $\dfrac{1}{x^5+x+1}$
6) $x^3-12x^2+45x+30$
7) $\dfrac{x^3}{(1+x)^2}$
8) $\dfrac{x^2}{2-x}$
9) $\sqrt{\dfrac{x}{2}+\dfrac{2}{x}}$
10) $\dfrac{x^2-x+1}{x^2+x+1}$

d) Calcula as cinco primeiras derivadas das funcións:

1) $f(x)=8x^7-3x^6+5x^5-x^4-x^3+x^2+1$
2) $g(x)=\operatorname{sen} x$
3) $h(x)=\ln x$

3. **Integrais**

a) Calcula e representa na mesma gráfica as seguintes funcións coas súas integrais indefinidas:

1) $\dfrac{1}{\cos^2 x}$
2) $\dfrac{1}{\sqrt{1-x^2}}$
3) $\arctan x$
4) $\dfrac{\ln x}{\sqrt{x}}$
5) $x\cos(3x)$
6) $\sqrt{1-x^2}$
7) $\dfrac{\cos x}{\operatorname{sen}^3 x+2\cos^2 x\operatorname{sen} x}$
8) $\dfrac{2^x}{1+4^x}$
9) $\dfrac{x-5}{(x-1)(x+1)^2}$
10) $4e^{x-1}$
11) x^2e^{2x}
12) $x^2\operatorname{sen} 2x$
13) $x^3(\ln x)^2$

14) $(2-x)\cos 2x$

15) $\dfrac{1}{9-4x^2}$

16) $\operatorname{sen}^3 x$

17) $\dfrac{x^2}{x+1}$

18) $\dfrac{x^3-8x^2-4x+8}{x^4-5x^2+4}$

19) $\dfrac{x^3-3x}{1-x^2}$

20) $\dfrac{1}{x(x-1)(x-2)}$

21) $\dfrac{x^3-x^2+1}{(x-1)^4}$

22) $\dfrac{\ln x}{x^2}$

b) Calcula as seguintes integrais definidas:

1) $\displaystyle\int_0^{\pi/2} \sqrt{1+\frac{1}{2}\operatorname{sen}^2 x}dx$

2) $\displaystyle\int_0^1 \sqrt{4-x^2}dx$

3) $\displaystyle\int_{-1}^1 \frac{1}{8+x^3}dx$

4) $\displaystyle\int_0^{2\pi} \frac{dx}{10+3\cos x}$

5) $\displaystyle\int_1^2 (x^2+x+1)dx$

6) $\displaystyle\int_0^{\pi} \operatorname{sen} x dx$

7) $\displaystyle\int_1^2 x\ln x dx$

8) $\displaystyle\int_0^{\pi} e^x \operatorname{sen} x dx$

9) $\displaystyle\int_0^{2\pi} \frac{\operatorname{sen} x}{1+\cos x}dx$

10) $\displaystyle\int_3^5 \frac{x}{x-1}dx$

11) $\displaystyle\int_{-\pi}^{\pi} x\operatorname{sen} x dx$

c) Calcula as seguintes integrais en intervalos non acotados:

1) $\displaystyle\int_0^{\infty} \frac{dx}{x^4}$

2) $\displaystyle\int_0^{\infty} e^{-x}\operatorname{sen} x dx$

3) $\displaystyle\int_{-\infty}^0 \frac{x^2+1}{x^4+1}dx$

4) $\displaystyle\int_0^{\infty} \frac{\operatorname{sin}^3 x}{1+\cos x+e^x}dx$

5) $\displaystyle\int_{-\infty}^0 e^{-x^2}dx$

d) Calcula as seguintes integrais de funcións non acotadas:

1) $\displaystyle\int_0^1 \frac{dx}{\sqrt{1-x^2}}$

2) $\displaystyle\int_0^1 x\ln x dx$

3) $\displaystyle\int_0^2 \frac{1}{x^2-4x+3}dx$

4) $\displaystyle\int_0^e \frac{1}{x\sqrt{\ln x}}dx$

e) Calcula as seguintes integrais dobres:

1) Sexa $A=\{(x,y)\in \mathbb{R}^2 : 1/2 \le xy \le 2, 1 \le x \le 3\}$

$$f(x,y) = \int\int_A \frac{e^{1/xy}dxdy}{y^2(x+1)^2} \tag{1}$$

2) $\displaystyle\int_0^1 \left[\int_0^y (x^2+y^2)dx\right] dy$

3) $\displaystyle\int_0^1 \left[\int_0^{\sqrt{1-x^2}} \sqrt{1-x^2-y^2}dy\right] dx$. Éste é o volume do elipsoide de ecuación $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2} = 1$ con $a=b=c=1$.

4) $\displaystyle\int_0^4 \left[\int_{-\sqrt{4-y}}^{\frac{y-4}{2}} (y+2x+20)dx\right] dy$

4. **Series numéricas**

a) Calcula as seguintes series e produtos numéricos infinitos:

1) $\displaystyle\sum_{n=0}^{\infty} \frac{2n+1}{2^n}$

2) $\displaystyle\sum_{n=1}^{\infty} \frac{n}{2^n}$

3) $\displaystyle\sum_{n=1}^{\infty} (-1)^n \frac{2n+1}{2^{n+1}} = -7/18$

4) $\displaystyle\sum_{n=1}^{\infty} 2^{n-1}\tan^2\frac{1}{2^n}\tan\frac{a}{2^{n-1}}$

5) $\displaystyle\sum_{n=1}^{\infty} ln\left[\frac{(n+1)^2}{n(n+1)}\right]$

6) $\displaystyle\sum_{n=1}^{\infty} ln\left(1+\tan^2\frac{1}{2^n}\right)$

7) $\displaystyle\prod_{n=1}^{\infty}\left[1-\frac{2}{n^4}\right]$

8) $\displaystyle\sum_{n=1}^{\infty} \frac{n-1}{(2n-1)(n+1)n^2}$

9) $\displaystyle\prod_{n=1}^{\infty}\left[1+\frac{1}{n^2+1}\right]$

10) $\displaystyle\prod_{n=2}^{\infty}\left[1+\frac{2n+1}{(n^2-1)(n+1)^2}\right] = 4/3$

11) $\displaystyle\sum_{n=1}^{\infty} ln\left(1+\tan^2\frac{a}{2^n}\right)$

5. **Series de potencias**

a) Calcula as expansións en serie de Taylor das seguintes funcións nos puntos e da orde que se indican (se non se indica nada, $x = 0, n = 5$):

1) $f(x) = \sin(\tan x) - \tan(\sin x), x = 0, n = 25$
2) $f(x) = x^3 - 1 - 4x^2 + 5x, x = 1, n = 10$
3) $f(x) = e^x \operatorname{sen} x$
4) $f(x) = \ln \dfrac{1}{\sqrt{1-x^2}}$
5) $f(x) = \dfrac{1 - x\cos x}{1 - 2x\cos x + x^2}$
6) $f(x) = \dfrac{\operatorname{arc\,sen} x}{\sqrt{1-x^2}}$
7) $f(x) = \operatorname{sen}\left(\dfrac{1}{1-x^2} - 1\right)$

b) Calcula as expansións en serie de potencias das seguintes funcións nos puntos e da orde que se indican:

1) $f(x) = \dfrac{1}{x(1+\sqrt{x})}, x = 0, n = 25$
2) $f(x,y) = \dfrac{1-y^2}{1-2xy+y^2}, x = 0, y = 0, n = 8$
3) $f(x) = \dfrac{1}{x-a}$ sen indica-la orde
4) $f(x) = \dfrac{1}{(x^2-1)(x-1)}$
5) $f(x) = \operatorname{arg\,senh} x$
6) $f(x) = \dfrac{1}{x^3+x^2+x+1}$

Semana 5

Traballo a desenvolver pol@ alumn@

1. **Representación gráfica**

a) Representa as superficies $z = x^2 - y^2$ e $z = -(x^2 - y^2)$

b) Representa a superficie $z = \cos xy$ en $x, y \in [-3, 3]$

c) Representa a curva paramétrica 3D $x(t) = t\cos 2\pi t, y(t) = t\operatorname{sen} 2\pi t, z(t) = 2 + t$

d) Representa a superficie catenoide, cuxa ecuación implícita é $\cosh z = \sqrt{x^2+y^2}$

e) Representa a curva animada $x^3 + y^3 - 5xy = 1 - \dfrac{t}{4}, t = 0, \ldots, 10$

SOLUCIÓN:

```
with(plots):
for t from 0 to 100 do P[t]:=implicitplot(x^3+y^3-5*x*y = 1 - t/4,x=-5..5,y=-5..5); od:
display([seq(P[t],t=0..100)], insequence=true);
```

f) Representa a función $f(x) = x!$ e a súa derivada en $[-4, 4]$

g) Representa a superficie 3D parametrizada por $x(u,v) = \cos u \operatorname{sen} 2v$, $y(u,v) = \operatorname{sen} u \operatorname{sen} 2v$, $z(u,v) = \operatorname{sen} u$, $u, v \in [0, 2\pi]$

h) Representa a superficie 3D parametrizada por:

$$x = \frac{u}{2} - \frac{u^3}{6} + \frac{uv^2}{2} \tag{2}$$

$$y = -\frac{v}{2} + \frac{v^3}{6} - \frac{vu^2}{2} \tag{3}$$

$$z = \frac{u^2}{2} - \frac{v^2}{2} \tag{4}$$

i) Representa a superficie 3D parametrizada por $x = u^2 + uv, y = u + v, z = u^3 + 3u^2v$.

j) Representa a función $f(x) = \int_{t=-\infty}^{x} e^{-t^4} dt$ no intervalo $[-2, 2]$

k) Representa a animación das curvas de Lissaojus, dadas por $x = \operatorname{sen} mt, y = \cos nt$, para $t \in [0, 2\pi]$ sendo m os 7 primeiros números de Fibonacci ($f_i = 1, i = 1, 2,\ f_i = f_{i-1} + f_{i-2}$; usa-la función `fibonacci(...)` de Maple), e $n = m + 1$.
SOLUCIÓN: (NOTA: teclear MAYÚSCULAS+RETURN entre as liñas 2 e 3)

```
with(plots):with(combinat,fibonacci):
for k from 1 to 7 do m := fibonacci(k): n := fibonacci(k+1):
P[k]:=plot([sin(m*t),cos(n*t), t=0..2*Pi]); od:
display([seq(P[k],k=1..7)], insequence=true);
```

l) Representa a lemniscata de Bernouilli, definida polas seguintes ecuacións cartesiana, polar e paramétrica:

$$(x^2 + y^2)^2 = (x^2 - y^2) \tag{5}$$

$$r^2 = \cos 2\theta \tag{6}$$

$$x = \frac{\cos t}{1 + \operatorname{sen}^2 t}, y = \frac{\cos t \operatorname{sen} t}{1 + \operatorname{sen}^2 t} \qquad t \in [-\pi, \pi] \tag{7}$$

m) Representa a función $z = f(x, y) = \dfrac{x}{x^2 + y^2}, x, y \in [-1, 1]$

n) Define a función $f(t) = \sum_{n=1}^{8} \dfrac{(-1)^{n+1}}{n} 2 \sin nt$. Represéntaa gráficamente e calcula $f(\pi/10)$

ñ) Representa gráficamente as curvas:

1) $x = \dfrac{t}{t^2 - 1}, y = \dfrac{t^2}{t^2 - 1}$
2) $x = e^{t-1} - t, y = t^3 - 3t$
3) $t^2 \log t, y = t(\log t)^2$

o) Representa gráficamente as curvas:

1) $x^3 + y^3 = 3xy$
2) $y^2(1 - x) = x^2(x + 3)$
3) $(x + y)^3(x - y) - 3xy + x = 0$
4) $x^7 + xy^4 - x^2y - y^2 = 0$

p) Representa gráficamente a superficie dada por $4(y - x^2)(xz - y^2) - (xy - z)^2 = 0$.

q) A ecuación de Schroedinger unidimensional, que na mecánica cuántica danos a función de onda $\Psi(x, t)$ dunha partícula sometida a un potencial $V(x)$, está dada por:

$$\frac{\partial^2 \Psi}{\partial x^2} + V(x)\Psi(x, t) = \frac{\partial \Psi}{\partial t} \tag{8}$$

(nesta ecuación eliminamos constantes que non afectan á forma da función solución). Se $V(x) = 0, |x| < 1$ e $V(x) = \infty, |x| > 1$ (pozo de potencial), a parte real da función de onda $\Psi(x, t)$ da partícula é a seguinte:

$$\Psi(x,t) = \begin{cases} \cos kx \cos \omega t & |x| < 1 \\ 0 & |x| > 1 \end{cases}$$

Representa gráficamente esta función para $k = 8, \omega = 5$ con $t = 0, \ldots, 10$, e 100 fotogramas.

r) Representa gráficamente o lugar xeométrico dos puntos dados pola ecuación $z = f(x,y,t) = \exp(-x^2 - y^2) \operatorname{sen} tx^2 y$.

2. **Resolución de ecuacións e sistemas de ecuacións**

a) Resolve simbólica e numéricamente as seguintes ecuacións e sistemas de ecuacións:

1) $a + \ln(x-3) - \ln x = 0$ (en función de a)
2) $\sqrt{x-8} + \sqrt{x} = 2$ (non ten solucións)
3) $x^3 - 5ax^2 + x - 1 = 0$ (en función de a)
4) $48x^5 + 8x^4 - 6x^3 + 114x^2 - 37x + 18 = 0$
5) $x^2 + y^2 = 5, xy = y^2 - 2$
6) $f(n+2) = xf(n+1) + yf(n), f(0) = 1, f(1) = 1$
7) $f(n+1) = \frac{8}{5} f(n) - f(n-1), f(0) = 0, f(1) = 1$
8) $f(n+1) = 3nf(n) - 2n(n-1)f(n-1), f(1) = 5, f(2) = 54$
9) Sistema de ecs:

$$z^2 - x^2 - y^2 + 2ax + 2az - a^2 = 0 \tag{9}$$
$$yz - ay - ax + a^2 = 0 \tag{10}$$
$$-2a + x + y = 0 \tag{11}$$

10) Calcula-los coeficientes enteiros p, q, r, s, t, u, v que axustan a ecuación química $pKMnO_4 + qH_2SO_4 + rH_2C_2O_4 \to sK_2SO_4 + tMnSO_4 + uH_2O + vCO_2$
11) $x^3 - (a-1)x^2 + a^2 x - a^3 = 0$. Logo, calcula as solucións para $a = 0, 1, 2$

b) Calcula $y = f(x)$ e representa gráficamente as seguintes curvas:

1) $x^2 y(y-x) + x^3 - 2y^2 = 0$
2) $(y-1)^3 - 4x^2(y+1) = 0$

c) Dado o sistema $x + y + z = 0, x - y - 2xz = 0$, calcula $x(z), y(z)$.

d) Dado o sistema $x^2 + y^2 + z^2 = 20, x - xy + z - 4 = 0$, calcula $y(x), z(x)$.

3. **Manipulación de polinomios e funcións racionais**

a) Comproba se as seguintes expresións son polinomios.

1) $(x+3)(5x^4 - 3x^2)$
2) $(x-2)(x+4)(7x+1)(3x-2)$
3) $(e^{3x} - 7)(x+1)$
4) $(5x - 6x^4 - 3x + 2)(4x^4 - 5x^2)$
5) $\cos(3x - 7\pi)(3x^2 - 4)$

Para os que sexan polinomios, calcula:

1) O grao do polinomio
2) O coeficiente do termo de maior grao
3) Obter o polinomio en forma canónica.
4) Evaluar as expresións nos puntos x=0

b) Asigna a variabel x o valor 1. Define o polinomio $(x-1)(x^3 - 4)$. Que aconteceu?

c) Divide o polinomio $4x^3 + 3x^2 + 2x + 1$ entre o polinomio $(x-2)$ dando o cociente e o resto. Multiplica o cociente polo polinomio $(x-3)(x-2)(x-1)$, dando o resultado en forma expandida. Calcula os graos de tódolos polinomios.

d) Calcula $(1-3x+x^3)^3$. De qué orde é o polinomio resultante?.

e) Dada a seguinte expresión racional $\dfrac{x^4+x^3-4x^2-4x}{x^4+x^3-x^2-x}$, calcula:

1) O cociente e o resto da división.
2) Escribe o polinomio do numerador e denominador como producto de factores irreducibles. Calcular o máximo común múltiplo e o mínimo común divisor do numerador e o denominador.
3) Transforma a expresión en $\dfrac{x^3+x^2-4x-4}{(x-1)(x+1)^2}$
4) Transforma a expresión en $\dfrac{(x+2)(x-2)}{(x-1)(x+1)}$
5) Transforma a expresión en $\dfrac{x^2-4}{x^2-1}$

f) Dado o polinomio $(x^2+xy+x+y)(x+y)$, transfórmao en:

1) $x^3+2x^2y+xy^2+x^2+2xy+y^2$
2) $(x+1)(x+y)^2$
3) $y^2+(2y+y^2)x+(1+2y)x^2+x^3$
4) $x^3+x^2+(2x^2+2x)y+(x+1)y^2$

Exercicios avanzados

1. Calcula as seguintes lonxitudes de arcos, áreas e volumes de recintos:

a) A lonxitude dun arco de curva está dado por:

$$L=\int_a^b \sqrt{1+f'(x)^2}dx \tag{12}$$

$$L=\int_{t_1}^{t_2} \sqrt{x'(t)^2+y'(t)^2}dt \tag{13}$$

$$L=\int_{\theta_1}^{\theta_2} \sqrt{\rho(\theta)^2+\rho'(\theta)^2}d\theta \tag{14}$$

$$L=\int_{t_1}^{t_2} \sqrt{x'(t)^2+y'(t)^2+z'(t)^2}dx \tag{15}$$

$$\tag{16}$$

Segundo sexa unha curva 2D en ecuación cartesiana, paramétrica $(x(t),y(t)$, polar $(\rho=\rho(\theta))$ ou paramétrica en 3D $(x(t),y(t),z(t))$. Calcula-la lonxitude do arco da curva $y^2=2px$ con $x\in[0,b]$

b) Calcula a lonxitude de arco da catenaria $y=a\cosh\dfrac{x}{a}$ con $x\in[0,b]$

c) Calcula a lonxitude de arco da curva $x=r(\cos t+t\,\mathrm{sen}\,t), y=r(\mathrm{sen}\,t-t\cos t)$ con $t\in[0,\pi]$

d) Calcula a lonxitude de arco da curva $\rho=a(\theta^2-1)/2$ con $\theta\in[0,\beta]$

e) Calcula a lonxitude de arco da curva 3D $x=3t, y=2\sqrt{2}t^{3/2}, z=3t^2/2$ con $t\in[0,b]$

f) Área entre $f(x)=\cosh x$ e $g(x)=\mathrm{senh}\,x$ con $x\geq 0$

g) Área entre as gráficas de $f(x)=x(x-1)$ e $g(x)=x/2$ con $x\in[0,2]$

h) Volume do sólido xerado ao rotar aredor do eixo OX o recinto limitado polas curvas $x=1$, $y=0$ e $y=1/\sqrt{1+x^2}$. Nota: este volume está dado por:

$$\pi\int_a^b f(x)^2dx \tag{17}$$

i) Volume do sólido limitado polas superficies $x^2 + y^2 = 1$, $z = 0$ e $z = x$

2. Os polinomios de Tchevyshev $T_n(x)$ verifican que:

$$\frac{1 - t^2}{1 - 2xt + t^2} = \sum_{n=0}^{\infty} \epsilon_n T_n(x) t^n \tag{18}$$

Con $\epsilon_0 = 1$ e $\epsilon_n = 2, n \geq 1$. Polo tanto, expandindo a función esquerda en serie en $t = 0$ podemos calcula-los polinomios $T_n(x)$. Calcula $T_2(x)$ e $T_{10}(x)$, e comproba a resposta cos comandos (que nos dan ambos polinomios):

```
with(orthopoly);2*T(2,x);2*T(10,x)
```

3. Dada a serie de potencias $\sum_{n=2}^{\infty} \frac{logn}{n} x^n$, calcula as series de potencias truncadas de orde 5 correspondentes ás series inversa, exponencial, logaritmo, derivada e integral.

4. Dadas as Funcións Integrais de Fresnel de coseno e seno:

$$F_c(t) = \int_0^t \cos \frac{\pi u^2}{2} du \qquad F_s(t) = \int_0^t \text{sen} \frac{\pi u^2}{2} du \tag{19}$$

Representa a curva paramétrica en $\mathbb{R}^2$ dada por $(F_c(t), F_s(t), t = 0, \ldots, 100)$

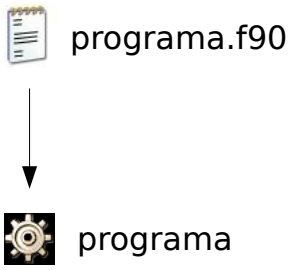
PROGRAMACIÓN ESTRUCTURADA EN FORTRAN

As mulleres na programación e na informática

Proceso de programación en Fortran

- Escritura dun programa fonte *programa.f90* co **editor** *kate*
- Compilación do programa dende unha **terminal** de comandos co compilador *f95*:

  f95 programa.f90 -o programa



- Corrección de erros de compilación
- Creación de programa executábel: *programa*
- Execución de *programa* (na terminal): *programa*
- Corrección de erros de execución e lóxicos (edición, compilación, execución)
- Coidado: non poñas *programa.f90* no canto de *programa*: se o fas, sobrescribes o programa

Programa básico en Fortran

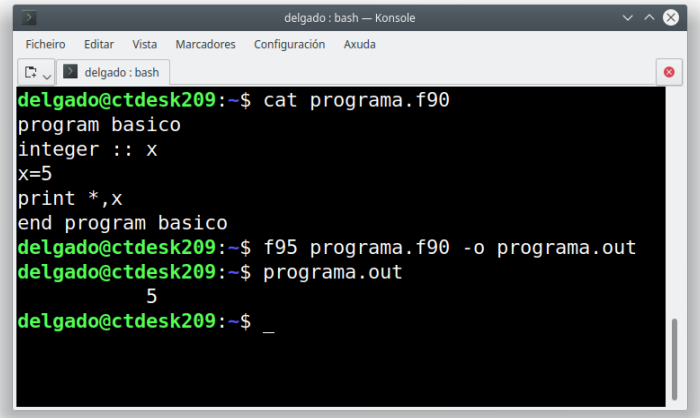
- Comeza con sentenza: "*program nome*"
- Remata con "*end program nome*"
- "*nome*" é o nome do programa: non pode haber variábeis nin subprogramas con ese nome
- Logo de *program*:
  - declaración de variábeis (ao principio)
  - sentenzas executábeis: operacións, entrada / saída, ...

Programa básico en Fortran

- Sentenza “*stop*”: remata a execución
  stop ‘mensaxe’: remata e imprime a mensaxe
- Exemplo de programa básico:

```
program basico
integer :: x
x=5
print *, x
end program basico
```

Imprime o nº 5 na terminal de comandos



- Liñas de comentarios: con ! ao comezo da liña

Entrada e saída estándar

- Entrada de datos estándar: le datos dende o teclado e almacénaos en variábeis
  - Sentenza *read*: *read *, x*
  - Pódese producir un erro se a variábel é numérica e introducimos un carácter (p.ex.)
- Saída de datos estándar: visualiza na terminal (pantalla)
  - Sentenza *print*: *print *, ‘resultado=‘, 2*x*
  - Con formato: *print ‘(“n=”,i0)’,n*
    print ‘(“x=”,f10.6)’,x

Ancho do nº enteiro

Ancho e nº de decimais

Estrutura de selección básica

- IF/ELSE: avalía unha serie de condicións e executa unhas sentenzas ou outras dependendo de que condicións se cumpren

```
if(condicion) then
   sentenzas1
else if(condición2) then
   sentenzas2
else
   sentenzas3
endif
```

```
if (x > 3) then
   print *, ‘alto’
else
   print *, ‘baixo’
end if
```


Estrutura iterativa *do* definida

- Permite repetir unha ou varias sentenzas un certo número de veces

```
do var = ini, fin, paso
     sentenzas
end do
```

```
do i = 1, 10
   print *, (i + 2)/3
end do
```

- Repite as sentenzas dende que a variábel *var* adopta o valor *ini*, ata que adopta o valor *fin*.
- En cada repetición, *var* increméntase en *paso*
- O incremento *paso* vale 1 por defecto

Acumulador

- Variable que aparece na esquerda e na dereita dunha asignación dentro dun bucle *do*: Exemplo: *x=x+i*; *p=p*j*
- Emprégase en operacións cun número variábel de operandos, que require un bucle *do*.
- O acumulador debe inicializarse co elemento neutro da operación.
- Acumulador de suma e de producto dun número variábel (*n*) de valores:

$\sum_{i=1}^{n} x_i$ →

```
suma=0
do i=1,n
    print *,'x? '
    read *,x
    suma=suma+x
end do
```

$\prod_{i=1}^{n} x_i$ →

```
prod=1
do i=1,n
    print *,'x? '
    read *,x
    prod=prod*x
end do
```


Estruturas iterativas *do* indefinidas

- Bucle *do/exit*: repite as sentenzas e remata cando se cumpre unha condición.

```
do
   sentenzas
   if(condición) exit
   sentenzas
end do
```

```
x=0;dx=0.1
do
   x=x+dx
   print *,x,x*x
   if(x>1) exit
end do
```

```
x=0;dx=0.1
do
   if(x>1) exit
   x=x+dx
   print *,x,sin(x)
end do
```

- Bucle *do while*: repite as sentenzas mentres se cumpre unha condición.

```
do while(condición)
   sentenzas
end do
```

```
x=0;dx=0.1
do while(x<1)
   print *,x,x*x
   x=x+dx
end do
```

Grace Murray Hopper (1906-1992)

- Inventora de compiladores das linguaxes de programación A0 e B0, para o cálculo de nóminas
- Contra-almirante da US Navy
- Muller do ano en Informática (1969), primeira muller na British Computer Society (1973)

Linguaxes de programación

- Linguaxe de programación: conxunto de sentenzas, regras sintácticas e palabras chave que permiten especificarlle ao ordenador unha tarefa a realizar
- Programa: arquivo de texto que contén sentenzas (comandos) na linguaxe de programación: programa fonte
- Estas sentenzas convértense en sentenzas executábeis (programa executábel) polo microprocesador usando un **compilador**, ou son executadas por un **intérprete**

Linguaxes compiladas e interpretadas

- Linguaxes de programación:
  - **Compiladas**: un programa traduce o programa fonte a programa executábel. Hai arquivos fonte e executábel (pódese executar só): Fortran, C/C++
  - **Interpretadas**: un programa interpreta o arquivo fonte, traduce e executa os seus comandos a código executábel. Non hai arquivo executábel. Só se pode executa-lo programa fonte co programa intérprete: Maple, Octave, Matlab, R, Python, Java

Linguaxes compiladas e interpretadas

- Linguaxe compilada: Fortran. O compilador (*f95*) traduce o programa fonte (*programa.f90*) a programa executábel (*programa*). Executas: *programa*

- Linguaxe interpretada: Maple, Octave e Matlab. Non hai arquivo executábel. Para executa-lo programa fonte, necesítase o intérprete para que o traduza a código executábel e o execute. Executas: *matlab programa.m*

Algoritmo

- Método para a resolución dun problema, detallado completamente en tódolos seus pasos
- Usualmente, tres etapas:
  - Entrada de datos: dende teclado ou arquivos
  - Procesamento: operacións cos datos
  - Saída de resultados: por pantalla ou a arquivos

Etapas do proceso de programación (I)

1) Análise do problema
2) Deseño do algoritmo
3) Codificación do programa fonte
4) Depuración do código
5) Proba do programa
6) Mantemento

Etapas do proceso de programación (II)

1) **Análise** do problema:

- Especificar: resultados a obter, datos de partida, posíbeis erros ou situacións límite, comportamento nestos casos, medidas de rendemento

2) **Deseño** do algoritmo:

- Técnica do deseño descendente: división do problema en subproblemas máis simples; resolución individual de cada subproblema se a súa complexidade o permite; caso contrario, nova división en subproblemas

Etapas do proceso de programación (III)

3) **Codificación** do programa fonte:

- Escritura do arquivo de texto nun editor (kate) ou entorno de desenvolvemento
- Escritura de sentenzas que resolven cada subproblema por separado
- Código intelixíbel: nomes adecuados para variábeis, coherencia, documentación, ...
- É inevitábel cometer erros que hai que correxir ...

Etapas do proceso de programación (IV)

4) **Depuración** do programa: corrección dos erros cometidos na codificación

- Erros de **sintaxe** (compilación)
- Erros de **execución**: prodúcense durante a execución do programa, aínda que non se viola a sintaxe da linguaxe: ex: división por cero, multiplicación de matrices non permitida, ... Producen o remate prematuro do programa
- Erros **lóxicos**: o programa remata ben pero non da resultados correctos

Etapas do proceso de programación (V)

5) **Proba**:

- Execución do programa con múltiples datos para comprobar que funciona ben en tódalas posíbeis situacións

6) **Mantemento**:

- Corrección de erros que aparezan durante a explotación do programa
- Face-los cambios necesarios para adaptarse a cambios no entorno (entradas / saídas / librarías usadas, ...).

Joan Clarke (1917-1996)

"A veces, la persona a la que nadie imagina capaz de nada, es la que hace cosas que nadie imagina"

Joan Clarke (1917)
Descrifó los mensajes alemanes de Enigma.

- Informática do exército británico especializada en criptografía
- Conseguiu descifrar o código Enigma dos alemáns durante a 2ª guerra mundial
- Nomeada cabaleira da Orde do imperio británico en 1947

Constantes e variábeis

- Os datos poden ser:
  - constantes: non se poden modificar durante a execución; dánselle valores no momento en que se comezan a usar: poden ter nome ou non
  - variábeis: pódense modificar, sempre teñen nome
- O nome dun dato pode ter letras, números, "_":
  - Non pode comezar por números
  - Non pode ter símbolos especiais: +?-=)/*$#

Nomes e tipos de datos

- Fortran NON distingue entre maiúsculas e minúsculas. Sempre usaremos minúsculas
- Os nomes deben ter significado: radio, temperatura, altura, ... Tamén deben ser curtos
- Os datos almacénanse en memoria RAM
- Datos de distintos tipos: enteiros, reais, reais de dobre precisión, complexos, lóxicos e carácter. Ocupan un nº de bytes distinto, e teñen un rango de valores distinto
- Fortran proporciona funcións intrínsecas para realizar operacións estándar (p. ex: funcións matemáticas estándar)

Declaración de variábeis e constantes

- En xeral, os datos deben ser declarados. Na declaración indícase o seu nome e tipo. Ex:

integer x

integer :: x

- Toda declaración debe estar antes de calquer outra sentenza que non sexa unha declaración. Se non, erro de compilación.
- Pódense inicializar na declaración: *integer :: x = -5*
- As constantes con nome decláranse co atributo *parameter* e sempre hai que inicializalas. Ex:

integer, parameter :: x = -10

- Tamén se poden usar constantes sen nome dos distintos tipos en expresións aritméticas: *print *, x + 3.5*

Funcións relacionadas cos tipos dos datos

- *huge(...)*: indica o valor máximo que pode acadar un dato do tipo indicado. Ex:

  integer x

  *print *, huge(x)* => 2147483647

- *precision(...)*: indica o nº de decimais

  real y

  real(kind = 8) z

  *print *, precision(y), precision(z)* => 6 15

- *kind(...)*: indica o nº de bytes que ocupa

  *print *, kind(x), kind(z)* => 4 8

Datos enteiros e reais

- Datos **enteiros**: *integer x*
  - ocupan 4 bytes, signo +/-, sen punto decimal
  - rango valores: $-2^{31}...2^{31}-1$
- Datos **reais**: *real x*
  - ocupan 4 bytes: signo +/-, partes enteira e decimal, expoñente (máx. 2 díxitos)
  - sen expoñente: *x=-1.2*
  - con expoñente: *x=-0.45e-18*
  - rango: $-3.4\cdot10^{38}...-3.4\cdot10^{-38}$, $3.4\cdot10^{-38}...3.4\cdot10^{38}$
  - precisión: 6 cifras decimais

Overflow e underflow; reais dobres

- Overflow / underflow: erro que se produce cando superamos o rango de valores dunha variábel real
  - Overflow: supérase o límite máximo: $\pm 3.4 \cdot 10^{38}$
  - Underflow: supérase o límite mínimo: $\pm 3.4 \cdot 10^{-38}$

```
real :: y = -1.2e+40
           1
Warning (128): Arithmetic overflow at (1)
```

- Reais de **dobre precisión**:
  - 8 bytes
  - Precisión: 15 cifras decimais

Exemplo: cálculo da media de 100 millóns de datos:
• Se sumamos e dividimos por N => overflow
• **Solución**: calcular 100 medias de 1 millón e a media destas 100

Reais dobres e complexos

- Reais de dobre precisión (continuación):
  - Declaración:

    real(8) x

    real(kind=8) x

    double precision :: x
  - Rango: $-1.79 \cdot 10^{308}$, ..., $-1.79 \cdot 10^{-308}$, $1.79 \cdot 10^{-308}$, ..., $1.79 \cdot 10^{308}$
- **Complexos**: parte real e imaxinaria

  complex c

  c=(-1.3, 1.5e-18)
  - Mostrar por pantalla: *print *, c* => (-1.3, 1.5e-18)

Lóxicos e carácter

- **Lóxicos**: só poden toma-los valores *.true.* e *.false.* (constantes)
  - Declaración: *logical x*
  - Inicialización: *x=.true.*
  - Mostrar por pantalla: *print *, x* => T
- **Cadeas de caracteres**:
  - Hai que indica-lo seu tamano máximo: non se pode superar
  - Se non se declara, vale 1: *character :: s='a'*
  - Declaración: *character(100) s*
  - Inicialización: s='ola que tal'
  - Lonxitude da cadea (nº de caracteres que realmente ten): *len_trim*(*s*)
  - Mostrar por pantalla: *print *, s* => hola que tal
  - Teste de igualdade: *s==t*
  - Relación de orde alfabética: *s<t*

Conversión entre tipos

- De enteiro a real: *x=i*
- De real a enteiro: *i=x*, pero podes perder información (p.ex. de 3.2 a 3).
- Podes redondear ao enteiro mais cercano con *i=nint(x)*, por defecto con *i=floor(x)*, por exceso con *i=ceiling(x)*.
- De carácter (p.ex. '32') a enteiro: *s='32';read (s,*) i*
- De carácter (p.ex. '3.2') a real: *s='3.2';read (s,*) x*
- De enteiro a carácter: *i=32; write (s,'(i0)') i*
- De real a carácter: *x=3.2; write (s,'(f3.1)') x*
- De carácter a real/enteiro, só funciona se o carácter ten un número real/enteiro. Se *s='ola'*, o *read* da erro de entrada/saída.

Declaración implícita (I)

- Na práctica, os datos básicos (reais e enteiros) non é necesario declaralos:
- Os datos enteiros non é necesario declaralos cando os seus nomes comezan polas letras *i j k l m n*
- Os datos reais non é necesario declaralos cando os seus nomes comezan polo resto de letras
- En resumo: se non declaramos un dato:
  - Se o seu nome comeza por {i j k l m n}, é enteiro
  - En caso contrario, é real

Declaración implícita (II)

- Se queremos datos doutro tipo (real dobre, complexo, lóxico, carácter, vector ou matriz), hai que declaralos
- A sentenza *implicit none* anula a declaración implícita no subprograma actual (*f95* da erro de compilación se hai variábeis sen declarar)
- A sentenza:

implicit tipo(rango), ..., tipo(rango)

permite asignar rangos de letras a tipos. Ex:

implicit integer(a-b), real(c-d), complex(e-z)

Vectores e matrices (I)

- Vector: **colección de datos** do mesmo tipo almacenados xuntos na memoria RAM. Cada elemento ten un índice.
- Matriz: cada elemento ten dous índices (fila e columna).
- Acceso a elementos e grupos de elementos: *x(i), x(i:j), x(i:), a(i,:), a(:,i), a(i:j,k:l)*. Vectores e matrices son **arrais**. Deben declararse.
- Inicialización de vector: *x=[1,2,3], x=[(i,i=1,10,2)], x=(/1,2,3/)*
- Inicialización de matriz *x*: a=*reshape(x,shape(a))*: o vector vai por columnas
- **Vector/matriz estático:** mesma lonxitude en tódalas execucións do programa.

```
real v(3)
integer :: a(3,3)
v(1)=2
v=[1,2,3]
a(2,3)=5
```

```
real v(10),a(2,2)
v=[(2*i+1,i=1,10)]
a=reshape(([((i*j,j=1,2),i=1,2)],shape(a)))
```


Vectores e matrices (II)

- **Vector/matriz dinámico**: distintas lonxitudes en distintas execucións.
- Resérvase a memoria cando se coñece o nº de elementos
- Se se accede a un elemento do matriz/vector **dinámico** antes do *allocate* ou despois do *deallocate* da erro de segmentación

```
real,allocatable :: v(:)
integer,allocatable :: a(:,:)
read *,n,nf,nc
allocate(v(n),a(nf,nc))
v(3)=2
a(2,3)=5
deallocate(v,a)
```

No *deallocate* non se indica o nº de elementos

- Pódese engadir elementos a un vector dinámico:

```
real,allocatable :: v(:)
v=[1]
do i=1,3
      v=[v,i]
end do
```

Vectores e matrices (III)

Dinámico

- **Declaración**: *real x(10)*, ou *real :: x(10)*, ou *real,allocatable :: x(:)*

  integer a(3,3), ou *integer :: a(3,3)*, ou *integer,allocatable :: a(:,:)*

- Alternativa: *tipo :: nome(N_1:N_2)*

  Se é dinámico: *allocate(x(-15:15))*

  Exemplo: *integer :: x(-5:5)*. Ten elementos x(-5)...x(5): $N_2 - N_1 + 1$ elementos: as funcións *lbound(x)* e *ubound(x)* dan N_1 e N_2

- Declaración vella (en exames resoltos):

  tipo,dimension(5) :: x
  tipo,dimension(-5:5) :: a

- **Ler** vector por teclado: *read *,x*

- **Ler** matriz:

```
do i = 1,n
   read *, (a(i, j),j=1,m)
end
```

Le a matriz por columnas: hai que traspoñer

```
read *,a
a=transpose(a)
```


Vectores e matrices (IV)

- **Inicialización** con bucle *do* explícito:

```
do i=1,n
   v(i)=i**2
end do
```

```
do i=1,n
   do j=1,m
      a(i,j)=i*j+i
   end do
end do
```

- **Inicialización** con bucle *forall*:

```
forall(i=1:10) v(i)=i*i+i-1
```

```
forall(i=1:5,j=1:6) a(i,j)=i*j-i+j
```

- **Escribir** vector: *print *,x* (tódolos elementos),

  *print *,x(1:n)*
  *print *,(x(i),i=1,n)*
  } só *n* primeiros elementos

- **Escribir** matriz:

```
do i = 1,2
   print *, (a(i, j),j=1,2)
end
```

Vectorización de expresións (I)

- Escribir unha expresión con vectores/matrices como se fose con números
- Sexan *x,y* dous vectores (de igual lonxitude) e *a,b* dúas matrices (de iguais dimensións).
- Dimensións: *size(x), size(a),size(a,1), size(a,2), shape(x), v=shape(a)*
- Asignación de valores a un arrai ou asignación dun arrai a outro: *x(:)=5; a(:,:)=7; x=y; b=a*
- Operacións aritméticas compoñente a compoñente: *x+2*y, x*y, x/y, 1/x, x**y, 1/a, a*b, 2**a, a**2, a**b*
- Suma e producto de elementos: *sum(x), sum(x(2:)), sum(a), sum(a,1), sum(a,2), product(x), product(a)*. Media dun vector: *sum(x)/size(x)*
- Valores e índices dos elementos máximo e mínimo dun vector: *maxval(x), minval(x), maxloc(x), minloc(x)*. O mesmo para matrices.
- Producto escalar: *dot_product(x,y)*
- Transposta dunha matriz: *b=transpose(a)*
- Producto de dúas matrices: *p=matmul(a,b)*

```
integer :: a(2,2)=reshape((/1,2,
    3,4/), shape(a)),x(2)
x=sum(a,2)
print *,sum(a,1)
```


Ada Lovelace (1815-1852)

- Inglaterra, século XIX (1815-1852)
- Inventora do primeiro programa de ordenador
- Precursora en máis de 100 anos da informática
- Linguaxe de programación de sistemas de tempo real chamado Ada na súa honra

Expresións aritméticas

- Operadores aritméticos: + - * / ** (exponenciación): x + 3, y – z, 4.3E-2*x, x/y, x**2
- Son binarios (2 operandos). O operador – pode ser unario: *print *, -x*
- Precedencia (prioridade):

  1) **

  2) * /

  3) - unario

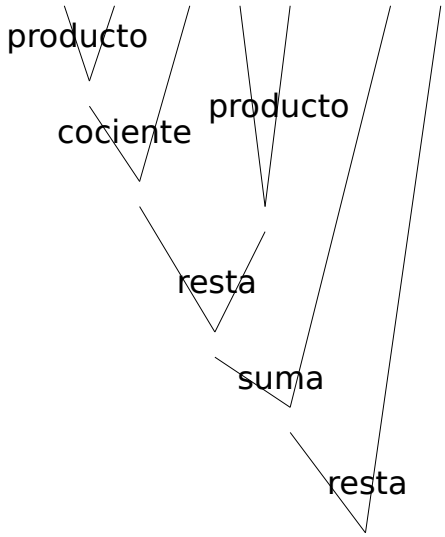
  4) + -
- Operadores de igual prioridade: execútanse de esquerda a dereita, agás a exponenciación (**), que se executa de dereita a esquerda

Prioridades na exponenciación

- Exponenciación: $x^{y^{z^t}}$ -> x**y**z**t
- Execútase de dereita a esquerda:
  - 1) z**t
  - 2) y elevado ao resultado de (1)
  - 3) x elevado ao resultado de (2)
  - Poderiamos escribir: x**(y**(z**t)), pero non é necesario poñer parénteses porque as prioridades xa fan que se execute nesa orde

Prioridades cos restantes operadores

a*b/c-4*y+3-c



As operacións combinan constantes e variábeis

Uso de parénteses

- As prioridades pódense modificar usando parénteses en función das nosas necesidades
- Exemplo: $\frac{\frac{x+y}{z-t}+1}{x+\frac{1}{y+z}}$ =>((x+y)/(z-t)+1)/(x+1/(y+z))
- Se o puxéramos sen parénteses: x + y/z – t + 1/x + 1/y + z=> $x+\frac{y}{z}-t+\frac{1}{x}+\frac{1}{y}+z$

Distinto de

- O nº de parénteses de apertura-peche debe coincidir. Se non coinciden, erro de compilación

Tipo (do resultado) dunha expresión

- Operación aritmética: dous operandos que poden ser de tipos distintos (p.ex., real e enteiro)
- Cando un operador actúa sobre datos de distintos tipos, o resultado é do tipo máis alto de ambos (*double* é máis alto que *real*, e *real* é máis alto que *integer*)
- Isto non impide a posíbel perda de información, dependendo dos seus valores.
- Exemplo: na división de enteiros, xa que o resultado será enteiro: perda de parte decimal, se existe: sexan *n=3,m=2* enteiros: *n/m* debería ser 1.5, pero como é enteiro, será 1
- Solución: declarar *n* e/ou *m* como reais; ou ben usa-la función *real(n)* para convertir *n* (ou *m*) en *real*

Tipos das expresións

- Sexan *x=1.2,y=1.0* reais, *n=3,m=2* enteiros. Na expresión:

$$x*y + n/m$$

- Erro común: pensar que na división *n/m* o resultado é real porque *x* e y son reais.
- Pero como *n* e *m* son enteiros, *n/m* é enteiro. Debería ser 1.5, pero vai ser 1 (enteiro): perda de información
- Hai que ver o tipo do resultado de cada operador individual (de dous operandos)

Sentenza de asignación

- A expresión da dereita pode ser unha constante, variábel, expresión aritmética ou función intrínseca
- Asigna o resultado da **expresión dereita** á **variábel esquerda**
- No lado esquerdo só pode haber unha variábel: non pode haber constante, expresión nin chamada a subprograma: erro de compilación
- Ambos lados poden ser de tipos distintos

Sentenza de asignación

- O valor da dereita convírtese ao tipo da esquerda: pode haber perda de información. Exemplo:

  n = x + 3.4 (*n* enteiro, *x* real)

A variábel da esquerda da asignación debe ter un tipo non **inferior** ao da expresión da dereita

integer < real < double

- Erro común: dúas asignacións consecutivas á mesma variábel. Ex:

  x = 3.4 – y

  x = z – y !pérdese o valor anterior de x

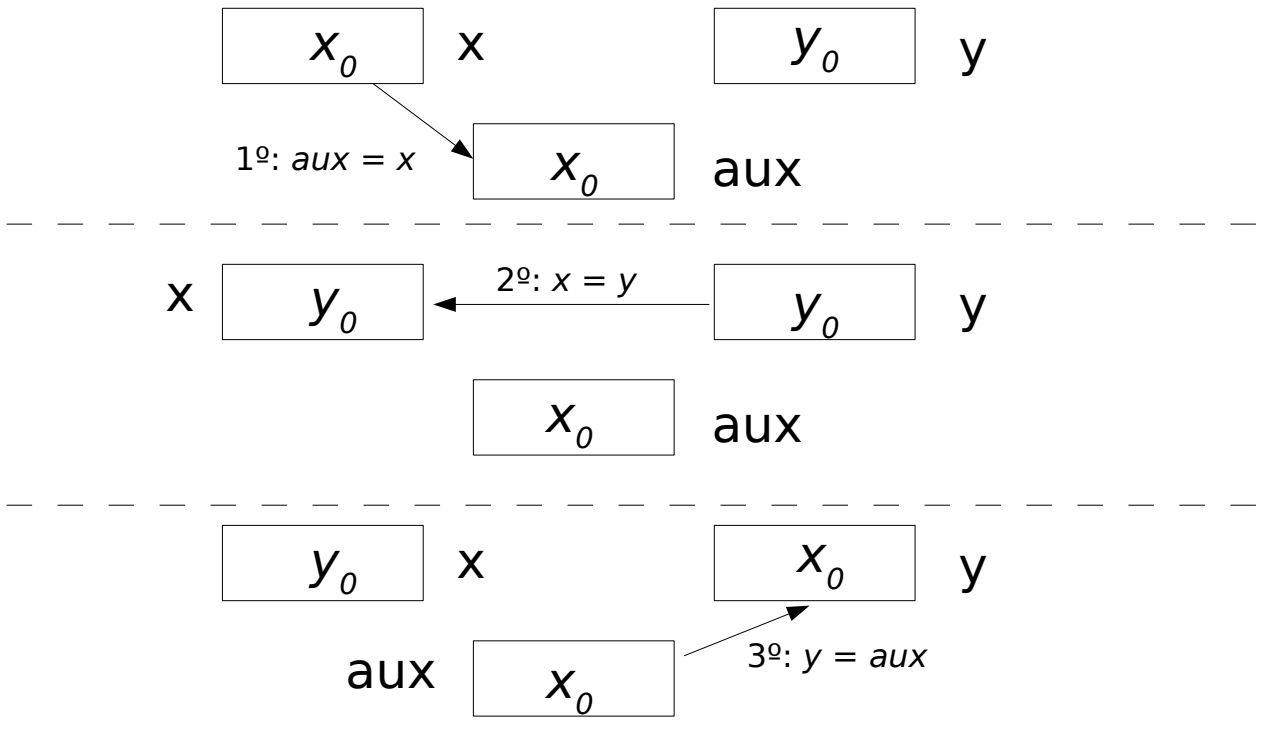
- Pódense encadear varias asignacións na mesma liña con “;”. Ex: *x = y + z; t = 2*x*

Exemplo: intercambio dos valores de dúas variábeis

- Temos dúas variábeis *x* e *y*: *x* ten o valor x_0, e *y* ten o valor y_0: queremos intercambiar os seus valores
- Distinguir entre o valor que se almacena nunha variábel e a variábel (que é o contedor do valor)
- Necesitamos unha variábel para almacenar temporalmente un dos dous valores: *aux*
- 1º paso: *aux* = *x* (almacena x_0 en *aux*)
- 2º paso: *x* = *y* (copia y_0 de *y* á variábel *x*)
- 3º paso: *y* = *aux* (copia x_0, que está almacenado na variábel *aux*, á variábel *y*)

Ollo: escribe na mesma variábel que salvou antes en *aux*

Intercambio de 2 variábeis



Intercambio de 2 variábeis

- Resumindo:

$aux = x$

$x = y$ ← Sempre escribe nunha variábel que se salvou na sentenza anterior a outra variábel

$y = aux$

- No intercambio, a variábel na dereita dunha sentenza de asignación debe aparecer na esquerda da seguinte sentenza de asignación
- Non pode estar a mesma variábel na esquerda de dúas sentenzas consecutivas de asignación

Algunhas funcións intrínsecas útiles de Fortran

- Valor absoluto: *abs(x)*. Argumento real/dobre.
- Raíz cadrada: *sqrt(x), csqrt(z)* para números complexos
- Resto da división enteira: *mod*(*m*,*n*) resto de *m*/*n*
- Trigonométricas: *sin*,*cos*,*tan*,*asin*,*acos*,*atan*
- Hiperbólicas: *sinh*,*cosh*,*tanh*,*asinh*,*acosh*,*atanh*
- Exponencial/logarítmica: *exp(x)*,*log(x)*
- Números aleatorios (0<*x*<1): *call random_number(x)*,*x=rand()*
- Arrai aleatorio: *real a*(3,3);*call random_number*(*a*)
- Truncamento a enteiro: *int*, *floor, ceiling, nint*
- Executar comando Linux/Windows: *call system('comando')*. Ex: *call system('ls')*: mostra o directorio actual

Funcións de sentenza

- Se queres definir unha función matemática, p.ex. *f(x,y)=x*y* que se poda calcular nunha única sentenza, podemos definir e chamar a unha función de sentenza (ou de liña) como a calquera función intrínseca
- Así non tes que poñer a expresión varias veces
- Hai que definila antes de ser chamada
- Os valores que se pasan como argumentos deben ter o mesmo tipo que o definido implícitamente polos argumentos da función de liña.

```
program exemplo
f(x,y)=x*y
print *,f(3.,2.5)
stop
end program exemplo
```

- Permite definir unha función como se fose intrínseca
- Só se pode chamar dende o programa principal no que se define.
- Verémolas de novo no tema de subprogramas

Programadoras da NASA no programa espacial de viaxe á lúa (1969)

Katherine Johnson: **matemática** e programadora da NASA
Primeira civil en recibir a **Medalla de Ouro** do Congreso USA

Mary Jackson: enseñeira da NASA

Película
“Figuras ocultas”

Dorothy Vaughan: programadora en **Fortran** do primeiro ordenador da NASA

Operadores relacionais (I)

- Definen as relacións de equivalencia (igualdade) e de orde numérica (con caracteres, a orde alfabética)
- Operadores: < <= > >= == /=
- Exemplos: x < 3, y >= z
- Operandos numéricos, resultado lóxico
- x < y: .*true*. se x < y, .*false*. en caso contrario
- x > y: .*true*. se x > y, .*false*. en caso contrario
- x == y: .*true*. se x e y son iguais, .*false*. en caso contrario
- *x /= y*: .*true*. se x e y son distintos, .*false*. en caso contrario

Operadores relacionais (II)

- Permiten definir condicións de igualdade / desigualdade ou maior/menor sobre números ou caracteres (orde alfabética)
- Úsanse en sentenzas de selección para crear distintas "rutas" de execución do programa, dependendo do cumprimento ou non dunha condición
- Sobre vectores e matrices, aplícanse por compoñentes:
  integer :: x(3)=(/1,2,3/)
  *print *,x>2 => F F T*

```
integer :: a(2,2)
forall(i=1:2,j=1:2) a(i,j)=i*i+j
print *,mod(a,2)==0
```

- Función *count*: nº de elementos dun vector/matriz que cumpren unha relación: *count(x>5)*, *count(x!=0)*, *count(mod(x,2)==1)*
- Caracteres: *character(10) :: s='ola'*

  s=='ola': T s>'pepe': F

```
integer :: a(2,2)
forall(i=1:2,j=1:2) a(i,j)=i*i+j
print *,count(mod(a,2)==1)
```

Operadores lóxicos (I)

- Combinan unha ou dúas condicións (definidas cos operadores relacionais) mediante relacións de:
  - **Conxunción**: *condición1* e tamén *condición2*
  - **Disxunción**: *condición1* ou *condición2*
  - **Negación**: non é certo *condición1*
  - **Equivalencia lóxica**: *condición1* e *condición2* son iguais (ou ben ambas se cumpren ou ben ningunha se cumpre)
  - **Non equivalencia lóxica**: *condición1* e *condición2* son distintas (se se cumpre unha non se cumpre a outra e ao revés)

Operadores lóxicos (II)

- Operadores:
  - Conxunción: .*and*.
  - Disxunción: .*or*.
  - Negación : .*not*.
  - Equivalencia lóxica (condicións iguais): .*eqv*.
  - Non equivalencia lóxica (condicións distintas): .*neqv*.

a	*.true.*	*.false.*	*.true.*	*.false.*
b	*.true.*	*.true.*	*.false.*	*.false.*
a.and.b	*.true.*	*.false.*	*.false.*	*.false.*
a.or.b	*.true.*	*.true.*	*.true.*	*.false.*
.not.a	*.false.*	*.true.*		
a.eqv.b	*.true.*	*.false.*	*.false.*	*.true.*
a.neqv.b	*.false.*	*.true.*	*.true.*	*.false.*

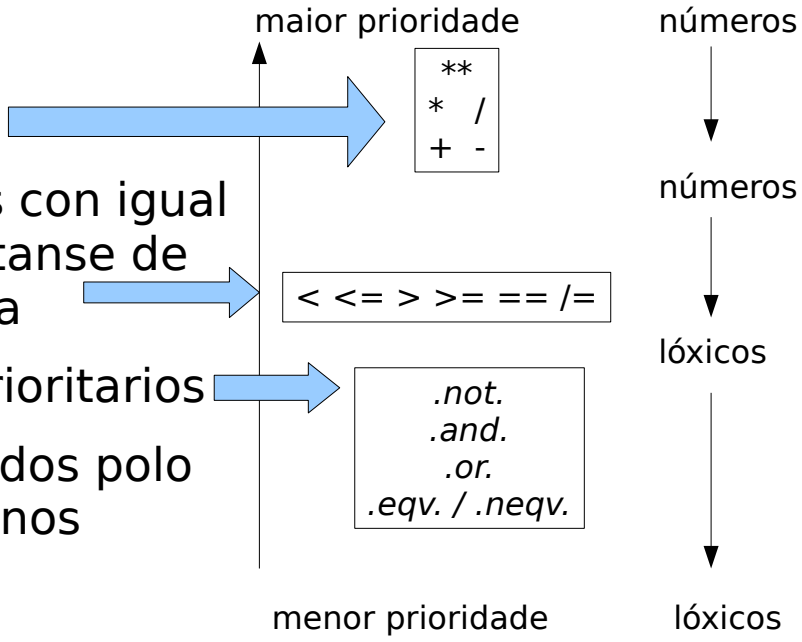
- Operandos lóxicos (resultado de operadores relacionais ou lóxicos). Resultado lóxico

Exemplos de operadores lóxicos

- AND: *x>5.and.x<=7*: x debe estar en (5,7] para que se cumpra a condición
- OR: *x==2.or.x==3*: x debe ser 2 ou 3 para que a condición sexa .true.
- NOT: *.not.(x>3)* é certa so se x<=3
- EQV: *mod(x,2)==0.eqv.mod(x,3)==0*: é certa se x é múltiplo de 2 e de 3, e tamén se x non é múltiplo de 2 nin de 3
- NEQV: *mod(x,2)==0.neqv.mod(x,3)==0*: é certa se x é múltiplo de 2 pero non de 3, e tamén se x non é múltiplo de 2 pero si de 3

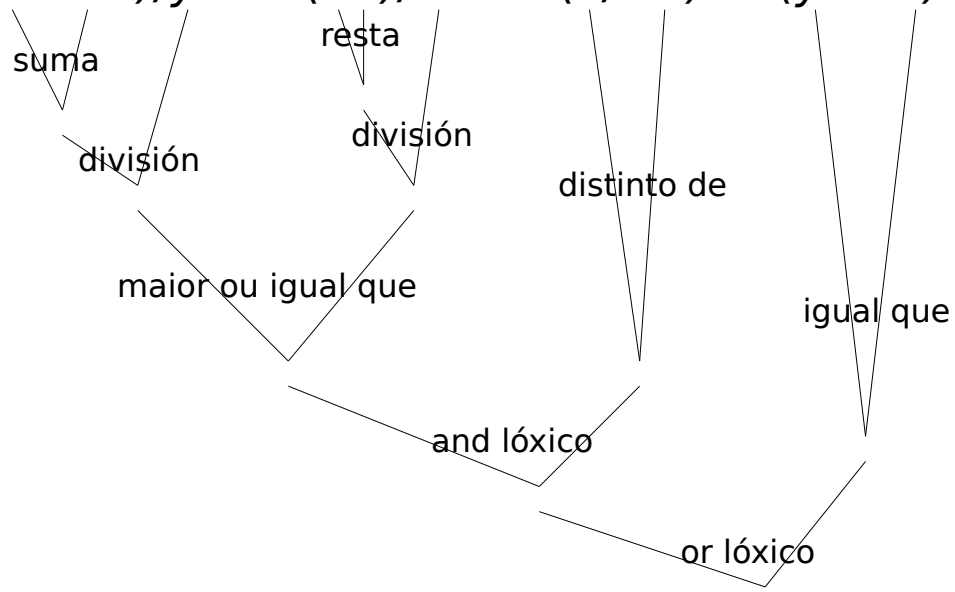
Precedencia de operadores

- Aritméticos: máis prioritarios
- Relacionais: todos con igual prioridade, execútanse de esquerda a dereita
- Lóxicos: menos prioritarios
- Operadores definidos polo usuario: os de menos prioridade



Exemplo de prioridades de execución

*(z+3.4)/y >= (z-t)/x.*and*.(x/=0).or.(y==z)*



Funcións lóxicas *all* e *any* para vectores/matrices

- *all(v/=0)*: *true.* se tódolos elementos do vector *v* son non nulos.
- *all(a>0)*: *true* se tódolos elementos da matriz *a* son positivos
- *all(a,1)*: executa o *all* por columnas (*a* debe ser *logical*)
- *all(a,2)*: executa o *all* por filas
- *any(a>5)*: *true* se *a* ten algún elemento maior que 5
- *any(mod(a,2)==0,1)*: *true* para columnas con elemento par.
- *any(a==3,2)*: *true* para as filas cun 3

```
integer :: a(2,3)=reshape((/1,-1,3,1,5,-2/),shape(a))
print *,all(a>0)        → False
print *,all(a>0,1)      → False  True  False
print *,any(a==3,2)     → True  False
```

```
1  3  5
-1 1 -2
```

Edith Clarke (1883-1959)

- Graduada en matemáticas e astronomía (1908)
- Calculadora (1912) en American Telephone and Telegraph (ATT)
- Creadora dunha calculadora gráfica patentada en 1925
- Enxeñeira eléctrica estadounidense e profesora de matemáticas, física e enxeñaría eléctrica

Sentenzas de selección

- Programa: execución secuencial de instrucións (*print*, *read*, asignacións, ...)
- Sentenza de selección: executa bifurcacións na execución secuencial do programa
- Avalía unha expresión lóxica: se se cumpre, executa unhas sentenzas; se non, executa outras ou non fai nada
- Permite obter programas complicados, con múltiples rutas de execución dependendo de datos de entrada
- Moi importantes na programación: permiten controla-lo fluxo de execución do programa

if lóxico / bloque *if*

- ***if* lóxico**: avalía unha condición (expresión lóxica) e dependendo do seu valor (.*true*./.*false*.) executa ou non unha única sentenza

```
if(condición) sentenza
```

  - A condición vai entre parénteses. Só válido cunha única sentenza. Se non se cumpre, non fai nada.
- **Bloque *if***: igual, pero permite varias sentenzas

```
if(condición) then
      sentenzas
end if
```

```
if(x>=10.or.y/=-5) then
    x = x+y; print *, x
end if
```

  - Atención ao *then*: se falta, erro de compilación

Bloque *if/else*

- Avalía a condición: se esta se cumpre, executa un bloque de sentenzas; se non se cumpre a condición, executa outro bloque distinto

```
if(condición) then
    sentenzas1
else
    sentenzas2
end if
```

```
if(x==0.or.y<-1) then
    x=x+y;print *, x
else
    y = y – x; print *, y
end if
```

- As sentenzas de selección poden conter outras sentenzas de selección, formando unha estructura complexa.

Bloque *if/else* múltiple

- Avalía *N* condicións: cada una ten un bloque de sentenzas, que se executa se se cumpre esa condición (entón xa non se avalía ningunha outra condición)
- A *i*-ésima condición so se avalía se non se cumpre a condicion (*i-1*)-ésima (nin as *i* condicións anteriores)
- Opcionalmente, pode haber un bloque *N+1* que se executa se ningunha condición se cumpre

```
if(condición1) then
    sentenzas1
else if(condición2) then
    sentenzas2
...
else if(condiciónN) then
    sentenzasN
else
    sentenzasN+1
endif
```


Sentenza *select*

- Compara secuencialmente unha *expresión* **enteira** con *N* selectores: valor único *(0)*, conxunto de valores *(1,2,3)* ou rango de valores *(:1)*
- Se a expresión é igual a algún selector, execútase ese bloque e remátase
- So se executa un bloque
- Pode haber opcionalmente un bloque *default* que se executará se a expresión non se corresponde con ningún selector

```
select case (expresión)
  case (selector1)
     sentenzas1
  case (selector2)
     sentenzas2
  ...
  case (selectorN)
     sentenzasN
  case default
     sentenzasN+1
end select
```

Exemplos de *if/else* e *select*

- Exemplo de *if/else* múltiple: función a cachos:

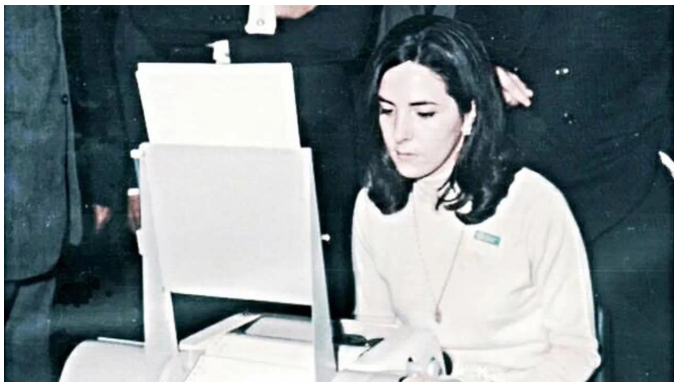
$$f(x)=\begin{cases}-1 & x<0\\ 0 & x=0\\ 1 & x>0\end{cases}$$

```
if(x < 0) then
    f = -1
else if(x == 0) then
    f = 0
else
    f = 1
end if
```

- Exemplo de *select*: código de operación: (*n* enteiro)

```
select case (n)
case (-1,0,1)
    print *, 'erro'
case (2:5)
    print *, 'correcto'
case default
    print *, 'apagado'
end select
```


Ana María Prieto López (1942-2018)



- Primeira programadora en España, de Santiago
- Dende os 21 anos traballou en IBM e foi a primeira programadora en España da empresa informática Bull
- Programou en COBOL e código máquina
- Traballou dende 1969 na Caixa de Aforros de Santiago

Sentenzas de iteración

- Repiten a execución dun bloque de sentenzas ...
  - Un certo nº de veces predeterminado (*iteración definida*)
  - Mentres se cumpra unha condición (ou ata que deixa de cumprirse): non se sabe de antemán o nº de repeticións (*iteración indefinida*)
- Grande importancia en programación
- Sentenza *do*: ambos tipos de iteración

Sentenza *do* definida

```
do var = ini, fin, paso
    sentenzas
end do
```

```
do i=1,10
    print *, v(i)
end do
```

imprime os elementos dun vector

- Antes de executar nada, asigna *ini* a *var*
- Repite as sentenzas mentres *var* <= *fin*
- Ao rematar unha iteración (repetición completa das sentenzas), executa *var = var + paso*
- O valor *paso*: opcional, por defecto vale 1
- Se modificamos *var* dentro das sentenzas: erro de compilación
- O parámetro *ini* debe ser enteiro, non real

Sentenza *do* definida

- Nº de iteracións do bucle *do*:

$$N = max\left\{0, \left\lfloor \frac{fin - ini + paso}{paso} \right\rfloor\right\}$$

- Non pode ser *paso = 0*: erro de compilación
- Se *paso < 0*, entón debe ser *fin < ini*, e a condición de continuidade do bucle é que *var >= fin*. Ex:

```
do i = 10, 1, -1
    print *, v(i)
end do
```

- Un bucles *do* pode estar dentro doutros bloques *do* ou *if/else*, ou conter bloques *if/else*.

Exemplo: cálculo de desviación típica dun vector

```
real,allocatable :: v(:)
real :: m
read *, n
allocate(v(n))
read *, v
m=sum(v)/n;d=0
do i=1,n
   t=v(i)-m;d=d+t*t
end do
print *,'desv=',sqrt(d/n)
deallocate(v)
```

Declara un vector dinámico
Le por teclado a lonxitude do vector
Reserva memoria para o vector
Le o vector por teclado
Calcula a media dos elementos do vector
Calcula a desviación típica dos elementos do vector de modo iterativo
Mostra a desviación por pantalla

Búsqueda de elementos comúns a dús vectores

- So con bucles *do*:

```
integer,parameter :: n=10
integer :: x(n),y(n),z(n)
k=0
do i=1,n
    u=x(i)
    do j=1,n
        if(u==y(j)) then
            k=k+1;z(k)=u;exit
        end if
    end do
end do
print *,(z(i),i=1,k)
```

- Vectorizado:

```
integer,parameter :: n=10
integer :: x(n),y(n),z(n)
k=0
do i=1,n
    u=x(i)
    if(any(u==y)) then
        k=k+1;z(k)=u
    end if
end do
print *,(z(i),i=1,k)
```


Erro de segmentación

- Usualmente a xestión de vectores e matrices realízase mediante bucles *do* definidos. Ex:

```
integer,dimension(3) :: v
do i = 1, 3
    v(i) = 2*i + 4
end do
```

- É posíbel que nos saiamos dos límites do vector, p.ex. se *i* chega a valer *10*
- En tal caso: *erro de segmentación*: é un exemplo de erro de execución (ver metodoloxía da programación), provoca o remate anticipado do programa

Detección de erros de segmentación

- Compila coa opción *-fcheck=all*
- Cando se produce o erro de segmentación durante a execución, indica en que liña se produce o erro
- Exemplo:
- Compilación:
  f95 -fcheck=all proba.f90 -o proba

```
program erro_segmentacion
real :: x(10)
do i=0,20
    x(i)=i
    print *,x(i)
end do
stop
end program erro_segmentacion
```

Erro de segmentación por chegar a ser i=0 ou i>10

```
At line 5 of file proba.f90
Fortran runtime error: Index '0' of dimension 1 of array
'x' below lower bound of 1
```


Uso de depurador gdb para atopar erros de segmentación

- Compila coa opción -g: f*95 -g programa.f90 -o programa* (engade información de depuración no executábel)
- Cargamo-lo executábel no gdb: *gdb programa*
- Executámolo no gdb: *run*
- Para ver ónde remata: *where*
- Seleccionar o nivel situado en *programa.f90*: *frame 3* (se o nivel #3 é o de *programa.f90*)
- Inspeccionar variábeis: *print i*
- Sair do gdb: *quit*

Sentenza *exit*

- Provoca o remate inmediato das iteracións
- Sempre vai dentro dunha sentenza de selección (asociada a unha condición)
- Se ésta se cumpre, execútase o *exit* e remata o bucle (se hai varios anidados, o bucle máis interno)
- A condición debe ter variábeis, e algunha delas debe cambiar o seu valor dentro do bucle para que remate

```
do i = 1, 10
   print *, “introduce un número (-1 para rematar):”
   read *, n
   if(-1 == n) exit  !remata a iteración neste intre
   suma = suma + n
end do
```


Bucle indefinido *do/exit*

```
do
   sentenzas
   if(condición) exit
end do
```

Executa as sentenzas antes de avaliar a condición (execútanse sempre polo menos unha vez)

```
x2=0.9
do
   x1=x2;x2=x1**2
   if(abs(x1-x2)<0.01) exit
   print *,x1,x2
end do
```

```
do
   if(condición) exit
   sentenzas
end do
```

Executa as sentenzas logo de avaliar a condición (poden non executarse nunca)

```
x=1
do
   if(x>10) exit
   x=x+0.1
   print *,x,x+1
end do
```

```
do
   sentenzas
   if(condición) exit
   sentenzas
end do
```

Remata cando se cumpre a condición: hai sentenzas antes e despois do *exit*: é máis xeral

O bucle *do/exit* executa as sentenzas ata que se cumpre a condición, que polo tanto é **condición de remate**.

Bucle indefinido *do/while*

- Executa as sentenzas mentres se cumpre a condición.
- A **condición é de continuidade**, non de remate.
- A condición avalíase antes de cada iteración.
- É menos flexíbel que o *do/exit*, porque a condición non se pode avaliar en calquera parte do bucle.
- Outro inconvinte: debes inicializar as variábeis da condición antes do *while*
- Tamén pode levar un *if/exit* dentro do bucle.

```
do while(condición)
   sentenzas
end do
```

```
s=0
do while(s<10)
   read *,x
   s=s+x
end
```

```
s=0
do while(s<10)
   read *,x
   if(x==-1) exit
   s=s+x
end
```


Bucle *do* indefinido

- O nº de iteracións depende do nº de veces que se non se cumpre a condición de remate

```
do
     print *, “introduce un nº (-1 para rematar)”
     read *, n
     if(n == -1) exit
end do
```

- Non se deben usar == nas condicións os erros en punto flotante poden evitar a igualdade: execución infinita. P. ex: ~~*if(x==y) exit*~~ *; if(abs(x-y)<1E-5) exit*
- Bucle *do* infinito:

```
do
     sentenzas
end do
```

```
do while(.true.)
   sentenzas
end do
```

O programa debería rematarse dende fora, p.ex. con CTRL+C

Bucle *do* indefinido

- **Exemplo**: aproximación dunha suma (serie) de infinitos sumandos: sumar só os sumandos > 1e-5

$$\sum_{n=1}^{\infty}\frac{1}{n^2}\simeq\sum_{n\in A}\frac{1}{n^2},A=\left\{n\in\mathbb{N}:\frac{1}{n^2}>10^{-5}\right\}$$

```
suma=0;n=1
do
   sumando=1./n**2
   if(sumando<1e-5) exit
   suma=suma+sumando
   n=n+1
end do
```

Debe ser real ou dará resultado 0

Versión con *do/exit*

Versión con *do-while*

```
suma=0;sumando=1;n=1
do while(sumando>1e-5)
   sumando=1./n**2
   suma=suma+sumando
   n=n+1
end do
```


Relación entre bucles definidos e indefinidos

- Un **bucle definido** pódese construir cun **bucle indefinido**, no cal a inicialización, actualización e teste de continuidade ou remate sobre a variábel fanse manualmente:

```
var = ini
do
     sentenzas
     var = var + paso
     if(var > fin) exit
end do
```

teste de remate

```
do var = ini, fin, paso
     sentenzas
end do
```

teste de continuidade

```
var = ini
do while(var<=fin)
     sentenzas
     var = var + paso
end do
```

- As versións indefinidas son útiles se queremos que a variábel a actualizar sexa real, xa que o bucle *do* definido so permite variábeis enteiras.

Bucle *do* híbrido

- É unha mestura de bucle definido e indefinido:

```
do var = ini, fin, paso
   sentenzas
   if(condición) exit
   sentenzas
end do
```

```
suma = 0
do n=1,100
   sumando = 1./n**2
   if(sumando < 1e-5) exit
   suma = suma + sumando
end do
```

```
sumando=1;suma = 0;n=1
do while(sumando>1e-5.and.n<100)
   sumando = 1./n**2
   suma = suma + sumando
   n=n+1
end do
```

- Executa as sentenzas un nº máximo de veces, pero pode rematar antes no *exit* se se cumpre a condición
- Evita unha execución infinita se a condición non está ben deseñada.
- Evita saírse dun vector ou matriz:

```
integer :: x(10)
s=0;i=1
do while(s<1.and.i<10)
   s=s+x(i);i=i+1
end do
```


Sentenza *cycle*

- Provoca que se salte as sentenzas que restan por executar na iteración actual. Sempre vai asociado a unha condición mediante unha sentenza de selección
- A execución salta ao comezo da seguinte iteración (con bucles *do*'s aniñados, salta á seguinte iteración do bucle *do* máis interno)

```
do i = 1, 10
      print *, “introduce un número natural:”
      read *, n
      if(n <= 0) then
         print *, “dixen que fora natural!!”
         cycle    !salta ao print na iteración i+1
      end if
      suma = suma + n
   end do
```

Nomeamento de bucles *do*

- Como pode haber bucles aniñados, pode resultar útil nomear os bucles:

```
nome1: do i = 1, 10,2
   nome2: do j = 1, 50, 4
      sentenzas
      if(condición) exit nome1
   end do nome2
end do nome1
```

- Permite, p. ex., rematar con *exit* o bucle *do* máis externo (se se desexa)
- Con *cycle*, permite saltar á seguinte iteración dun bucle *do* que non é o máis interno: *cycle nome1*

Sentenza *where/elsewhere*

- Executa sentenzas de asignación para tódolos elementos dun vector/matriz nos que se cumpre (ou non) unha condición.

```
where condición-arrai
   asignacións_arrai_1
elsewhere
   asignacións_arrai_2
end where
```

```
real :: a(3,3),b(3,3)
where(a==b)
      a=b+3
end where
```

```
real,dimension(10) :: v,w
where(v>w)
   v=v+w
elsewhere
   v=v-w
end where
```

Permite vectorizar expresións, evitando bucles *do* e executando operacións para moitos elementos á vez.

Sentenza *forall*

- **Sentenza *forall***: executa iterativamente sentenzas sobre arrai

  forall (var=ini:paso:fin,condición) sentenzas

  forall (var1=ini1:paso1:fin1,var2=ini2:fin2,condición)

  sentenzas

  end forall

```
forall (i=1:n,j=1:m,y(i,j)/=0.and.i/=j)
   x(i,j)=1./y(i,j)
   ...
end forall
```

- **Bucle *do* implícito**: *print *, (v(i), w(i), i = 1, 10)*
  - Imprime un vector nunha única liña
  - Imprime unha matriz, cada fila nunha liña da terminal

```
do i = 1, 3
   print *, (m(i, j), j = 1, 3)
end do
```


Sentenza *pack* con vectores

- Retorna un vector *k* cos índices, ou os valores, dos elementos do vector que cumpren unha condición. Se ningún a cumpre, vector baleiro (*size(k)*=0): similar á función *find* de Octave
- *pack(x,condición):* valores que cumpren a condición
- *pack([(i,i=1,n)],condición)*: índices dos elementos

  n=nº elementos do vector ou matriz que se testea

  condición: expresión lóxica na que hai un vector *x*: *x>=2*

Retorna valores que cumpren a condición

```
integer :: x=(/2,3,5,2/)
integer,allocatable :: z
print *,'x>2: ',pack(x,x>2)
z=pack(x,mod(x,2)==0)
print *,'x par: ',z
```

```
integer :: x=(/2,3,5,2/)
integer,allocatable :: k(:)
n=size(x)
k=pack([(i,i=1,n)],x==2)
print *,k  ! k(1)=1, k(2)=4
```

Retorna os índices dos elementos que cumpren a condición

Se queres almacenar os valores ou índices nun vector, hai que declaralo como allocatable sen reservar memoria

Sentenza *pack* con matrices

Retorna os valores da matriz que cumpren a condición

```
integer :: a(2,3)=reshape((/7,2,1,8,10,3/),shape(a))
integer,allocatable :: z
print *,'a>2: ',pack(a,a>2)
z=pack(a,mod(x,2)==0)
print *,'índices de valores pares: ',z
```

Retorna os índices dos elementos que cumpren a condición

```
integer :: a(2,3)=reshape((/7,2,1,8,10,3/),shape(a))
integer :: i(2,3)=reshape((/1,2,3,4,5,6/),shape(i))
integer,allocatable :: k(:)
k=pack(i,mod(a,2)==0)  ! k=(2,4,5)
k=pack(i,a>2)  ! k=(1,4,5,6)
k=pack(i,a==7)  ! k=1
```


Uso de *pack* para vectorizar expresións

- Eliminar elementos dun vector:

```
integer,allocatable :: x(:),y(:),j(:),k(:)
x=[(i**2,i=1,10)]
y=pack(x,x>=3.and.x<=5)  ! baseándose no valor de x(i)
j=[(i,i=1,10)]
k=pack(j,j/=2)
x=x(k) ! baseándose no índice (borra x(2)
```

- Eliminar filas e columnas dunha matriz

```
integer :: a(3,3)=reshape((/1,2,3,4,5,6,7,8,9/),shape(a))
integer,allocatable :: b(:,:),k(:),m(:)
k=[(i,i=1,3)]
m=pack(k,k/=2)  ! borra o elemento 2 de vector k
b=a(m,m)    ! colle filas 1,3 e columnas 1,3 de a
```

Conversión entre vectores e matrices

Función *reshape(x,forma)*: o argumento *forma* pode ser *(/n/)* (para convertir a vector de lonxitude *n*), ou (/*nf*,*nc*/), para convertir a unha matriz *nf*x*nc*.

- Conversión de vector a matriz: *a=reshape(v,(/nf,nc/))*
- Conversión de matriz a vector: *v=reshape(a,(/n/))*

```
real :: v(4)=(/1,2,3,4/)
real,dimension(2,2) :: a
a=reshape(v,(/2,2/))
```

Convirte de vector a matriz por columnas

```
real :: v(4)
real,dimension(2,2) :: a=reshape((/1,2,3,4/),shape(a))
v=reshape(a,(/4/))
print *,'v=',v
```

Función *shape(v), shape(a)*: retorna un vector coas dimensións do vector/matriz.

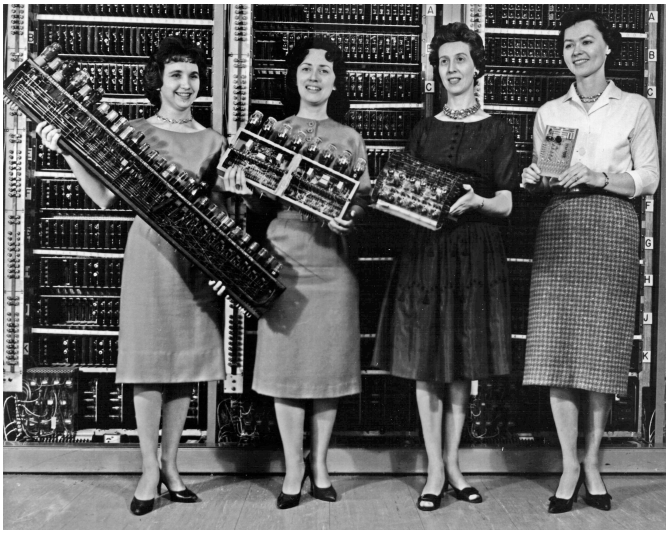
Programa de conversión de vector a matriz

```
program vector_matriz
real,allocatable :: x(:)
real,allocatable :: a(:,:)
real,parameter :: rand_max=2147483647
n=5;m=floor(sqrt(real(n)));k=1
allocate(x(n),a(m,m))
do i=1,n
      x(i)=floor(10.*irand()/rand_max)       !   nº enteiro aleatorio entre 0 e 10
end do
do i=1,m  ! conversión por filas
      do j=1,m
            a(i,j)=x(k);k=k+1            ! para convertir por columnas: a(j,i)=x(k)
      end do
end do
print *,'x=',x                ! imprime vector
print *,'a='                  ! imprime matriz
do i=1,m
      print *,(a(i,j),j=1,m)
end do
deallocate(x,a)
stop
end program vector_matriz
```

Programa de conversión de matriz a vector

```
program matriz_vector
real,allocatable :: x(:)
real,allocatable :: a(:,:)
n=3;m=n**2;k=1
allocate(x(m),a(n,n))
do i=1,n
      do j=1,n
            a(i,j)=i**2*j+i     !   inicializa a(i,j)
      end do
end do
do i=1,n  ! conversión por filas
      do j=1,n
            x(k)=a(i,j);k=k+1           ! para convertir por columnas: x(k)=a(j,i)
      end do
end do
print *,'a='                  ! imprime matriz
do i=1,n
      print *,(a(i,j),j=1,n)
end do
print *,'x=',x                ! imprime vector
deallocate(x,a)
stop
end program matriz_vector
```


Programadoras do ENIAC



- Betty Snyder Holberton (1917-2001)
- Betty Jean Jennings Bartik (1924-2011)
- Ruth Lichterman Teitelbaum (1924-1986)
- Kathleen McNulty (1921-2006)
- Frances Bilas Spence (1922-2012)
- Marlyn Wescoff Meltzer (1922-2008).

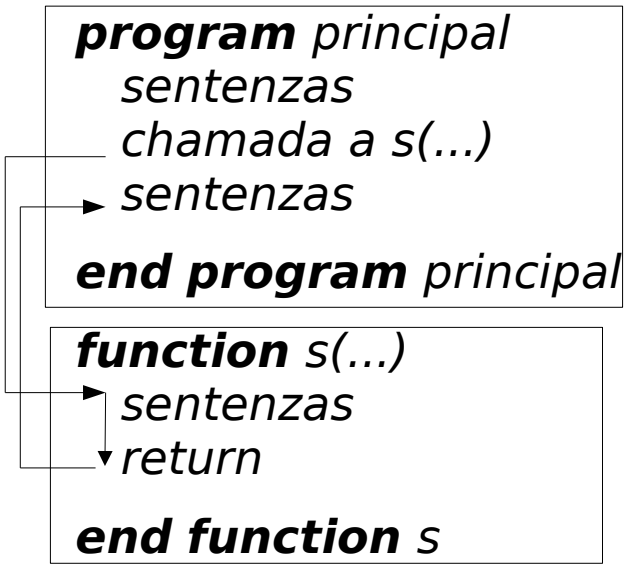
- ENIAC: Electronic Numerical Integrator And Computer
- Primeiro ordenador de propósito xeral (1946-1955)
- Elas desenvolveron as bases da programación de ordenadores

Subprogramas (I)

- Subprograma: conxunto de sentenzas que realizan unha tarefa clara, cunhas entradas (datos) e saídas (resultados) ben definidas. Exemplos:
  - subprograma que recibe un vector e calcula a súa media aritmética
  - subprograma que recibe unha matriz e calcula o seu determinante
- Hai dúas cousas:
  - Chamada ao subprograma dende o programa principal (*program*)
  - Corpo do subprograma: sentenzas do mesmo
- Argumentos: entradas e saídas do subprograma
- Poden estar no mesmo arquivo *.f90* ou en arquivos distintos

Subprogramas (II)

- O corpo do subprograma debe estar fóra do programa principal
- A chamada estará no programa principal (subprograma chamador)
- O subprograma ten as súas propias variábeis, inaccesíbeis dende o programa principal (so pode acceder aos argumentos)
- O programa principal tampouco non pode acceder ás variábeis do subprograma
- Dous tipos principais de subprogramas: **subrutinas** (non retornan valores) e **funcións externas** (retornan valores)



Sentenza *return* (opcional): retorna ao programa chamador (pode haber varias *return* no mesmo subprograma)

Tipos de subprogramas

Principais

- **Subrutina**: non retorna ningún valor, so ten argumentos *in-out-inout*
- **Función externa**: retorna un único valor (*integer*, *real*, *complex*, *double*, *character*, vector ou matriz), ten argumentos *in-out-inout*
- **Subprograma interno** (función ou subrutina): en programa principal, logo de *contains*. So se pode chamar dende o subprograma onde se define
- **Función de sentenza**: ten unha única sentenza, so se pode chamar dende o subprograma onde se define.

Cando usar funcións e cando subrutinas?

- **Función**: cando o subprograma so ten que calcular un único resultado (enteiro, real, complexo, lóxico ou carácter). A función retorna este resultado, almacenado na variábel co nome da función ou na variábel indicada no *result(...)*.

- **Subrutina**: cando o subprograma ten que calcular máis dun resultado, ou un único resultado pero éste é vector ou matriz. A subrutina debe ter algún argumento de saída ou entrada/saída (ver páxina seguinte) no que almacenar os resultados.

Argumentos (I)

- Tipo dos argumentos: *integer*, *real,* vector, matriz...
  - *intent(in)*: só lectura, o subprograma non pode modificalo (entrada): hai que inicializalos antes da chamada ao subprograma
  - *intent(out)*: non se pode ler (o chamador non lle dou ningún valor), só escribir nel (saída)
    - Non se pode inicializar antes da chamada
    - Logo de chamar ao subprograma hai que usa-lo seu valor
  - *intent(inout)*: o subprograma pode ler o seu valor e tamén modificalo (entrada/saída)

Argumentos (II)

- Os argumentos poden ser constantes, variábeis ou expresións
- Cando se pasa un argumento constante, variábel, ou expresión, hai que ter en conta o que espera o subprograma (*intent in*, *out* ou *inout*):
  - Se espera un argumento *out* ou *inout*, na chamada hai que pasarlle unha **variábel**: non constante ou expresión (daría un erro de compilación), porque vai ser modificado
  - Se o subprograma espera un argumento *in*, pódeselle pasar unha **constante, variábel ou expresión** (porque non vai modificarse)
- Os argumentos deben coincidir en número, tipo e orde na **chamada** e no **corpo do subprograma**, pero poden ter nomes distintos nos dous sitios
- Os argumentos *intent*(*in*) son constantes dentro do subprograma, e poden ser usados como dimensión de arrais estáticos.

Subrutina

- Non retorna ningún valor ao chamador: as entradas e saídas son a través dos argumentos. Axeitadas cando hai múltiples saídas

```
subroutine nome(arg1, ..., argN)
   tipo1, intent(...) :: arg1
   tipoN, intent(...) :: argN
   declaracións
   sentenzas executábeis

end subroutine nome
```

Variábeis locais do subprograma

- Chamada á subrutina: ***call*** *nome(arg1, ..., argN)*

Función externa

- Retorna un único valor, do *tipo* indicado:

```
tipo function nome(arg1, ..., argN) result(var)
tipo1, intent(...) :: arg1
tipoN, intent(...) :: argN
declaracións
sentenzas executábeis
var=expresión    !valor retornado: var
end function nome
```

Variábeis locais do subprograma

- A función chámase dende unha sentenza de asignación:
  tipo :: nome
  var=nome(arg1, ..., argN)
- Se non leva *tipo*, retorna un valor do tipo implícito de *nome*
- Se falta ***result***(*var*), a función retorna a variábel *nome*

Subprogramas e programa principal en arquivos distintos

- Os subprogramas e programa principal poden ir no mesmo arquivo *.f90*, ou en arquivos distintos, normalmente un arquivo distinto para cada subprograma
- Podes compilar co comando: *f95 *.f90 -o executabel*
- Tamén podes compilar cada arquivo coa opción *-c*:

f95 -c principal.f90
f95 -c subprograma1.f90
...
f95 -c subprogramaN.f90

- Crea un arquivo *.o* para cada arquivo *.f90*: non se executa
- Para crear o executábel:

*f95 *.o -o executabel*

Paso de vectores como argumentos

- **Neste curso**: pasar o nome do arrai e as súas dimensións (*explicit-shape arrays*)
- Cursos posteirores: pasar o nome do arrai (*assumed-shape arrays*) sen as súas dimensións: 3 posibilidades:
  - Subprograma interno
  - Módulo
  - Interface

```
program proba
integer :: x(10)
call sub(x,10)
end program proba
```

```
subroutine sub(x,n)
integer,intent(out) :: x(n)
integer,intent(in) :: n
...
end subroutine sub
```

```
program proba
integer,allocatable :: x(:)
...
call sub(x)
...
contains
    subroutine sub(x)
    integer,intent(out) :: x(:)
    n=size(x)
    ...
    end subroutine sub
end program proba
```

```
module aux
contains
    subroutine sub(x)
    integer,intent(out) :: x(:)
    n=size(x)
    ...
    end subroutine sub
end module
```

```
program proba
use aux
integer,allocatable :: x(:)
...
call sub(x)
...
end program proba
```

```
program proba
interface
    subroutine sub(x)
        integer,intent(out) :: x(:)
    end subroutine
end interface
integer,allocatable :: x(:)
call sub(x)
end program proba
```

```
subroutine sub(x)
integer,intent(out) :: x(:)
n=size(x)
...
end subroutine sub
```

Paso de matrices como argumentos

- Tamano explícito (**neste curso**):
- Tamano asumido (cursos posteriores):
  - Subprograma interno
  - Módulo
  - Interface

```
program proba
integer,allocatable :: a(:,:)
y=fun(a,n,m)
end program proba
```

```
function fun(a,n,m)
integer,intent(...) :: a(n,m)
integer,intent(in) :: n,m
...
end function fun
```

```
program proba
integer,allocatable :: a(:,:)
...
y=fun(a)
...
contains
    function fun(a)
    integer,intent(...) :: a(:,:)
    nf=size(a,1);nc=size(a,2)
    ...
    end function fun
end program proba
```

```
module aux
contains
    function fun(a)
    integer,intent(...) :: a(:,:)
    nf=size(a,1);nc=size(a,2)
    ...
    end function fun
end module
```

```
program proba
use aux
integer,allocatable :: a(:,:)
...
y=fun(a)
...
end program proba
```

```
program proba
interface
    function fun(a)
        integer,intent(...) :: a(:,:)
    end function
end interface
integer,allocatable :: a(:,:)
y=fun(a)
end program proba
```

```
function fun(a)
integer,intent(...) :: a(:,:)
nf=size(a,1);nc=size(a,2)
...
end function fun
```


Interface

- Bloque no que se indican subprogramas (tipo, nome, argumentos, valores retornados).

```
interface
    bloque subprograma1
    ...
    bloque subprograma N
end interface
```

- Non son necesarias, agás que os subprogramas teñan cousas especiais:

  Retornar arrais dinámicos
  Argumentos arrai sen a súa dimensión
  Argumentos nomeados ou opcionais

```
interface
    function fun(x,y,n) result(z)
        integer,intent(in) :: x(n),y(n)
        real :: z
    end function fun
    subroutine sub(x,y)
        real,intent(in) :: x
        real,intent(out) :: x
    end subroutine sub
end interface
```

Módulo

- Bloque de código que contén datos e subprogramas: ***use*** *modulo*

```
module nome
   tipo :: var
   interface
      tipo subprograma(...)
      ...
      end fipo
   end interface
contains:
   tipo subprograma(...)
   ...
   end tipo
end module
```

```
module proba
     integer :: x
contains
     subroutine sub(y)
          integer,intent(in) :: y
          print *,y
     end subroutine sub
end module
```

```
program modulo
use proba
call sub(5)
end program modulo
```

- Pode estar en arquivo distinto do programa principal
- Se está en arquivo distinto: *f95 modulo.f90 principal.f90*

antes do arquivo que o usa!

Exemplo de **subrutina**: ordeamento dun vector de números:

```
subroutine ordea_seleccion(x,n)
real, intent(inout) :: x(n)
integer,intent(in) :: n
do i = 1, n
   aux = x(i); k = i
   do j = i + 1, n
      if(x(j) < aux) then
         aux = x(j); k = j
      end if
   end do
   x(k) = x(i); x(i) = aux
end do
return
end subroutine ordea_seleccion
```

vector: argumento *inout* xa que se os seus elementos lense e tamén se modifican

```
real :: x(5) = (/5,2,4,3,1/)
call ordea_seleccion(x,5)
print *, "ordeado=", x
```

chamada ao subprograma

Exemplo de **función externa** sen cambiar o tipo do valor devolto

```
program proba
real :: x(5)=(/1,2,3,4,5/)
t=f(x,5)
print *,t
end program proba
```

Programa principal

Chamada á función

```
function f(x,n)
real,intent(in) :: x(n)
integer,intent(in) :: n
f=0
do i = 1, n
     f=f+i*x(i)
end do
end function f
```

Subprograma

$$f = \sum_{i=1}^{n} i\, x_i$$

o valor calculado debe retornarse almacenándoo na variábel co mesmo nome ca función

Exemplo de **función externa** cambiando o tipo e nome do valor devolto

Cambiando so o tipo do valor devolto

```
program proba
real :: x(5)=(/1,2,3,4,5/)
integer :: f  ! valor devolto enteiro
n=f(x,5)
print *,n
end program proba

integer function f(x,n)
real,intent(in) :: x(n)
integer,intent(in) :: n
f=0
do i = 1, n
     f=f+i*x(i)
end do
end function f
```

$$f = \sum_{i=1}^{n} i\, x_i$$

Cambiando o tipo e nome do valor devolto

```
program proba
real :: x(5)=(/1,2,3,4,5/)
integer :: f  ! valor devolto
enteiro
n=f(x,5)
print *,n
end program proba

integer function f(x,n) result(y)
real,intent(in) :: x(n)
integer,intent(in) :: n
y=sum([(i,i=1,n)]*x) !vectorizado
end function f
```

Subprograma que ten que calcular un vector ou matriz

- Pódese facer cunha subrutina ou cunha función, pero é máis fácil cunha subrutina, porque coa función necesitas unha **interface**
- Polo tanto, usa unha **subrutina** cun argumento *out* (ou *inout*) para o vector ou matriz

```
program exemplo_vector
integer,allocatable :: x(:)
read *,n
allocate(x(n))
call sub(x,n)
print *,'x=',x
deallocate(x)
end program exemplo_vector
!-----------------------------------------
subroutine sub(x,n)
integer,intent(out) :: x(n)
integer,intent(in) :: n
do i=1,n
      x(i)=i**2
end do
end subroutine sub
```

```
program exemplo_matriz
integer :: a(2,3)
call sub(a,2,3)
print *,'a='
do i=1,2
      print *,(a(i,j),j=1,3)
end do
end program exemplo_matriz
!-----------------------------------------
subroutine sub(a,n,m)
integer,intent(out) :: a(n,m)
integer,intent(in) :: n,m
forall(i=1:2;j=1:3) a(i,j)=i**2+j**3
end subroutine sub
```


Subrutina que reserva un vector ou matriz *intent*(*out*) cunha interface

Con vector

```
program exemplo
interface
      subroutine sub(x)
         integer,allocatable,intent(out) :: x(:)
      end subroutine sub
end interface
integer,allocatable :: x(:)
call sub(x)
print *,'x=',(x(i),i=1,size(x))
deallocate(x)
end program exemplo
!-----------------------------------------
subroutine sub(x)
integer,allocatable,intent(out) :: x(:)
allocate(x(5))
do i=1,5
      x(i)=i*i
end do
return
end subroutine sub
```

Con matriz

```
program exemplo
interface
      subroutine sub(a)
            integer,allocatable,intent(out) :: a(:,:)
      end subroutine sub
end interface
integer,allocatable :: a(:,:)
call sub(a)
do i=1,size(a,1)
      print *,(a(i,j),j=1,size(a,2))
end do
deallocate(a)
end program exemplo
!-----------------------------------------------
subroutine sub(a)
integer,allocatable,intent(out) :: a(:,:)
allocate(a(3,3))
do i=1,3
      do j=1,3
            a(i,j)=i*j+i-j
      end do
end do
return
end subroutine sub
```

Función externa que retorna un vector ou matriz reservado dinámicamente cunha interface

Con vector

```
program exemplo
interface
      function func(x) result(y)
         integer,intent(in) :: x(:)
         integer,allocatable :: y(:)
      end function func
end interface
integer :: x(5)=(/1,2,3,4,5/)
integer,allocatable :: y(:)
print *,'x=',x
y=func(x)
print *,'y=',y
deallocate(y)
end program exemplo
!-----------------------------------------------
function func(x) result(y)
integer,intent(in) :: x(:)
integer,allocatable :: y(:)
n=size(x);allocate(y(n))
do i=1,n
      y(i)=x(n-i+1)
end do
end function func
```

Con matriz

```
program exemplo
interface
      function func(a) result(b)
            integer,intent(in) :: a(:,:)
            integer,allocatable :: b(:,:)
      end function func
end interface
integer :: a(5,5)
integer,allocatable :: b(:,:)
b=func(a)
deallocate(b)
end program exemplo
!-----------------------------------------------
function func(a) result(b)
integer,intent(in) :: a(:,:)
integer,allocatable :: b(:,:)
nf=size(a,1);nc=size(a,2)
allocate(b(nf,nc))
do i=1,nf
      do j=1,nc
            b(i,j)=a(nf-i+1,nc-j+1)
      end do
end do
end function func
```


Exemplo de función que retorna un vector dinámico: *find*

- Atopa os índices dos elementos dun vector que cumpren unha condición (expresión lóxica)
- Moi útil para vectorizar expresións
- Retorna un vector reservado dinámicamente na función

```
function find(x) result(y)
logical,intent(in) :: x(:)
integer,allocatable :: y(:)
n=count(x);m=size(x)
allocate(y(n));j=1
do i=1,m
      if(x(i)) then
            y(j)=i;j=j+1
      end if
end do
end function find
```

```
program proba
interface
      function find(x) result(y)
         logical,intent(in) :: x(:)
          integer,allocatable :: y(:)
      end function find
end interface
integer :: x(5)=(/1,2,3,4,5/)
print *,count(x>2) !nº elementos >2
print *,find(x>2)    ! índices de elementos >2
print *,pack([(i=1,5)],x>2) !alternativa
print *,x(find(x>2))  ! valores de elementos >2
print *,pack(x,x>2) !alternativa
end program proba
```

Paso de subprogramas como argumentos

- Subprograma *external*: pode pasarse como argumento doutro subprograma. Ex: integral definida

```
program integral_definida
f(x)=sin(x); h(x)=cos(x)
real :: integral
external :: f, h
print *, integral(f, 0., 1.)
print *, integral(h, -1., 1.)
end program integral_definida
```

$$\int_0^1 \sin x\,dx$$

$$\int_{-1}^1 \cos x\,dx$$

```
real function integral(g,a,b)
real, intent(in) :: a, b
integral=0;x=a; h = 0.001
do
    integral=integral+g(x)
    x = x + h
    if(x > b) exit
end do
integral=integral*h
end function integral
```


Variábeis estáticas

- Son variábeis locais dos subprogramas que conservan o seu valor entre chamadas sucesivas a un subprograma
- Son estáticas porque se almacenan na mesma posición de memoria en tódalas chamadas ao subprograma
- As variábeis locais por defecto non son estáticas
- Para ser estática, dúas opcións alternativas:
  - Inicialización na declaración: *real :: x = 5*
  - Atributo *save: real, save :: x*
- Na 1ª chamada ao subprograma, o seu valor é 0 agás que se inicialice na declaración: *real :: x = 1*

Exemplo de uso das variábeis estáticas

```
program exemplo
do i=1,10
   call s()
end do
end program exemplo
```

```
subroutine s()
integer :: n=0
print *, "n=", n
n=n+1
end subroutine s
```

Subprograma que mostra por pantalla o nº de veces que se leva executado usando unha variábel estática

Variábel local estática por inicializarse na declaración

Execución: imprime por pantalla os números do 0 ao 9 dende o subprograma (sen que se lle pasen argumentos)

Atención: se inicializamos na declaración unha variábel local dun subprograma:
- Pasa a ser **estática** (e conservar o seu valor entre chamadas a subprograma)
- A inicialización só vale para a 1ª chamada ao subprograma

Función de sentenza

- É unha función que so ten unha sentenza cunha expresión matemática.
- Sintaxe: *nome(arg1,...argN)=expresion*
- Exemplo:

```
f(x)=x**2+x-2
print *,x,y,f(x),f(y)
```

```
integer :: f
f(x)=x**2+x-2
print *,x,f(x)
```

- O seu resultado é do tipo definido implícitamente polo seu nome. Para cambialo:
- So se pode usar no subprograma no que se declara.
- Unha función de sentenza pode ser usada na definición doutra función de sentenza:

```
length(r)=2*pi*r
area(r)=pi*r*r
key(r)=length(r)*area(r)
```

Recursividade (I)

- Subprograma recursivo: subprograma que se chama a si mesmo. Pode ser función ou subrutina
- Debe ter o atributo *recursive*: se non, erro de compilación
- **Funcións recursivas**:

```
recursive function nome(args) result(var)
  x = nome(...)   ! chamada recursiva
  var = ...       ! valor retornado
```

- O nome da función non pode ser retornado, porque debe ser usado para chamarse a si mesma
- A función debe chamarse a si mesma nalgún lugar do corpo

Recursividade (II)

- **Subrutinas recursivas**:

```
recursive subroutine nome(args)

call nome(...)
```

- Dentro da subrutina debe chamarse a si mesma
- Subprogramas recursivos: debe haber sentenza(s) de selección para distinguir entre:
  - Caso recursivo (hai chamada recursiva)
  - Caso non recursivo (non hai chamada)
- Se non hai caso non recursivo, secuencia infinita de chamadas: *stack overflow* (erro de execución)

Exemplo de función recursiva: factorial dun número enteiro

```
program exemplo
integer :: factorial
fn = factorial(5)
end program exemplo

recursive integer function factorial(n) result(fn)
integer, intent(in) :: n
if(n <= 1) then
   fn = 1
else
   fn = n*factorial(n-1)
end if
end function factorial
```

$$n! = n(n-1)\cdot(n-2)...2\cdot1$$

$$n! = n(n-1)!$$

Caso non recursivo

Caso recursivo

Implementación iterativa:

```
fn=1
do i=2,n
   fn=fn*i
end do
```

Vectorizada:

```
fn=product([(i,i=2,n])
```


Versión con subrutina recursiva

```
program exemplo
call factorial(5,fn)
print *,fn
stop
end program exemplo
!-------------------------------------------------------------
recursive subroutine factorial(n,fn)
integer, intent(in) :: n
integer, intent(out) :: fn
if(n <= 1) then
   fn=1
else
   call factorial(n-1,m)
   fn=n*m
end if
end subroutine factorial
```

Argumento *out* para almacenar o resultado

Caso non recursivo

Caso recursivo

$$n! = n(n-1)!$$

Argumentos nomeados

- Na chamada ao subprograma pódese especificar os nomes dos argumentos.
- Se na chamada se nomea un argumento, hai que nomealos todos.
- Pode haber chamadas con argumentos nomeados e outras sen nomealos.
- O subprograma debe declararse nun bloque *interface* no programa principal.

```
program argumentos_nomeados
interface
   subroutine nomeados(x,y)
      real,intent(in) :: x,y
   end subroutine
end interface
call nomeados(y=3.2,x=2.1)
call nomeados(1.1,2.2)
end program
argumentos_nomeados
!-----------------------------------------
subroutine nomeados(x,y)
real,intent(in) :: x,y
print *,'x=',x,'y=',y
end subroutine nomeados
```


Argumentos opcionais

- Teñen o atributo *optional*, de modo que se pode chamar ao subprograma indicando este argumento ou non.
- Non pode haber ningún argumento obrigatorio logo dun argumento opcional.
- O subprograma debe declararse nun bloque *interface* no programa principal.
- Función intrínseca *present(x)*: retorna *.true.* se o argumento *x* está presente, e *.false.* en caso contrario.

```
program argumentos_opcionais
interface
   subroutine opcionais(x)
      real,intent(in),optional :: x
   end subroutine
end interface
call opcionais(2.1)
call opcionais()
end program argumentos_opcionais
!-------------------------------------------------
subroutine opcionais(x)
real,intent(in),optional :: x
if(.not.present(x)) then
   print *,'argumento y non presente'
else
   print *,'argumento y presente',y
end if
end subroutine opcionais
```

Evelyn Berezin (1925-2018)

- Inventora (1957) do primeiro ordenador de oficina, gardaba libros e contas e automatizaba un sistema bancario nacional
- Creadora (1971) do primeiro software procesador de texto
- Desenvolvedora (1962) do primeiro sistema software de reservas de pasaxeiros (60 cidades) para a liña aérea United Airlines, o sistema informático máis grande construído ata entón

Formatos de E/S (I)

- Os datos (enteiros, reais, carácter, ...) pódense ler / escribir (teclado/pantalla ou arquivos) de distintas formas (ancho, nº de decimais, modo exponencial ou non, etc.)
- Sentenza *format*:

```
print  2, x, y, z    ! escritura en pantalla con formato 2
read 2, x, y, z   ! lectura por teclado co formato 2
print '(códigos de formato)', x, y, z
read '(códigos de formato)', x, y, z
2 format(códigos de formato)
```

- Os códigos de formato indican como se le / escribe cada dato, e débense axustar en nº aos datos (constantes / variábeis / expresións) a ler ou escribir.

Formatos de E/S (II)

- Ancho de campo: nº de caracteres máximo que pode ocupar un dato

Códigos de formato

- **Enteiros**: código I: **3i5**: 3 enteiros con 5 caracteres Ex: *print '(i2)', n; read '(i5)', m*

  O mellor: *print '(i0)',n*: escribe co ancho necesario.
- **Reais sen expoñente**: código F: **2f6.1**: 2 reais sen expoñente con 6 cifras e 1 decimal
- **Reais con expoñente**: código E: **4e10.3**: 4 reais con expoñente, con 10 cifras (incluíndo signo, parte enteira e fraccionaria, E do expoñente, signo e expoñente) e 3 decimais

Formatos de E/S (III)

- **Reais de dobre precisión**: código D: **2d10.4**: 2 datos de dobre precisión, igual que con código F
- **Caracteres**: código A: **2a10**: 2 cadeas de caracteres, cada unha con 10 caracteres. Se pos o código **a** sen ancho nun *print*, escribe a cadea co seu ancho
- **Espazos en branco**: código X: **5x**: 5 espazos en branco
- **Supresión de nova liña**: código $: *print '(“n? ”,$)'*
- **Tabuladores**: código T: **t10**: tabulador que chega até a columna 10-1=9

  print 1, x, y, n, a
  1 format(f6.2,e10.1,i7,a20)

1 real de ancho 6 e 2 decimais
1 real con expoñente, ancho 10 e 1 decimal
1 enteiro de ancho 7
1 carácter de ancho 20

Apertura de arquivos: *open* (I)

Sentenza *open*: permite abrir un arquivo e asocialo a unha unidade (representada por un nº enteiro):

```
open(1,file='datos.dat')
open(1,file='datos.dat',status='old'|'new',err=1)
```

- *status='old'*: o arquivo xa debe existir (p.ex. para ler): se non existe, *open* da erro para evitar ler dun arquivo baleiro
- *status='new'*: o arquivo non debe existir (p.ex. para crealo e escribir nel): se xa existe, *open* da erro para evitar sobrescribilo

Apertura de arquivos: *open* (II)

- En caso de erro, salta á sentenza con etiqueta 1:

```
open(1, file='datos.dat', status='new', err=1)
...
stop
1 stop 'erro en open abrindo datos.dat'
end program
```

- Neste exemplo, a unidade é a nº 1. Hai unidades reservadas (non se poden usar):
  - Erro estándar (terminal): 0
  - Entrada estándar (teclado): 5, *
  - Saída estándar (terminal): 6, *

Lectura de arquivos: *read* (I)

- Sentenza *close*: desvincula arquivo e unidade:

close(1)

- Lectura dende un arquivo: *read*

```
read (1, *) x, y, z
read (1, fmt=1,end=2) a, b, c
1 format(3f5.2)
...
2 close(1)
```

moi importante: variábeis nas que se almacenan os datos lidos (hai que saber previamente a forma do arquivo): fácil de esquecer

- Le unha liña do arquivo e almacena os datos lidos nas variábeis indicadas
- Unidade (1), formato: * (por defecto), etiqueta ou códigos de formato

Lectura de arquivos: *read* (II)

- Cada *read* faise nunha liña distinta do arquivo
- Opción *end=2*: salta á sentenza con etiqueta 2 cando atopa o final do arquivo
- Exemplo: le un arquivo até que atopa o final:

```
character(100) :: a
open(1, file='datos.dat', status='old', err=1)
do
  read (1, *, end=2) a
end do
2 close(1)
stop
1 stop 'erro lendo de datos.dat'
```

Exemplo: lectura dun arquivo dato a dato

```
program lectura_arquivo_dato_a_dato
open(1,file='arquivo_dato_a_dato.dat',status='old',err=1)
do
    read (1,'(i2,$)',end=2) n
    print '(i0," ",$)',n
end do
2 close(1)
print *,''
stop
1 stop 'arquivo_dato_a_dato non existe'
end program lectura_arquivo_dato_a_dato
```

Lectura dun enteiro sen pasar á seguinte liña

Remate do bucle ao chegar á fin de arquivo

arquivo_dato_a_dato.dat

```
1 2 3 4 5 6 7 8 9
8 7 6
```


Uso de *read* para extraer variábeis de cadeas de texto formatadas

- Supón que tes unha cadea de texto na cal hai valores numéricos (enteiros/reais) e caracteres: ‘x= 5.32 n=512 s=ola caracola’
- Para extraer os valores 5.32, 512 e ‘ola caracola’ a variábeis real, enteira e carácter debes usar a sentenza *read*:

```
character(100) :: s='x= 5.32 n=512 s=ola caracola'
character(50) :: t,u,v,w
print *,s
read (s,'(a3,f4.2,a3,i3,a3,a)') u,x,v,n,w,t
print *,x,n,t
```

Extrae os valores 5.32. 512 e ‘ola caracola’ ás variábeis *x*,*n* e *t*

Os a3 correspóndense con ‘x= ’, ‘n= ’ e ‘s= ‘

O *read* ten un carácter (*s*) no canto dunha unidade de arquivo

Escritura de arquivos: *write*

write (1,) x, y, z*

- Escribe no arquivo da unidade 1 as variábeis indicadas co formato indicado (neste caso, por defecto, *)
- Para escribir na saída estándar (terminal), poñer unidade=5 ou *; ou *print *, x, y, z*
- Pódese engadir a opción *err=n* (*n*=etiqueta de sentenza á que salta se hai erro na escritura)
- Posíbeis erros: tentar escribir en arquivos nos que non hai permiso de escritura, ou en arquivos de directorios que non existen

Exemplos de formatos de E/S

```
real :: x = -3.2, y = -1.1234
integer :: n = -10, m = 19
write (1, fmt=1) x, n, y, m
1 format(e10.3, t15, i4, f8.2, 6x, i8)
```

```
-0.320E+01    -10   -1.12          19
```

Saída do *write*

```
write(1,fmt=2) n, m, x
2 format(2(i5,4x), e8.1)
```

```
 -10       19    -0.3E+01
```

Saída do *write*

```
write(1,fmt=3) x, y
3 format(2f3.1)
```

non hai ancho de campo dabondo para os datos

```
***  ***
```

Saída do *write*

Exemplo: creación e escritura dun arquivo

```
open(1,file='datos.dat',status='new',err=1)
do i = 1, 10
   write (1, *) i
end do
close(1)
stop
1 print *,'erro en open abrindo datos.dat'
stop
```

- Se non lle poñemos *status='new'*, non nos daría erro en caso de xa existi-lo arquivo. Neste caso, sobrescribiría o arquivo, se este existe, **perdéndose**. Isto é perigoso

Uso de *write* para crear cadeas de texto formatadas a partir de variábeis

- A sentenza *write* tamén permite escribir en cadeas de texto, non so en unidades de arquivo.
- Supoñámos que tes as variábeis *x* (real), *n* (enteira) e *s* (cadea de caracteres).
- Para crear unha cadea de caracteres *t* da forma '*x=X n=N s=S*', onde X, N e S son os valores das variábeis *x* (con 5 cifras decimais), *n* (con 7 cifras) e *s*, tes que facer o seguinte:

```
character(20) :: s='ola caracola'
character(100) :: t
x=3.141592;n=1830
!t debe ter lonxitude dabondo para todo (x,n,s)
write (t,'(a,f9.5,a,i7,a,a)') 'x=',x,' n=',n,' s=',s
print *,t
```

Escribe en *t* a cadea formatada 'x=*X* n=*N* s=*S*'

Escritura de arquivos: pregunta antes de sobrescribir

```
program exemplo
character(5) :: s
open(1,file='datos.txt',status='new',err=1)
2 write(1,*) 'ola'
close(1)
stop
1 print '("datos.txt existe: sobrescribir?(s/n) ",$)';read *,s
if(s/='n') then
   open(1,file='datos.txt');goto 2
end if
stop
end program exemplo
```

Vai a sentenza con etiqueta 2

Escritura ao final dun arquivo (I)

- Tes que empregar a subrutina:

  fseek(unidade, desprazamento, onde)

- Move o cursor polo arquivo, en función do argumento *onde*: *onde=0* (comezo do arquivo), *1* (posición actual no arquivo), *2* (final do arquivo);
- Argumento *desprazamento*: nº de bytes (ou caracteres) logo do comezo / actual / final.
- Exemplos:

  call fseek(1,0,0): sitúa o cursor ao comezo do arquivo

  call fseek(1,10,1): sitúa o cursor 10 bytes logo da posición actual

  call fseek(1,-20,2): sitúa o cursor 20 bytes antes do final do arquivo

Escritura ao final dun arquivo (II)

```
open(1, file='datos.dat', status='old', err=1)
call fseek(1,0,2)  ! sitúase ao final do arquivo
write (1, *) x, y, z  ! engade ao final do arquivo
close(1)
stop
1 print *, "erro en open abrindo datos.dat"
```

- Función *ftell*: retorna a posición (en nº de bytes ou caracteres dende o comezo do arquivo) do cursor:

  pos = ftell(unidade)

- Ex: *print *,'pos=', ftell(1)*
- Función *rewind(unidade)*: vai ao comezo do arquivo

Subrutina que le un vector columna dende un arquivo

Hai que reservar memoria para o vector na subrutina e definir a interface axeitada:

```
program proba
interface
    function le_vector(nf) result(x)
        character(len=100),intent(in) :: nf
        real,allocatable :: x(:)
    end function
end interface
character(len=100) :: nf
real,allocatable :: x(:)
print '(a,$)','nf? '
read *,nf  ! usa 'vector.dat'
x=le_vector(nf)
print *,'x=',x
deallocate(x)
stop
end program proba
```

```
function le_vector(nf) result(x)
character(len=100),intent(in) :: nf
real,allocatable :: x(:)
open(1,file=nf,status='old',err=1);n=0
do
    read (1,*,end=2);n=n+1
    print *,n
end do
2 allocate(x(n))
rewind(1)
do i=1,n
    read (1,*) x(i)
end do
return
1 print *,'open: arquivo',nf,'non existe'
stop
end function le_vector
```

Le arquivo e conta datos

Reserva memoria

Le datos e almacena no vector

Arquivo vector.dat:

```
1
2
3
4
5
```

Subrutina que escribe un vector nun arquivo

```
subroutine escribe_vector(v,n)
real, intent(in) :: v(:)
integer, intent(in) :: n
open(1, file = “vector.dat”, status=’new’)
write (1, *) (v(i), i = 1, n)
close(1)
end subroutine escribe_vector
```

```
real, dimension(3) :: v=(/1,2,3/)
call escribe_vector(v,3)
...
```

Inicialízase o vector no programa principal

Subrutina que le unha **matriz cadrada** dende un arquivo

```
1 2 3
4 5 6
7 8 9
```

Arquivo matriz_cadrada.dat

```
program principal
interface
    function le_matriz(f) result(b)
        integer,dimension(:,:),allocatable :: b
        character(100),intent(in) :: f
    end function le_matriz
end interface
integer,dimension(:,:),allocatable :: a
character(100) :: f='matriz_cadrada.dat'
a=le_matriz(f)
print *,'a='
n=size(a,1)
do i=1,n
    print *,(a(i,j),j=1,n)
end do
stop
end program principal
```

```
function le_matriz(f) result(a)
character(100),intent(in) :: f
integer,dimension(:,:),allocatable :: a
character(100) :: s
open(1,file=f,status='old',err=1)
n=0
do
    read (1,*,end=2)
    n=n+1
end do
2 allocate(a(n,n))
rewind(1)
do i=1,n
    read (1,*) (a(i,j),j=1,n)
end do
close(1)
return
1 print *,'erro: arquivo ',f,'non existe'
stop
end function le_matriz
```

Le arquivo e conta liñas

Reserva memoria

Le datos e almacena en matriz

Programa que le unha matriz **non cadrada** dende un arquivo

```
program proba
character(100) :: s
integer,allocatable :: a(:)
open(1,file='matriz.dat',status='old',err=1)
read (1,'(a)') s
nf=1;nc=1
do i=1,len_trim(s)
   if(s(i,i)==' ') nc=nc+1
end do
do
   read (1,*,end=2)
   nf=nf+1
end do
2 rewind(1)
allocate(a(nf,nc))
do i=1,nf
   read (1,*) (a(i,j),j=1,nc)
end do
print *,'a='
do i=1,nf
   do j=1,nc
      print '(i5,$)',a(i,j)
   end do
   print *,''
end do
close(1);deallocate(a)
stop
1 stop 'erro: matriz.dat non existe'
end program proba
```

1 2 3
4 5 6

Arquivo matriz.dat

Conta as columnas da matriz

Conta as filas da matriz

Le a matriz

Mostra a matriz por pantalla

O nº de columnas é o nº de espazos da columna máis 1

len_trim(*s*): lonxitude de *s*

Subrutina que escribe unha matriz nun arquivo

```
program principal
real, dimension(3,4) :: a
forall(i=1:3,j=1:4) a(i,j)=i*j+j/2
call escribe_matriz(a,3,4)
end program principal
```

Inicialízase a matriz no programa principal

```
subroutine escribe_matriz(a)
real, intent(in) :: a(:,:)
nf=size(a,1);nc=size(a,2)
open(1, file='matriz.dat',status='new',err=1)
do i = 1, nf
   write (1, *) (a(i, j), j = 1, nc)
end do
close(1)
return
1 print *,'erro escribindo matriz.dat'
stop
end subroutine escribe_matriz
```

Exemplo: lectura de arquivo en formato *write.table* de R

```
program le_arquivo
character(10),allocatable :: entrada(:),saida(:)  !vectores de strings
real,allocatable :: a(:,:)   !matriz cos datos
nf=2;ne=4
allocate(entrada(ne),a(nf,ne),saida(nf))
open(1,file='arquivo.dat',status='old',err=1)
read (1,*) (entrada(i),i=1,ne)  ! descarta a última cadea de caracteres ('Saida')
do i=1,nf
      read (1,*) j,(a(i,j),j=1,ne),saida(i)
end do
close(1)
print *,'a='
do i=1,nf
      print *,(a(i,j),j=1,ne)
end do
print *,'saida=',(saida(i),i=1,nf)
deallocate(entrada,a,saida)
stop
1 print *,'erro en open abrindo arquivo.dat'
stop
end program le_arquivo
```

Pre-define o nº de filas e de columnas

arquivo.dat: columnas separadas por tabuladores

	E1	E2	E3	E4	Saida
1	0.25	0.33	1.23	-0.51	Branco
2	-0.34	1.3E5	0.22	4.3	Negro

Le arquivo en formato R sen pre-definir o nº de filas e columnas

```
program le_arquivo_formato_R
character(100) :: fich='arquivo.dat'
real,allocatable :: x(:,:)
character(100),allocatable :: y(:),none(:)
call conta_filas_columnas(fich,nf,nc)
open(1,file=fich,status='old',err=1)
read (1,*) (nome(i),i=1,nc)
do i=1,nf
      read (1,*) t,(x(i,j),j=1,nc),y(i)
end do
close(1)
do i=1,nc
      print '(a10,$)',nome(i)
end do
print *,'saída')
do i=1,nf
      do j=1,nf
            print '(f10.2,$)',x(i,j)
      end do
      print *,y(i)
end do
deallocate(x,y)
stop
1 print *,fich,'non atopado';stop
end program le_arquivo_formato_R
```

Le datos do arquivo

Mostra datos por pantalla

```
subroutine conta_filas_columnas(fich,nf,nc)
character(100),intent(in) fich
integer,intent(out) :: nf,nc
logical :: dato
open(1,file=fich,status='old',err=1)
nf=1;nc=1;dato=.false.
read (1,*) s
do i=1,len_trim(s)
      if(dato) then
            if(s(i:i)==' ') dato=.false.
      else
            if(s(i:i)/=' ') then
                  dato=.true.; nc=nc+1
            end if
      end if
end do
nc=nc+1
do
      read (1,*,end=2);nf=nf+1
end do
close(1)
return
1 print *,nf,'non atopado';stop
end subroutine conta
```

Conta columnas na cabeceira

Conta as filas de datos

Mary Allen Wilkes (1937)

- Creadora (1965) del primer ordenador personal para trabajo en casa
- Desarrolladora de sistemas operativos e linguaxe ensamblador (LAP6)
- Traballou no MIT e na Univ. Washington

Tipos de datos definidos por usuario

- Podemos definir tipos de datos agregados.

```
type nome
    tipo1 :: var1
    tipoN :: varN
end type nome
```

- Defínense en módulos para poder usarse en varios subprogramas.

```
module modulo_persoa
type persoa
      character(10) :: nome
      integer :: idade
end type persoa
end module modulo_persoa
```

```
program exemplo_tipos
use modulo_persoa
type(persoa) :: p=persoa('carlos',14)
call mostra(p)
end program exemplo_tipos

subroutine mostra(p)
use modulo_persoa
type(persoa),intent(in) :: p
print *,'nome=',p%nome,'idade=',p%idade
end subroutine mostra
```

- Declaración: *type(nome) :: x=nome(y,z)*
- Acceso a campos: *x%y,x%z*

Sobrecarga de operadores

- Define un operador a medida para un dato agregado.
- Bloque *interface operator*.
- Función que define o operador aritmético ou relacional.

```
program principal
use exemplo
type(persoa) :: x=persoa('alba',65.2,1.84)
type(persoa) :: y=persoa('clara',70.1,1.98)
if(x>y) then
      print *,x%nome,'>',y%nome
else
      print *,x%nome,'<=',y%nome
end if
stop
end program principal
```

```
module exemplo
      type persoa
            character(10) :: nome
            real :: peso,altura
      end type persoa
      interface operator (>)
            module procedure maior
      end interface
contains
      logical function maior(x,y)
      type(persoa),intent(in) :: x,y
      if(x%peso/x%altura > y%peso/y%altura) then
            maior=.true.
      else
            maior=.false.
      end if
      end function maior
end module
```

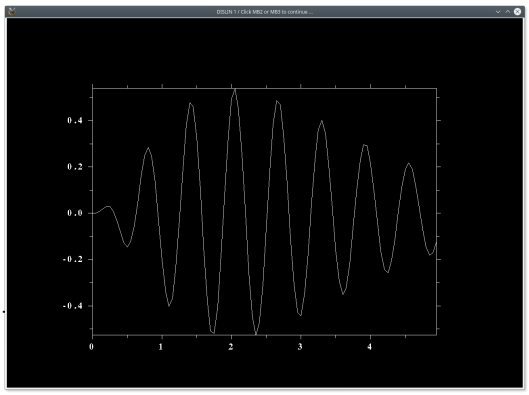

Representación gráfica (I): curva 2D (función de 1 variábel)

- Libraría Dislin: https://www.dislin.de
- Tamén o podes descargar dende: http://ftp5.gwdg.de/pub/grafik/dislin/linux/i586_64/

Helmut Michels: **The data plotting software DISLIN : Version 11,** ISBN 9783868585179, Sinatura 1203 232 (Bibl. Fac. Matemáticas)

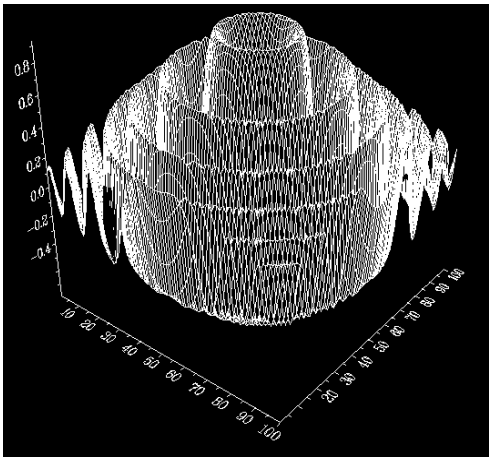
- Instala paquete *libmotif-dev*
- Descargar o arquivo .deb dende https://www.dislin.de/i586_64.html e executa como root: *dpkg -i arquivo.deb*
- *export LD_LIBRARY_PATH=/usr/local/dislin* (directorio donde está o arquivo *libdislin.so*)
- Compilación: *f95 -I/usr/local/dislin/gf curva2d.f90 -ldislin*
- Curva 2D: $y=f(x)=t^2e^{-t}sen\ 10t$: vectores x, y con coordenadas de puntos

```
program curva2d
use dislin
integer,parameter :: n=100
real,dimension(n) :: x,y
a=0;b=5;h=(b-a)/n;t=a
do i=1,n
      x(i)=t;y(i)=t**2*exp(-t)*sin(10*t)
      t=t+h
end do
call metafl('xwin')
call disini()
call qplot(x,y,n)
stop
end program curva2d
```



Representación gráfica (II): superficie 3D (función 2 variábeis)

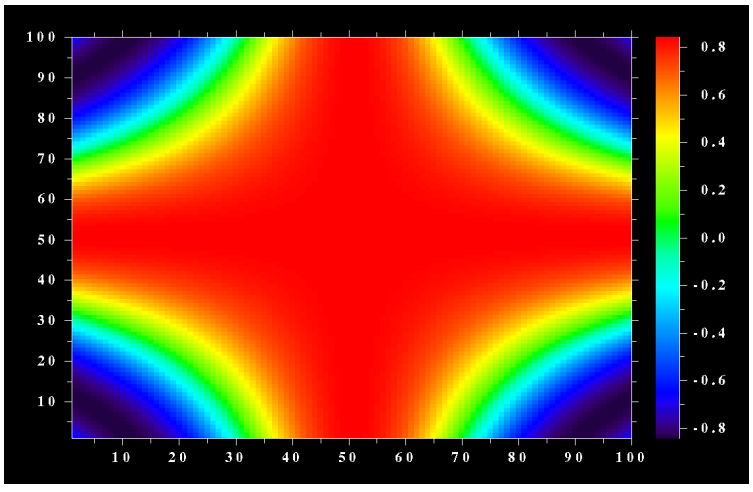
- Superficie 3D: matriz *z* con valores de función *z=f(x,y)*, *n* valores en cada dimensión.
- Exemplo: $z=f(x,y)=sen(5(x^2+y^2))exp(-(x^2+y^2)/4)$.



```
program superficie3d
use dislin
integer,parameter :: n=100
real,dimension(n) :: x,y
real,dimension(n,n) :: z
a=-2;b=2;h=(b-a)/n;t=a
do i=1,n
      x(i)=t;y(i)=t;t=t+h
end do
do i=1,n
      do j=1,n
            tx=x(i);ty=y(j);
          z(i,j)=sin(5*(tx**2+ty**2))*
              exp(-(tx**2+ty**2)/4)
      end do
end do
call metafl('xwin')
call disini()
call qplsur(z,n,n)
stop
end program superficie3d
```

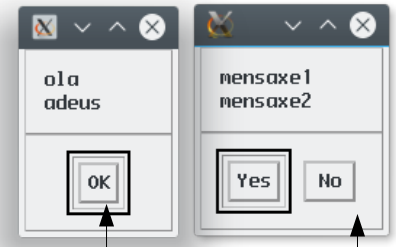

Representación gráfica (III): mapa de calor (función 2 variábeis)

- Mapa de calor: diagrama de cores que representan o valor dunha función *z=f(x,y)*, azul=valores baixos, vermello=valores altos.
- Exemplo: $z=f(x,y)=sen(5(x^2+y^2))exp(-(x^2+y^2)/4)$.



```
program mapa_calor
use dislin
integer,parameter :: n=100
real,dimension(n) :: x,y
real,dimension(n,n) :: z
a=-2;b=2;h=(b-a)/n;t=a
do i=1,n
      x(i)=t;y(i)=t;t=t+h
end do
do i=1,n
      do j=1,n
          tx=x(i);ty=y(j);
          z(i,j)=sin(cos(tx*ty))
      end do
end do
call metafl('xwin')
call disini()
call qplclr(z,n,n)
stop
end program mapa_calor
```

Interface gráfica de usuario: ventá emerxente con mensaxe e botón



export LD_LIBRARY_PATH=/usr/local/dislin
Compilación: *f95 -I/usr/local/dislin/gf curva2d.f90 -ldislin*

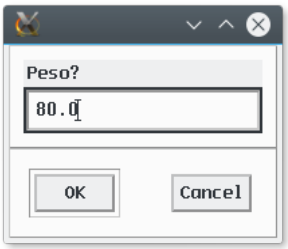
- Compoñentes gráficas (*widgets*).
- Comezando: ventás emerxentes.
- Mensaxe de varias liñas (separadas con |) e un botón OK.
- Mensaxe con dous botóns (Yes e No), podes comprobar que botón se pulsou.

```
program mensaxe
use dislin
call dwgmsg('ola|adeus')
stop
end program mensaxe
```

```
program boton
use dislin
call dwgbut('mensaxe1|mensaxe2', ival)
if(ival==0) then
      print *,'pulsaches non'
else
      print *,'pulsaches si'
end if
stop
end program boton
```


Ventá emerxente con entrada de texto/números

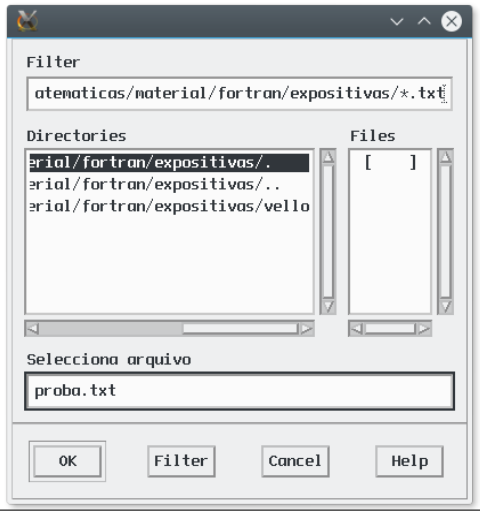
- Entrada de texto/números.
- A entrada almacénase como cadea de caracteres.
- Logo podes converter a número (enteiro, real).
- Botóns *OK* e *Cancel*: o programa pode comprobar se se cancelou ou se introduciu un dato.




```
program entrada_texto
use dislin
character(100) :: s='80.0'
integer :: estado
call dwgtxt('Peso?',s)
call dwgerr(estado)
if(estado==0) then
      read (s,*) p
      print *,'introduciches ',p
else
      print *,'cancelaches'
end if
stop
end program entrada_texto
```

Ventá emerxente con selector de arquivos

- Permite seleccionar un arquivo no directorio actual e navegar polos directorios
- Indica un tipo de arquivo e nome por defecto.
- Permite cancelar.




```
program selector_arquivo
use dislin
character(100) :: f='proba.txt'
integer :: estado
call dwgfil('Selecciona arquivo',f,'*.txt')
call dwgerr(estado)
if(estado==0) then
      print *,'seleccionaches ',f
else
      print *,'cancelaches'
end if
stop
end program selector_arquivo
```


Ventá emerxente con lista de opcións

- Mostra unha lista e permite seleccionar un dos seus elementos, indicando o elemento seleccionado por defecto.
- Permite cancelar.


```
program lista_opcions
use dislin
integer :: sel=2,estado
call dwglis('lista','Carlos|Luis|Alberto',sel)
call dwgerr(estado)
if(estado==0) then
      print *,'seleccionaches ',sel
else
      print *,'cancelaches'
end if
stop
end program lista_opcions
```

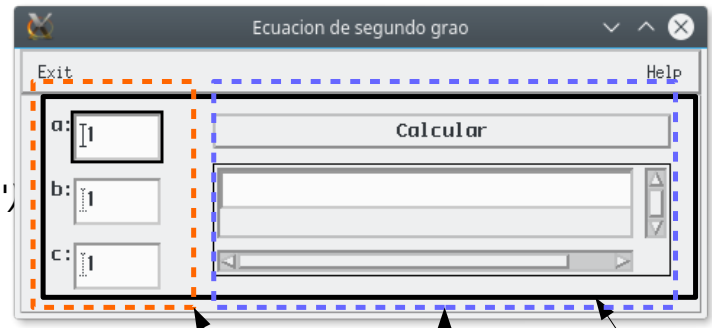
Interfaz de usuario con deseño complexo

- Subrutina *wgini(tipo,id)*: inicializa as compoñentes gráficas e crea un contedor de compoñentes, tipo=’vert’ ou ’hori’, segundo o aliñamento das compoñentes.

call wgini('vert',v)
call wgltxt(v, 'x: ', '1', 50,t1) ! Cadro de texto nº 1
call wgltxt(v,'x^0.3+x-1: ',s,50,t2) ! Cadro de texto nº 2 (debaixo do actual)

- Se uso ‘*hori*’ (‘*vert*’) en *wgini*, as compoñentes gráficas engádense de esquerda a dereita (de arriba a abaixo).
- Cada contedor ou compoñente ten un *id* (enteiro): *v*,*t1*,*t2*.
- Podes poñer un contedor horizontal dentro doutro vertical, e viceversa.
- Remata as compoñentes gráficas con *wgfin*.

Deseño da interfaz: contedores horizontais e verticais con compoñentes

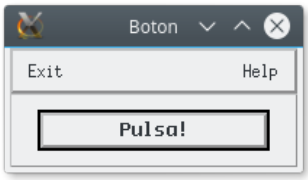


program ec2grao
use dislin
integer :: ip, ipv1, ipv2, iph
call swgtit ('Ecuacion de segundo grao')
call wgini('vert', ip) ! inicializa GUI
call wgbas(ip, 'hori', iph) ! horizontal
call swgwth(10) ! pon anchura
call wgbas(iph, 'vert', ipv1) ! vertical
call wgltxt(ipv1, 'a:', '1', 80, id_a) ! pon widget para ler texto
call wgltxt(ipv1, 'b:', '1', 80, id_b)
call wgltxt(ipv1, 'c:', '1', 80, id_c)
call swgwth(40) ! pon anchura
call wgbas(iph, 'vert', ipv2) ! contedor vertical
call wgpbut(ipv2,'Calcular', id_button) ! botón no GUI
call wgstxt(ipv2, 2, 1, id_txt) ! cadro para mostrar texto
call wgfin() ! mostra GUI
stop
end program ec2grao

Contedor vertical *ipv2*
Contedor horizontal *ip*
Contedor vertical *ipv1*

Execución de accións asociadas a compoñentes

- Cadros de texto, botóns: ao pulsar *Intro* (p.ex. cadro de texto) ou co rato (p.ex. botón), executar accións.
- Subprogramas retro-chamados (*callback*). Asocia subprograma con compoñente gráfica.
- *call swgcbk(id,subprograma)*: asocia o subprograma á pulsación da compoñente gráfica *id*.
- O subprograma debe ser declarado *external*.

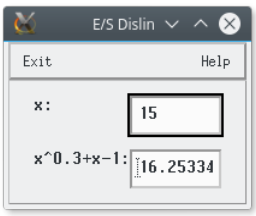



```
subroutine pulsa(b)
integer,intent(in) :: b
print *, 'has pulsado!'
return
end subroutine pulsa
```

```
program widget_boton
use dislin
integer :: v,b
external pulsa
call swgtit ('Boton')
call wgini('vert',v)
call wgpbut(v,'Pulsa!',b)
call swgcbk(b,pulsa)
call wgfin
stop
end program widget_boton
```


Exemplo: entrada e saída gráfica

- Libraría **Dislin** (non incluída en **Gfortran**).
- Cadros de texto para ler *x* e para mostrar $x^{0.3}+x-1$. Escribe o número *x* e pulsa *Intro*.




```
program entrada_saida_grafica
use dislin
character(100) :: s
integer :: v,t1,t2
external calcula
common t1,t2
call swgtit('E/S Dislin')
call wgini('vert',v)
call wgltxt(v, 'x: ', '1', 50,t1)
call swgcbk(t1,calcula)
write (s,*) x**0.3+x-1
call wgltxt(v,'x^0.3+x-1: ',s,50,t2)
call wgfin
stop
end program entrada_saida_grafica
```

```
subroutine calcula(id)
integer,intent(in) :: id
common t1,t2
character(len=50) :: s
call gwgflt(t1,x)
y=x**0.3+x-1
call swgflt(t2,y,5)
return
end subroutine calcula
```

Le no 1º cadro

Escribe no 2º cadro

Cadro de texto para ler x

Executase cando se pulsa Intro no 1º cadro de texto

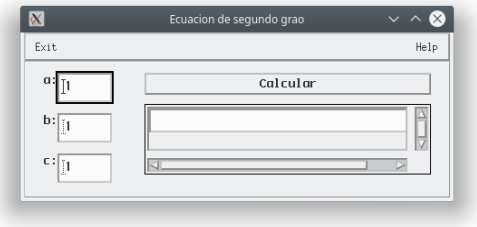
Cadro de texto para escribir $x^{0.3}+x-1$

Interfaz de usuario para o programa da ecuación de 2º grao

```
module comun ! para definir variables globais o programa
integer :: id_a, id_b, id_c, id_txt
end module comun
```

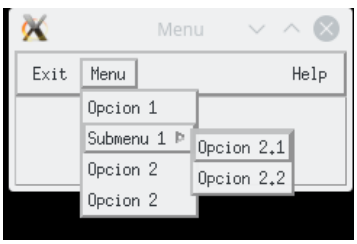
```
program ec2grao
use dislin
use comun
integer :: ip, ipv1, ipv2, iph, boton
external :: calcula
call swgtit('Ecuacion de segundo grao')
call wgini('vert', ip) ! inicializa GUI
call wgbas(ip, 'hori', iph) ! horizontal
call swgwth(10) ! pon anchura
call wgbas(iph, 'vert', ipv1) ! vertical
call wgltxt(ipv1, 'a:', '1', 80, id_a) ! pon widget para ler texto
call wgltxt(ipv1, 'b:', '1', 80, id_b)
call wgltxt(ipv1, 'c:', '1', 80, id_c)
call swgwth(40) ! pon anchura
call wgbas(iph, 'vert', ipv2) ! contedor vertical
call wgpbut(ipv2,'Calcular', boton) ! botón no GUI
call swgcbk(boton, calcula) ! conecta botón-subprograma
call wgstxt(ipv2, 2, 1, id_txt) ! cadro para mostrar texto
call wgfin ! mostra GUI
stop
end program ec2grao
```

```
subroutine calcula(id)
use comun
integer, intent(in) :: id
character(50) :: s, ecuacion
! lee o número da etiqueta de texto
call gwgflt(id_a, a)
call gwgflt(id_b, b)
call gwgflt(id_c, c)
! calcula a solución (carácter)
s=ecuacion(a,b,c)
! mostra a solución no cadro de texto
call swgtxt(id_txt,s)
return
end subroutine calcula
```



Compoñente gráfica menú

- Menú con ítems e submenús:
- *call wgpop(id1,'nome',id)*: crea menú ou submenú no contedor *id1*
- *call wgapp(m,'nome',id)*: engade elemento *id* ao menú ou submenú *m*.
- Cada elemento debe ter un subprograma que se execute cando é seleccionado.



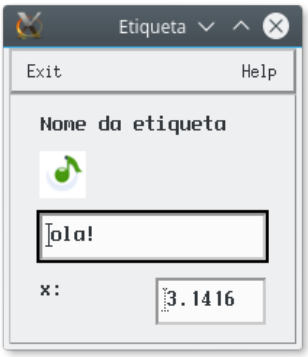

```
program widget_menu
use dislin
integer :: v,m,o1,sm1,o2,o21,o22,o3
call swgtit ('Menu')
call wgini('vert',v)
call wgpop(v,'Menu',m)
call wgapp(m,'Opcion 1',o1)
call wgpop(m,'Submenu 1',sm1)
call wgapp(sm1,'Opcion 2.1',o21)
call wgapp(sm1,'Opcion 2.2',o22)
call wgapp(m,'Opcion 2',o2)
call wgapp(m,'Opcion 2',o3)
call wgfin
stop
end program widget_menu
```

Etiqueta e cadro de texto

- Etiqueta: *call wglab(id1,'nome',id)*. Tamén con icono no canto de texto: *wgicon*
- Cadro de texto: *call wgtxt(id1,'texto',id)*
- Tamén con etiqueta: *wgltxt*

```
program widget_varios
use dislin
integer :: v,l,i,t1,t2
external le_t1
call swgtit ('Etiqueta e cadro de texto')
call wgini('vert',v)
call wglab(v,'Nome da etiqueta',l)
call wgicon(v,'Nome do icono',0,0,'icono.ico',i)
call wgtxt(v,'ola!',t1)
call swgcbk(t1,le_t1)
call wgltxt(v,'x: ','3.1416',50,t2)
call wgfin
stop
end program widget_varios
```

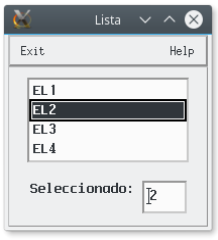
```
subroutine le_t1(t1)
integer,intent(in) :: t1
character(100) :: s
call gwgtxt(t1,s)
print *, 'escribiches ',s
return
end subroutine le_t1
```



Le o cadro de texto ao pulsar Intro e móstrao na terminal

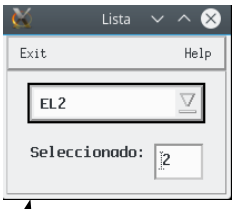
Lista e lista despregábel

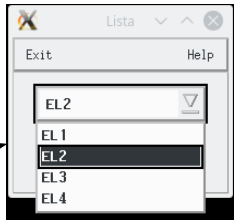
- *call wglis(id1,'E1|E2|E3|E4',sel,id)*: crea lista en *id1*.
- *call swgcbk(id,accion)*: acción ao seleccionar un elemento da lista.
- *call gwglis(id,i)*: i=elemento seleccionado.




```
program widget_lista
use dislin
integer :: v,l,sel=3,t
character(30) :: s
common t
external le_lista
call swgtit ('Lista')
call wgini('vert',v)
call wglis(v,'EL1|EL2|EL3|EL4',sel,l)
write (s,'(i0)') sel
call wgltxt(v,'Seleccionado: ',s,30,t)
call swgcbk(l,le_lista)
call wgfin
stop
end program widget_lista
```

```
subroutine le_lista(l)
integer,intent(in) :: l
common t
integer :: sel
character(20) :: s
call gwglis(l,sel)
write (s,'(i0)') sel
call swgtxt(t,s)
return
end subroutine le_lista
```

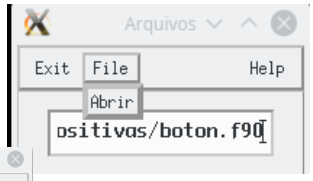


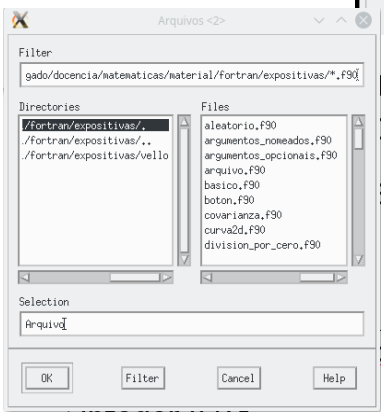


- Lista despregábel: *wdglis(...)*

Menú con selector de arquivos

- *call wgfil(id1,'Abrir','Arquivo','*.f90',id)*: crea un menú *File* cun elemento *Abrir*.
- Cando se pulsa, abre un selector de arquivos, mostrando so os arquivos coa extensión indicada (neste caso, **.f90*).
- Seleccionado o arquivo, aparece no cadro de texto

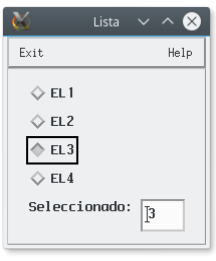





```
program widget_selec_arquivo
use dislin
integer :: v,f
call swgtit ('Arquivos')
call wgini('vert',v)
call wgfil(v,'Abrir','Arquivo','*.f90',f)
call wgfin
stop
end program widget_selec_arquivo
```


Lista de elementos excluintes

- *call wgbox(id1,'E1|E2|E3|E4', sel,id)*: crea lista con elementos que son botóns activábeis excluíntes (se activas un, desactívanse os demáis).
- *call swgcbk(id,accion)*: acción ao seleccionar un elemento da lista.
- *call gwgbox(id,i)*: i=elemento seleccionado.




```
subroutine le_lista(l)
integer,intent(in) :: l
common t
integer :: sel
character(20) :: s
call gwgbox(l,sel)
write (s,'(i0)') sel
call swgtxt(t,s)
return
end subroutine le_lista
```

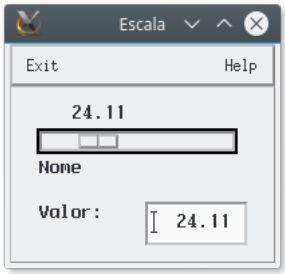
```
program widget_wgbox
use dislin
integer :: v,l,sel=3,t
character(30) :: s
common t
external le_lista
call swgtit ('Lista')
call wgini('vert',v)
call wgbox(v,'EL1|EL2|EL3|EL4',sel,l)
write (s,'(i0)') sel
call wgltxt(v,'Seleccionado: ',s,30,t)
call swgcbk(l,le_lista)
call wgfin
stop
end program widget_wgbox
```

Escala

- Regra horizontal na que podes seleccionar un valor.
- call wgscl(id1,'nome',vmin,vmax,vdef,ndix,id): valores mínimo, máximo e por defecto (reais), nº díxitos.

```
program widget_scale
use dislin
integer :: v,sc,t
character(10) :: s
common t
external le_scale
call swgtit ('Escala')
call wgini('vert',v)
call wgscl(v,'Nome',0.,100.,75.0,2,sc)
call gwgscl(sc,sel)
write (s,'(f7.2)') sel
call wgltxt(v,'Valor: ',s,50,t)
call swgcbk(sc,le_scale)
call wgfin
stop
end program widget_scale
```

```
subroutine le_scale(l)
integer,intent(in) :: l
common t
character(20) :: s
call gwgscl(l,sel)
write (s,'(f7.2)') sel
call swgtxt(t,s)
return
end subroutine le_scale
```

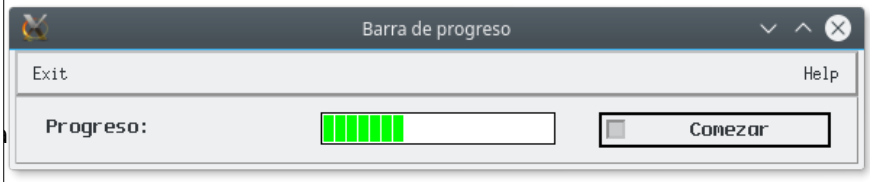


Barra de progreso

- Regra horizontal que indica o progreso dun cálculo (0-100%)
- Moi útil para cálculos numéricos lentos.
- *call wgpbar(id1,ini,fin,paso,id)*
- Retro-subprograma, asociado a un botón de inicio, que actualiza o progreso da barra.

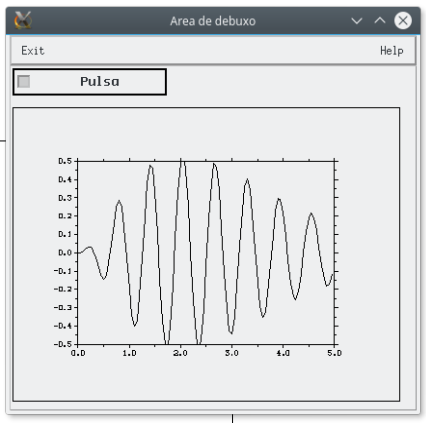
```
program widget_progress_bar
use dislin
integer :: h,l,pb,b,n=20
common pb,n,b,l2
external actualiza_pb
call swgtit ('Barra de progreso')
call wgini('hori',h)
call wglab(h,'Progreso:',l)
call wgpbar(h,0.,100.,100./n,pb)
call wgbut(h,'Comezar',0,b)
call swgcbk(b,actualiza_pb)
call wgfin
stop
end program widget_progress_bar
```

```
subroutine actualiza_pb(id)
integer,intent(in) :: id
common pb,n,l2
do i=1,n
      call sleep(1)
      call swgval(pb,100.*i/n)
end do
call swgbut(id,0)
return
end subroutine actualiza_pb
```



Área de debuxo (*draw*)

- *call wgdraw(id1,id)*: a retro-subrutina do botón representa unha función na compoñente gráfica.
- Fixa o seu tamano.




```
program widget_draw
use dislin
integer :: v,b,dr
common dr
external representa
call swgtit ('Area de debuxo')
call wgini('form',v)
call wgbut(v,'Pulsa',0,b)
call swgcbk(b,representa)
call swgpos(0,40)
call swgsiz(400,300)
call wgdraw(v,dr)
call wgfin
stop
end program widget_draw
```

```
subroutine representa(id)
integer,intent(in) :: id
integer,parameter :: n=100
real :: x(n),y(n)
common dr
call metafl('cons')
call setxid(dr,'widget')
call scrmod('reverse')
call disini
a=0;b=5;h=(b-a)/n;t=a
call graf(a,b,a,1.,-0.5,0.5,-0.5,0.1)
do i=1,n
   x(i)=t;y(i)=t**2*exp(-t)*sin(10*t);t=t+h
end do
call curve(x,y,n)
call disfin
return
end subroutine representa
```


Creación dunha libraría en Fortran

- Libraría: código máquina de moitos subprogramas (en Fortran) empaquetado nun arquivo
- Non contén o código fonte (non se pode depurar ou ver como opera).
- Proporciónanos subprogramas que permiten facer operacións (p.ex., determinantes, resolución de sistemas de ecuacións, ...)
- Para usar unha libraría, hai que enlazar (*link*) o noso programa coa libraría
- O compilador *f95* xa usa algunhas librarías incluídas co compilador (p.ex. funcións intrínsecas)

Librarías estáticas

- Ficheiro con extensión *.a*: *libproba.a*. Usado só na compilación:

  f95 -L. programa.f90 -lproba

- O código máquina da libraría empótrase no programa executábel.
- Non se necesita nada para a execución: *a.out*
- Programa executábel de tamano grande (carga en memoria RAM máis lenta).
- Listado de arquivos *.o* contidos en libraría *.a*: *ar tv libproba.a (ou nm X.a*, ou *readelf -s X.a)*

Librarías dinámicas

- Ficheiro con extensión *.so*: *libproba.so*
- O código máquina da libraría úsase na compilación e na execución:

  f95 -L. programa.f90 -lproba

  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.

  a.out

- Executábel máis pequeno (carga máis rápida en memoria).
- Listado de arquivos *.o*: *nm X.so* ou *readelf -s X.so*.
- Non se pode executar sen que a variábel de entorno indique a ruta onde se atopan o(s) arquivo(s) **.so**
- Ver a variábel: *echo $LD_LIBRARY_PATH*

Creación dunha libraría estática en Fortran (I)

- Edición/compilación/depuración de código
- Compilación sen enlazado dos arquivos que conteñen subprogramas (non incluir o programa principal). Crea só arquivos obxecto (.o), non crea executábel:

  f95 -c arquivo.f90 (crea *arquivo.o*)

- Empaquetado de arquivos *.o* e creación de libraría (***libXX.a***, debe comezar por *lib*)

  *ar qv libXX.a *.o*

- Para ver arquivos .o empaquetados: *ar tv libXX.a*

Creación dunha libraría estática en Fortran (II)

- É recomendábel meter unha función ou subrutina en cada *arquivo .o*
  - Alomenos os subprogramas que van ser usados dende fóra da libraría).
- Así coinciden os nomes dos arquivos *.o* empaquetados en *libXX.a* cos nomes dos subprogramas que proporciona a libraría.
- Así podemos saber, co comando *ar tv libXX.a*, os subprogramas que contén esa libraría.

Exemplo de creación e uso dunha libraría estática

- Libraría *libestatistica.a* que proporcione subprogramas para calcula-la media, desviación, mediana e ordear un vector
- Arquivos media.f90, desviacion.f90, mediana.f90, ordea.f90, principal.f90 (descárgaos)
- Comando de compilación (sen enlazado):

  f95 -c media.f90 desviacion.f90 mediana.f90 ordea.f90
- Empaquetado da libraría: *ar qv libestatistica.a *.o*
- Listado de arquivos *.o*: *ar tv libestatistica.a* (ou *readelf -s X.a*)
- Para enlazar o programa *principal.f90* coa libraría:

  f95 -L. principal.f90 -lestatistica
- Execución: *a.out*

Exemplo de creación e uso dunha libraría dinámica

- Libraría *libestatistica.so*. Mesmos arquivos *.f90* de antes.
- Comando de compilación (sen enlazado):

  f95 -fpic -c media.f90 desviacion.f90 mediana.f90 ordea.f90
- Empaquetado: *f95 -shared -o libestatistica.so *.o*
- Listado de arquivos *.o*: *nm libestatistica.so* ou *readelf -s X.so*
- Uso da libraría dende programa *principal.f90*:

  f95 -L. principal.f90 -lestatistica
- Engade a ruta de libraría ao *LD_LIBRARY_PATH*:

  export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:.
- Execución: *a.out*

media.f90 e desviacion.*f90*

```
real function media(x, n)
   real, intent(in) ::  x(n)
   integer, intent(in) :: n
   media=0
   do i=1, n
      media=media+x(i)
   end do
   media=media/n
   return
   end function media
```

```
function desviacion(x, n, media)
real,intent(in):: x(n)
integer, intent(in) :: n
real, intent(in) :: media
desviacion=0
do i=1, n
    y = x(i) - media
    desviacion = desviacion + y*y
end do
desviacion = sqrt(desviacion/n)
return
end function desviacion
```


mediana.f90

```
real function mediana(x, n)
   real, intent(in) :: x(n)
   integer, intent(in) :: n
   call ordear(x, n)
   if(mod(n, 2) == 0) then
      mediana = (x(n/2)+x(n/2+1))/2
   else
      mediana = x(n/2+1)
   endif
   return
   end function mediana
```

ordea.f90

```
subroutine ordea(x, n)
real, intent(inout) :: x(n)
integer, intent(in) :: n
do i = 1, n
    vmin = x(i); imin = i
    do j = i + 1, n
        if(x(j) < vmin) then
            vmin = x(j); imin = j
        end if
    end do
    x(imin) = x(i); x(i) = vmin
end do
return
end subroutine ordear
```


principal.f90

```
program principal
real,allocatable :: x(:)
real :: media, mediana, m
print *, "introduce n: "
read *, n
allocate(x(n))
print *, "introduce x: "
read *, x
y = media(x, n)
d = desviacion(x, n, y)
m = mediana(x, n)
call ordear(x, n)
print *, "media= ", y, "desviacion= ", d, "mediana= ", m
print *, "vector ordeado= ", x
end program principal
```

Uso dunha libraría feita por outros (I)

- **Instalación** da libraría como administrador ou usuario. Dúas alternativas:
  - Paquetes precompilados (binarios), co synaptic,apt-get,rpm,...
  - Compilación do código fonte: *configure*, *make*, *make install* (como administrador)
- Coñecer o directorio no que se atopa o(s) arquivo(s) **libXX.a** ou **libXX.so**
- Acceder á **documentación** da libraría, para saber que subprogramas ten, os seus argumentos, valores retornados, etc.

Uso dunha libraría feita por outros (II)

- Dende o noso programa, chamamos aos subprogramas da libraría (ver documentación), pasándolle os argumentos axeitados (en número e tipo) e usando os valores retornados
- Coñecido o directorio do arquivo **libXX.a** ou **libXX.so**:

f95 -Ldir programa.f90 -lXX

dir: directorio no que *f95* debe buscar *libXX.a/libXX.so*

-lXX: ***libXX.a*** é a librería que usará *programa.f90*

- Con librarías dinámicas:
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:dir
- Librarías libres en Fortran: http://www.fortran.com/tools.html
- Destacamos: Lapack, Linpack (http://www.netlib.org), Cernlib, Plplot, Libg2

Programación estruturada en Fortran
Exercicios clases interactivas

Semana 2

Traballo en clase

1. **Variábeis. Expresións aritméticas. Entrada/saída básicas**. Escribe un programa no editor Kate en Fortran chamado `expresions.f90` que lea un número real x por teclado e mostre por pantalla $3x-1$, $x^2+\sqrt{x-2}$ e $(\operatorname{sen} x - 3)/(\ln x + \mathrm{e}^x - 1)$. Gárdao no teu directorio persoal. Para compilar o programa, executa o comando:

   ```
   f95 expresions.f90
   ```

   Deste modo xérase o programa executable `a.out`. Para executalo, teclea `a.out` na terminal.

   ```
   program expresions
   print '("x? ",$)'
   read *, x

   print *, 'Os valores son: '
   print '("3x-1=",f6.3)',3*x-1
   print '("x^2+sqrt(x-2)=",f6.3)',x**2+sqrt(x-2)
   print '("(sin(x)-3)/(ln(x)+exp(x)-1)=",f6.3)',(sin(x)-3)/(log(x)+exp(x)-1)

   end program expresions
   ```

   Se queres que o executable se chame, por exemplo, `expresions`, hai que executar:

   ```
   f95 expresions.f90 -o expresions
   ```

   Finalmente, copia o programa `expresions.f90` ao directorio `/Z/rai/nome.apelidos`, onde `nome.apelidos` son o teu nome e apelidos, co comando:

   ```
   cp expresions.f90 /Z/rai/nome.apelidos
   ```

   Copia tamén o programa á memoria flash (se a tes):

   ```
   cp expresions.f90 /media/nome.apelidos/nome_memoria_flash
   ```

2. **Ecuación de 2^o grao. Sentenzas de selección**. Escribe un programa chamado `ec2grao.f90` que lea os coeficientes (reais) dunha ecuación de segundo grao $ax^2+bx+c=0$ e calcule as súas solucións segundo a fórmula:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \tag{1}$$

   Sexa $d=b^2-4ac$ o discriminante. Se $d<0$, entón hai dúas solucións complexas conxugadas $x = \frac{-b}{2a} \pm i\frac{\sqrt{-d}}{2a}$, onde i é a unidade imaxinaria $i=\sqrt{-1}$. Se $d=0$, entón hai dúas solucións reais iguais $x=\frac{-b}{2a}$. Se $d>0$ hai dúas solucións reais distintas $x=\frac{-b\pm\sqrt{d}}{2a}$. Se $a=0$ e $b\neq 0$, entón temos unha ecuación de 1^o grao con solución $x=-c/b$. Se $a=b=c=0$ todo $x\in\mathbb{R}$ é solución, e se $a=b=0, c\neq 0$ non hai solución. Para comprobar que o programa funciona correctamente, proba cos exemplos da táboa 1:

   ```
   program ec2grao
   print '("a,b,c? ",$)';read *, a,b,c
   if(0 == a) then
       if(0 == b) then
           if(0 == c) then
   ```

a	b	c	Nº solucións	Solucións
1	1	1	2 complexas	$x = -\frac{1}{2} \pm i\frac{\sqrt{3}}{2}$
1	-2	1	2 reais iguais	$x = 1$
1	0	-1	2 reais distintas	$x = \pm 1$
0	1	-1	ec. 1^o grao	$x = 1$
0	0	0	∞	$x = \mathbb{R}$
0	0	1	0	$x = \varnothing$

Cuadro 1: Valores dos coeficientes a, b, c e das solucións que debe obter o programa.

```
            print *, 'infinitas solucions reais: x=IR'
        else
            print *, 'non existen solucions'
        endif
    else
        print *, 'solucion= ', -c/b
    endif
else
    d=b*b-4*a*c; a2=2*a; rd=sqrt(abs(d)); u=-b/a2; v=rd/a2
    if(d < 0) then
        print *,'2 solucions complexas conxugadas: x=',y,'+/-I*',abs(v)
    else if(0 == d) then
        print *,'2 solucions reais iguais: x=',u
    else
        print *,'2 solucions reais: x=',u-v,u-v
    endif
endif
end program ec2grao
```

Versión usando a libraría gráfica Dislin (**https://www.dislin.de**) para a entrada e saída de datos mediante compoñentes gráficas (cadros e etiquetas de texto, botóns). Podes descargar este programa (`ec2grao_dislin.f90`) desde este **enlace**.

```
module comun ! para definir variables globais o programa
integer :: id_a, id_b, id_c, id_txt
end module comun

!-----------------------------------------------
program menu
use dislin
use comun
integer :: ip, ipv1, ipv2, iph, boton
external calcula

call swgtit ('Ecuacion de segundo grao')
call wgini('vert', ip) ! inicializa GUI
call wgbas(ip, 'hori', iph) ! pon contedor horizontal
call swgwth(10) ! pon anchura
call wgbas(iph, 'vert', ipv1) ! pon contedor vertical
call wgltxt(ipv1, 'a:', '1', 80, id_a) ! pon widget para ler texto
call wgltxt(ipv1, 'b:', '1', 80, id_b)
call wgltxt(ipv1, 'c:', '1', 80, id_c)
call swgwth(40) ! pon anchura
call wgbas(iph, 'vert', ipv2) ! pon contedor vertical
call wgpbut(ipv2,'Calcular', boton) ! pon un boton no GUI
```

```
call swgcbk(boton, calcula) ! conecta a pulsacion do boton con un subprograma
call wgstxt(ipv2, 2, 1, id_txt) ! cadro para mostrar texto
call wgfin ! para mostrar GUI

end program menu

!-----------------------------------------------------
subroutine calcula(id)
use comun
integer, intent(in) :: id
character(50) :: s

call gwgflt(id_a, a) ! lee o numero flotante da etiqueta de texto
call gwgflt(id_b, b)
call gwgflt(id_c, c)

! inserta a solucion nunha cadea de caracteres
if(a==0) then
    if(b==0) then
        if(c==0) then
            s='Todo IR e solucion'
        else
            s='Non hai solucion'
        end if
    else
        write(s,'(a,f6.3)') 'x=', -c/b
    end if
else
    d=b*b-4*a*c;a2=2*a;rd=sqrt(abs(d));
    if(d<0) then
        write(s,'(a,f6.3,a,f6.3)') 'x=', -b/a2,' +- I*', rd/abs(a2)
    else if (d>0) then
        write(s,'(a,f6.3,2a,f6.3)') 'x1=', (-b+rd)/a2,char(10),'x2=', (-b-rd)/a2
    else
    write(s,'(a,f6.3)') 'x1=x2=', -b/a2
    end if
end if
call swgtxt(id_txt,s)  ! mostra a solucion no cadro de texto
end subroutine calcula
```

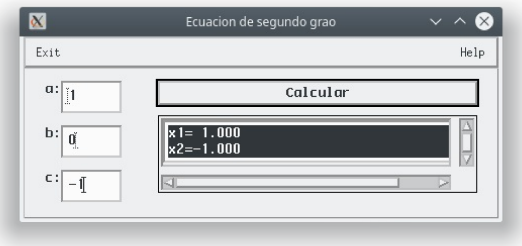


Figura 1: Programa en Fortran para resolver unha ecuación de 2^o grao usando unha interface gráfica creada coa libraría Dislin.

Debes compilar e executar este programa dende a terminal cos seguintes comandos:

```
f95 -I/usr/local/dislin/gf ec2grao_dislin.f90 -l dislin
```

```
export LD_LIBRARY_PATH=/usr/local/dislin
a.out
```

e debes obter a interface gráfica que se mostra na figura 1.

3. **Sentenza de iteración definida. Acumulador. Polinomio de Tchevyshev. Constantes con nome**. O valor do polinomio de Tchebyshev $T_n(x)$ de grao n nun punto x pódese calcular usando a seguinte expresión:

$$T_n(x) = 2^{n-1} \prod_{k=1}^{n} \left[x - \cos\left(\frac{(2k-1)\pi}{2n} \right) \right] \tag{2}$$

Escribe un programa en Fortran chamado `tchevyshev.f90` que lea n e x por teclado e calcule o valor de $T_n(x)$. Como $T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$, proba con $n = 9, x = 2$ e debes obter $T_9(2) = 70225{,}9531$.

```
program tchevyshev

real, parameter :: pi = 3.141592

print '("n,x? ",$)';read *,n,x

tnx = 2**(n - 1); t=pi/(2*n)
do k = 1, n
    tnx = tnx*(x - cos((2*k - 1)*t))
end do
print *, "t_n(x)= ", tnx


end program tchevyshev
```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa que calcule a distancia entre un punto $\mathbf{v} = (x_0, y_0, z_0)$ e un plano π dado pola ecuación $ax + by + cz + d = 0$. Esta ecuación tamén se pode escribir como $\mathbf{w}^T\mathbf{x} + d = 0$, onde $\mathbf{w} = (a, b, c)$ é o vector director do plano, $\mathbf{w}^T$ é o trasposto de $\mathbf{w}$, perpendicular ao plano π, e $\mathbf{x} = (x, y, z)$ é un punto pertencente ao plano. A distancia entre $\mathbf{v}$ e π pódese calcular como:

$$D(\mathbf{v}, \pi) = \frac{|ax_0 + by_0 + cz_0 + d|}{|\mathbf{w}|} \tag{3}$$

O valor absoluto é coa función `abs(...)`; ademáis, $|\mathbf{w}| = \sqrt{a^2 + b^2 + c^2}$. O programa debe ler por teclado os valores $a, b, c, d, x_0, y_0, z_0$.

2. Escribe un programa que calcule a posición $x(t)$, velocidade $v(t)$ e aceleración $a(t)$ dun móvil en movemento armónico, dado por:

$$x(t) = b\,\mathrm{sen}(\omega t + \theta) \tag{4}$$
$$v(t) = b\omega\cos(\omega t + \theta) \tag{5}$$
$$a(t) = -b\omega^2\,\mathrm{sen}(\omega t + \theta) \tag{6}$$

sendo ω= 0.1 radiáns/s, $\theta = \pi/2$ radiáns, $b = 2.5\ m$ para tempos $0 \le t \le 100$ segs. separados 1 seg. entre si.

3. Escribe un programa que lea n números $x_1, \ldots, x_n$ por teclado e calcule a súa media e o seu produto (non uses arrais):

$$Media = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad Produto = \prod_{i=1}^{n} x_i \tag{7}$$

Semana 4

Traballo en clase

1. **Sumatorio dobre. Vectores dinámicos**. Escribe un programa chamado `sumatorio.f90` que lea por teclado un número enteiro n e logo dous vectores n-dimensionais $\mathbf{v}$ e $\mathbf{w}$, reservados dinámicamente. O programa debe calcular, usando vectores:

$$s = \sum_{i=1}^{n} \sum_{j=1}^{i} v_i w_j \tag{8}$$

Proba con $n = 3$, $\mathbf{v} = (1, 2, 1)$ e $\mathbf{w} = (-1, 0, 1)$ e tes que obter $s = -3$.

```
program sumatorio_dobre

real,allocatable :: v(:),w(:)

print '(a,$)', 'n? ';read *, n
allocate(v(n),w(n))
print '("v? ",$)';read *,v
print '("w? ",$)';read *,w

suma=0
do i=1,n
    do j=1,i
        suma=suma+v(i)*w(j)
    end do
end do

! alternativa simple vectorizada
!suma=0
!do i=1,n
!    suma = suma + v(i)*sum(w(1:i))
!end do

! alternativa optima
!suma=0;s=0
!do i=1,n
! s=s+w(i);suma=suma+v(i)*s
!end do

print*, "O resultado e: ", suma

deallocate(v, w)

end program sumatorio_dobre
```

2. **Produto vector-matriz-vector. Matrices dinámicas**. Escribe un programa chamado `producto.f90` que lea por teclado un número enteiro n, dous vectores $\mathbf{v}$ e $\mathbf{w}$ e unha matriz $\mathbf{A}$ de orde n, e calcule o produto

$$p = \mathbf{v}^T \mathbf{A} \mathbf{w} = [v_1 \dots v_n] \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix}$$

onde $\mathbf{v}^T$ denota o vector trasposto de $\mathbf{v}$ (os vectores considéranse por defecto vectores columna). Proba con $n = 3$, $\mathbf{v} = (1, 2, 1), \mathbf{w} = (-1, 0, 1)$ e $\mathbf{a}$ =(1 2 3; 4 5 6; 7 8 9) (filas separadas por ;) e tes que obter $p = 8$.

```
program producto

real,allocatable :: v(:),w(:),a(:,:),p(:)

print '("n? ",$)'; read *,n
allocate(a(n,n), v(n), w(n), p(n))

print '("v? ",$)'; read *,v
print '("w? ",$)'; read *,w
print *,'a? '
do i=1,n
    read *, (a(i,j),j=1,n)
end do

!p=vA
do i=1,n
    p(i)=0
    do j=1,n
        p(i)=p(i)+v(i)*a(j,i)
    end do
end do

!r=pw
r=0
do i=1,n
    r=r+p(i)*w(i)
end do

print*, "O resultado e: ", r
deallocate(a, v, w, p)


end program producto
```

Versión optimizada, que non necesita o vector **p**:

```
program producto

real,allocatable :: a(:,:)
real,allocatable :: v(:),w(:)

print '("n? ",$)'; read *,n
allocate(a(n,n),v(n),w(n))
print '("v? ",$)'; read *,v
print '("w? ",$)'; read *,w
print 'a? '
do i=1,n
    read *, (a(i,j),j=1,n)
end do

r = 0
do i=1,n
    s = 0
    do j=1,n
        s = s + v(i)*a(j,i)
```

```
    end do
    r = r + s*w(i)
end do
print*, "O resultado e: ", r

deallocate(a,v,w)

end program producto
```

Versión usando funcións intrínsecas `dot_product` e `matmul` de Fortran:

```
program producto
real,allocatable :: v(:),w(:),a(:,:)

print '("n? ",$)'; read *,n
allocate(v(n),w(n),a(n,n))

print '("v? ",$)'; read *,v
print '("w? ",$)'; read *,w
print *,'a? '
do i=1,n
    read *,(a(i,j),j=1,n)
end do
print *, "vAw'=",dot_product(v,matmul(a,w))
! print *, "vAw'=",dot_product(matmul(v,a),w)   ! alternativa
deallocate(v,w,a)

end program producto
```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa que lea un número enteiro n e dous vectores $\mathbf{v}$ e $\mathbf{w}$ n-dimensionais con valores reais (usa vectores reservados dinámicamente). O programa debe calcula-lo produto escalar (ou interior) de ambos:

$$\mathbf{v}^T\mathbf{w} = \sum_{i=1}^{n} v_i w_i \tag{9}$$

2. Escribe un programa que lea un vector estático $\mathbf{v} = (v_1, \ldots, v_5)$ con 5 valores reais. O programa debe calcula-la norma (módulo) de $\mathbf{v}$:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{n} v_i^2} \tag{10}$$

   NOTA: a raíz cadrada dun valor `x` calcúlase en Fortran coa función `sqrt(x)`.

3. Escribe un programa que lea dous vectores $\mathbf{x}$ e $\mathbf{y}$ de dimensión n por teclado (usa vectores dinámicos) e calculen a súa distancia $|\mathbf{x} - \mathbf{y}|$ definida como:

$$|\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{11}$$

4. Escribe un programa que lea por teclado un número enteiro n e un vector $\mathbf{v}$ de dimensión n (usar vectores reservados dinámicamente), e calcule o vector transformado $\mathbf{w}$, tamén de dimensión n, definido por:

$$w_i = \sum_{j=1}^{i} v_j, \quad i = 1, \ldots, n \tag{12}$$

Semana 6

Traballo en clase

1. **Análise dunha matriz. Variábeis lóxicas. Bucles nomeados. Sentenza `exit`. Vectorización. Función `all`.** Escribe un programa chamado `matriz.f90` que lea por teclado un número enteiro n e unha matriz $\mathbf{A}$ cadrada de orde n e calcule:

   - A súa traza (suma dos elementos da diagonal principal), definida pola ecuación:

   $$tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} \tag{13}$$

   - A suma dos elementos do seu triángulo superior (sen a diagonal).
   - Determine se a matriz é simétrica, é decir, se $a_{ij} = a_{ji}$ para $i, j = 1, \ldots, n$.

```
program matriz
integer,allocatable :: a(:,:)
logical :: simetrica

print '("n? ",$)';read *, n
allocate(a(n,n))
print *,"a?"
do i=1,n
    read *, (a(i,j),j=1,n)
end do

!Calculo da traza menos eficiente
! m = 0
! do i=1,n
!    do j=1, n
!       if(i==j) m = m + a(i,j)
!    end do
! end do

! calculo mais eficiente
m = 0
do i=1,n
    m = m + a(i,i)
end do
print *, "traza=", m

!suma do triangulo superior menos eficiente
!m = 0
!do i = 1, n
!   do j = 1, n
!      if(j > i) m = m + a(i, j)
!   end do
!end do
!print *, "sts=", m

!suma do triangulo superior mais eficiente
!m = 0
!do i = 1, n
```

```
!    do j = i + 1, n
!       m = m + a(i, j)
!    end do
!end do
!print *, "sts=",m

!suma do triangulo superior ainda mais eficiente con funcion sum()
m = 0
do i=1,n
   m = m + sum(a(i,i+1:))
end do
print *, "sts= ", s

! E a matriz simetrica
simetrica=.true.
filas: do i=1,n
    do j=i+1, n
        if(a(i,j)/= a(j,i)) then
            simetrica=.false.
            exit filas
        end if
    end do
end do filas
if(simetrica) then
    print *, 'simetrica'
else
    print *, 'non simetrica'
end if

! forma mellor: transpose() para transpor e all() para comprobar igualdade
!if(all(a==transpose(a))) then
! print *,'simetrica'
!else
! print *,'non simetrica'
!end if

deallocate(a)
end program matriz
```

2. **Mínimo común múltiplo de dous números enteiros. Vectores estáticos. Iteración indefinida. Subrutina. Función externa. Resto da división de dous números enteiros. Paso de vector como argumento**. Escribe un programa chamado `mcm.f90` que presente na pantalla o mínimo común múltiplo (mcm) de dous números enteiros positivos. O mcm calcúlase como o produto dos factores primos de ambos números, procedendo da seguinte maneira: se un factor primo está presente nunha das factorizacións e non na outra, inclúese no cálculo do mcm; se un factor primo está presente nas dúas factorizacións, tómase aquel que ten un expoñente maior. Usa unha función `mcm(x,y)` para calcular o mínimo común múltiplo de dous números `x` e `y`, e unha subrutina `factores(...)` para descompoñer un número en factores primos.

```
! Proba con x = 120 e y = 252
! factores de 120= 2^3 * 3^1 * 5^1
! factores de 252= 2^2 * 3^2 * 7
! Factores comuns= 2^3 * 3^2 * 5^1 * 7^1
! O mcm e 2520
program principal
integer :: x,y
print '("x,y? ",$)';read *,x,y
m=mcm(x,y)
```

```
print '("mcm= ",i0)',m
end program principal


!-----------------------------------------
function mcm(x,y)
integer,intent(in) :: x,y
integer :: bx(100),ex(100),by(100),ey(100)
call factores(x,bx,ex,nx)
call factores(y,by,ey,ny)
mcm=x
do i=1,ny
  k=by(i);l=ey(i)
  do j=1,nx
    if(k==bx(j)) exit
  end do
  if(j<=nx) then
    if(l>ex(j)) mcm=mcm*k**(l-ex(j))
  else
    mcm=mcm*k**l
  end if
end do
end function mcm

!-----------------------------------------
subroutine factores(x,b,e,nf)
integer,intent(in) :: x
integer,intent(out) :: b(100),e(100),nf
k=2;nf=0;m=x
do
  if(mod(m,k)==0) then
    nf=nf+1;b(nf)=k;e(nf)=1;m=m/k
    do while(mod(m,k)==0)
      e(nf)=e(nf)+1;m=m/k
    end do
  end if
  if(m==1) exit
  k=k+1
end do
print '("factores de ",i0,"= ",$)',x
do i=1,nf
  print '(i0,"^",i0," ",$)',b(i),e(i)
end do
print *,''
end subroutine factores
```

3. **Paso de vector a unha subrutina usando interfaces.**

```
program subrutina_vector
integer,allocatable :: v(:)
interface
    subroutine sub(v)
        integer,intent(out) :: v(:)
    end subroutine sub
end interface
print '("n? ",$)';read *,n
allocate(v(n))
```

```
call sub(v)
print *,'v=',v
deallocate(v)

end program subrutina_vector
!-------------------------------
subroutine sub(v)
integer,intent(out) :: v(:)
n=size(v)
forall(i=1:n) v(i)=i*i
end subroutine sub
```

4. **Paso de matriz a unha subrutina usando interfaces.**

```
program subrutina_matriz
integer,allocatable :: a(:,:)
interface
    subroutine sub(a)
        integer,intent(out) :: a(:,:)
    end subroutine sub
end interface
print '("n,m? ",$)';read *,n,m
allocate(a(n,m))
call sub(a)
print *,'a='
do i=1,n
    print *,(a(i,j),j=1,m)
end do
deallocate(a)

end program subrutina_matriz
!-------------------------------
subroutine sub(a)
integer,intent(out) :: a(:,:)
n=size(a,1);m=size(a,2)
forall(i=1:n,j=1:m) a(i,j)=i**2*j
end subroutine sub
```

5. **Paso de vector a unha función usando interfaces.**

```
program funcion_vector
integer,allocatable :: v(:)
integer :: s
interface
    integer function fun(v) result(s)
        integer,intent(out) :: v(:)
    end function fun
end interface
print '("n? ",$)';read *,n
allocate(v(n))
s=fun(v)
print *,'v=',v,'s=',s
deallocate(v)

end program funcion_vector
!-------------------------------
integer function fun(v) result(s)
integer,intent(out) :: v(:)
```

```
n=size(v)
forall(i=1:n) v(i)=i*i
s=sum(v)
end function fun

program function_matriz
integer,allocatable :: a(:,:)
interface
    integer function fun(a) result(s)
        integer,intent(out) :: a(:,:)
    end function fun
end interface
n=2;m=3
allocate(a(n,m))
s=fun(a)
print *,'s=',s,' a='
do i=1,n
    print *,(a(i,j),j=1,m)
end do
deallocate(a)

end program function_matriz
!-------------------------------
integer function fun(a) result(s)
integer,intent(out) :: a(:,:)
n=size(a,1);m=size(a,2)
forall(i=1:n,j=1:m) a(i,j)=i**2*j
s=sum(a)
end function fun
```

6. **Progreso dun programa. Formatos. Código ASCII dun carácter non imprimíbel**. Escribe un programa chamado `progreso.f90` que mostre por pantalla o progreso dun bucle como un porcentaxe na mesma liña da terminal. Podes descargar este programa desde este **enlace**.

```
program progreso
integer,parameter :: n=10000000
do i=1,n
    write (*,'(1a1,f6.2,"%",$)') char(13),100.*i/n
end do
write (*,*) ''

end program progreso
```

Ampliando este programa podemos estimar o tempo que queda para que remate (**enlace**):

```
program retorno_carro
real(8) :: t0,t1,dt,i=1,n=1000000000.   !10000000
character(40) :: strtime
print '(a10,"    ",a)','Progreso','Tempo restante'
call cpu_time(t0)
do
    call cpu_time(t1)
    dt=(t1-t0)*(n-i)/i
    !char(13): codigo para retorno de carro
    write (*,'(1a1,f10.2,"%    ",a,$)') char(13),100.*i/n,strtime(dt)
    i=i+1
    if(i>n) exit
end do
```

```
write (*,*) ''
end program retorno_carro
!-----------------------------------------
character(40) function strtime(t) result(str)
real(8),intent(in) :: t
integer :: year,month,d,h,m,s
if(t<60) then  ! seconds in a minute
    s=floor(t)
    write(str,'(i2," s")') s
else if(t<3600) then  ! seconds in an hour
    m=floor(t/60);s=floor(t-60*m);
    write(str,'(i2," m ",i2," s")') m,s
else if(t<86400) then  ! seconds in a day
     h=floor(t/3600);m=floor((t-3600*h)/60);s=floor(t-3600*h-60*m);
     write(str,'(i2," h ",i2," m ",i2," s")') h,m,s
else if(t<2592000) then  ! seconds in a month
     d=floor(t/86400);h=floor((t-86400*d)/3600);m=floor((t-86400*d-3600*h)/60);s=floor(t-8640
     write(str,'(i2," d ",i2," h ",i2," m ",i2," s")') d,h,m,s
else if(t<31536000) then  ! seconds in a year
     month=floor(t/2592000);d=floor((t-2592000*month)/86400);h=floor((t-2592000*month-86400*d
     m=floor((t-2592000*month-86400*d-3600*h)/60);s=floor(t-2592000*month-86400*d-3600*h-60*m
     write(str,'(i2," month ",i2," d ",i2," h ",i2," m ",i2," s")') month,d,h,m,s
else
    y=floor(t/31536000);month=floor((t-31536000*y)/2592000);d=floor((t-31536000*y-2592000*mon
    h=floor((t-31536000*y-2592000*month-86400*d)/3600);m=floor((t-31536000*y-2592000*month-86
    s=floor(t-31536000*y-2592000*month-86400*d-3600*h-60*m)
    write(str,'(i2," y ",i2," month ",i2," d ",i2," h ",i2," m ",i2," s")') month,d,h,m,s
end if
return
end function strtime
```

7. **Validación de datos lidos por teclado**. Escribe un programa chamado `valida.f90` que pida por teclado un número enteiro maior que 2 e valide o valor introducido, voltando a pedilo se éste non cumpre a condición.

```
program valida
do
  print '("n(>2)? ",$)'
  read *,n
  if(n>2) exit
  print *,'valor non aceptado'
end do
print '("valor ",i0," aceptado")',n
end program valida
```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa que lea por teclado un número n e un vector $\mathbf{v}$ enteiro de lonxitude n. Logo, o programa debe ler outro número enteiro m e mostrar por pantalla os índices das ocurrencias de m en $\mathbf{v}$.

2. Escribe un programa que lea por teclado catro números enteiros n_a, m_a, n_b e m_b e dúas matrices A e B de orde $n_a \times m_a$ e $n_b \times m_b$ respectivamente, e calcule o produto matricial de ambas. O programa debe comprobar que son multiplicábeis.

3. Escribe un programa que lea por teclado un número enteiro n e logo dous vectores n-dimensionais $\mathbf{v}$ e $\mathbf{w}$. O programa debe invocar a unha subrutina chamada `calcula_produto_exterior(...)`, que calcule e proporcione como saída a matriz A resultante de multiplicar o vector columna $\mathbf{v}$ polo vector fila $\mathbf{w}$: $a_{ij} = v_i w_j; i, j = 1, \ldots, n$. O programa debe mostra-la matriz A por pantalla dende o programa principal.

$$\mathbf{v}'\mathbf{w} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} [w_1 \dots w_n] = \begin{bmatrix} v_1w_1 & \dots & v_1w_n \\ \dots & \dots & \dots \\ v_nw_1 & \dots & v_nw_n \end{bmatrix}$$

4. Escribe un programa que lea por teclado un número enteiro n, un vector $\mathbf{v}$ e unha matriz $\mathbf{A}$, ambos de orde n. O programa principal debe chamar a un subprograma `prod_vector_matriz(...)` (debes decidir o seu tipo e argumentos) que calcule o resultado do produto matricial $\mathbf{vA}$ (sendo $\mathbf{v}$ un vector fila).

5. Escribe un programa que lea por teclado un número enteiro n, un vector $\mathbf{v}$ e unha matriz $\mathbf{A}$, ambos de orde n. O programa debe calcula-lo resultado do produto matricial $\mathbf{Av}$ (sendo $\mathbf{v}$ un vector columna).

Semana 8

Traballo en clase

1. **Cálculo de límite dunha función nun punto finito. Bucle indefinido**. Escribe un programa chamado `limite.f90` que calcule o límite:

$$\lim_{x \to 2} \frac{x^2 + x - 6}{x^2 - 4} \tag{14}$$

```
program limite
!-------------------------
! version con variabel real: da warning por variable e paso non enteiros
! do x = 1, 3, 0.05
!       print *, x, (x*x+x-6)/(x*x-4)
! end do
!-------------------------
! version con bucle indefinido
x=1
do
    print *, x, (x*x+x-6)/(x*x-4)
    x=x+0.05
    if (x>3) exit ! para evitar pasar de x=3
end do


end program limite
```

2. **Representación gráfica empregando programas externos**. Para isto, redirixe a saída do programa anterior a un arquivo co comando de Linux:

   `a.out >limite.dat`

   E logo representa gráficamente esta saída co `octave`: executa o comando `octave -q --no-gui` e, unha vez dentro do octave, executa:

```
load limite.dat
plot(limite(:,1), limite(:,2))
exit
```

   Ou co `gnuplot`: executa o comando `gnuplot` e, unha vez dentro do gnuplot, executa:

```
plot "limite.dat" using 1:2 with linespoints
exit
```

3. **Representación gráfica en Fortran usando o programa GnuFor2**. A mesma representación gráfica pódese facer dende Fortran. Para isto, descarga o programa `gnufor2.f90` dende este **enlace**. O código orixinal de Alexey Kuznetsov está dispoñíbel neste outro enlace (arquivo gnufor2.zip, descomprímeo con `unzip gnufor2.zip`; a documentación está no arquivo `index.html`). O seguinte programa `limite_gnufor2.f90` representa gráficamente con esta libraría a función do exercicio anterior, usando a subrutina `plot(x,y)`, sendo `x` e `y` dous vectores (con $n = 100$ elementos cada un) coas coordenadas X e Y dos puntos da función, sendo $x \in [1, 3]$ e $y = f(x) = \dfrac{x^2 + x - 6}{x^2 - 4}$.

```
program limite_gnufor2
use gnufor2  ! indica que se use o modulo gnufor2 (arquivo gnufor2.mod)
integer,parameter :: n=100
f(x)=(x**2+x-6)/(x**2-4)
real(8) :: x(n),y(n)  ! hai que usar reais de dobre precision
a=1;b=3;h=(b-a)/n;t=a
do i=1,n
  x(i)=t;y(i)=f(t);t=t+h
end do
call plot(x,y)  ! subrutina da libraria gnufor2
end program limite_gnufor2
```

Compila o programa co comando (necesitas ter os arquivos gnufor2.f90 no mesmo directorio que limite_gnufor2.f90):

```
f95 gnufor2.f90 limite_gnufor2.f90
```

Executa o programa co comando `a.out`: crea a ventá da figura 2, na que podes comprobar que o $\lim\limits_{x \to 2} \dfrac{x^2 + x - 6}{x^2 - 4} = 1.25$.

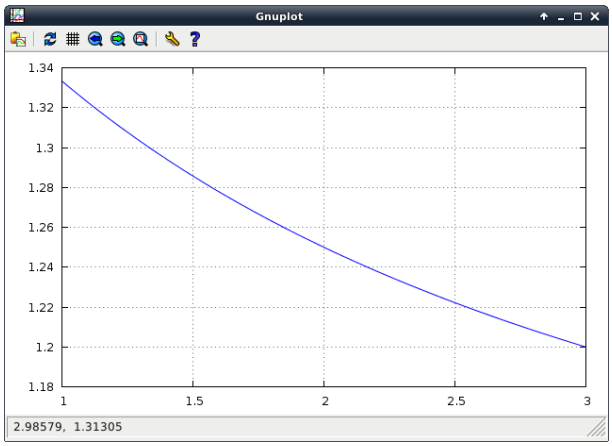


Figura 2: Representación gráfica dunha función dunha variábel usando gnufor2.

4. **Representación gráfica de función de dúas variábeis en $\mathbb{R}^3$**. Escribe un programa en Fortran chamado `grafica3D.f90` que represente a función $f(x, y) = \operatorname{sen} 5(x^2 + y^2) \exp\left(-\dfrac{x^2 + y^2}{4}\right)$, con $x, y \in [-2, 2]$. Podes descargar este programa dende este **enlace**.

```
program grafica3D
use gnufor2
integer,parameter :: n=100
real(8) :: x(n),y(n),z(n,n)
f(x,y)=sin(5*(x**2+y**2))*exp(-(x**2+y**2)/4)
a=-2;b=2;h=(b-a)/n;t=a
do i=1,n
  x(i)=t;y(i)=t;t=t+h
end do
forall(i=1:n,j=1:n) z(i,j)=f(x(i),y(i))
```

```
call surf(x,y,z)
end program grafica3D
```

Podes compilar este programa co comando `f95 gnufor2.f90 grafica3D.f90`, obtendo a ventá da figura 3.

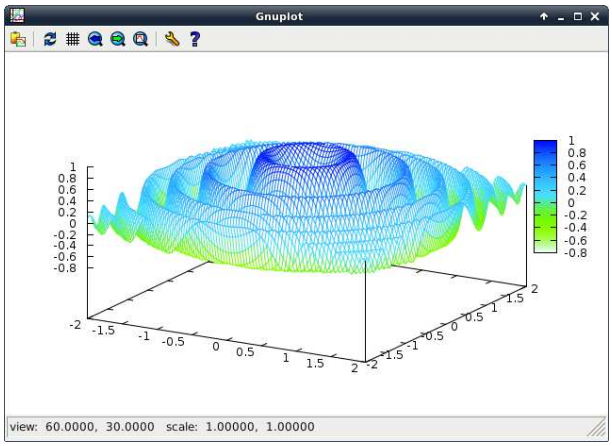


Figura 3: Representación gráfica da función $f(x,y) = \operatorname{sen} 5(x^2+y^2) \exp\left(-\dfrac{x^2+y^2}{4}\right)$ usando gnufor2.

5. **Cálculo da derivada dunha función. Función de sentenza. Escritura en arquivo.**. Escribe un programa chamado `derivada.f90` que calcule e represente gráficamente a derivada da función $f(x) = \mathrm{e}^{-x} \operatorname{sen} 2x$ no intervalo $[0, 10]$. Para isto, ten en conta, pola definición de derivada dunha función nun punto, que, usando $h = 0^+$:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \simeq \frac{f(x+h) - f(x)}{h} \tag{15}$$

```
program derivada
real, parameter :: a = 0, b = 10
f(x)=exp(-x)*sin(2*x)  ! funcion de sentenza
fp(x)=-exp(-x)*sin(2*x)+2*exp(-x)*cos(2*x)  ! derivada analitica
open(1, file="derivada.dat", status="new", err=1)
h=0.01;x=a;fx=f(x)
do
    xh=x+h;fxh=f(xh);df=(fxh - fx)/h
    write (1, *) x, fx, df, fp(xh)
    x=xh;fx=fxh
    if(x > b) exit
end do
close(1)
stop
1 stop "derivada.dat xa existe"
end program derivada
```

Para representar a función e a derivada co `octave` (o símbolo $ significa que tes que teclear na terminal, e o símbolo $\gg$ significa que tes que teclealo no octave):

```
$ a.out
>> octave -q --no-gui
>> x=load("derivada.dat");
>> subplot(2,1,1)
>> plot(x(:,1),x(:,2),';f(x);','linewidth',5)
>> subplot(2,1,2)
>> plot(x(:,1),x(:,3),';df(x);','linewidth',5)
```

```
>> hold on
>> plot(x(:,1),x(:,4),'r;fp(x);','linewidth',5)
>> quit
```

6. **Cálculo de integrais indefinidas**. Escribe un programa chamado `primitiva.f90` que calcule a integral indefinida (primitiva) dunha función *f(x)* no intervalo $[a, b]$. Sabes que se $p(x) = \int_a^x f(t)dt$, con $a \leq x \leq b$, é unha primitiva de $f(x)$, entón $p'(x) = f(x)$. Pola definición de derivada temos que:

$$f(x) = p'(x) = \lim_{h \to 0} \frac{p(x+h) - p(x)}{h} \tag{16}$$

Se tomamos $h \simeq 0^+$ podemos aproximar:

$$f(x) \simeq \frac{p(x+h) - p(x)}{h} \tag{17}$$

e despexar na ec. anterior $p(x+h) \simeq p(x) + hf(x)$. Como coñecemos $f(x)$ e queremos a súa integral indefinida (é dicir, $p(x)$ tal que $p'(x) = f(x)$), fixando un valor inicial $p(a)$ podemos calcular $p(x), \forall x > a$. Este valor inicial $p(a)$ prefixado é equivalente á constante C que se lle pode sumar á función primitiva $p(x)$. A fórmula anterior indica que o valor novo $p(x+h)$ da primitiva calcúlase como o valor en $x+h$ da liña recta que pasa polo punto $(x, p(x))$ e ten pendente $f(x)$. Deste modo, a derivada da primitiva $p(x)$ é a función orixinal $f(x)$, como se mostra na figura 4. O valor de h debe verificar que para $x \in [a, b]$ a función $f(x)$ pode aproximarse entre x e $x+h$ por unha liña recta con pendente $f(x)$. No programa, calcula $p(x) = \int_a^x f(t)dt$ no intervalo $[a, b]$ usando $a = 0, b = 1, f(t) = t$, $p(a) = 0$. Repite o cálculo para $a = 0, b = \pi, f(t) = \operatorname{sen} t, p(a) = 0$. Se queres calcular outra integral indefinida, so tes que cambiar a, b, $p(a)$ e $f(x)$.

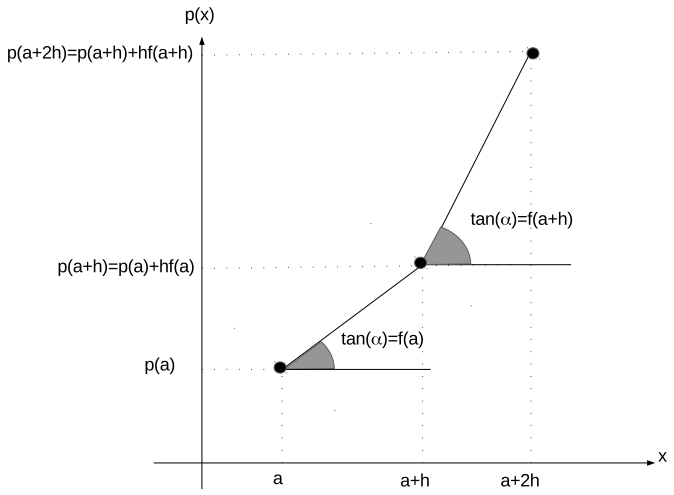


Figura 4: Aproximación numérica á primitiva $p(x)$ dunha función $f(x)$.

```
program primitiva
f(x)=x  !f(x)=sin(x)
open(1, file="integral.dat", status="new", err=1)
a=0;b=1;px=0  ! modificar para cada caso
x=a;h=0.01
do
    fx=f(x)
    write (1, *) x,fx,px
    x=x+h;px=px+h*fx
    if(x > b) exit
```

```
end do
close(1)
stop
1 stop "integral.dat xa existe"
end program primitiva
```

Tamén se pode calcular a primitiva dunha función nun intervalo sen que se coñeza a súa expresión analítica pero si os seus valores nese intervalo. Supón que a separación h entre dous valores consecutivos é h=0.01. O seguinte exemplo calcula a primitiva lendo os valores dende o arquivo `valores_funcion.dat`, que podes descargar dende este **enlace**.

```
program primitiva2
open(1, file="integral.dat", status="new", err=1)
open(2, file="valores_funcion.dat", status="old", err=2)
a=0;b=1;px=0;x=a;h=0.01
do
    read (2,*,end=3) fx
    write (1,*) x,fx,px
    x=x+h;px=px+h*fx
end do
3 close(2)
close(1)
stop
1 stop "integral.dat xa existe"
2 stop "valores_funcion.dat non existe"
end program primitiva2
```

7. **Cálculo dunha integral definida. Reais de dobre precisión**. Escribe un programa chamado `integral.f90` que calcule a integral definida funha función $f(x)$ no intervalo $[a, b]$. Prueba con $\int_{-1}^{1} \frac{\arccos x}{1+x^2} dx$. Usa reais de dobre precisión.

```
program integral
real(8) :: a=-1,b=1,h=1d-004,s=0,x,f  ! modificar a,b,h para cada caso
f(x)=acos(x)/(1+x*x)   ! modificar para cada caso
x=a
do
    s=s+f(x);x=x+h
    if(x > b) exit
end do
s=h*s
print *,'h=',h
print *,'integral=', s
print *,'valor correcto= 2.46740110027234'
print *,'diferencia=',abs(s-2.46740110027234)
end program integral
```

Compara o resultado co proporcionado polo octave, executando:

```
$ octave -q --no-gui
>> f=@(x) acos(x)/(1+x*x)
>> format long
>> quad(f,-1,1)
>> quit
```

Derivada, integral indefinida e integral definida dunha función dada como un vector de puntos.

A partir dos tres exercicios anteriores, consideremos unha función $f(x)$ definida no intervalo $[a, b]$ por un vector de n valores $\mathbf{f} = (f_1, \ldots, f_n)$, onde $f_i = f(x_i)$ con $x_i = a + h(i-1)$ con $i = 1 \ldots n$ e $h = \frac{b-a}{n-1}$. Consideraremos que o número n

de puntos é suficientemente elevado como para que a función $f(x)$ poda aproximarse con precisión por unha recta entre x_i e x_{i+1} ou, equivalentemente, que h é suficientemente pequeno. Entón temos que:

- A súa derivada $d(x) = f'(x)$ pode describirse polo vector $\mathbf{d} = (d_1, \ldots, d_{n-1})$, onde $d_i = \dfrac{f_{i+1} - f_i}{h}$, con $i = 1 \ldots n-1$.
- A súa primitiva $p(x) = \int f(x)dx$ pode describirse polo vector $\mathbf{p} = (p_1, \ldots, p_n)$, onde $p_1 = p(a)$ (prefixado por nós arbitrariamente, p.ex. $p(a) = 0$) e $p_{i+1} = p_i + hf_i$ con $i = 1 \ldots n-1$.
- A súa integral definida $\int_a^b f(x)dx$ pode aproximarse pola suma $h\sum_{i=1}^{n} f_i$.

Traballo a desenvolver pol@ alumn@

1. Escribe un programa en Fortran que calcule os valores da seguinte función definida por intervalos:

$$f(x) = \begin{cases} 1+x & x \leq 0 \\ x & 0 < x < 1 \\ 2-x & 1 \leq x \leq 2 \\ 3x - x^2 & x > 2 \end{cases}$$

2. Escribe un programa que represente gráficamente en $\mathbb{R}^3$ a curva $x(t) = \mathrm{e}^{-t/10}\operatorname{sen} 2t, y(t) = t^2, z(t) = \operatorname{sen} 3t$, con $t = 1 \ldots 10$. Usa a subrutina `plot3d` da libraría `gnufor2`.
3. Escribe un programa que calcule límite $\lim_{x\to 0} \mathrm{e}^{|x|/x}$.
4. Escribe un programa que calcule a derivada de $f(x) = \dfrac{x}{x^2 + 2x + 9}$
5. Xeraliza o programa visto en clase para calcular integrais definidas de modo que, usando funcións `external`, poida calcular a integral de calquer función (definida como función externa no programa).

Semana 9

Traballo en clase

1. **Determinante dunha matriz cadrada de orde n. Subprogramas recursivos. Paso de matrices a subprogramas. Arquivos. Módulos. Interfaces**. Escribe un programa chamado `determinante.f90` que lea dende un arquivo de texto unha matriz cadrada de orde n. Logo, o programa principal debe chamar a un subprograma recursivo `det(...)` que calcule o determinante da matriz lida usando o desenvolvemento por adxuntos da primeira fila da matriz. Proba cun arquivo chamado `determinante_orde3.dat` que conteña a matriz [0 2 3; 4 5 6; 7 8 9] (filas separadas por ;) con determinante 3. Proba logo con outro arquivo `determinante_orde4.dat` coa matriz [1 0 2 -1; 1 1 1 1; 3 2 0 1; 5 3 1 0], que ten determinante 4.

   Versión pasando matrices e as súas dimensións, cunha interface para a función `le_matriz(...)`:

```
program determinante
interface
    function le_matriz(nf) result(a)
        character(*),intent(in) :: nf
        integer,allocatable :: a(:,:)
    end function le_matriz
end interface
integer,allocatable :: a(:,:)
integer :: det
a=le_matriz('matriz.dat')
n=size(a,1);m=det(a,n)
```

```
print '("det(a)=",i0)',m
deallocate(a)
end program determinante
!-------------------------------
recursive integer function det(a,n) result(m)
integer,intent(in) :: a(n,n),n
integer,allocatable :: b(:,:)
if(n==1) then
    m=a(1,1)
else if(n==2) then
    m=a(1,1)*a(2,2)-a(1,2)*a(2,1)
else
    m=0;k=1;l=n-1
    allocate(b(l,l))
    do i=1,n
        call adxunta(a,n,i,b,l)
        m=m+k*a(1,i)*det(b,l);k=-k
    end do
    deallocate(b)
end if
end function det
!-------------------------------
subroutine adxunta(a,n,i,b,l)
integer,intent(in) :: a(n,n),n,i,l
integer,intent(out) :: b(l,l)
integer :: p
do j=2,n
    m=j-1;p=1
    do k=1,n
        if(k/=i) then
            b(m,p)=a(j,k);p=p+1
        end if
    end do
end do
call imprime(b,l)
end subroutine adxunta
!-------------------------------
function le_matriz(nf) result(a)
character(*),intent(in) :: nf
integer,allocatable :: a(:,:)
open(1,file=nf,status='old',err=1);n=0
do
    read (1,*,end=2);n=n+1
end do
2 allocate(a(n,n));rewind(1)
do i=1,n
    read (1,*) (a(i,j),j=1,n)
end do
call imprime(a,n)
close(1)
return
1 print *,'arquivo',nf,'non atopado';stop
end function le_matriz
!-------------------------------
subroutine imprime(a,n)
integer,intent(in) :: a(n,n),n
do i=1,n
```

```
    do j=1,n
        print '(i0," ",$)',a(i,j)
    end do
    print *,''
end do
print *,'-----------------------'
end subroutine imprime
```

Versión co cálculo da matriz adxunta vectorizada:

```
program determinante
interface
    function le_matriz(nf) result(a)
        character(*),intent(in) :: nf
        integer,allocatable :: a(:,:)
    end function le_matriz
end interface
integer,allocatable :: a(:,:)
integer :: det
a=le_matriz('matriz.dat');n=size(a,1)
m=det(a,n)
print '("det(a)=",i0)',m
deallocate(a)
end program determinante
!-------------------------------------
recursive integer function det(a,n) result(d)
integer,intent(in) :: a(n,n),n
integer,allocatable :: b(:,:),j(:),l(:)
select case (n)
case (3:)
    d=0;m=n-1;k=1;j=[(i,i=1,n)]
    do i=1,n
        l=pack(j,j/=i);b=a(2:,l)
        d=d+k*a(1,i)*det(b,m);k=-k
    end do
case (2)
    d=a(1,1)*a(2,2)-a(1,2)*a(2,1)
case (1)
    d=a(1,1)
case default
    print *,'erro: orde ',n,' invalida';stop
end select
end function det
!-------------------------------------
function le_matriz(nf) result(a)
character(*),intent(in) :: nf
integer,allocatable :: a(:,:)
open(1,file=nf,status='old',err=1);n=0
do
    read (1,*,end=2);n=n+1
end do
2 allocate(a(n,n))
rewind(1)
do i=1,n
    read (1,*) (a(i,j),j=1,n)
end do
call imprime(a,n)
close(1)
```

```
return
1 print *,'erro: ',nf,'non existe';stop
end function le_matriz
!-------------------------------------
subroutine imprime(a,n)
integer,intent(in) :: a(n,n),n
do i=1,n
    do j=1,n
        print '(i0," ",$)',a(i,j)
    end do
    print *,''
end do
print *,'-------------------------------------'
end subroutine imprime
```

Versión con módulo. Arquivo `determinante_modulo.f90`, co módulo que contén tódolos subprogramas:

```
module detmod
contains
!-------------------------------------
function le_matriz(nf) result(a)
character(len=100),intent(in) :: nf
real,allocatable :: a(:,:)
open(1,file=nf,status='old',err=1);n=0
do
  read(1,*,end=2);n=n+1;
end do
2 rewind(1)
allocate(a(n,n))
do i=1,n
  read (1,*) (a(i,j),j=1,n)
end do
close(1)
call imprime(a)
return
1 stop 'arquivo non existe'
end function le_matriz
!-------------------------------------
subroutine imprime(a)
real,intent(in) :: a(:,:)
n=size(a,1);print *,'a='
do i=1,n
  do j=1,n
    print '(f5.2," ",$)',a(i,j)
  end do
  print *,''
end do
end subroutine imprime
!-------------------------------------
recursive function det(a) result(d)
real,intent(in) :: a(:,:)
real,allocatable :: b(:,:)
n=size(a,1)
if(1==n) then
  d=a(1,1)
else if(2==n) then
  d=a(1,1)*a(2,2)-a(1,2)*a(2,1)
else
```

```
    d=0;m=n-1;allocate(b(m,m));k=1
    do i=1,n
      call adxunto(a,i,b)
      d=d+k*a(1,i)*det(b);k=-k
    end do
    deallocate(b)
  end if
  end function det
  !--------------------------------
  subroutine adxunto(a,i,b)
  real,intent(in) :: a(:,:)
  integer,intent(in) :: i
  real,intent(out) :: b(:,:)
  n=size(a,1)
  do j=2,n
    l=1
    do k=1,n
      if(k/=i) then
        b(j-1,l)=a(j,k);l=l+1
      end if
    end do
  end do
  call imprime(b)
  end subroutine adxunto
  end module detmod
```

Arquivo `determinante.f90` que contén o programa principal que usa o módulo anterior:

```
program determinante
use detmod
real,allocatable :: a(:,:)
character(len=100) :: nf='determinante_orde3.dat'
a=le_matriz(nf)
d=det(a)
print '("determinante=",f10.2)',d
deallocate(a)
end program determinante
```

Para compilalo, executa:

```
f95 determinante_modulo.f90 determinante.f90 -o determinante
```

2. **Persistencia dun número enteiro (descomposición en cifras)**. Escribe un programa chamado `persistencia.f90` que lea por teclado un número enteiro e calcule a súa persistencia. Para isto, o programa debe separar o número nas súas cifras e multiplicalas entre si. Este producto dividirase novamente nas súas cifras, e éstas multiplicaranse entre si, continuando o proceso ata obter un resultado dunha única cifra. A **persistencia** será o número de veces que se repetiu o proceso. `Exemplo: o número 715 ten persistencia 3 (715 ->35 ->15 ->5)`

```
program persistencia
print '("n? ",$)'; read *,n
m=n;k=0
do
  i=1
  do
    i=i*mod(m,10);m=m/10
    if(m==0) exit
  end do
  print *,i
  k=k+1
```

```
   if(i<10) exit
  m=i
end do
print '("persistencia de ",i0,": ",i0)',m,k
end program persistencia
```

Versión usando cadeas de caracteres:

```
program persistencia
character(100) :: n
print '("n? ",$)'
read *,n
k=0
do
     i=1;k=k+1
     do j=1,len_trim(n)
          read (n(j:j),'(i1)') l
          i=i*l
     end do
     print '("i=",i0)',i
     if(i<10) exit
     write (n,'(i0)') i
end do
print '("persistencia=",i0)',k
end program persistencia
```

3. **Cálculo numérico: resolución de ecuacións non lineares co método da bisección**. Escribe un programa chamado `biseccion.f90` que implemente este método, descrito no enlace:

   `https://es.wikipedia.org/wiki/Método_de_bisección`

   Este método busca solucións dunha ecuación non linear $f(x) = 0$, nun intervalo $[a, b]$ tal que $f(a)f(b) < 0$, sendo $f(x)$ unha función continua. Primeiro debes comprobar que $f(a)f(b) < 0$. En caso contrario, remata cunha mensaxe de erro. Partiendo de $x_1 = a, x_2 = b$, este método calcula $x_m = \frac{a+b}{2}$ e avalía o signo de $f(a)f(x_m)$ e $f(x_m)f(b)$. Se o primeiro producto é negativo, entón repite o proceso con a e x_m. Se o negativo é o segundo producto, repite o proceso con x_m e b. Así vas reducindo o intervalo de búsqueda ata que $|b - a| < \varepsilon$ (usa $\varepsilon = 10^{-5}$), caso no que $f(x_m) \simeq 0$ e o proceso de interación remata, sendo x_m unha aproximación á solución. Proba con: 1) con $f(x) = xe^{-x}-0.2$ usando $a = 0, b = 1$, cuxa solución é x^*=0.259171102; 2) con $f(x) = x - e^{-x}$ usando $a = 0, b = 1$, con solución x^*=0.567143290; 3) con $f(x) = x \sin x$ e $a = 0, b = 1$, con solución $x^* = a = 0$; e 4) con $f(x) = x - 1/2$, con solución $x^* = x_m$ =0.5.

```
program biseccion
!-------------------------------------------
character(100) :: s='x*exp(-x)-0.2'
f(x)=x*exp(-x)-0.2
!-------------------------------------------
niter=100;eps=1e-5
print *,'f(x)=',s
do
     print '("introduce a,b con f(a)f(b)<0: ",$)'; read *,a,b
     fa=f(a);fb=f(b)
     if(fa*fb<0) exit
     if(abs(fa)<eps) then
          print '("x=a=",f13.9)',a; stop
     end if
     if(abs(fb)<eps) then
          print '("x=b=",f13.9)',b; stop
     end if
     print *,'f(a)f(b)>0: non hai ceros entre ',a,' e ',b
end do
do i=1,niter
```

```
    xm=(a+b)/2; fxm=f(xm)
    if(abs(fxm)<eps) then
          print '("converxeu en ",i0," iteracions: x=xm=",f13.9)',i,xm; stop
      end if
    if(fa*fxm<0) then
      b=xm
    else
      a=xm; fa=fxm
    end if
      print '("i=",i0," a=",f13.9," b=",f13.9)',i,a,b
      if(b-a<eps) then
          print '("converxeu en ",i0," iteracions: x=",f13.9)',i,a; stop
      end if
  end do
  print '("non converxeu en ",i0," iteracions: a=",f13.9," b=",f13.9," dif=",f13.9)',
    niter,a,b,b-a

  end program biseccion
```

4. **Cálculo da inversa dunha matriz cadrada de orde n usando a libraría Lapack**. Descarga o arquivo `inversa_lapack.tar.gz` dende este **enlace** e descomprímeo co comando `tar zxvf inversa_lapack.tar.gz`. O programa chámase `inversa.f90`, e calcula a inversa dunha matriz cadrada de calquera orde usando a libraría Lapack (o programa le a matriz dende un arquivo incluido no arquivo `inversa_lapack.tar.gz`).

```
! ORDE 3: matriz a=
!    2.00   1.00  -1.00
!    2.00   0.00   1.00
!    1.00   2.00   4.00
!  inv(a)=
!    0.13   0.40  -0.07
!    0.47  -0.60   0.27
!   -0.27   0.20   0.13
!-------------------------------------
! ORDE 4: a =
!     1    2    3    4
!    -1    2    0    3
!     4    1    9    5
!     4    3    2    1
! inv(a) =
!     3.08   -2.40   -1.00   -0.12
!    -3.32    2.60    1.00    0.48
!    -2.80    2.00    1.00    0.20
!     3.24   -2.20   -1.00   -0.36
program inversa
real(8),allocatable :: a(:,:),b(:,:)

open(1, file="matriz_orde4.dat", status="old")
read (1,*) n
allocate(a(n,n),b(n,n))
do i = 1, n
  read (1, *) (a(i, j), j = 1, n)
  do j = 1, n
    print '(f6.2,$)', a(i, j)
  end do
  print *, '' ! para pasar a seguinte linha
end do
close(1)
```

```
call inv(a, n, b)

print *, "inv(a)="
do i = 1, n
  do j = 1, n
    print '(f6.2,$)', b(i, j)
  end do
  print *, ''
end do

deallocate(a,b)

end program inversa

!-----------------------------------
! Retorna a inversa dunha matriz calculando a descomposicion LU con Lapack
subroutine inv(A, n, Ainv)
real(8),intent(in) :: A(n,n)
real(8),intent(out) :: Ainv(n,n)
integer,intent(in) :: n
real(8) :: work(n)  ! work array for LAPACK
integer :: ipiv(n)   ! pivot indices
integer :: info

! procedementos external definidos en Lapack
external DGETRF
external DGETRI

! Almacena A en Ainv para evitar que lapack a sobrescriba
Ainv = A

! DGETRF calcula a factorizacion LU da matriz usando pivote parcial con intercambio de fila
call DGETRF(n, n, Ainv, n, ipiv, info)
if (info /= 0) stop "error: matriz singular"

! DGETRI calcula a matriz inversa usando a factorizacion LU calculada por DGETRF.
call DGETRI(n, Ainv, n, ipiv, work, n, info)
if (info /= 0) stop "error en la inversion"

end subroutine inv
```

Compila co comando:

```
f95 inversa.f90 -llapack
```

Executa o programa probando cos arquivos matriz_orde3.dat e matriz_orde4.dat indicados nos comentarios do comezo do programa, que están incluidos no arquivo `inversa_lapack.tar.gz` que descargaches. Comproba a solución co octave (p.ex. para orde 3):

```
$ octave -q --no-gui
> a = [2 1 -1; 2 0 1; 1 2 4];
> inv(a)
ans =
   0.133333   0.400000  -0.066667
   0.466667  -0.600000   0.266667
  -0.266667   0.200000   0.133333
> quit
```

5. **Resolución dun sistema de n ecuacións lineares usando a libraría Lapack**. Descarga e descomprime o arquivo `sistema_lapack.tar.gz` dende este **enlace** e descomprímeo co comando `tar zxvf sistema_lapack.tar.gz`. Este arquivo contén o seguinte programa `sistema.f90` e os ficheiros de datos `sistema_orde3.dat` e `sistema_orde4.dat`:

```
!sistema_orde4.dat: x=1 y=0 z=-1 t=2; sistema={x+y+z+t=2, x-y-z+t=4, -x+z=-2, y+z+t=1}
! 1 1 1 1 2
! 1 -1 -1 1 4
! -1 0 1 0 -2
! 0 1 1 1 1
!-----------------------------------------------------
!sistema_orde5.dat: x=1 y=2 z=0 t=-1 u=0; sistema={x+y-z+t+u=2;x-y+t+u=1;x+z-u=0;
! x-y-z+t+u=2;x-y+z-t+u=1}
!    1   1   1   1   1   2
!    2  -1   0  -1  -1   1
!   -1   0   1  -1   1   0
!    1   1   0   1  -1   2
!    0   1  -1   1   0   1
program sistema_linear
real,allocatable :: a(:,:)
real,allocatable :: b(:)
integer,allocatable :: pivote(:)
character(1),imension(10) :: incog(10)=(/'x','y','z','t','u','v','w','r','s','p'/)
open(1,file="sistema_orde4.dat",status="old",err=1)
read (1,*) n
allocate(a(n,n),b(n),pivote(n))
do i = 1, n
  read (1, *) (a(i, j), j = 1, n), b(i)
end do
close(1)

print '(a,i3,a)', "sistema de orde", n, " ="
do i = 1, n
  print '(f7.2,a,$)', a(i,1), incog(1)
  do j = 2, n
    if(a(i,j) >= 0) then
      print '(a,f7.2,a,$)', ' + ', a(i, j), incog(j)
    else
      print '(a,f7.2,a,$)', ' - ', -a(i, j), incog(j)
    endif
  end do
  print '(a,f7.2)', ' = ', b(i)
end do

call sgesv(n, 1, a, n, pivote, b, n, info)

print *, "solucion:"
do i = 1, n
  print '(2a,f6.2)', incog(i), ' = ', b(i)
end do

deallocate(a,b,pivote)
stop
1 stop "erro en open"
end program sistema_linear
```

Descarga tamén dende o curso virtual os arquivos `sistema_orde4.dat` e `sistema_orde5.dat`, cos coeficientes e termos independentes dos sistemas lineares de ecuacións. Compila o programa co comando:

`f95 sistema_linear.f90 -llapack`

Execútao con `a.out`. Comproba que a solución é correcta calculándoa co octave (p.ex. para o sistema de orde 4):

```
$ octave -q --no-gui
> a = [1 1 1 1; 1 -1 -1 1; -1 0 1 0; 0 1 1 1]; b=[2; 4; -2; 1];
> a\b
ans =
   1
  -0
  -1
   2
> quit
```

6. **Xerador de números aleatorios**. Descarga o programa `aleatorio.f90` dende este **enlace**. Este programa imprime por pantalla 10 números aleatorios no intervalo $[a, b)$ empregando a subrutina `random_number()` de `f95`. Usa $a = -10, b = 10$. O programa usa a subrutina `init_random_seed()` para inicializar o xerador de números aleatorios co reloxio do sistema. Proba con e sen a chamada a esta subrutina.

```
program aleatorio
real :: x(10),m(3,3),s(3)=(/0,0,0/)
print '(a,$)', "introduce a,b: "
read *, a, b
rango = b - a
! call init_random_seed_clock()     ! non-reproducible
call init_random_seed_default()  ! reproducible
call random_number(x)
print '("real: ",10f8.4)', rango*x + a
call random_number(x)
print '("enteiro: ",10i5)', int(rango*x + a)
call random_number(m)
m=int(rango*m+a)
print *,'matriz de enteiros:'
do i=1,3
  print *,(int(m(i,j)),j=1,3)
end do
stop
end program aleatorio
!-----------------------------------
subroutine init_random_seed_clock()
integer :: i, n, clock
integer, allocatable :: seed(:)
call random_seed(size = n)
allocate(seed(n))
call system_clock(count = clock)
seed = clock + 37 * (/ (i - 1, i = 1, n) /)
call random_seed(put = seed)
deallocate(seed)
return
end subroutine
!-----------------------------------
subroutine init_random_seed_default()
integer :: i, n
integer,allocatable :: seed(:)
call random_seed(size=n)
allocate(seed(n))
seed=0
call random_seed(put=seed)
```

```
deallocate(seed)
return
end subroutine
```

7. **Medida do tempo consumido por un programa en Fortran**. Descarga o programa `tempo.f90` dende este **enlace**. Este programa executa un bucle de 10^8 iteracións e mostra o tempo consumido:

```
program tempo
real(8) :: inicio, fin, n, i
n=1e8; i=0
print '("medindo tempo consumido por ",d8.1," iteracions ...")',n
call cpu_time(inicio)
do
    i=i+1
    if(i>n) exit
end do
call cpu_time(fin)
print '("n=",d8.1, " tempo= ",f10.4," s.")',n,fin-inicio
end program tempo
```

8. **Funcións para produto escalar de vectores e para produto matricial**. Descarga o programa `exemplos_funcions.f90` dende este **enlace**. Este programa define un vector **v** e unha matriz cadrada **a**, ambos de orde 3, e calcula o produto escalar $\mathbf{v}^T\mathbf{v}$ e o produto matricial **aa**.

```
program exemplos_funcions
integer :: v(3) = (/1,2,3/)
integer :: a(3,3) = reshape((/1,2,3,4,5,6,7,8,9/),shape(a)), b(3,3)
interface
  subroutine imprime_matriz(a)
    integer,intent(in) :: a(:,:)
  end subroutine imprime_matriz
end interface
print '("v=",3(i0," "))', v
print *, "a="
call imprime_matriz(a)
print '("dot(v,v)=",i0)', dot_product(v,v)
b = matmul(a,a)
print *, "a*a="
call imprime_matriz(b)
b = transpose(a)
print *, "a^T="
call imprime_matriz(b)

end program exemplos_funcions


!-----------------------------------
subroutine imprime_matriz(a)
integer,intent(in) :: a(:,:)
n=size(a,1)
do i=1,n
  do j=1,n
    print '(i0," ",$)',a(i,j)
  end do
  print *,''
end do
end subroutine imprime_matriz
```

9. **Creación dunha libraría**. Descarga os programas **media.f90**, **mediana.f90**, **desviacion.f90**, **ordea.f90** e **principal.f90**.

   a) **Libraría estática**. Para crear unha libraría estática `libstat.a`, executa os comandos:

   ```
   f95 -c media.f90 mediana.f90 desviacion.f90 ordea.f90
   ar qv libstat.a *.o
   ```

   Para listar os arquivos `*.o` contidos na libraría `libstat.a` executa `ar tv libstat.a`. Para compilar o programa `principal.f90` enlazado coa libraría `libstat.a`, usa o comando:

   ```
   f95 -L. principal.f90 -lstat
   ```

   b) **Libraría dinámica**. Para crear unha libraría dinámica `libstat.so`, executa os comandos:

   ```
   f95 -fpic -c media.f90 desviacion.f90 mediana.f90 ordea.f90
   f95 -shared -o libstat.so *.o
   ```

   Para listar os arquivos `*.o` contidos na libraría `libstat.so` executa `nm libstat.so`. Para compilar o programa `principal.f90` enlazado coa libraría `libstat.so`, usa o comando:

   ```
   f95 -L. principal.f90 -lstat
   ```

   Para executar o programa:

   ```
   export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:.
   a.out
   ```

   c) **Compilación separada**. Tamén se pode compilar separadamente, sen crear ningunha libraría, cos comandos:

   ```
   f95 -c media.f90 mediana.f90 desviacion.f90 ordea.f90
   f95 principal.f90 *.o
   ```

10. **Derivada dun polinomio**. Escribe un programa chamado `polinomio.f90` que lea por teclado a orde n e os coeficientes $a_0, \ldots, a_n$, dun polinomio $p(x)$. Usa un vector dinámico de $n+1$ compoñentes con índices $0, \ldots, n$. O programa debe crear o arquivo `polinomio.dat` (inicialmente baleiro). Logo, debe chamar n veces a un subprograma `calcula_derivada(...)`: na chamada k-ésima (con $k = 1, \ldots, n$), este subprograma debe calcula-los coeficientes da derivada k-ésima de $p(x)$. Para isto hai que ter en conta que:

$$p(x) = \sum_{i=0}^{n} a_i x^i, \quad p'(x) = \sum_{i=1}^{n} i a_i x^{i-1}$$
$$p''(x) = \sum_{i=2}^{n} i(i-1) a_i x^{i-2} \quad p'''(x) = \sum_{i=3}^{n} i(i-1)(i-2) a_i x^{i-3}$$

E, polo tanto, a derivada k-ésima do polinomio está dada por:

$$p^{(k)}(x) = \sum_{i=k}^{n} \left[\prod_{j=0}^{k-1} (i-j) \right] a_i x^{i-k}, \qquad k = 1, \ldots, n \tag{18}$$

Deste modo, o coeficiente de x^{i-k} en $p^{(k)}(x)$ para $i = k, \ldots, n$, está dado por:

$$a_i \prod_{j=0}^{k-1} (i-j) \tag{19}$$

O subprograma `calcula_derivada(...)` anterior debe engadir ao arquivo `polinomio.dat` os coeficientes dos polinomios derivados (un polinomio en cada liña do arquivo). Finalmente, o programa principal debe pecha-lo arquivo e libera-la memoria reservada.

EXEMPLO: dado o polinomio $p(x) = x^4 + x^3 + x^2 + x + 1$, resulta que $n = 4$ e as derivadas do polinomio son:

$$\begin{aligned} p'(x) &= 4x^3 + 3x^2 + 2x + 1 \\ p''(x) &= 12x^2 + 6x + 2 \\ p^{(3)}(x) &= 24x + 6 \\ p^{(4)}(x) &= 24 \end{aligned}$$

e polo tanto o arquivo `polinomio.dat`, logo de executa-lo programa, debe almacena-lo seguinte contido:

```
1  2  3  4
2  6  12
6  24
24
```

```
program polinomio
real,allocatable :: a(:)
print '("n? ",$)'; read *,n
allocate(a(0:n))
print '("a(0:n)? ",$)'; read *,a
open(1,file='polinomio.dat',status='new',err=1)
do k=1,n
    call calcula_derivada(a,n,k)
end do
close(1)
deallocate(a)
stop
1 stop 'erro: polinomio.dat xa existe'
end program polinomio

!-------------------------------
subroutine calcula_derivada(a,n,k)
real,intent(in) :: a(0:n)
integer,intent(in) :: n,k
do i=k,n
    d=a(i)
    do j=0,k-1
        d=d*(i-j)
    end do
    write (1,'(f5.1,$)') d
end do
write (1,*) ''
end subroutine calcula_derivada
```

11. **Cálculo numérico: resolución de ecuacións non lineares polo método de Newton**. Este método, que podes atopar descrito neste enlace:

    `https://es.wikipedia.org/wiki/Método_de_Newton`

    resolve unha ecuación non linear $f(x) = 0$ partindo dunha aproximación inicial x_0 para x, e calculando aproximacións sucesivas x_i, con $i = 1, 2, \ldots$, dadas pola seguinte fórmula:

    $$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad i = 0, \ldots \tag{20}$$

    Esta operación iterativa execútase ata que $|x_{i+1} - x_i| < \varepsilon$. Escribe un programa chamado `newton.f90` que defina dúas constantes $\varepsilon = 10^{-5}$ e `niter=100`, e lea por teclado o punto inicial x_0 e execute iterativamente (ata `niter` iteracións) a operación 20 ata que $|x_i - x_{i-1}| < \varepsilon$, mostrando a solución x^* ou unha mensaxe indicando que non converxeu. Proba coas ecuacións: 1) $x\mathrm{e}^{-x}$ =0.2 usando $x_0 = 0$ e debes obter x^*=0.259171102; 2) con $f(x) = x - \mathrm{e}^{-x}$ usando $x_0 = 0$, con

solución x^*=0.567143290; 3) $x^3 - x^2 - x + 1 = 0$, usando $x_0 = 2$, para calcular a raíz dobre $x = 1$, e con $x_0 = -3$ para calcular a raíz simple $x = -1$; e 4) $e^{-x^2} = 0$ con $x_0 = 1$ (da erro por derivada nula) e $x_0 = 1$ (executa `niter` iteracións sen atopar solución, porque non existe).

```
program newton
real,parameter :: eps=1e-5
integer,parameter :: niter=100
!---Funcion e derivada-----------------------------------------------------
character(100) :: s='x*exp(-x)-0.2'
f(x)=x*exp(-x)-0.2;df(x)=(1-x)*exp(-x)
!-------------------------------------------------------------------------
print *,'ecuacion ',s
print '("x0? ",$)';read *, xi
do i=1,niter
    dfx=df(xi)
    if(dfx==0) then
        print *,'dfx=0 en x=',xi,':rematado'; stop
    end if
    xi1=xi-f(xi)/dfx
    print '("iter=",i0," x=",f13.9)', i, xi1
    if(abs(xi1-xi)<eps) exit
    xi=xi1
end do
if(i<=niter) then
    print '("x=",f13.9," en ",i0," iteracions")',xi1,i
else
    print '("non converxeu en ",i0," iteracions")',niter
end if

end program newton
```

12. **Cálculo numérico: resolución de ecuacións non lineares co método do punto fixo**. Este método, que podedes consultar en:

    http://es.wikipedia.org/wiki/Metodo_del_punto_fijo

    permite resolver ecuacións non lineares do tipo $f(x) = 0$ se podes poñelas na forma $x = g(x)$ con $|g'(x)| \leq 1$. Para isto, le por teclado un valor x_0 que verifique $|g'(x_0)| < 1$, e logo executas, na iteración i:

$$x_{i+1} = g(x_i), \quad i = 0, \ldots \tag{21}$$

    O proceso repítese ata que $|x_i - x_{i-1}| < \varepsilon$ (entón considérase que o proceso de busca da solución converxeu), e a solución é $x^* = x_i$. Escribe un programa chamado `punto_fixo.f90` que execute este método para $f(x) = x + xe^x + 1$, usando $g(x) = -1/(1 + e^x)$, de modo que $g'(x) = e^x/(1 + e^x)^2$, que verifica $|g'(x)| < 1, \forall x$, aínda que non necesitas calcular $g'(x)$ no programa de Fortran. Usa ε =0.001. O programa debe executar a operación 21 ata que se cumpra a condición de remate, mostrando por pantalla a solución e o n.º de iteracións. Debes obter a solución x^*= -0.659852.

```
program punto_fixo
print '("x0? ",$)'; read*, xi
eps=1e-4;iter=0
do
    print '("iteracion ",i0," x=",f10.6)',iter,xi
    xi1=-1/(1+ exp(xi))
    if (abs(xi1 - xi) < eps) exit
    xi=xi1;iter=iter + 1
end do
print '("solucion x=", f10.6, " usando ", i0, " iteracions")',xi,iter

end program punto_fixo
```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa que lea por teclado o grao n dun polinomio $p(x) = a_n x^n + \ldots a_2 x^2 + a_1 x + a_0$ e os seus coeficientes $a_0, \ldots, a_n$ e calcule as raíces enteiras do polinomio. Ter en conta que as posíbeis raíces enteiras do polinomio son divisores do termo independente a_0.

2. Escribe un programa que calcule a matriz de covarianza Σ dun conxunto de N vectores d-dimensionais $\{\mathbf{x}_i, i = 1, \ldots, N\}$, sendo $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})$. O elemento ij da matriz de covarianza Σ (cadrada de orde d) defínese como:

$$\Sigma_{ij} = \frac{1}{N} \sum_{k=1}^{N} (x_{ki} - \langle x_i \rangle)(x_{kj} - \langle x_j \rangle) \tag{22}$$

   Onde $\langle x_i \rangle$ é o valor medio da compoñente i dos vectores $\mathbf{x}_k$:

$$\langle x_i \rangle = \frac{1}{N} \sum_{k=1}^{N} x_{ki} \tag{23}$$

   O programa debe, dados $N = 12$ e $d = 5$, debe chamar a un subprograma onde abra o arquivo e lea os vectores (cada vector está almacenado nunha liña distinta no arquivo). Logo, debe chamar a outro subprograma que calcule as medias $\langle x_i \rangle, i = 1, \ldots, d$. Por ltimo, debe chamar a un subprograma que calcule cada elemento $\Sigma_{ij}, i, j = 1, \ldots, d$, mediante a fórmula 3. Finalmente, debe chamar a outro subprograma que imprima a matriz Σ (fila a fila).

   NOTA: Empregar o arquivo `vectores.dat` que se proporciona ($N = 12$, $d = 5$).

```
1.3 0.4 1.5 0.4 1.2
1.9 0.4 1.5 0.7 1.3
1.2 0.4 0.9 0.5 1.2
1.5 0.4 2.1 0.8 1.1
1.1 0.4 2.2 0.9 1.0
1.0 0.4 2.3 0.2 0.9
0.6 0.4 2.5 0.1 0.8
1.1 1.4 1.9 0.4 0.7
0.4 0.3 1.5 0.3 0.6
0.5 1.2 1.3 0.2 0.5
0.8 1.4 1.6 0.4 0.4
1.3 0.9 1.2 0.7 0.2
```

Tema 2 Programación estruturada en Fortran
Exercicios propostos

1. **Programas básicos**

   a) Escribir un programa que lea a latitude β e lonxitude λ eclípticas dun obxecto astronómico en ascenso directo α e declinación δ usando as fórmulas:

$$\alpha = \arctan \frac{\operatorname{sen}\lambda \cos\epsilon - \tan\beta \operatorname{sen}\epsilon}{\cos\lambda} \tag{1}$$

$$\delta = \operatorname{arc\,sen}(\operatorname{sen}\beta \cos\epsilon + \cos\beta \operatorname{sen}\epsilon \operatorname{sen}\lambda) \tag{2}$$

   onde ϵ= 0.4091 e tódalas magnitudes están en radiáns.

   b) Escribe programas en Fortran que calculen os valores das seguintes funcións nos intervalos que se indican:

$$f(x) = \frac{\operatorname{sen} x}{x}, [-10, 10] \tag{3}$$

$$f(x) = \frac{1 - \sqrt{1 - x^2}}{x^2}, [-1, 1] \tag{4}$$

$$f(x) = \frac{x}{1 + x^2}, [-100, 100] \tag{5}$$

$$f(x) = x^2 e^{-x}, [0, 10] \tag{6}$$

2. **Límites, derivadas, integrais e series**

   a) Escribe programas en Fortran que calculen os seguintes límites:

$$\lim_{x\to\infty} \frac{2x - \operatorname{sen} x}{\cos 2x} \tag{7}$$

$$\lim_{x\to 0} \frac{x \arctan(x/2)}{\cos x (\operatorname{sen} 2x)^2} \tag{8}$$

$$\lim_{x\to\infty} x \log \frac{x+1}{x-1} \tag{9}$$

$$\lim_{x\to 1} \frac{\log x}{x - \sqrt{x}} \tag{10}$$

$$\lim_{x\to 1} \frac{x-1}{\arg\operatorname{sen}(x-1)} \tag{11}$$

$$\lim_{x\to 1} \frac{x^{1/3} - 1}{x - 1} \tag{12}$$

   b) Calcula as derivadas das seguintes funcións:

$$f(x) = \log \frac{\sqrt{1+x} + \sqrt{1-x}}{\sqrt{1+x} - \sqrt{1-x}}, \quad 0 < x < 1 \tag{13}$$

$$f(x) = \arctan \sqrt{\frac{1 - \cos x}{1 + \cos x}}, 0 < x < \pi \tag{14}$$

$$f(x) = \sqrt{x + \sqrt{x + \sqrt{x}}}, \quad x > 0 \tag{15}$$

c) Calcula as seguintes integrais definidas:

$$\int_{-1}^{1} \frac{\arccos x}{1+x^2} dx \tag{16}$$

$$\int_{0}^{\pi/2} \frac{1+\operatorname{sen} x}{1+\cos x} dx \tag{17}$$

$$\int_{-\infty}^{0} e^{-x^2} dx \tag{18}$$

$$\int_{1}^{2} \frac{x+1}{2x\sqrt{x}}$$

d) Dada unha curva de ecuacións paramétricas $x = x(t), y = y(t)$ (t é o parámetro), a lonxitude do arco de curva que vai dende o punto $(x(a), y(a))$ ao punto $(x(b), y(b))$ está dado pola seguinte fórmula:

$$L = \int_a^b \sqrt{\frac{dx(t)}{dt}^2 + \frac{dy(t)}{dt}^2}\, dt$$

Se a curva ten ecuación explícita $y = f(x)$, a lonxitude entre os puntos $(a, f(a))$ e $(b, f(b))$ é:

$$L = \int_a^b \sqrt{1 + \frac{df(x)}{dx}^2}\, dt$$

Se a curva ten ecuación en coordenadas polares $\rho = \rho(\theta)$, a lonxitude do arco entre os ángulos θ_1 e θ_2 é:

$$L = \int_{\theta_1}^{\theta_2} \sqrt{\rho(\theta)^2 + \frac{d\rho(\theta)}{d\theta}^2}\, d\theta$$

Escribir programas en Fortran que calculen as seguintes lonxitudes de arcos:

1)

$$x = x(t) = \cos t + t \operatorname{sen} t \tag{19}$$

$$y = y(t) = \operatorname{sen} t - t \cos t \tag{20}$$

$$t \in [0, \pi]$$

2)

$$x = x(t) = (t^2 - 2) \operatorname{sen} t + 2t \cos t \tag{21}$$

$$y = y(t) = (2 - t^2) \cos t + 2t \operatorname{sen} t \tag{22}$$

$$t \in [0, \pi]$$

3)

$$y = f(x) = 2 \cosh\left(\frac{x}{2}\right); \qquad \cosh x = \frac{e^x + e^{-x}}{2}$$

4)

$$\rho = \rho(\theta) = \frac{1}{2}(\theta^2 - 1)$$

5)

$$\rho = \rho(\theta) = 1(1 + \cos\theta)$$

e) Dada unha curva en coordenadas polares $\rho = \rho(\theta)$, a área pechada pola curva entre os ángulos θ_1 e θ_2 ven dada pola ecuación:

$$A = \frac{1}{2}\int_{\theta_1}^{\theta_2} \rho(\theta)^2 d\theta$$

Escribir programas en Fortran que calculen o área do recinto limitado polos arcos de curva seguintes no rango de θ indicados:

$$\rho = \theta, \theta \in [0, 2\pi] \tag{23}$$
$$\rho = e^{\theta}, \theta \in [0, \pi] \tag{24}$$
$$\rho = 2(1 + \cos\theta), \theta \in [0, 2\pi] \tag{25}$$
$$\rho = 3\cos 2\theta, \theta \in [0, 2\pi] \tag{26}$$

f) As series numéricas son sumas infinitas definidas da seguinte forma:

$$\sum_{n=1}^{\infty} x_n = \lim_{k\to\infty} \sum_{n=1}^{k} x_n \tag{27}$$
$$\tag{28}$$

Escribe programas en Fortran que calculen as seguintes series numéricas.

$$\sum_{n=1}^{\infty} \arctan\frac{1}{n^2 - n + 1} = \frac{\pi}{4} \tag{29}$$
$$\sum_{n=0}^{\infty} \frac{1}{(2n+1)(2n+3)} = \frac{1}{2} \tag{30}$$
$$\sum_{n=0}^{\infty} 3^n \operatorname{sen}^3 \frac{1}{3^{n+1}} = \frac{1 - \operatorname{sen} 1}{4}$$

Fdez-Viñas, Exercicios e Complementos de Análisis Matemático, Vol. 1, exercs. 1262, 1255, 1256, 1264, 1275 (pax. 540-546)

3. **Exercicios diversos**

a) Escribir un programa en Fortran que resuelva un sistema de n ecuacións lineais con n incógnitas empregando o Método de Eliminación Gaussiana. Dado o sistema seguinte:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \tag{31}$$
$$\ldots \tag{32}$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \tag{33}$$

O método de eliminación transforma este sistema no seguinte:

$$x_1 + a'_{12}x_2 + \ldots + a'_{1n}x_n = b'_1 \tag{34}$$
$$0 + x_2 + \ldots + a'_{1n}x_n = b'_1 \tag{35}$$
$$\ldots \tag{36}$$
$$0 + 0 + \ldots + x_n = b'_n \tag{37}$$

Onde se pode despexar directamente x_n, sustituir na $(n-1)$-ésima ecuación e despexar x_{n-1} e así sucesivamente ata calcula-las n incógnitas. As únicas transformacións permitidas son:

- Dividir tódolos elementos dunha fila polo mesmo número.
- Sumar a tódolos elementos dunha fila o produto dun escalar polo elemento correspondente doutra fila

b) Codificar un programa en Fortran que calcule cal das filas dunha matriz A ten menor valor medio. Tamén deberá informar ó usuario de cal é o valor máximo da dita fila. A matriz de datos A está gardada nun arquivo, que o programa deberá ler. Usar subprogramas.

c) Escribe un programa que realice unha operación de "suavizado" dos elementos dunha matriz cadrada **b**. A dita operación consiste en obter unha nova matriz **a** da mesma dimensión ca orixinal. Cada elemento a_{ij} da matriz transformada obtense como a media aritmética dos 9 elementos contidos nunha submatriz 3x3 centrada na compoñente correspondente b_{ij} da matriz orixinal. Para aquelas compoñentes da matriz orixinal b con menos de 8 elementos mais próximos (bordes) considerarase que son ceros na matriz "suavizada" **a**.

d) Escribe un programa que resuelva un sistema de 3 ecuacións lineais con 3 incógnitas empregando o Método de Cramer. Dado o sistema seguinte:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \tag{38}$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \tag{39}$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \tag{40}$$

As solucións (x_1, x_2, x_3) están dadas por:

$$x_1 = \frac{\begin{vmatrix} b_1 & x_{12} & x_{13} \\ b_2 & x_{22} & x_{23} \\ b_3 & x_{32} & x_{33} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}}, \quad x_1 = \frac{\begin{vmatrix} x_{11} & b_1 & x_{13} \\ x_{21} & b_2 & x_{23} \\ x_{31} & b_3 & x_{33} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}}, x_1 = \frac{\begin{vmatrix} x_{11} & x_{12} & b_1 \\ x_{21} & x_{22} & b_2 \\ x_{31} & x_{32} & b_3 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}} \tag{41}$$

Sempre supoñendo que:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \neq 0 \tag{42}$$

Os coeficientes $\{a_{ij}\}$ e os termos independentes $\{b_i\}$ deberán lerse dun ficheiro (`sistema.dat`). Deberá empregarse un subprograma para calcular o determinante dunha matriz 3x3, e outra para ler os datos do ficheiro. Probar co seguinte sistema:

$$3x_1 + 2x_2 + 8x_3 = 17 \tag{43}$$
$$2x_1 - x_2 + 7x_3 = -9 \tag{44}$$
$$x_1 + x_2 - 2x_3 = 13 \tag{45}$$

Coas solucións $x_1 = 3, x_2 = 8, x_3 = -1$.

e) Escribir un programa para realizar operacións con filas e columnas nunha matriz de números enteiros. Debe haber un menú coas seguintes operacións (a realizar en subprogramas):

1) Ler unha matriz 3x4 almacenada nun arquivo (`matriz.dat`).
2) Intercambiar de orde das filas.
3) Sumar unha das filas a todas as demais da matriz.
4) Intercambiar de orde das columnas da matriz actual.
5) Garda-la matriz no seu estado actual nun arquivo distinto do orixinal.
6) Sair do programa.

f) Escribir, usando subprogramas, un programa que pida ó usuario un número enteiro N e mostre na pantalla a seguinte información (Probar o programa cos números 15, 1024, 14, 4e-12.):

- Suma das cifras de N.
- Números primos menores que N.
- Suma dos enteiros da serie $1, 2, \ldots, N$.
- Divisores de N.

g) Escribe un programa que codifique o **algoritmo da vida**. Mediante unha matriz cadrada representarase unha poboación aleatoria inicial de individuos. Un "1" nunha compoñente da matriz representará a existencia dun individuo nesa posición, mentres ca un "_" representará a non existencia de individuos nesa posición. O número de veciños dun individuo é o que determina o seu destino na seguinte xeración. As regras que gobernan a evolución das sucesivas xeracións dunha poboación inicial son as seguintes:

- Un individuo con mais de 3 veciños nas posicións máis próximas morre por superpoboación.
- Un individuo con menos de 2 veciños máis próximos morre por aillamento.
- Aparece un individuo en calquer posición baleira que ten exactamente 3 veciños próximos.

Estas regras aplícanse sobre a poboación inicial para determina-la seguinte xeración, e así sucesivamente, determinando a evolución das seguintes xeracións. O programa deberá presentar no monitor a poboación inicial e as sucesivas xeracións obtidas aplicando as regras anteriores. Para visualiza-la seguinte xeración será necesario que o usuario pulse unha tecla. Usar subprogramas

h) Escribe un programa para realiza-lo escrutinio dos acertos en apostas da lotería primitiva (sen considerar o número complementario nin apostas múltiples). O programa debe presentar as seguintes utilidades:

- Introducción das apostas a escrutar, que poden ser mais de unha, comprobando que as apostas conteñan números válidos (no intervalo [1, 49]).
- Introducción dende o teclado da combinación gañadora (débese comprobar que as apostas conteñen números no intervalo [1, 49]).
- Presentación da estatística do número de acertos para cada columna de apostas, e un resumo das apostas con premios (3 ou mais acertos).

i) Escribe un programa para realiza-lo reparto de escanos nas eleccións seguindo a Ley d'Hont. Para face-lo reparto segundo esta lei, utilízanse os seguintes criterios:

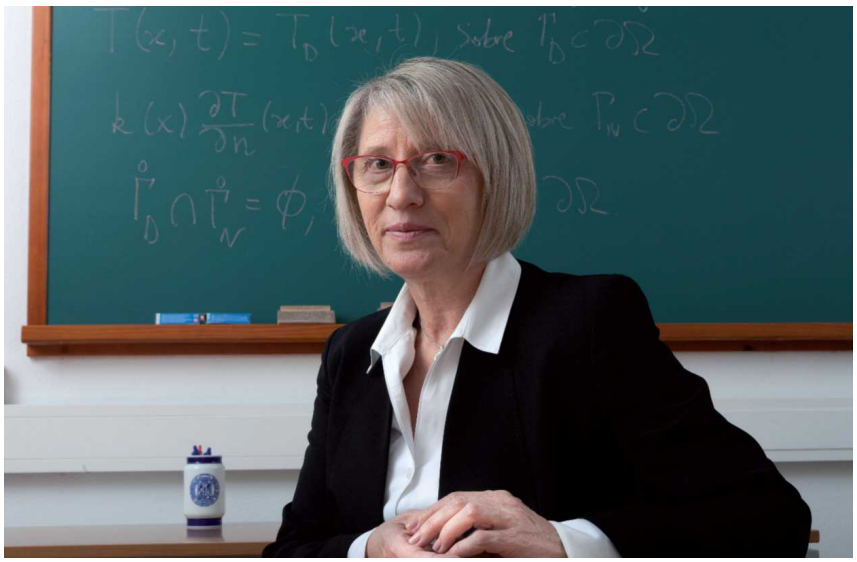
- Os partidos que obteñan menos dun 10 % do total dos votos válidos quedan excluidos do reparto.
- Para cada partido, calclase un vector de "cocientes" que resultan de dividir o total de votos obtidos polos números enteiros $k = 1, \ldots, N$
- O escano k-ésimo atribúese ó partido que ten o "cociente" mais grande. Unha vez atribuido ese escano, o "cociente" correspondente xa non se volve ter en conta no proceso. O proceso finaliza no momento en que se repartiron os N escanos.

O programa deberá ler un arquivo solicitado ó usuario cos dados do escrutinio das eleccións, e presentar en pantalla o reparto de escanos aplicando os criterios citados anteriormente.

CÁLCULO NUMÉRICO

EN MATLAB/OCTAVE

Peregrina Quintela Estévez (1960)

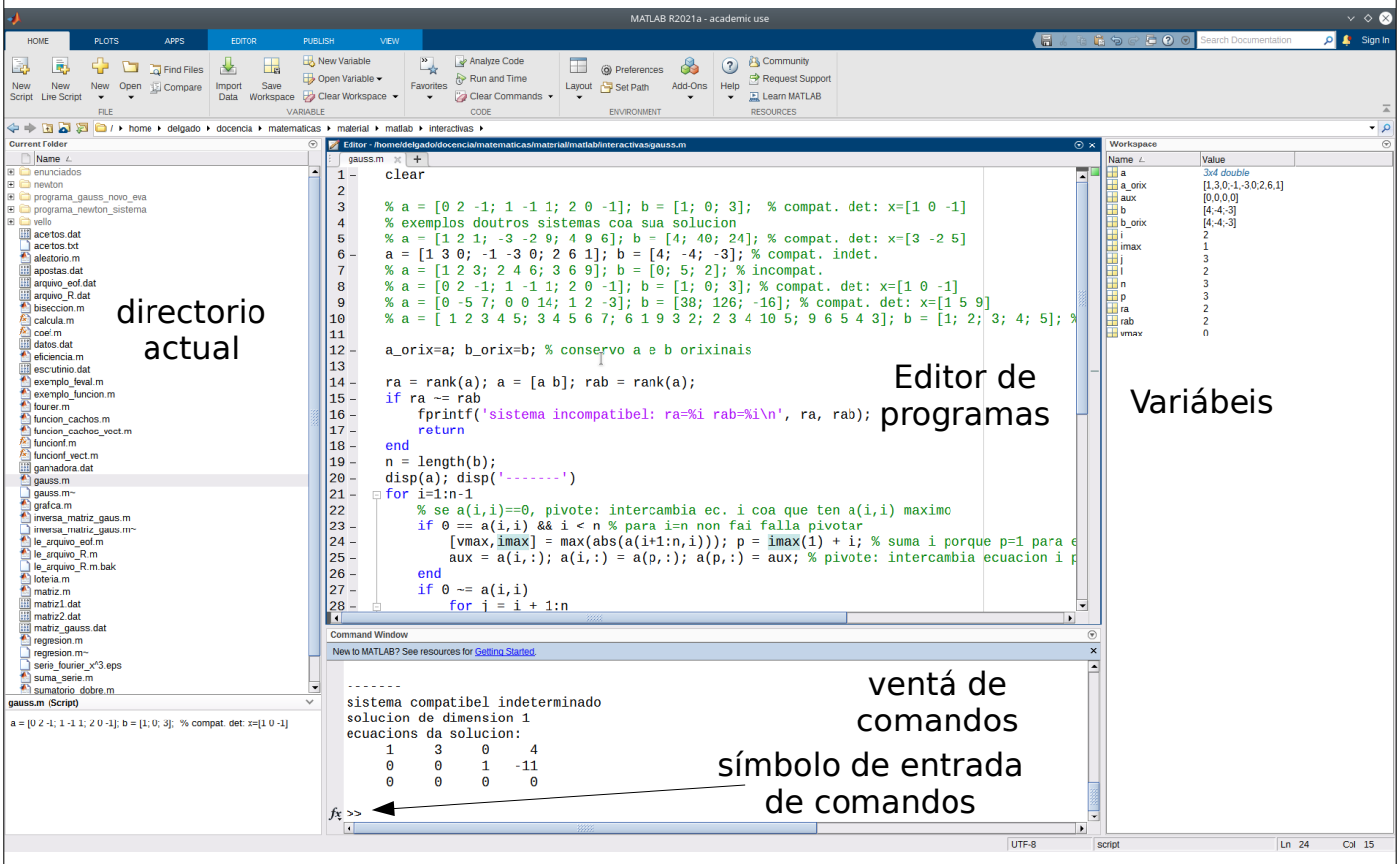


- Catedrática de Matemática Aplicada da USC
- Directora do Instituto Tecnolóxico de Matemática Industrial da USC
- Escritora de varios libros sobre Matlab

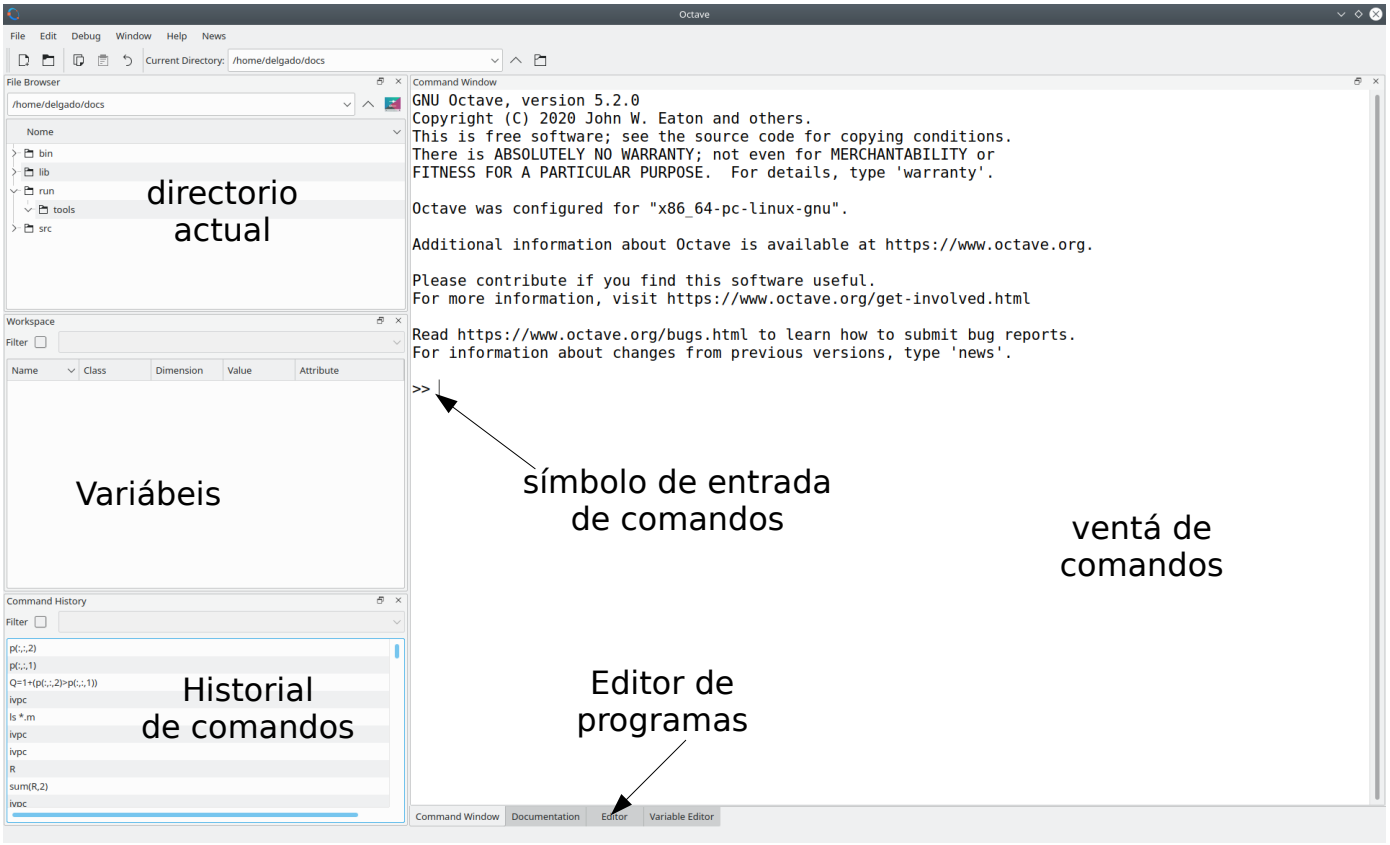
Características de Matlab

- Linguaxe de cálculo científico e numérico, visualización e programación
- Octave: versión libre de Matlab
- Librerías de funcións moi amplas
- Cálculos matemáticos
- Desenvolvemento de algoritmos
- Análise e representación gráfica de datos
- Simulación
- Desenvolvemento de interfaces de usuario

Interface gráfica de Matlab



Interface gráfica de Octave



Comandos básicos

- Execución de operacións: *ans* é unha variábel predefinida que almacena o resultado da última operación (se éste non se almacena noutra variábel)
- Comando rematado en ; non mostra o resultado
- Repetición de comandos anteriores: ↑
- *clc*: limpia a ventá de comandos
- *clear*: borra a memoria (workspace)
- Pódense encadear varias ordes con ;

  x=-1:0.1:1;plot(sin(x))

Variábeis (I)

- Variábeis: non hai declaración, só hai que asignarlle un valor; antes desta asignación, non existe, e non pode ser referenciada (erro)
  - Enteiras e reais (con / sen expoñente)
  - Complexas: i,j= unidade imaxinaria: 2+2*i;
- Os nomes poden conter letras, números e o signo "_", pero só poden comezar por letras. Non poden ter signos especiais (+&%$(/?*, etc.). Matlab distingue entre maiúsculas e minúsculas
- Almacénanse internamente como reais de dobre precisión (8 bytes, 16 cifras decimais, rango $\pm 10^{\pm 308}$)

Variábeis (II)

- Comando *diary*: almacena a historia de comandos
- *diary ficheiro.txt*: comeza a almacenar en *ficheiro.txt*
- *diary off*: remata o almacenamento
- Variábeis predefinidas: *ans*, *pi*, *eps* (menor diferencia entre números=-1.2E-16), *inf* (∞), *i*, *j*, *NaN* (Not a Number: 0/0), *realmax/realmin* (nº real máximo e mínimo)
- Asignación de valor a unha variábel: *x = 5.4;*
- Cadeas de caracteres: entre comiñas simples: s = 'cadea de caracteres'

```
>> whos s
  Name      Size                Bytes  Class
  s         1x5                    10  char array
```

Funcións básicas (I)

- *sqrt*, *abs* (valor absoluto), *exp*, *log*, *log10*, *sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sinh*, *cosh*, *tanh*, *asinh*, *factorial*
- Redondeo de real a enteiro: *round* (cara enteiro máis cercano), *fix* (ídem cara 0), *floor*(ídem cara -inf), *ceil* (ídem cara +inf)
- Exemplo: a= [-1.9 –0.2 3.4 5.6 7 2.4+3.6i]
  round(a) →[-2 0 3 6 7 2 +4 i]
  fix(a) → [-1 0 3 5 7 2 +3 i]
  floor(a) →[-2 -1 3 5 7 2 +3 i]
  ceil(a) → [-1 0 4 6 7 3 +4 i]
- *conj(z)*: conxugado dun nº complexo z

Funcións básicas (II)

- *real*(z), *imag*(z): partes real e imaxinaria de z
- *factorial*(x): factorial de nº enteiro
- *rem*(x, y): resto de división enteira x/y
- *rats*(x): aproxima x polo nº racional máis cercano
- *factor*(x): factores primos dun nº enteiro
- *isprime*(x): determina se x é primo
- *primes*(x): números primos menores que x
- Poden operar sobre vectores e matrices (operan elemento a elemento)

Formatos e operacións aritméticas

- Formatos: comando *format*:
  - *short* (*short e*): 5 decimais (exponencial)
  - *long* (*long e*): 15 decimais (exponencial)
  - *compact*: suprime liñas en branco
- Operacións aritméticas: +-*/^ (*power(x,y)*=x^y; *nthroot(x,y)*=$\sqrt[y]{x}$).
- Prioridades: as usuais: ^ */ +-
- Axuda: *help/doc comando*, tecla F1
- Tempos: *tic* (inicializa reloxio) e *toc* (mide o tempo transcurrido dende *tic*); *cputime*, *etime*, *clock*

Vectores

- Almacenamento de comandos en ficheiro: *diary ficheiro.txt*; *diary on*; *diary off*.
- Definición entre corchetes: *v = [1 2 3]*: *e*lementos separados por espacios ou comas. Separación entre filas mediante ;
- Vector columna: *v = [1;2;3]*
- Trasposición dun vector: *v*'
- Definición con compoñentes equiespaciadas (v_{i+1}-v_i=*cte*) nun intervalo [a,b]: *v=a:paso:b* (por defecto *paso*=1): *v= 0:0.1:1*: elementos de 0 a 1 separados 0.1
- *linspace(a,b,n)* *n*=lonxitude
- Vector con compoñentes logarítmicamente espaciadas: *v = logspace(a,b,n)*: *n* mostras logarítmicamente equiespaciada entre 10^a e 10^b: ($log_{10}x_{i+1}$ – $log_{10}x_i$=*cte* independente de i):

$$\log_{10} v_i = \frac{(b-an)+i(a-b)}{1-n}; i=1,\ldots,n$$

Acceso e edición dun vector

- Acceso a elementos dun vector:
  - v(1) elemento nº 1
  - v(*end*) último elemento
  - v(1:5) elementos de 1 a 5
  - v(1:2:10) elementos de 1 a 10 de 2 en 2
  - v(:) o vector completo
  - v(*1*:*end*~=*k*): o vector menos o elemento *k*-ésimo
- Adición / supresión de elementos:
  - Adición de elementos: v = [v 5 6]. Tamén: v=1:3; v(6)=9
  - Concatenación de vectores; v=[1 2 3];w=[4 5 6]; *z=[v w]* ou *z=[v' w']*
  - Supresión de elementos: v(5:8) = [];
- Lonxitude dun vector: *length(v)*; nº elementos: *numel(v).*
- Producto escalar de 2 vectores: *dot(v,w)*, *v*w'* ou *sum(v.*w)*

Funcións con vectores

- Lectura de vector/matriz dende arquivo: *load datos.dat;* ou ben v=*load('datos.dat')*.
- Almacena en vector/matriz *datos* ou *v*. O arquivo debe conter unha matriz numérica (non *char*). Tódalas liñas coa mesma cantidade de valores.
- Suma/producto de elementos dun vector: *sum(v)*, *prod(v)*
- *min(v)* e *max(v)*: valores mínimo e máximo dun vector.

  [vmax imax] = max(v): valor máximo e índice do máximo

  [~,imax]=max(v): só o índice do máximo
- *sort(v): ordea un vector por orde crecente (con matrices, ordea cada columna); sort(v, 'descend') ->* orde decrecente; [v2,i]=sort(v): v2=vector ordeado, i=vector cos índices dos elementos de v ordeados
- *mean(v)*, *var(v)*, *std(v), median(v)*: media, varianza, desviación típica e mediana.
- *unique(v)*: elementos non repetidos de *v* ordeados (crecente).
- Invertir un vector: *flip*(1:4) -> 4 3 2 1

Matrices

matriz=[elem 1ª fila; ...; elems. nª fila]

- *a = [1 2 3; 4 5 6; 7 8 9]* : matriz 3x3: columnas separadas por ;
- Tipo da matriz (nº filas e cols): *whos a*

Name	*Size*	*Bytes*	*Class*
a	*3x3*	*72*	*double array*

Ollo: podes escribir a(1,2) ou a(5), onde o índice incrémentase por columnas

- *eye(m, n)*: matriz identidade; *eye(n)*: cadrada identidade
- *zeros(m, n)*: matriz *mxn* con ceros; *3+zeros(m,n)*: con 3s
- *ones(m, n)*: matriz *mxn* con 1s; *5*ones(m, n)*: con 5s
- *rand(m, n)*: con valores reais aleatorios en [0, 1]

  *a + (b – a)*rand(m, n)* : aleatorios en [a, b]
- *randi([m n],nf,nc)*: matriz *nf* x *nc* con valores enteiros aleatorios entre *m* e *n; randi(n,nf,nc): valores entre 1 e n*
- Inicializa xerador de números aleatorios:
  - *rng('default')* ← Sempre igual
  - *rng('shuffle')* ← Distinto (co tempo)

Octave:
rand('*seed*',0)
rand('*seed*','*reset*')

Acceso a elementos dunha matriz e inserción / borrado de filas e columnas

- a(1,2): elemento 1ª fila e 2ª columna
- a(5): elemento 5º percorrendo por columnas
- a(1,:): elementos da 1ª fila
- a(:, 2): elementos da 2ª columna
- *a(1:2,2:3)*: [a_{12} a_{13}; a_{22} a_{23}]
- *a = rand(10,10); a(1:2:5, 2:3:10)*

$$\begin{bmatrix} a_{12} & a_{15} & a_{18} \\ a_{32} & a_{35} & a_{38} \\ a_{52} & a_{55} & a_{58} \end{bmatrix}$$

- Adición dunha fila: *a=ones(3);a=[a; [1 2 3]]*. Ou ben: a=ones(3);a(5,:)=[1 1 1]
- Adición de columna: *a=[a [1;2;3]]* ou a(:,6)=zeros(3,1)
- Supresión dunha fila: *a(1,:)=[]*
- Supresión dunha columna: *a(:,1)=[]*

Funcións con matrices (I)

- Tamano: *[nf nc] = size(a); nf = size(a,1); nc=size(a,2);*
- *numel(a)*: nº elementos de matriz; *length(a)*: *max*(*nf*,*nc*)
- Matriz diagonal cun vector: *v=[1 2 3];a= diag(v)*
- Vector coa diagonal dunha matriz: *v = diag(a)*
- *diag(diag(a))*: matriz diagonal coa diagonal de *a*
- *magic(n)*: matriz cadrada máxica de orde *n* (igual suma de filas e columnas).
- Suma de elementos por columnas: *sum(a)* ou *sum(a,1); por filas*: *sum(a,2)* ou *sum(a')*; suma completa: *sum(a(:))* ou *sum(sum(a))*
- Producto de elementos por columnas/filas: *prod(a)/prod(a,2)*
- Triángulo superior / inferior: *triu(a)* / *tril(a)*

Transformación de matriz en vector ou matriz doutra orde

- Útil para transformar unha matriz nun vector e procesar os seus elementos cun so bucle, evitando bucles dobres.
- Conversión de matriz *a* en vector fila por columnas: *v=a(:)'*. Se queres por filas: *b=a';b*(:)'
- Función *reshape(matriz,nf,nc)*: transforma a matriz noutra de orde *nf* x *nc* por columnas. O número *nf* x *nc* debe ser igual ao nº de elementos de *a*.
- Se queres que sexa por filas: *reshape(a',nf,nc);*
- Transformar de matriz a vector por filas: *v=reshape(a,1,nf * nc);* Se queres por columnas: *v=reshape(a',1,nf * nc);*
- Transformar *a* -> *b* (con *n* filas): *b=reshape(a,n,[]);* o número *n* debe ser divisor do nº de elementos de *a*; o nº de columnas será o necesario para almacenar os *nf* x *nc* elementos de *a* nunha matriz *b* de *n* filas.

Repetición dunha matriz

- Función *repmat*(*a*,*n*,*m*): repite a matriz a *n* veces verticalmente e *m* veces horizontalmente
- Ex: *a=[1 2;3 4]*

  repmat(a,2,3): repite a matriz dúas veces verticalmente e 3 horizontalmente:

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

Operacións con matrices (I)

- Operacións por compoñentes: punto antes do operador: *a.*b, a./b, a.^b*: ambas matrices deben coincidir en nº de filas e de columnas
- Operacións matriz-escalar:
  - Suma / resta / produto / cociente con escalar: tódolos elementos da matriz se operan co escalar
  - Cociente escalar-matriz por compoñentes: $b=k./a \rightarrow b_{ij} = k/a_{ij}$
  - Potencia escalar-matriz por compoñentes: $b=k.\hat{}a \rightarrow b_{ij} = k^{aij}$
  - Potencia matriz-escalar por compoñentes: $b=a.\hat{}k \rightarrow b_{ij} = a_{ij}^{k}$
  - Potencia matriz-escalar matricial: *b=a^k* (*a·…·a*, *a* debe ser cadrada)

Operacións con matrices (II)

- Operacións entre matrices:
  - Suma *a* + *b* e resta *a* - *b*: *a* e *b* deben coincidir en nº de filas e de columnas
  - Producto matricial: *a***b*: o nº de columnas de *a* debe coincidir co nº de filas de *b*
  - Producto por compoñentes: *c*=*a*.**b*: *a* e *b* deben coincidir en nº filas e de columnas, e $c_{ij} = a_{ij} \cdot b_{ij}$
  - División matricial a esquerda: $a\backslash b \rightarrow a^{-1} \cdot b \rightarrow pinv(a)*b$
  - División matricial a dereita: $a/b \rightarrow a \cdot b^{-1} \rightarrow a*pinv(b)$
  - Cociente por compoñentes: $c=a./b \rightarrow c_{ij} = a_{ij}/b_{ij}$
  - Potencia por compoñentes: $c=a.\hat{}b \rightarrow c_{ij} = a_{ij}\hat{}b_{ij}$

Funcións con matrices (II)

- *unique(a)*: elementos non repetidos de *a* ordeados (crecente) como vector columna.
- Determinante dunha matriz cadrada: *det(a)*.
- Inversa dunha matriz cadrada: *inv(a)*, so cando $det(a) \neq 0$.
- Pseudoinversa de Moore-Penrose: *pinv(a)*, existe para matrices non cadradas e cadradas con *det(a)=0*.
- Nun sistema de ecuacións lineares a*x=b -> x=inv(a)*b. Se x non existe ou hai infinitas (*det*(*a*)=0 ou *a* non cadrada), x=pinv(a)*b verifica que |a*x-b| é mínima (solución de erro, non nulo, mínimo).
- Autovalores dunha matriz cadrada: *v=eig(a)*
- Autovectores: *[v d] = eig(a)*
  - *v*= matriz con autovectores de matriz *a* por columnas
  - *d*=matriz diagonal con autovalores de matriz *x*: $det(a - d_{ii}\mathbf{1}) = 0$; $xv_i^T = d_{ii}v_i$ (v_i= columna *i* de *v*), *i=1,...,n*

Funcións con matrices (III)

- Mínimo e máximo:
  - Por columnas: *min(a)* e *max(a)*
  - Por filas: *min(a,[],2)* ou *min(a')*, menos eficiente (análogo para *max*)
  - Matriz completa: *min(a(:))* ou *min(min(a))*
- Valores mínimos/máximos e índices dos elementos mín/máx:
  - Por columnas: *[v,i]=min(a)*
  - Por filas: *[v,i]=min(a,[],2)*
  - Matriz completa: *[v,i]=min(a(:))*
  - *v*: vector con valores mínimos
  - *i*=vector con índices de elementos mínimos
- Índices de fila e columna do elemento mínimo/máximo dunha matriz:

[~,i]=min(a(:)); [f,c]=ind2sub(size(a),i)

Funcións con matrices (IV)

- Media, varianza desviación típica e mediana:
  - Por columnas: *mean(a), var(a)*, *std(a*), *median(a)*:
  - Por filas: *mean(a,2), var(a,[],2)*, *std(a,[],2*), *median(a,2)*
  - Matriz completa: *mean(a(:)), var(a(:)), std(a(:)), median(a(:))*
- Ordeamento:
  - Por columnas: *sort(a)*, *sort(a,'descend')*
  - Por filas: *sort(a,2), sort(a,2,'descend')*
  - Matriz completa: *sort(a(:))*, *sort(a(:),'descend')*
- Matrices dispersas (moitos elementos nulos): *a = sparse(i, j, c, m, n); full(a)*: mostra matriz; i(j) = índices de filas (columnas) de elementos non nulos; c= vector con valores de elementos non nulos; m(n)= nº filas(columnas)

Resolución dun sistema de ecuacións lineais

- Sexa o sistema en forma matricial **b** = **Ax**, con *n* ecuacións e *n* incógnitas
- Resolución en Matlab:

```
a = [a11 ... a1n; ...; an1 ... ann];
b = [b1; ...; bn];
rank(a)
rank([a b])
x = a\b          % se rank(a)==rank([a b])
x = inv(a)*b     % alternativa
```

rango dunha matriz: se *rango(a)==n*, o sistema é compatíbel determinado; se *m=rango(a)<n*, o sistema é compatíbel indeterminado (se *rango([a b])=m*) ou incompatíbel (se *rango([a b])≠m*)

- Se o sistema é **incompatíbel**, a pseudoinversa *x=pinv(a)*b* permite atopar unha solución de norma mínima, é dicir, *norm(a*x-b)* é mínima, aínda que ≠0 e pode ser elevada

x+2y+3z=0
2x+4y+6z=5
3x+6y+9z=2

Non existe **x** con **Ax**-**b**=**0**
x=$\mathbf{A}^{+}$**b** verifica |**Ax**-**b**| é mínima

Sistema compatíbel indeterminado (I)

- As infinitas solucións pódense escribir como unha **solución individual do sistema** máis unha **combinación linear de solucións do sistema homoxéneo** asociado.
- Podes calcular unha **solución individual** usando a matriz pseudoinversa: ***x0=pinv(a)*b***. Podes comprobar que é solución calculando *norm(A*x0-b)*.
- **Solucións do sistema homoxéneo**: cerne da aplicación linear asociada á matriz dos coeficientes: ***k=null(a)*** retorna unha matriz onde cada columna é un vector dunha base ortonormal deste subespazo linear (nulo).
- **Solución xeral** do sistema indeterminado: ***x0+k*c***, sendo *x0* unha solución individual e *c* o vector de coeficientes da combinación linear (p.ex. *c=ones(r,1)*, sendo *r=size(k,2)* a dimensión do espazo solución (nº columnas de *k*).

Sistema compatíbel indeterminado (II)

- Sistema *ax=b* con *a=[1 2 3;4 5 6;7 8 9]; b=[1;2;3]*
- *rank(a)=2,rank(b)=2<3: s*ist. compat. indet.
- A solución ten dimensión 3-2=1 (é unha recta en $\mathbb{R}^3$)
- *x0=pinv(a)*b* : solución individual
- *k=null(a)*: *k*=vector columna director da recta
- Solucións da forma *x=x0+c*k* onde *c*=escalar
- O vector *x* é solución porque *norm(a*x-b)* $\simeq$ 0

Exercicios

1) Define un vector x con 10 compoñentes espaciadas logarítmicamente entre 1 e 100; suprímelle as compoñentes 3-5; engádelle o vector [3 4 5] polo comezo; selecciona os elementos de índice múltiplo de 3; calcula a lonxitude de x
2) Calcula a suma, producto, máximo e mínimo, media e desviación típica do vector x do exercicio anterior
3) Crea co editor de Matlab un arquivo de *datos.dat*. Cárgao en Matlab ao vector x e representa as dúas columnas de x
4) Define os vectores (1,2,3,4,5) e (5,4,3,2,1) e calcula o seu producto escalar

Exercicios

5) Crea unha matriz 3x3 con elementos=1 e outra 2x2 con elementos=5. Logo pégaas e obtén a seguinte matriz:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 5 & 5 \end{bmatrix}$$

6) Define unha matriz de orde 3x4 con números aleatorios no intervalo $[-1, 1]$:

7) Dada unha matriz A cadrada de orde 5: selecciona a submatriz de A coas filas 2-3 e as columnas 1-3; amplía a matriz engadíndolle unha fila ao comezo da matriz; bórralle as filas 1 e 4

8) Dadas as matrices A e B:

$$A=\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \quad B=\begin{bmatrix} 1 & 4 & 5 \\ -3 & 2 & 1 \\ 0 & 5 & 4 \end{bmatrix}$$

calcular $A \cdot B$, $A^{-1} \cdot B$, $A \cdot B^{-1}$, $|A|$, suma,min e max por columnas de A; triángulo superior e inferior de B

Exercicios

9) Define unha matriz dispersa 10x10 con valores non nulos (8,1)=-3 e (3,4)=-9

10) Define unha matriz 5x5 con valores aleatorios no intervalo [-3,1]

11) Partindo da matriz identidade 7x7 e usando o : obtén a seguinte matriz:

$$\begin{bmatrix} 2 & 2 & 2 & 0 & 5 & 5 & 5 \\ 2 & 2 & 2 & 0 & 5 & 5 & 5 \\ 3 & 3 & 3 & 0 & 5 & 5 & 5 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \end{bmatrix}$$

12) Crea unha matriz 5x7 coa 1ª fila 1 2 3 4 5 6 7, 2º fila 8 9 10 11 12 13 14, 3ª fila 15-21, etc. A partir dela crea outra matriz 3x4 coas filas 2-4 e columnas 3-6 da matriz orixinal

13) Resolve este sistema de ecuacións:

$$\begin{aligned} -44x+10y+16z &= 20 \\ 10x-43y+6z+12t &= 0 \\ 16x+6y-30z+8t &= 12 \\ 12y+8z-34t &= -40 \end{aligned}$$

Solucións aos exercicios (I)

1)x=logspace(1,2,10);x(3:5)=[];x=[3 4 5 x];
x(3:3:10);length(x) ou size(x,2)

2)sum(x);prod(x);max(x);min(x);mean(x);
std(x)

3)x=load('datos.dat');plot(x(:,1),x(:,2),'o-')

*4)x=[1 2 3 4 5]; y=[5 4 3 2 1];dot(x,y);x*y'*

*5)a=ones(3,3);b=5*ones(2);[[a zeros(3,2)] ;*
[zeros(2,3) b]]

*6)a=-1+2*rand(3,4)*

7)a=magic(5);a(2:3,1:3);a=[ones(1,5); a];a(1:3:4,:)=[];

Solucións aos exercicios (II)

8)a=[1 2 3; 0 1 2; 0 0 1]; b=[1 4 5; -3 2 1; 0 5
*4];a*b; a\b ou inv(a)*b; a/b ou a*inv(b);*
det(a);sum(a);min(a);max(a);a-tril(a);a-triu(a)

9)a=sparse([8 3],[1 4],[-3 -9],10,10); full(a)

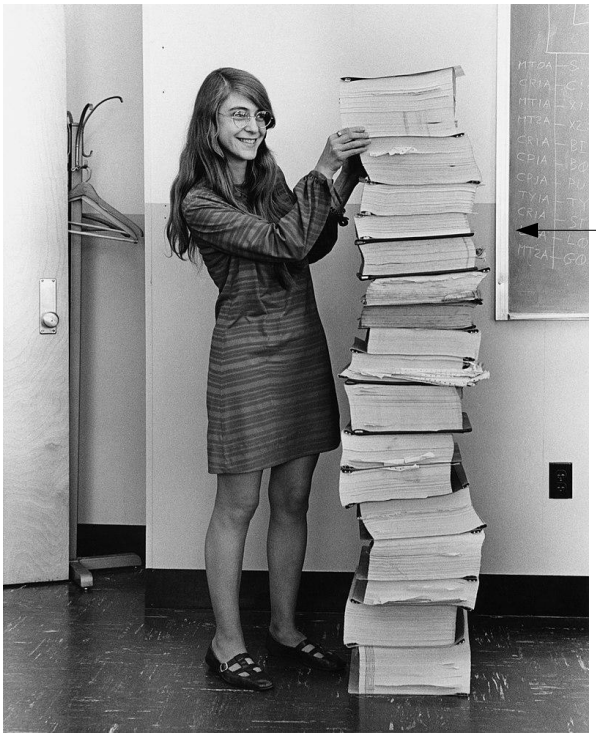
*10)a=-3+4*rand(5)*

11)a=eye(7);a(1:2,1:3)=2; a(3,1:3)=3; a(1:3,5:7)=5;
a(5:7,1:2)=4; a(5:7,3)=7; a(5:7,5:7)=9

12)a=zeros(5,7);x=1;for i=1:5; for j=1:7; a(i,j)=x;
x=1+1;end; end; b=a(2:4,3:6)

13)a = [-44 10 16 0; 10 -43 6 12; 16 6 -30 8; 0 12 8 -
34]; b = [20;0;12;40];a\b

Margaret Hamilton (1936)



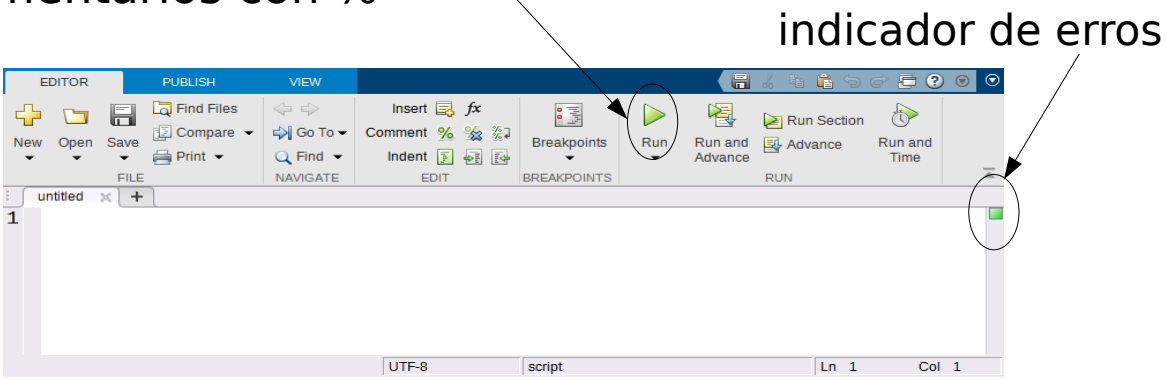
- Primeira enxeñeira do software
- Desenvolveu o **software de navegación** para o programa espacial EEUU (años 60)
- Xefa da equipa de programación da NASA para a viaxe á lúa
- Medalla Presidencial de Liberdade (2016)

Programas

- Ficheiros coa extensión .m: conteñen comandos que se executan secuencialmente
- Execución: escribe o seu nome (sen .m) na ventá de comandos
- O arquivo debe estar no directorio actual (ou nun directorio incluido na variábel *path*, que se pode consultar co comando *path*)
- Pódese engadir directorios a *path* no menú *File* submenú *Set Path* ou co comando *addpath(dir)*
- Execución alternativa: na ventá de directorio, seleccionar arquivo e *Run* (F5) no menú contextual
- Ou dende o editor de Matlab, menú *Debug -> Run*

Edición do programa

- Botón *New* en barra botóns: abre o editor de Matlab (dende o cal tamén se pode executa-lo programa)
- Execución: botón *Run* (F5) no editor ou escribi-lo nome do programa + Intro na ventá de comandos
- Permite depura-lo programa durante a execución
- Comentarios con %



Execución do programa (I)

- Matlab: linguaxe interpretado (o programa necesita ao Matlab para executarse)
- Non hai erros de compilación (non hai compilación): só erros de execución e léxicos
- Matlab só atopa un erro de sintaxe cando executa o programa e chega ao erro
- Nembargantes, o editor de Matlab indica con cores vermello, laranxa e verde se o programa ten erros (vermello), advertencias (laranxa) ou é correcto (verde) antes de executalo
- Pode haber erros aínda que o indicador sexa verde

Execución do programa (II)

- Co Matlab dende a terminal de comandos de Linux (ponlle *exit* ao final do programa para que o Matlab retorne á terminal):

  matlab -nosplash -nodesktop -r programa

- Dentro do Octave: escribe o nome do programa sen a extensión *.m*
- Co Octave dende a terminal de comandos de Linux:

  octave programa.m

- Outra forma co octave dende a terminal de Linux:

  1) Engade a liña *#!/usr/bin/octave* ao comezo do programa para indicarlle ao *bash* que interprete o programa co Octave.
  2) Na terminal (*bash*), executa *chmod u+x programa.m* para darlle permiso de execución.
  3) Executa *programa.m* dende a terminal.

Depurador de Matlab

- No editor de Matlab, abre o programa a depurar
- Establece un punto de ruptura (breakpoint) pulsando no marxe esquerdo na liña desexada
- Executa o programa (F5 ou menú Debug->Run no editor): detense a execución no punto de ruptura
- Podes inspecciona-las variábeis poñendo o rato sobre o variábel no programa (ou no workspace)
- Podes executa-lo programa sentenza a sentenza con F10 (ou menú Debug->Step)
- Entra nunha función: F11 ou Debug->Step into
- Continua a execución: F5 ou Debug->Continue

Variábeis e entrada de datos

- O programa usa as variábeis globais (as do workspace)
- Recomendábel executar *clear* ao comezo do programa (borra as variábeis existentes)
- Entrada de datos por teclado:

var = input('introduce un valor: ');

cadea = input('introduce unha cadea: ', 's');

- Exemplo:

x = input('introduce x: '):

- Podes introducir un vector/matriz entre corchetes.

Saída de información por pantalla

- Comando *disp*:

disp(var); disp('mensaxe de texto');

- Comando *fprintf*:

fprintf('formato', var1, ..., varN);

  - 'formato': cadea con caracteres, e tamén ...
  - códigos de formato (ver páx. seguinte)
  - secuencias de control: \n para nova liña, \t para tabulador, \r para retorno de carro, \b para borrar un carácter impreso, \\ (carácter '\'), %% (carácter '%'), \” (carácter “);

Códigos de formato

- **%c**: carácter simple
- **%s**: cadea de caracteres
- **%i** ou **%d**: enteiro; **%5i** para enteiro de ancho 5.
- **%f**: nº real sen expoñente; %.6f: con 6 decimais; %10.3f: con ancho 10 (incluindo o punto decimal) e 3 cifras decimais
- **%e**: nº real en formato exponencial; %10.2e: real exponencial con ancho 10 e 2 decimais (tamén con %n.dE, neste caso E no expoñente)
- **%g**: nº real na forma máis compacta entre e/f
- Exemplo: *x=3.5;t=5; s='ola';*
  fprintf('x= %.3f x=%.2e t= %4i s=%s\n', x, x, t, s);

Función *fprintf*

- *fprintf* está vectorizada: se unha variábel é vector ou matriz, repítese a función até que se imprimen tódolos elementos (por columnas) na mesma liña, agás que se poña \n.
- Ex: *x = [1 2; 3 4]; fprintf('x=%i\n', x);*
  x=1 % en cada liña por ter \n
  x=3
  x=2
  x=4
- Ex: *fprintf('%i ', x);*
  1 3 2 4 % na mesma liña por non ter \n

Función *sprintf*

- *fprintf* tamén permite almacenar nun arquivo
- A función *sprintf* opera igual que *fprintf* pero non mostra a cadea por pantalla, senón que retorna a cadea formateada, para logo facer cousas con ela:

  s=sprintf('cadea formato', var1, ..., varn)

- Útil cando se quere manipular cadeas (concatenar con outras, etc.)
- Ex: *s = sprintf('x= %i y= %f\n', x, y);*
  mensaxe=[s ' ' sprintf('a= %c\n', a)];
  disp(mensaxe)

Estrutura de selección básica

- Similar a IF de Fortran
- Avalía unha condición definida polos operadores >, >=, <, <=, ==, ~=

```
if x<=0
    disp('baixo);
elseif x<=1
    disp('medio');
else
    disp('alto');
end
```

- Sentenza IF-ELSE IF:

```
if condición1
    sentenzas1;
elseif condición2
    sentenzas2;
...
else
    sentenzasN;
end
```

Estrutura iterativa básica

- Definida:

```
for var = ini:paso:fin
    sentenzas;
end
```

Similares a *do* definido e indefinido e *do-while* de Fortran

```
for k = 1:10
    fprintf('k= %i\n', k);
end
```

- Indefinida:

```
while condición
    sentenzas;
end
```

```
n=0; suma=0;
while n ~= -1
    n=input('introduce n (-1 para rematar)');
    suma = suma + n;
end
```


Remate dun programa

- *return*: remata a execución do programa (ou retorna dende unha función)
- *error*('*mensaxe*'): remata a execución e mostra a *mensaxe* de erro en cor vermella (que pode estar formateada como con *fprintf*):
  - Ex: *error('erro: x=%i < 0!\n', x)*
- *break*: remata un bucle *for*/*while* (análogo a *exit* en Fortran) cando se cumpre unha condición
- *break* remata a execución se está fora dun bucle
- Se usas *exit,* remata o Matlab (isto é útil se executas o programa dende a terminal de comandos)

Frances Allen (1932-2020)

- Pioneira na optimización de compiladores, optimización automática de código e programación paralela
- Creadora de linguaxes de programación e códigos de seguridade para a NSA
- Premio Turing de Informática en 2006

Operadores relacionais

- Operadores relacionais: > (maior) >= (maior ou igual) < (menor) <= (menor ou igual) == (teste igualdade), ~= (teste desigualdade)
- Se comparamos escalares, o resultado é 1 (certo) se se cumpre o teste ou 0 (falso) se non se cumpre
- Se comparamos matrices (deben ser da mesma dimensión en filas e columnas), a comparación faise elemento a elemento
- O resultado é unha matriz de 0s (nos elementos onde falla o teste) e 1s (nos elementos onde se cumpre) coa mesma dimension cas orixinais
- Precedencia: todos teñen a mesma, e avalíanse de esquerda a dereita

Operadores lóxicos

- NOT Lóxico: ~: P. ex: ~*x* da 1 se *x* é 0, e 0 se x é distinto de 0
- AND lóxico: & (para vectores/matrices), && (para escalares)
- OR lóxico: | (para vectores/matrices), || (para escalares)
- Operandos numéricos (0 é falso, ≠0 é certo)
- Con escalares, dan 0 ou 1; con matrices, operan elemento a elemento e dan unha matriz da mesma orde
- Se actúan cun escalar e unha matriz, cada elemento da matriz opérase co escalar

Función lóxica *all*

- *all(x)*: retorna 1 se tódolos elementos do vector *x* son non nulos, 0 se algún elemento de *x* é nulo
- *all*(*a*), *all*(*a*,*1*): vector de lonxitude *size*(*a*,*2*), ou sexa, nº de columnas de *a*, con valores 1 nas columnas de *a* con tódolos elementos non nulos e 0 nas restantes
- *all*(*a*,*2*): vector de lonxitude *size*(*a*,*1*), nº de filas de *a*, con valores 1 nas filas de *a* con tódolos elementos non nulos e 0 nas restantes
- *all*(*all*(*a*)) ou *all*(*a*(:)): retorna un número, que é 1 se tódolos elementos de *a* son non nulos, ou 0 se a ten algún elemento nulo
- *all* pode aplicarse a unha expresión: *all(rem(a,2)==0)* vale 1 se tódolos elementos de *a* son pares

Funcións lóxicas *xor* e *any*

- *xor(a,b)*: retorna 1 se un dos operandos é 0 e o outro non, ou viceversa; con vectores/matrices opera elemento a elemento
- *any(x)*: retorna 1 se algún elemento do vector *x* é non nulo
- *any*(*a*), *any*(*a*,1): vector de lonxitude *size*(*a*,*2*) con valores 1 nas columnas de *a* con alomenos un elemento non nulo e 0 nas restantes
- *any(a,2)*: vector de lonxitude *size*(*a*,*1*) con valores 1 nas filas de *a* con alomenos un elemento non nulo e 0 nas restantes
- *any(any(a))* ou *any(a(:))*: actúa para toda a matriz *a*
- *xor* ou *any* poden aplicarse sobre expresións: *any(a>2)*

Función *find*

- *find(v)*: retorna os índices dos elementos non nulos do vector *v*
- *find(v>0 & v<5)*: índices dos elementos no intervalo [0,5]
- *v(v>0)* ou *v(find(v))* => elementos non nulos
- *[i,j]=find(rem(a,2)==0)*: índices de fila (*i*) e columna (*j*) dos elementos pares da matriz *a*
- *find(v>0,1,'first')*: índice do primeiro elemento positivo de vector *v*
- *find(isprime(v),3,'last')*: índices dos 3 últimos elementos primos de *v*

Sentenzas de selección (I)

As condicións elabóranse con operadores relacionais e lóxicos

- Sentenza IF:

```
if condición
    sentenzas;
end
```

- Cunha soa sentenza:

```
if condición; sentenza; end
```

- Sentenza IF-ELSE

```
if condición
    sentenzas1;
else
    sentenzas2;
end
```

- Sentenza IF-ELSE IF:

```
if condición1
    sentenzas1;
elseif condición2
    sentenzas2;
...
else
    sentenzasN;
end
```

```
if condición; sentenza1; else; sentenza2; end
```


Sentenzas de selección (II)

- Sentenza *switch*:
- Compara a expresión cos distintos valores *val1*... e executa as sentenzas asociadas ao valor co cal coindice
- Se non coincide con ningún valor, execútanse *sentenzasN* (isto é opcional, pero permite aforrar un caso)

```
switch expresión
case val1
    sentenzas1;
case val2
    sentenzas2;
...
otherwise:
    sentenzasN;
end
```

Sentenza de iteración definida (I)

- Se *x* é un vector fila, *a* é unha matriz:

```
for i=x
    sentenzas
end
```

```
for i=a(:)'
    sentenzas
end
```

Poño a(:)' para que sexa un vector fila, porque a(:) é un vector columna

- Nas sentenzas, *i* percorre os elementos do vector *x* ou da matriz *a* (por columnas)

```
for i=x
    disp(i)
end
```

```
for i=a(:)'
    disp(i)
end
```

- Para percorrer *a* por filas:

```
b=a';
for i=b(:)'
    disp(i)
end
```


Sentenza de iteración definida (II)

- O vector x pode ser da forma *ini:paso:fin*

```
for var = ini:paso:fin
    sentenzas;
end
```

Cando so é unha sentenza

```
for var=ini:paso:fin; sentenza; end
```

- Pódense usar variábeis *i*, *j* (por defecto son a unidade imaxinaria: *i*^2=-1)
- Inicializa *var = ini*
- En cada iteración executa as sentenzas
- Se *var + paso > fin* rematan as iteracións.
- En caso contrario, executa *var=var+paso* e continúa coa seguinte iteración

Sentenza de iteración definida (III)

- Se *paso > 0*, entón debe ser *ini <= fin* para que haxa iteracións; se *paso < 0*, debe ser ini >= *fin*
- A *var* pódenselle asignar valores específicos. Ex:

```
for k = [1 3 6 -4]
    fprintf('k=%i\n', k);
end
```

- Non se lle debe cambia-lo valor a *var* dentro do bucle *for* (Matlab non o detecta)
- Exemplo: suma da serie $\sum_{k=0}^{n} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$

```
x=input('x?');n=input('nº de elementos? '); suma=0;
for k=0:n
    suma=suma+(-1)^k*x^(2*k+1)/factorial(2*k+1);
end
```


Bucle *for* con varios índices

- O *for* pódese poñer con dous ou tres índices (deben ser vectores fila)

```
i=1:3; j=4:6;
for k=[i;j]
    disp(k)
end
```

```
Iter. 1: k=[1;4]
Iter. 2: k=[2;5]
Iter. 3: k=[3;6]
```

```
i=1:3; j=4:6;
for k=[i;j]
    fprintf('%i %i\n',k(1),k(2))
end
```

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

- Con tres índices (caracteres):

```
x='abc'; y='def'; z='ghi';
for k=[x;y;z]
    fprintf('%c %c %c\n',k(1),k(2),k(3))
end
```

```
a d g
b e h
c f i
```

Exemplos de estruturas iterativas definidas

- Mostrar por pantalla un vector na mesma liña:

```
v=randi([vmax vmin],1,n);
fprintf('%7.3f ',v);fprintf('\n')
```

Estrutura iterativa implícita (vectorizada)

- Mostrar por pantalla unha matriz (unha fila en cada liña):

```
a=randi([vmin vmax],n,m);
for i=1:n
    fprintf('%10.2f ',a(i,:)); fprintf('\n')
end
```

- Cálculo do máximo dunha serie de números (para o mínimo debes inicializar *m=inf*):

```
m=-inf;
for i=1:n
    x=input('x? ');m=max(m,x);
end
```

Se os números están almacenados nun vector *x*: *min*(*x*), *max*(*x*)

Sentenza de iteración indefinida

- Sentenza *while* (iteración indefinida): ten unha condición para continuar coa iteración

```
while condición
    sentenzas;
end
```

```
while condición;sentenza; end
```

- A condición debe ter alomenos unha variábel (se é nula, a condición é falsa, e certa en caso contrario)
- As variábeis da condición deben inicializarse antes (en caso contrario, erro de execución)
- Dentro das sentenzas débese modifica-lo valor de alomenos unha das variábeis da condición: en caso contrario, entrará nun bucle infinito

Sentenza de iteración indefinida

- As modificacións nestas variábeis deben garantir que a iteración acade un final: en caso contrario, iteración infinita (isto non o detecta Matlab)
- Non poñer condicións de igualdade estricta entre variábeis con valores reais, xa que o redondeo poden levar a que nunca se cumpran
- Ex: suma de serie $\sum_{n=0}^{\infty} \frac{x^n}{n!}$ con sumandos > 0.0001

```
suma = 0;sumando = 1;n = 0;x=1;
while sumando > 0.0001
   sumando = x^n/factorial(n);
   suma = suma + sumando; n = n + 1;
end
```

```
n=1:1000;
sum(x.^n./factorial(n))
```

Versión vectorizada

Sentenzas *break* e *continue*

- Sentenza *break*:
  - Provoca o remate inmediato da estrutura iterativa, se está dentro dun bucle *for* ou *while*
  - Se non está dentro dun bucle, remátase o programa (p.ex., se se introducen datos inválidos)
- Sentenza *continue*: provoca o paso inmediato á seguinte iteración, saltando a execución do que resta da iteración actual
- O *continue* debe estar dentro dunha estrutura de selección (para que so se execute cando se cumpra a condición)

Exemplo de *break* en bucle dobre con matrices

- ¿Cómo rematar un bucle dobre?
- Exemplo: busca un elemento impar nunha matriz:

```
clear all
% a=matriz nxn enteira aleatoria en {1..10}
n=5;a=randi(10,n,n);impar=0;
for i=1:n
    for j=1:n
        if rem(a(i,j),2)==1
            impar=1;break
        end
    end
    if impar; break end
end
fprintf('elemento (%i,%i) con valor %g é impar\n',i,j,a(i,j));
```

Podes vectorizalo: *any(rem(a(:),2)==1)*
disp(find(rem(a(:),2)==1,1,'first'))

Sentenzas de iteración híbrida

- Teñen un número máximo de iteracións (parte definida) pero poden rematar antes se se cumpre unha condición (parte indefinida).
- Pódese facer cun *for+break* ou cun *while+and* lóxico.
- Exemplo: executa *n* iteracións pero remata cando *s>10*:

```
s=0;
for i=1:n
   s=s+x(i);
   if s>10
      break
   end
end
```

```
s=0;i=1;
while s<=10 && i<=n
   s=s+x(i);i=i+1;
end
```

Outro exemplo: método de Newton

```
x1=input('x1? ');i=1;dif=inf;
while dif>=1e-5 && i<=n
   x2=x1-f(x1)/df(x1);i=i+1;
   dif=abs(x2-x1);x1=x2;
end
```

```
x1=input('x1? ');
for i=1:n
   x2=x1-f(x1)/df(x1);
   if abs(x2-x1)<1e-5
      break
   end
   x1=x2;
end
```

- O *while+and* é útil para controlar os índices dun vector ou matriz, evitando superar os rangos dos seus índices.

Exemplo: Conversión dun vector por filas/columnas a unha matriz

- Manualmente (conversión por filas): vector *v* de *n* elementos, matriz *a* de orde *mxm*, con *m=ceil(sqrt(n))*

```
clear all
n=10;v=randi(10,1,n);m=ceil(sqrt(n));a=zeros(m);k=1;
for i=1:m
      for j=1:m
            a(i,j)=v(k);k=k+1;
            if k>n; break; end   % para evitar sairse do vector
      end
      if k>n; break; end
end
disp(v);disp(a)
```

- Para convertir por columnas: *a(j,i)=v(k)*
- Automáticamente: *reshape([1 2 3 4],2,2)* por columnas; *reshape([1 2 3 4],2,2)'* por filas. O vector debe ter exactamente o mesmo nº de elementos que a matriz

Exemplo: Conversión dunha matriz nun vector

- Manualmente: matriz *a* de orde *nxn* a vector *v* de $m=n^2$ elementos (conversión por filas):

```
clear all
n=5;a=randi(10,n,n);m=n^2;v=zeros(1,m);k=1;
for i=1:n
      for j=1:n
            v(k)=a(i,j);k=k+1;
      end
end
disp(a);disp(v)
```

- Para convertir por columnas: *v(k)=a(j,i)*
- Automáticamente: *a*(:), como vector columna; *a*(:)', como vector fila; *reshape(a,1,m)* por columnas; *reshape(a',1,m)* por filas.

Ann Zeilinger Karakristi (1921-2016)

- Especialista en criptografía con ordenadores da NSA (axencia de seguridade nacional dos EEUU)
- 2ª guerra mundial (contra Japón) e a guerra fría
- Directora adxunta da NSA (1980)
- Premio de servizos civís distinguidos dos EEUU

Funcións

- Pode estar nun arquivo *.m* co mesmo nome que a función:

```
function [ret1,...,retN] = nome(arg1,...,argM)
%nome: liña de axuda (liña H1)
%texto da axuda
sentenzas da función;
ret1=...;...;retN=...;
end
```

Se so hai un valor devolto, os corchetes sobran

- Pode chamarse dende calquera programa se o directorio onde se atopa está no path
- *arg1, ..., argM*: argumentos de entrada
- *ret1, ..., retN*: valores devoltos pola función
- A función pode estar tamén logo do programa principal. Esta función é **local**, é decir, non se pode chamar dende outros arquivos *.m*

```
clear
v=randi(20,1,10);
w=f(v);disp(w)

function y=f(x)
y=x.^2;
end
```

Estructura dunha función

- Cando chamamos a unha función, Matlab búscaa no arquivo actual, nos arquivos da carpeta actual e nos arquivos das carpetas do *path*
- Debe haber sentenzas que asignen valores a *ret1*, ..., *retN*
- En **octave**, a función tamén pode estar no programa principal, pero antes da chamada á función
- Se a función comeza na primeira liña do arquivo, éste será considerado un arquivo de función, e non un programa

```
clear

function y=f(x)
y=x.^2;
end

v=randi(20,1,10);
w=f(v);disp(w)
```


Exemplo de función

Función que calcule $\frac{1}{n}\sum_{k=1}^{n}\cos(kx)$ dados *x*, *n*

```
function y = f(x, n)
y = 0;
for k=1:n
    y = y + cos(k*x);
end
y=y/n;
%y = mean(cos((1:n)*x));
end
```

Título da función, variábel devolta e argumentos

Dáselle o valor á variábel devolta

Versión vectorizada

Exemplo de función (varios valores devoltos)

```
function [b x np ni] = funcionf(a)
[nf nc]=size(a);
b = zeros(nf,nc); x = zeros(1,nf);
for i=1:nf
   for j=1:nc
      b(i,j)=a(i,j)*a(j,i); % b_ij=a_ij a_ji
   end
   x(i) = a(i,:)*a(:,i); %prod. escalar fila i por columna i
end
i=mod(a(:),2);
np = sum(i==0); %nº elementos pares
ni = sum(i==1); % nº elementos impares
end
```


Función con varios valores devoltos

- Se non imos empregar algún dos valores devoltos pola función, na chamada á función podemos non almacenalos en ningunha variábel. Ex:
  - Corpo da función:

    ```
    function [z t] = calcula(x,y)
       z=x+y;t=x*y;
    end
    ```

  - Chamada a función: *[~, b]=calcula(3,4);*
- Neste exemplo, o primeiro valor devolto non se almacena en ningunha variábel.

Chamada á función

- A función pode ser chamada dende a ventá de comandos, dende outra función ou dende un programa (*script*)
- Chamada á función:

[var1 ... varN] = f(arg1, ..., argM);

Ex: a = randi(10,4,4);
[b x np ni] = f(a);

- Tamén se pode chamar á función dende unha expresión ou dende a chamada a outra función:

z = y + f(arg1, ..., argM);

fprintf('y = %f\n', sin(f(arg1, ..., argM)));

Función con varios puntos de retorno

- Sentenza *return*: provoca o retorno inmediato da función:

```
function y=le_arquivo(fich)
f=fopen(fich,'r');
if -1==f
    y=NaN;return
end
n=fscanf(f,'%g',1);y=zeros(n);
fclose(f);
end
```

- Pode haber varias sentenzas *return* (que poden estar asociadas a diferentes valores retornados) na mesma función.

Función con argumentos opcionais

- Matlab almacena o número de argumentos na variábel *nargin* (non tes que definila).

```
function z=exemplo(x,y)
   if nargin==2
      z=x+y;
   else
      z=2*x;
   end
end
```

```
function z=exemplo(varargin)
   if numel(varargin)==2
      z=varargin(1)+varargin(2);
   else
      z=2*varargin(1);
   end
end
```

- Chamada á función: *z=exemplo(2)* ou *z=exemplo(2,3)*
- Alternativa: argumento (vector) chamado *varargin*: tes que comprobar a súa lonxitude dentro da función.

Variábeis globais

- As variábeis dunha función son **locais** (non se coñecen fóra da función)
- Na función non se pode acceder ás variábeis do *workspace* nin doutras funcións
- Para acceder a unha variábel do *workspace* na función, hai que declarala **global** na función:
- Para acceder a unha variábel da función dende fóra (*workspace* ou outra función), hai que poñela como **global** onde se quera usar

```
x=5;y=6;
z=f(y)

function z=f(y)
global x
z=f+x
end
```

```
global z
x=5;
y=f(x)

function y=f(x)
global z
z=5;y=z+x
end
```

Paso dunha función como parámetro a *feval*

- Dende a mesma sentenza (que debería estar dentro dun bucle), *feval* permite chamar a unha función distinta en cada iteración do bucle
- O nome da función chamada por *feval* pode ser un carácter ou unha referencia a función (ver máis adiante)

[ret1 ... retM]=feval('nome función', arg1, ..., argN);

```
function y=nome(f,x)
...
y=eval(f, x);
end
```

nesta función o argumento *f* é unha función que en cada chamada a *nome(...)* pode ser distinta

Permite facer o mesmo que as funcións *external* en Fortran

Exemplo de *feval*: integral definida

Cálculo da integral indefinida dun vector de funcións (varias funcións á vez) usando *feval*:

```
clear all
f = {@sin, @cos};
for i = 1:2
   a=0;b=pi;x=a;h=0.001;integral=0;
   for x=a:h:b
      integral=integral+h*feval(f{i}, x);
   end
   fprintf('integral de %s en [%g, %g]= %g\n', char(f{i}),
     a, b, integral);
end
```

Vector de celdas de Matlab

Referencia a función

Transforma a referencia a función nunha cadea de caracteres

Función anónimas

- Funcións simples (dunha única liña, análogas a **funcións de sentenza** en Fortran). Pódense crear en calquer parte (dentro dunha función, arquivo ou liña de comando).
- Definición: *f=@(argumentos) expresión;*
- Chamada: *f(argumentos)*
- Aínda que non se lles pasen argumentos teñen que levar os parénteses tanto na creación como na chamada. Exemplos:
  - Definición: *f=@(x) x^2+1, f=@(x,y) sin(x+y)*
  - Chamada: *f(4), f(pi/2,pi/3)*

Referencias a función

1) Co ***operador @*** (“at” ou “arroba”), precedendo o operador @ ó nome dunha función predefinida de Matlab: *f=@sin*

2) Coa función ***str2func('funcion')*** que toma como argumento o nome de función e devolve a referencia á función: *f=str2func('sin'); f(pi)*

   Esta función permite transformar unha **expresión** (cadea de caracteres) en referencia a función:

   expr=’x^2’;f=str2func(sprintf(‘@(x) %s’,expr))

Funcións *inline* (I): obsoletas

- Funcións simples (dunha única liña, análogas a **funcións de sentenza** en Fortran) para cálculos matemáticos que requiren computación extensiva e deben ser eficientes, xa que se executan moitas veces
- Defínense dentro do programa (non nun arquivo separado)

nome = inline('expresión matemática')
nome = inline('expresión', 'var1',...,'varN')

- Pode incluír funcións de Matlab ou propias
- As *var1...varN* son as variábeis independentes
- Debe respeta-la dimensión do argumento (escalar / vector/matriz). Se é vector/matriz, hai que facer as operacións compoñente a compoñente (.*, .^, ./)

Funcións *inline* (II)

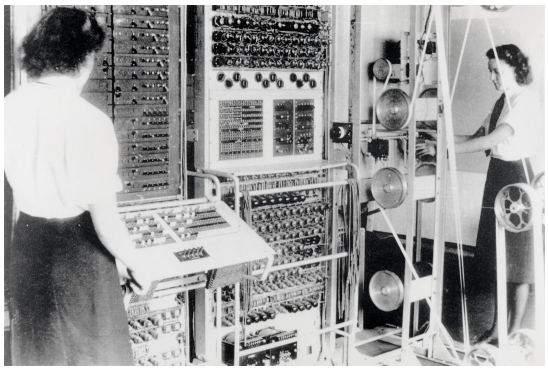
- A expresión pode ter varias variábeis independentes (non *i* ou *j*, unidade imaxinaria)
- Ex: *f = inline('exp(x^2)/sqrt(x^2+5)');*
  f(2)
  ans=18.1994
- Ex: *f=inline('x^2+y^2+z^2','x','y','z');*
  f(1,2,1) => resultado: 6
- *Ex: f=inline('exp(-x.^2)./(x.^2+5)') % con vectores*
  x=-1:0.1:1; f(x)
- Se as variábeis independentes non se indican, Matlab asume que son as letras da expresión por orde alfabética
- Son equivalentes ás funcións anónimas

Resumen de referencias a función

- Función anónima: f=@(x) x^2
- Operador @: f=@*sin*
- Inline: f=*inline*('x^2');
- Con *str2func*: f=*str2func*('sin(x)^2')
- Todas poden ser chamadas: f(5)
- Agás *inline*, podes obter a súa expresión como cadea de caracteres coa función *func2str*(f):

printf('f=%s\n',*func2str*(*f*))

Programadoras dos ordenadores Colossus (1943)



- Primeiras calculadoras electrónicas
- Programadas por 273 mulleres programadoras do Women's Royal Naval Service
- Entre outras (2016): Irene Dixon, Lorna Cockayne, Shirley Wheeldon, Joanna Chorley and Margaret Mortimer
- Usados por Inglaterra para descifrar comunicacións alemanas na 2ª guerra mundial

Arquivos

Lectura de datos dende un arquivo:

- Comando ***load***: lectura de arquivos numéricos (sen letras nin símbolos) onde tódalas liñas teñen o mesmo número de elementos
- ***Non*** necesita abrir e pechar o arquivo
- Exemplos:

  load datos.dat => carga os datos á matriz *datos*

  x=load('datos.dat'); => carga os datos á matriz *x*

  Mellor a segunda forma con nomes de arquivos longos.

Entrada e saída a arquivos

- Apertura: *f=fopen('arquivo.dat', 'permisos');*
- Retorna *f=-1* en caso de erro, *f>0* noutro caso.
- Permisos:
  - '*r*': abre o arquivo para lectura (por defecto)
  - '*w*': escritura: borra o arquivo se xa existe
  - '*a*': abre o arquivo para escribir ao final (conserva o que xa está)

```
nf='arquivo.txt';f=fopen(nf,'r');
if -1==f; error('fopen %s',nf); end
```

- Peche do arquivo: *fclose(f);*

Escritura / lectura en arquivos

- **Escribir**: función *fprintf*:

fprintf(f, 'formato', datos);

  - A mesma función para saída por pantalla pero con *f* para enviar a arquivo. Ex: *fprintf(f,'n=%i x=%f\n',n,x);*
  - Se mostra en pantalla, non retorna nada; se almacena en arquivo, retorna o nº de bytes escritos (rematar en **;**)
  - Vectorizada: *x=randi(100,1,20); fprintf(f,'%i ',x);*

- **Ler**: Función *fscanf*: ten dúas formas:

a=fscanf(f, 'formato');
[a m]=fscanf(f, 'formato', n);

Le *n* datos. Só con arquivos nos que as liñas teñen distintos números de elementos ou conteñen texto (noutro caso, usa *load*)

Se non lle indicas *n*, le tódolos datos do arquivo

Lectura dende un arquivo con *fscanf*

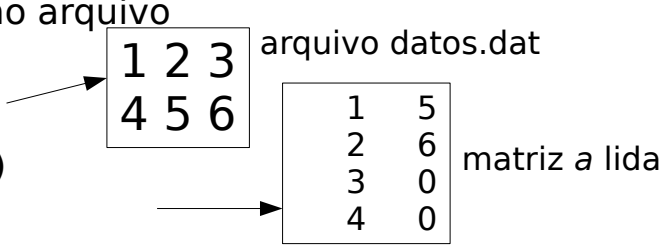
[a,m] = fscanf(f, 'formato', n)

- *f* : identificador de arquivo retornado por *fopen*
- '*formato*': cadea de formato: igual que en *fprintf*
- Se non ten nada que ler (p. ex., no final do arquivo) retorna *a=[]*
- *m: nº de datos realmente lidos*
- Argumento *n* (opcional, se non está vale *inf*): nº de datos a ler: pode ser:
  - Un enteiro: neste caso, le *n* datos (ou m<n se non hai máis datos no arquivo), que se almacenan no vector columna *a*
  - *inf*: Le tódolos datos do arquivo e almacénaos no vector columna *a* (igual que se non se especifica *n*).
  - *[nf nc]:* Le *nf* x *nc* datos do arquivo e méteos nunha matriz *a* de orde *nf* x *nc*: Mete a 1ª fila do arquivo na 1ª columna, etc.

Lectura dende un arquivo con *fscanf*

- *Se hai menos de nf x nc datos no arquivo, le os que haxa e enche os elementos restantes con ceros*
- *O valor nc* pode ser *inf*, de modo que *a* ten *nf* filas e o número mínimo de columnas para mete-los datos lidos en *a*
- *O valor nf* non pode ser *inf*
- Se *n*=[*nf nc*], entón *m* pode ser menor que *nf* x *nc* cando hai menos de *nf* x *nc* datos no arquivo
- Exemplo:

```
f=fopen('datos.dat');
a=fscanf(f,'%d', [4 inf])
fclose(f);
```



- *A matriz a* é 4x2 porque en *datos.dat* hai 6 elementos e non collen nunha matriz 4x1, necesita 2 columnas

Exemplos de lectura/escritura

- Lectura dunha matriz [*nf* x *nc*] dende un arquivo:
  a = load('arquivo.txt');

```
f = fopen('arquivo.txt', 'r');
a = fscanf(f, '%g', [nf nc]);
fclose(f);
```

- Escritura nun arquivo:

```
f = fopen('arquivo.txt', 'w');
for i=1:nf
    fprintf(f, '%g ', a(i,:));
    fprintf(f,'\n');
end
fclose(f);
```

- Lectura dun arquivo irregular completo:

```
1 2 3
4 5
6 7 8 9 8 7 6
```

```
f= fopen('arquivo.txt', 'r');
x=fscanf(f, '%g');
fclose(f);
```

- Adición ao final dun arquivo:

```
x = [1 2 3 4];
f = fopen('arquivo.txt', 'a');
fprintf(f, '%g ', x);
fclose(f);
```

Lectura de cadeas de caracteres con *fgetl* e *strsplit*

- Sintaxe: *s=fgetl(f);* le unha liña como cadea de caracteres
- Se a liña está baleira, ou está ao final do arquivo, retorna -1 como número
- Divides en palabras con *strsplit*(*s*): *s*{*1*},...,*s*{*n*}
- Se a liña ten números, hai que dividir a cadea en palabras e convertir os números en double (función *str2double*)
- Se a cadea de caracteres non é un número, *str2double* retorna *NaN*, e podes comprobalo coa función *isnan*

```
clear all
f=fopen('datos.dat');
if -1==f; error('fopen datos.dat'); end
while ~feof(f)
   s=fgetl(f);
   fprintf('liña= <%s>\n',s);
   t=strsplit(s);n=numel(t);
   fprintf('palabras:');
   for j=1:n
      x=str2double(t{j});
      if ~isnan(x)
         fprintf('%f\n',x);
      else
         fprintf('%s\n',t{j});
      end
   end
end
fclose(f);
```

Divide a cadea e convirte a números se procede

Funcións *feof* (fin de arquivo) e *frewind* (rebobinado)

- Cando lemos, é importante saber cando chegamos á fin de arquivo
- A función *fscanf* retorna 0 bytes cando chega á fin do arquivo
- A función *feof*(*f*) retorna 1 se atopou a fin do arquivo, ou 0 en caso contrario
- Función *frewind*(*f*): retorna ao comezo do arquivo

Exemplo: programa que le todo o arquivo e mostra liña a liña:

```
f=fopen('datos.dat');
if -1==f
   error('fopen datos.dat')
end
while ~feof(f)
   s=fgetl(f);fprintf('%s\n',s);
end
fclose(f);
```

Funcións *fseek* e *ftell*

- *fseek*(*f*,*n*,*w*): sitúase *n* bytes (caracteres) á dereita (se *n*>0) ou esquerda (se *n*<0) de *w*, que pode ser:
  - 'bof' ou -1: comezo do arquivo (*beginning of file*)
  - 'cof' ou 0: posición actual no arquivo (*current position on file*)
  - 'eof' ou 1: final do arquivo (*end of file*)
- *ftell*(*f*): retorna a posición actual (onde se vai ler ou escribir) en caracteres dende o comezo do arquivo

Exemplo de lectura de arquivo en formato *write.table* de R con *fscanf*

- Arquivo co seguinte contido:

	E1	E2	E3	E4	Saida
1	0.25	0.33	1.23	-0.51	Branco
2	-0.34	1.3E5	0.22	4.3	Negro

```
clear all
f=fopen('arquivo.dat','r');
if -1==f
    error('erro abrinfo arquivo.dat')
end
s=strsplit(fgetl(f));s(1)=[];nc=numel(s)-1;nf=0;
while ~feof(f)
    fgetl(f);nf=nf+1;
end
dato=zeros(nf,nc);saida=cell(1,nf);
frewind(f);nomes=fscanf(f,'%s',nc+1);
for i=1:nf
    fscanf(f,'%i',1);  % le e descarta (non almacena) o nº de liña
    dato(i,:)=fscanf(f,'%g',nc);  % le as ne entradas (tamén formato exponencial)
    saida{i}=fscanf(f,'%s',1);  % le a saída (cadea de caracteres)
end
fclose(f);
for i=1:nf
    fprintf('dato %i: ',i);fprintf('%g ',dato(i,:));fprintf('saida=%s\n',saida{i});
end
```

Le o nº de filas e columnas

Jean Sammet (1928-2017)

- Matemática e informática
- Desenvolveu (1962) a primeira linguaxe de programación simbólica FORMAC (formula manipulation compiler) en IBM
- Primeira persoa que escribiu extensamente (1969) sobre a historia e clasificación das linguaxes de programación

Función *find* e operadores relacionais

- Obter elementos que cumpren unha condición: *vector(condición)*. Ex: *a(rem(a,2)==0), v(v>3 && v<=5)*

Elementos con valores pares

Elementos con valores no intervalo (3,5]

- Obter índices de elementos: *find(condición)*. Ex: *find(rem(5*v-3,4)==3)*

Índices de elementos que dan resto 3 cando divido 5v-3 entre 4

- Obter nº de elementos que cumpren unha condición: *length(v(condición))* ou *sum(condición).*

Índices de elementos con valores impares

Ex: *length(v(rem(v,2)==1))*, ou *sum(rem(v,2)==0)*

- Modificar elementos que cumpren unha condición: *v(cond)=F(v(cond)).*

Incrementa en 5 os elementos de v maiores ca 3

Ex: *v(v>3)=v(v>3)+5*, ou *t=v>3;v(t)=v(t)+5*

Vectorización de expresións (I)

- Executa un comando que cree unha matriz *a* de orde 5 e poña os elementos con valores pares a 7 e os elementos con valores impares a -1:

  a = magic(5);

  a(rem(a,2)==1)=-1

  a(rem(a,2)==0)=7

- Se fago *a=magic(5);rem(a,2)==0* devólveme unha matriz de 1s nos elementos pares e 0 nos restantes
- Hai que executar primeiro *rem(a,2)==1* e logo *rem(a,2)==0*: se facemos primeiro *rem(a,2)==0*, poñemos os elementos pares a valores impares

Vectorización de expresións (II)

- Dada unha matriz cadrada, manipúlaa de modo que os elementos a_{ij} que verifiquen que $i \cdot j$ é par pasen a valer -1, e os elementos con $i \cdot j$ impar pasen a valer 3

  a=magic(5)

  *i=1:5;j=i; b=i'*j;*

  a(rem(b,2)==1)=3

  a(rem(b,2)==0)=-1

- Matriz *b=i'*j,* de orde *nxn*: $b_{kl}=i(k)*j(l)$

Vectorización de expresións (III)

- Crea un vector con 10 elementos enteiros aleatorios no rango [-10,10]: *v=round(-10+20*rand(1,10))*
- Mostra os índices dos elementos positivos: *find(v>0)*
- Mostra so os elementos positivos: *v(v>0)*
- Mostra tódolos elementos positivos do vector e ceros nos negativos: *v(v<0)=0* ou *v.*(v>0)*
- Crea un vector con valores -3 onde *v>0* e 5 onde *v<=0*: *-3*(v > 0) + 5*(v<=0)*

Vectorización de expresións (IV)

- Crea dous vectores *v* e *w* de orde 10: *a=magic(10); v=a(1,:); w=a(2,:);*
- Atopa os índices dos elementos de *v* maiores que os seus correspondentes de *w* : *find(v > w)*
- Atopa os elementos de *v* maiores que o seu correspondente de *w* : *v(v > w)*
- Crea un vector que valia 4 nos elementos *i* nos que $v_i > w_i$ e -6 nos restantes: *4*(v > w) - 6*(v <= w)*
- Selecciona os elementos de *v* nas posicións *i* nas que v_i e w_i sexan meirandes ca 5: *v(v > 5 & w > 5)*

Carol Shaw (1955)

- Icono da industria dos videoxogos
- Programadora pioneira de videoxogos en Atari (1978)
- Enxeñeira microprocesadora de videoxogos
- Creadora dos videoxogos River Raid (1982, Activision; 1983, Atari 800), Happy Trails (1983, Intellivision)
- Industry Icon Award (2017) polas súas contribucións á industria dos videoxogos

Vectores e matrices de celdas (*cell arrays*)

- Vectores/matrices onde cada elemento pode ser dun tipo distinto.
- Creación: utilizando chaves { }
  - *s={'ola',1:3,17,1+3*i}; disp(s{1})*: define un vector de 4 celdas
  - Define un vector de 3 celdas:

    1)vc(1)={[1 2 3]} ou *vc{1}=[1 2 3]*

    2)vc(2)={'unha cadea'} ou *vc{2}='unha cadea'*

    3)vc(3)={ones(3)} ou *vc{3}=ones(3)*
- Acceso a un elemento: s{1} para imprimir: *fprintf*('%s', s{1}); s(1) para eliminar: s(1)=[ ]; s{1}=[ ] asigna a matriz baleira

Vectores e matrices de celdas

- Funcións de manipulación:
  - *cell(m,n)*: crea un cell array baleiro de m filas e n columnas
  - *celldisp(ca)*: mostra o contido de tódalas celdas de ca
  - *iscell(ca)*: indica se ca é un vector de celdas
  - *num2cell(v)*: convirte un vector numérico **v** nun cell array
  - *cellstr(s)*: crea un vector de celdas a partires dun vector de caracteres.

Cadeas de caracteres

- Definición: *str='son a cadea 450'*
- Se queremos ter varias cadeas (p.ex. un vector de cadeas) non podemos facer *s=['ola','adeus']* xa que entón *s='olaadeus'*.
- Non podemos facer *s=['ola' ; 'adeus']*, xa que as dúas filas (ou cadeas) teñen que ter a mesma lonxitude
- Por iso o usual é usar unha celda de cadeas: *s={'ola', 'adeus'}*
- Para acceder á 1ª cadea: *disp(s{1})* ou *disp(s(1))*
- O elemento *s{1}* é de tipo *char*, pero *s(1)* é *cell*:
  - *fprintf('%s\n',s{1})* non da erro
  - *fprintf('%s\n',s(1))* si da erro
- Outra opción: s=*char*({‘ola’,’adeus’}); s(1,:)->’ola’; s(2,:)->’adeus’

Funcións de manipulación de cadeas de caracteres (I)

- Función **strsplit**(s,t): divide unha cadea s en palabras usando o delimitador t (espazo, por defecto)
- Retorna un vector de celdas, cada elemento é unha cadea de caracteres (palabra)
- O nº de palabras é o nº de elementos do vector de celdas
- Exemplo: s=’valor 3.65’
  p=*strsplit*(s)
  p{1}=’valor’
  p{2}=’3.65’; *str2num*(p{2}) da 3.65 como número e podes facer operacións

Funcións de manipulación de cadeas de caracteres (II)

- *ischar(str):* devolve 1 se *str* é unha cadea de caracteres e 0 se non o é.
- *isletter(str)*: devolve un vector de igual dimensión a str con 1 se é unha letra do abecedario e 0 en caso contrario.
  - Ex: *s='ola que tal'; isletter(s) -> 1 1 1 0 1 1 1 0 1 1 1*
    (Nota: -> denota a resposta que da Matlab á función *isletter*)
- *isspace(str)*: igual que *isletter()* pero con caracteres de espacio.
  - Ex: *s='ola que tal'; isspace(s) -> 0 0 0 1 0 0 0 1 0 0 0*
- *char(x), char(c), char(t1, t2, t3, ...):* devolve un vector de caracteres a partires de: *x* un vector de enteiros (códigos Unicode), *c* un vector de celdas de caracteres, as cadeas *t1, t2, t3* ....
  - Ex: *c={'ola','adeus'};s=char(c); s(1,:)->ola; s(2,:)->adeus*

Funcións de manipulación de cadeas de caracteres (III)

- *num2str(x,n):* convirte o número *x* con *n* (opcional) cifras nunha cadea de caracteres.
- *str2num(x):* convirte a cadea de caracteres *x* a un número.
- *strcmp('str1','str2'), strcmp('str1',C2), strcmp(C1, C2):* compara as cadeas *str1* e *str2* e devolve 1 se son iguais e senon 0. Se temos o vector de celdas *C1*, compara *str1* con tódolos elementos de *C1*. No último caso compara os dous vectores de celdas. *strcmp* distingue entre maiúsculas e minúsculas; a función *strcmpi* non.
  - Ex: *strcmp('ola','ola') -> 1; strcmp('ola',{'ola','adeus'}) -> 1 0; strcmp({'ola','pepe'},{'ola','adeus'}) -> 1 0*
- *strncmp('str1', 'str2', n), strncmp('str', C, n), strncmp(C1, C2, n)*: igual que strcmp pero comparando so os *n* primeiros caracteres.

Funcións de manipulación de cadeas de caracteres (IV)

- *strmatch('str', C)* ou *strmatch('str', C, 'exact')*: devolve un vector columna cos índices de *C* onde se atopa *str*.
  - Ex: *C={'N', 'H', 'O', 'He', 'Ca'}; strmatch('H', C)* -> vector columna cos índices 2 e 4 (que corresponden ó '*H*' e '*He*')
  - Ex: *strmatch('H', C, 'exact')*-> vector columna co índice 2
- Lonxitude dunha cadea: *s='ola caracola'; numel(s),length(s) -> 11*
- Concatenación de cadeas: *strcat(s1,s2,s3,...); strvcat(s1,s2,s3,...)*
  - Horizontal: *strcat('ola','adeus') -> 'olaadeus'*
  - Vertical: *strvcat('ola','adeus') -> ola*
    adeus
  - Tamén se pode facer concatenando vectores (supoñendo que unha cadea de caracteres é un vector de caracteres).

Funcións de manipulación de cadeas de caracteres (V)

- *strfind(str, patrón)* e *strfind(cellstr, patrón)*: devolve un vector coas posicións de comezo onde se atopou a cadea *patrón* na cadea *str* ou vector de celdas *cellstr*.
  - Ex: *strfind('ola caracola', 'ol') -> 1 10* (o patrón *'ol'* aparece nas posicións 1 e 10 da cadea *'ola caracola'*)
  - Ex: *strfind({'ola','caracola'},'ol') -> [1] [6]* (o patrón *'ol'* aparece nas posicións 1 e 6 do 1º e 2º elementos do arrai de celdas)
- *regexprep('str', 'expr', 'repstr'):* reemplaza a expresión *expr* por *repstr* na cadea de caracteres *str* e devolve a cadea resultante.
  - Ex: *cad='H2(g)+O2(g)=H20'; regexprep(cad, '\(g\)', '') -> H2+O2=H20*: substitúe '(g)' pola cadea baleira, é dicir, elimina '(g)'.

Función *textscan*

- Le un arquivo aberto con *fopen* e almacena nun vector de celdas os datos lidos (números ou cadeas de caracteres):
- Sintaxe:
  f=fopen('arquivo.dat','r');
  c=textscan(f,formato); % le todo o arquivo
  c=textscan(f,formato,n); % le n datos
- O formato é como en *fprintf*: *%i,%s,%f,%g*
- O valor *c* é un vector de celdas: *c{1}*,...,*c{n}*

Función *textscan*: exemplo

```
clear all;
fid=fopen('taboa.txt');
t=textscan(fid, '%s', 4); % ler primeira liña
celldisp(t);
datos=textscan(fid, '%d %s %s %f'); % ler resto arquivo
celldisp(datos);
fclose(fid);
```

posición	símbolo	elemento	pesoAtómico
1	H	HIDRÓXENO	1
6	C	CARBONO	12
7	N	NITRÓXENO	14.01
8	O	OSÍXENO	16
17	Cl	CLORO	35.45

t{1}{1} =posición
t{1}{2} =símbolo
t{1}{3} =elemento
t{1}{4}=pesoAtómico

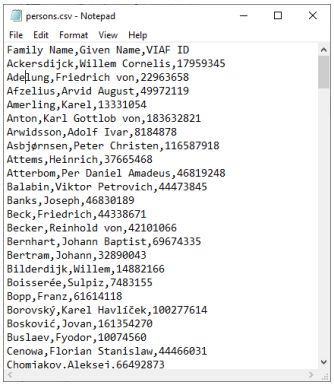
datos{1}' = [1 6 7 8 17] (vector)
datos{2}' = { 'H' 'C' 'N' 'O' 'Cl'} (vector celdas)
datos{3}' ={'HIDRÓXENO' 'CARBONO' 'NITRÓXENO' 'OSÍXENO' 'CLORO'} (vector celdas)
datos{4}' =[1.0000 12.00 14.010 16.000 35.450] (vector)

Ollo:
datos{1}(2) -> 6
datos{2}{3} -> 'N'

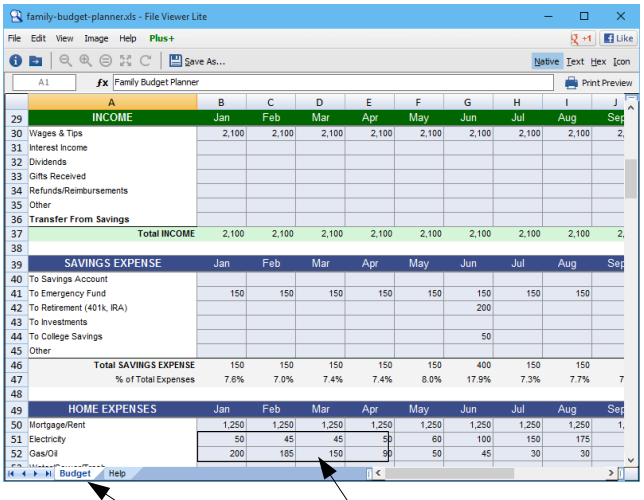
Outras funcións de lectura de arquivos

- Arquivos CSV (comma separated values):

x=readtable(‘arquivo.csv’)



- Arquivos XLS ou XLSX (follas de cálculo):



x=xlsread(‘arquivo.xls’,’páxina’,’rango’)

rango: ‘A2:C3’

Estructuras (*struct*)

- Conxunto de datos heteroxéneo (ten datos de distintos tipos, p.ex. real e carácter)

a=struct('nome','Carlos','idade',19);

- *a* é unha estructura con dúas variábeis: *a.nome* é unha cadea, e *a.idade* é un número. Podes imprimir os seus campos con:

fprintf('nome=%s idade=%i\n',a.nome,a.idade)

- Normalmente se usa un vector ou matriz de estructuras:

```
v=cell(1,10);
for i=1:10
    v{i}=struct('nome',nome(i),'idade',idade(i));
end
```

- Logo podes facer *v{i}.nome* (accedes ao nome i-ésimo) ou *[v{:}].nome* (accedes a todos os nomes): *fprintf('%s ',[v{:}].nome)*
- É máis cómodo que usar un vector distinto para cada campo.

Cálculo numérico e simbólico

- Resolución **numérica** de ecuación non linear dunha variábel: *f(x)=0*:

Función anónima

f=@(x) expresión; x = fzero(f, x0)

  - *x0* é un punto de inicio
  - Ex: $xe^{-x}=0.2$: *f=@(x) x*exp(-x)-0.2; fzero(f, 0.7)*

- Mínimo dunha función:

```
f=@(x) expr
xmin=fminbnd(f,a,b);
[xmin vmin] = fminbnd(f, a,b)
```

  - Busca o valor mínimo de *f* no intervalo *[a,b]*
  - Retorna punto e valor mínimo *xmin* e *vmin*
  - Exemplo: *f=@(x) x^3-12*x^2+3*x-1;*
    [x v]=fminbnd(f,0,10)

Integración numérica

x = quad('función', a, b)

- A función pode ser unha expresión, función predefinida de Matlab ou función definida polo usuario. Ídem en *fzero()* e *fminbnd()*
- A función debe escribirse considerando que *x* é un vector (operandos compoñente a compoñente)
- Ex: *quad('x.*exp(-0.8*x) + 0.2', 0, 8)* → $\int_0^8 (xe^{-0.8x}+0.2)dx$
- Para integral de función definida polos puntos $\{(x_i,y_i),i=1...n\}$ mediante o método dos trapecios: *trapz(x, y)*

```
x=0:0.01:8;
y=x.*exp(-0.8*x)+0.2;
trapz(x, y)
```

- Simbólicamente: *syms x; eval(int(x*exp(-0.8*x)+0.2,x,0,8))*

Cálculo simbólico (I)

- Definición de variábeis simbólicas: *syms v1 ... vn*
- En **Octave**: antes de definilas, executa *pkg load symbolic*.
- Límites: *limit(expresión, variábel, valor, lado)*
  - *var*, *valor* e *lado* son opcionais (*var=x,valor=0* por defecto; *lado* = '*left*' ou '*right*', por defecto calcúlase o límite por ambos lados)
  - Ex: *syms x; limit(1/x); limit(1/x,inf); limit(1/x, x, 0, 'left');*
- Derivación: *diff(expresión, var, orde)*
  - *var* e *orde* son opcionais (*var=x* e *orde=1*)
  - Ex: *syms x; diff(x^2); diff(x^2, x); diff(x^2,x,2)*
    syms x y; diff(x^2+y^2,x,y)

Cálculo simbólico (II)

- Integración indefinida: *int(expresión, var)*
- Integración definida: *int(expresión, var, ini, fin)*

$$\int_0^{\infty} e^{-x^2} dx$$

  - Ex: *int(cos(x), x); int(exp(-x^2), 0,inf); eval(ans)*
- Series numéricas: *symsum(expresión, var, ini, fin)*
  - Ex: *syms n; symsum(1/(n^2-1),n,2,inf)* $\longrightarrow \sum_{n=2}^{\infty} \frac{1}{n^2-1}$
- Substitución de variábeis simbólicas por valores numéricos:

  res = subs(expresión, var, valor(es))

  - Ex: *subs(x^2+exp(-x), x, pi/2)*
- *Avaliación de expresión simbólica en punto flotante: eval(expr)* ou double(expr). En octave: *double(expr)*.

Conversión entre cadeas de caracteres, expresións simbólicas e referencias a función

- Conversión de cadea de caracteres (cunha expresión matemática) a expresión simbólica:

  syms x;f=str2sym('x^2')

  Non podes chamala (p.ex. f(5)), pero si podes derivala: *diff(f,x)*, e calcular o seu valor con *subs(f,x,5)*

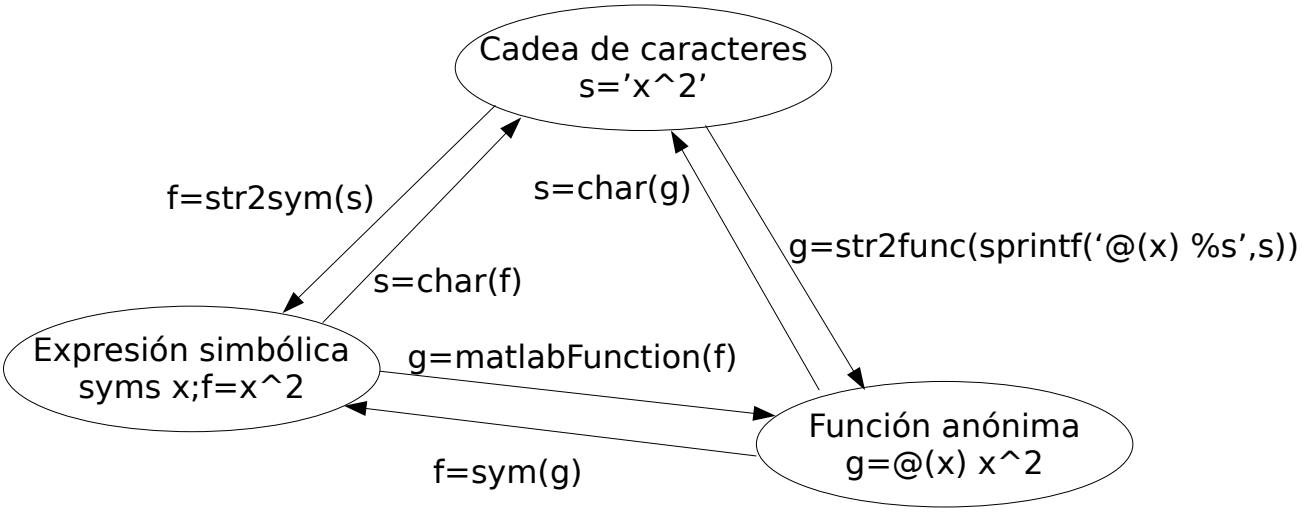
  Non funciona *diff('x^2')*: *diff(str2sym('x^2'))*
- Podes facer operacións de cálculo simbólico:

  con funcións *inline*: *f=inline('x^2'); diff(f(x))*

  con funcións anónimas: *f=str2func('@(x) x^2'); diff(f(x))*
- Conversión de expresión simbólica a función anónima:

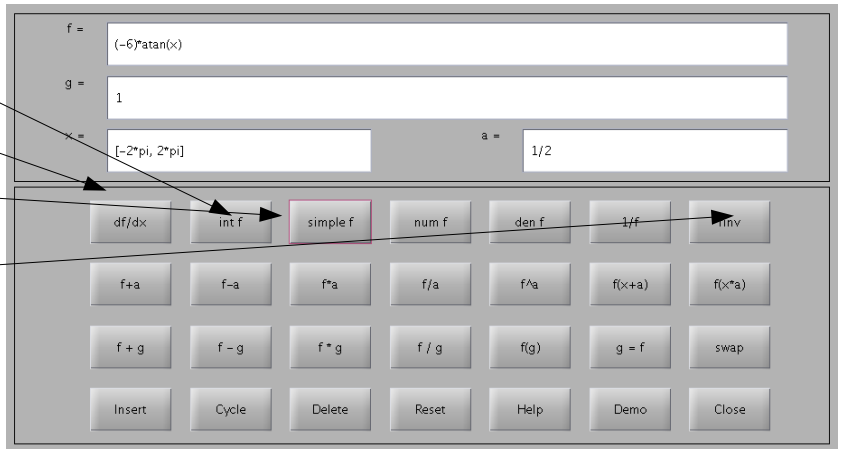
  syms x;f=x^2;g=matlabFunction(f)

Esquema de conversión: cadea de caracteres ↔ función anónima ↔ expresión simbólica



Cálculo simbólico (III)

- *funtool*: calculadora de funcións
  - Integrar
  - Derivar
  - Simplificar
  - Invertir
  - *help funtool*

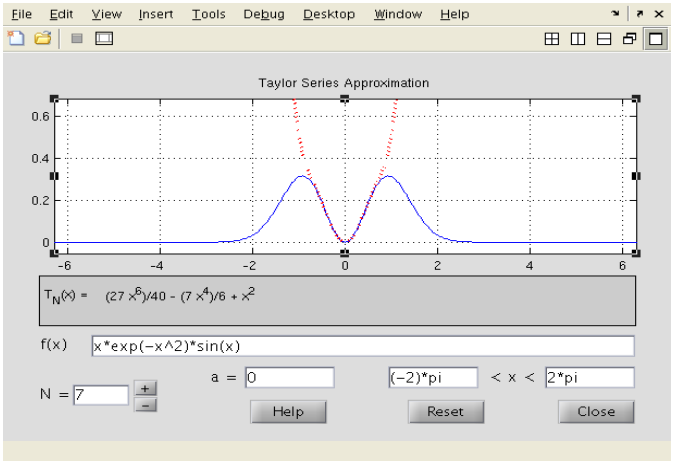


Cálculo simbólico (IV)

Polinomio de Taylor orde *n-1* dunha función *f* en *x=a*:

taylor(f,x,'ExpansionPoint',a,'Order',n)

taylor(f): en x=0,orde 5
taylor(f,x):en x=0,orde 5
taylor(f,x,'ExpansionPoint',1);
en x=1, orde 5
taylor(f,x,'Order',7);
en x=0, orde 6
Ex: syms x; f=(x-1)/(x+1);
taylor(f,x,'ExpansionPoint',7,
'Order',1)



taylortool: calcula e representa graficamente a función e o seu polinomio de Taylor

Cálculo simbólico (V)

- Resolución **simbólica** de ecuacións e sistemas de ecuacións:

  syms x; h = solve(expresión, var)

  - A ecuación é da forma *expresión=0*
  - Ex: *syms x; h = solve(x*exp(2*x)-5)*
- *Resolución simbólica dun sistema de varias ecuacións con varias variábeis:*

  syms x1...xn; [v1 ... vn]=solve(eq1,...,eqn, x1, ..., xn)

  - Ex: *syms x,y,z; [xz yz]= solve(10*x+12*y+16*z, 5*x-y-13*z,x,y)* %x,y función de z
  - Ex: *syms x, y; [x y] = solve(x*exp(y)-3,y*exp(x)-2,x,y)*

Combinatoria

- Función *nchoosek(n, k)*: calcula $\frac{n!}{(n-k)!\,k!}$
- **Combinacións** de *n* elementos dun vector *v* tomados en grupos de *k* (*n* < 15): *nchoosek(v, k), nchoosek('abcde',3)*, *nchoosek({'a','b','c','d','e'},5)*
- Retorna unha matriz de *n*!/((*n* – *k*)! *k*!) filas e *k* columnas; *n* < 15 (evitar explosión combinatoria)
- **Permutacións** de *n* elementos: *perms(1:n)*, *perms('abcd')*, *perms({'a' 'b' 'c'})*. Retorna unha matriz de *n*! filas e *n* columnas; *n* debe ser < 15
- Selección aleatoria dunha permutación de *n* números de 1 a *n*: *randperm(n)*. Ex: *i=randperm(n);v(randperm(n))*: barallamento aleatorio dos elementos do vector *v* con *n* elementos.

Exercicios

1) Representa gráficamente $f(x)=e^{-x^2/10}\sin(x^2)$ e atopa o punto $(x_0, f(x_0))$ que minimiza *f* en *[-10, 10]*
2) Resolve as ecuacións $4\cos 2x - e^{x/2}+5=0$; $2\,sen\,x-\sqrt{x}=-2.5$ $\cos x = 2x^3$
3) Calcula θ tal que $92=88/(\cos\theta + 0.45\,sen\,\theta)$
4) Calcula numéricamente a integral $\longrightarrow \int_0^5 \frac{1}{0.6x^2+0.5x+2}\,dx$
5) Calcula simbólicamente:

$$\lim_{x\to\infty}\frac{\ln x}{\sqrt{x}} \qquad \lim_{x\to 1}\left(\frac{x}{x-1}-\frac{1}{\ln x}\right)$$

$$\frac{d}{dx}\left[\frac{\sqrt{x^2+x+2}}{x-1}\right] \qquad \int_0^{\pi/2}\left(1+\frac{1}{2}sen^2 x\right)dx \qquad \sum_{n=1}^{\infty}\frac{2^n+3^n}{n^2+\log n+5^n}$$

Solucións aos exercicios (I)

1) *Represento e atopo o mínimo en [-10, 10]*
   *fplot('exp(-x^2/10)*sin(x^2)', [-10,10])*
   *[xmin fmin] = fminbnd('exp(-x^2/10)*sin(x^2)', -10, 10)*
   x =2.1477, fmin =-0.6274

2) *a) fzero('4*cos(2*x)-exp(x/2)+5', 0) -> 1.2374*
   *comprobación: subs('4*cos(2*x)-exp(x/2)+5', 1.2374) -> 2.4960e-04*
   *syms x; solve(4*cos(2*x)-exp(x/2)+5, x) -> -1.557-0.258*i*
   *comprobación: subs('4*cos(2*x)-exp(x/2)+5', ans) -> -0.14e-30 – 0.272e-30*i*
   *b) Solución numérica: fzero('2*sin(x)-sqrt(x)+2.5', 2) -> 3.4664*
   *comprobación: subs('2*sin(x)-sqrt(x)+2.5', 3.4664) -> -4.4409e-16*
   *c) Solución simbólica: fzero('cos(x)-2*x^3', 0) -> 0.7214*
   *comprobación: subs('cos(x)-2*x^3', 0.7214) -> 2.2821e-05*

Solucións aos exercicios (II)

3) *fzero('92-88/(cos(x)+ 0.45*sin(x))', 0) -> -0.0881*

4) *quad('1./(0.6*x.^2+0.5*x+2)', 0, 5) -> 0.9596*
   *x=0:0.01:5; y=1./(0.6*x.^2+0.5*x+2); trapz(x, y) -> 0.9596*
   *syms x; eval(int(1/(0.6*x^2+0.5*x+2), x, 0, 5)) -> 0.9596*

5) *syms x; limit(x/(x-1)-1/log(x), x, 1) -> ½*
   syms x; diff(sqrt(x^2 + x + 2)/(x-1), x, 1)
   *syms x: int(1 + sin(x)^2/2, x, 0, pi/2) -> 5/8*pi*
   syms n; symsum((2^n + 3^n)/(n^2+log(x)+5^n), n, 1, inf) -> sum((2^n+3^n)/(n^2+log(x)+5^n),n = 1 .. Inf) (non a resolve, pero converxe)

Versión vectorizada:

```
n=1:100;
sum((2.^n + 3.^n)./(n.^2 + log(n)+5.^n))
```

Resultado: 1.8918

```
clear all
suma = 0;sumando = inf; n = 1
while sumando > 1e-5
      sumando = (2^n + 3^n)/(n^2 + log(n) + 5^n);
      suma = suma + sumando; n = n + 1;
end
fprintf('n= %i suma= %g\n', n, suma);
```

Gráficos 2D

- Sentenza *plot*: varias formas
- Representa os elementos do vector x fronte ao nº de compoñente: *plot(x)*
- Se x e y son vectores co mesmo nº de elementos, representa y frente a x: *plot(x, y)*
- Indicando propiedades do gráfico:

plot(x, y, 'r-')*

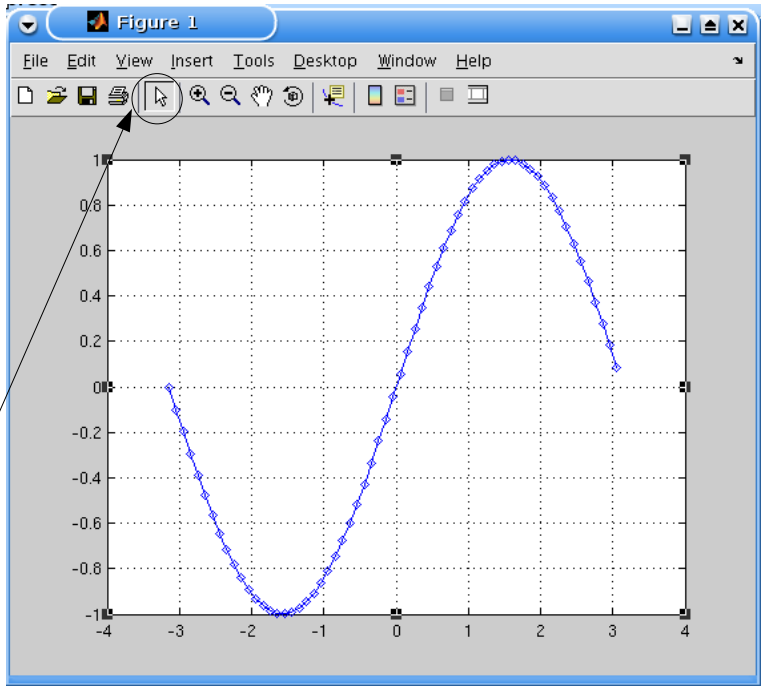
r=red, -= liña, *=punto con forma de asterisco

- Se *a* é unha matriz, *plot(a)* representa cada columna como unha gráfica distinta (e cores distintas)
- Podes representar varias gráficas no mesmo *plot*:

plot(x,sin(x),'bs-',x,cos(x),'or-',x,x.^2,'vg-')

Ventá de figuras

- Como en Maple, non é necesario nin eficiente saber tódalas opcións (cores, tipos de liña e punto, propiedades dos eixos, etc.). Pódense modificar na ventá do plot



Exportación de figuras a ficheiros de imaxe

- Manualmente: gardar ou imprimir en ventá de figura.
- Automáticamente dende ventá de comandos ou dende programa: función *print*.
- A arquivo en formato encapsulado postscript (EPS): *print('-depsc','arquivo.eps');*
- Formato portable network graphics (PNG):

  print('-dpng','arquivo.png');
- Formato PDF: *print('-dpdf','arquivo.pdf);*
- Formato TIFF: *print('-dtiff','arquivo.tif')*
- Formato JPEG: *print('-djpeg','arquivo.jpg')*

Representación de funcións (expresión analítica)

- Función *ezplot:*

  ezplot('función', [min, max])

  ezplot('función'): en [-2π, 2π]

  Ex: *ezplot('exp(-x^2)*sin(x)', [0,2*pi]), ezplot('sin(x)')*

- Función *fplot:*

  fplot(@(x) x.^2,[min max])

  - Función anónima con operacións compoñente a compoñente: .* ./ .^

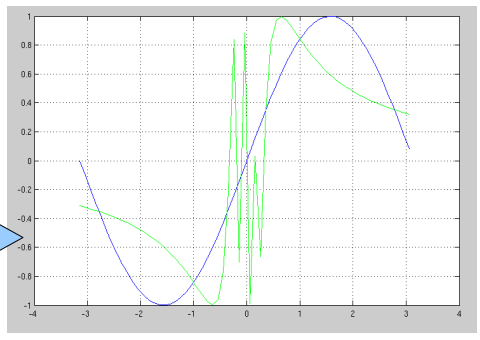
Comandos relacionados

- *xlabel('título eixo X'),* ídem con *ylabel*
- *title('título do gráfico')*
- *clf: borra a gráfica actual*
- *figure(2)*: crea unha nova ventá de figura
- *hold on: permite representar unha gráfica mantendo a(s) anterior(es); hold off: desactiva isto*
- *axis([xmin xmax ymin ymax])*: establece rangos en ambos eixos
- *axis equal*: igual lonxitude para a unidade en X e Y
- *axis square*: figura cadrada e non rectangular
- *grid on (off)*: pon (quita) o enreixado; *grid*: conmuta

Múltiples gráficas na mesma ventá

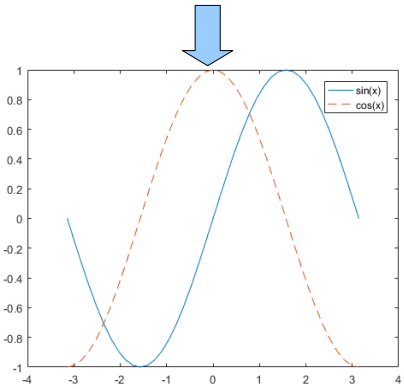
- Con *hold on*:

```
clf
x=-pi:0.1:pi;
y=sin(x);
z=sin(1./x);
plot(x,y)
grid on
plot(x,z,'g')
plot(x,y)
grid on
hold on
plot(x,z,'g')
```



- Comando *legend*:

legend({'gráfica1' 'gráfica2'}, 'location', 'northwest')
Outras *location*: *east*, *southwest*,...



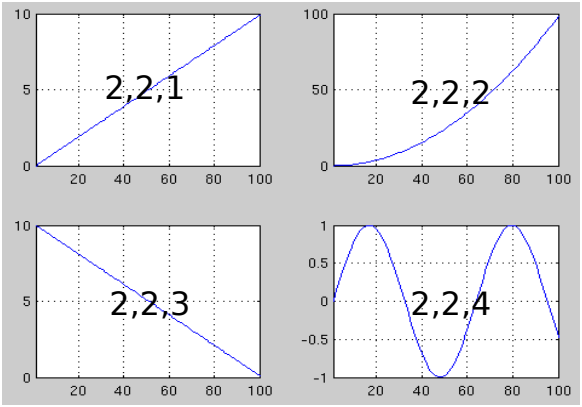
Múltiples gráficas en distintas sub-ventás

- Crea unha matriz de figuras (2x2, 3x2, ...)

  subplot(filas,columnas,actual)

- A 1ª execución de subplot crea a matriz (sen figuras) e crea a figura (1,1). Execútase unha vez por cada figura, e crea a figura actual ($1 \leq actual \leq filas*columnas$)

```
x=0:0.1:10;y=x.^2;z=10-x;
t=sin(x);
subplot(2,2,1); plot(x)
subplot(2,2,2); plot(y)
subplot(2,2,3); plot(z)
subplot(2,2,4);plot(t)
```



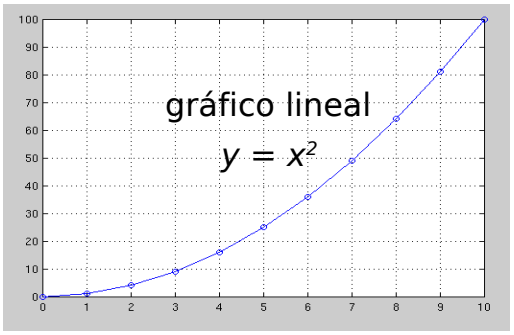
Outros comandos

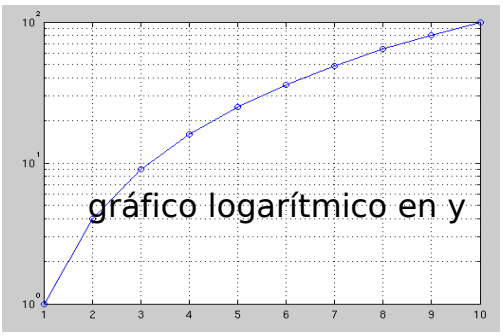
- Engadir texto na figura: *text(x,y,'texto no gráfico en (x,y)'); gtext('texto en posición indicada co rato');*

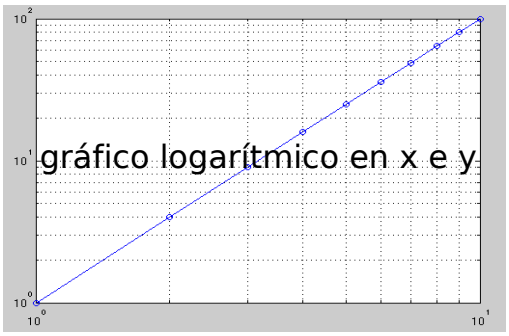
Gráficos logarítimos

- Ás veces é bon usar unha escala logarítmica (isto é, representar *log(x)* no canto de y, ou o mesmo para y) xa que o rango de valores é moi grande.
- *semilogx(x, y)*: escala log só en x
- *semilogy(x, y)*: escala log só en y
- *loglog(x, y)*: escala log en x e y
- Os valores nulos ou negativos non se poden representar nestos gráficos.

Gráficos logarítmicos





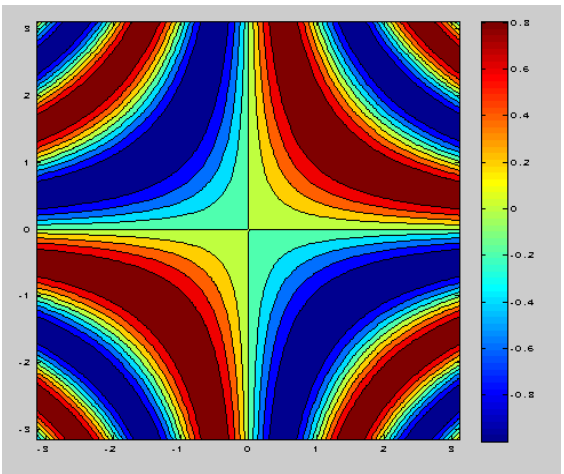


Mapa de calor

- Representa unha función *f(x,y)* cun código de cores (temperaturas: vermello=alto, azul = baixo, verde-marelo=medio).

```
[x y]=meshgrid(-3:0.05:3);
z=sin(x.*y);
contourf(x,y,z)
colorbar
```

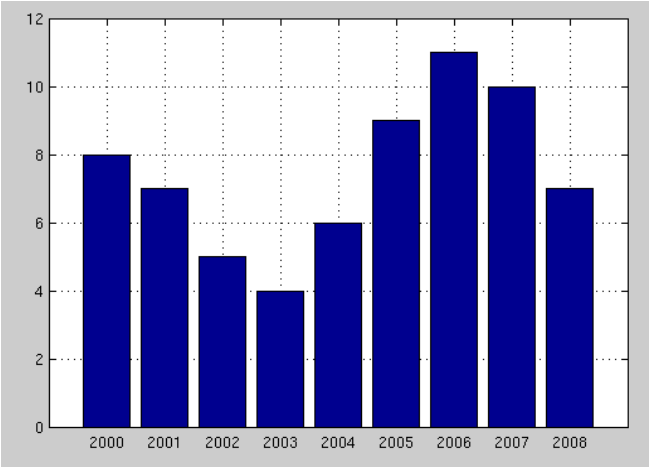
Engade a barra de cores na dereita



Gráficos especiais

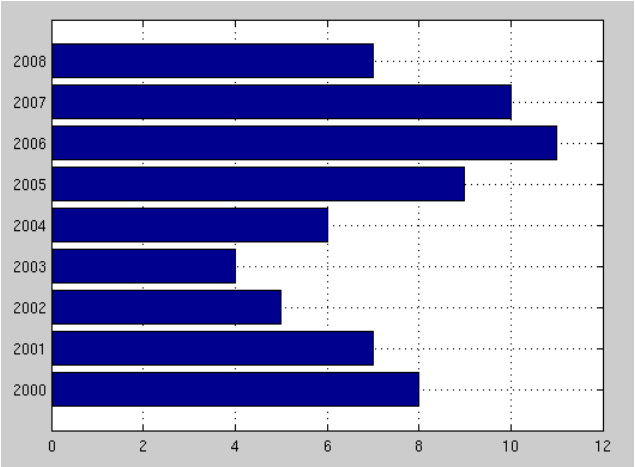
- Barras verticais:

x = [2000:2008]; y = [8 7 5 4 6 9 11 10 7];
bar(x, y); % x crecente



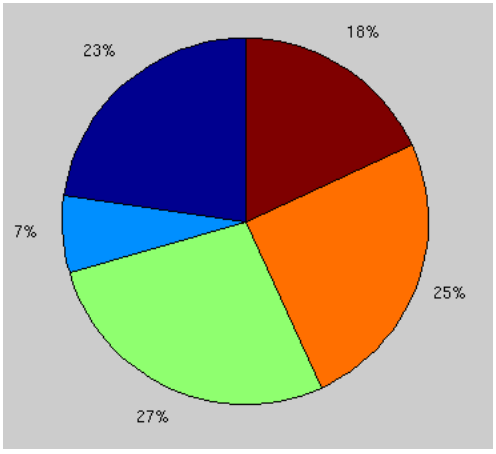
- Barras horizontais:

x = [2000:2008]; y = [8 7 5 4 6 9 11 10 7];
barh(x, y); % x crecente

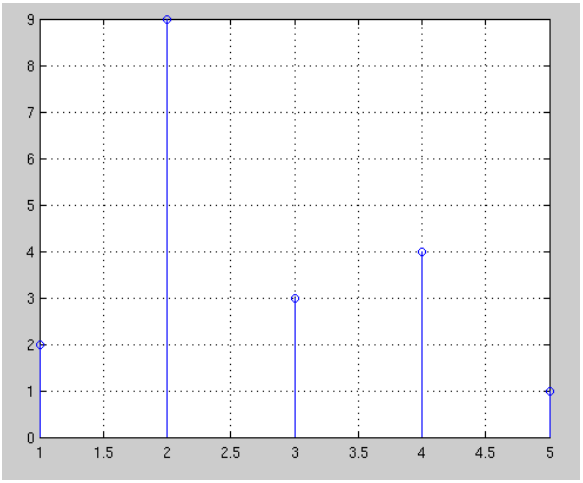


Gráficos especiais

- Tartas:
x=[10 3 12 11 8];
pie(x) % calcula os porcentaxes;sentido antihorario

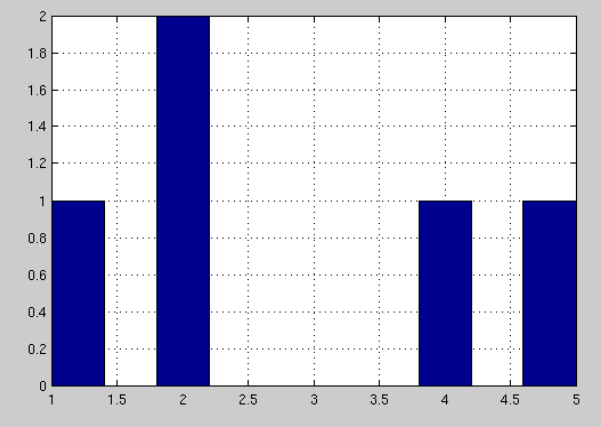


- Diagrama de troncos:
x=[1:5];y=[2 9 3 4 1];
stem(x,y) %x crecente



Histogramas (I)

- Cada barra vertical está asociada a un intervalo dos datos
- A altura de cada barra representa o nº de datos que se atopan no seu intervalo asociado
- Divide o rango dos elementos do vector en intervalos (10, por defecto)
- Ex: *y=[1 2 4 5 2]; hist(y)*
- Con nº de intervalos: *hist(y, 5)*

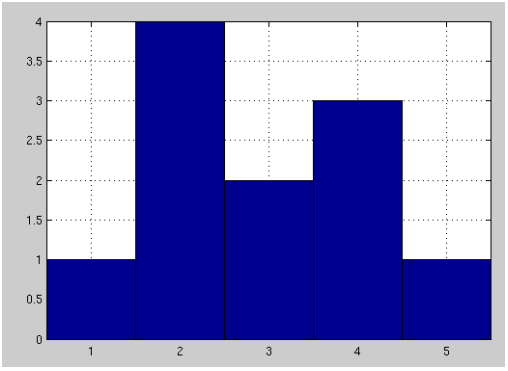


Histogramas (II)

- Usando 2 vectores, para X e Y:

x=[1 2 3 4 5];y=[1 2 3 2 3 4 4 2 5 4 2];

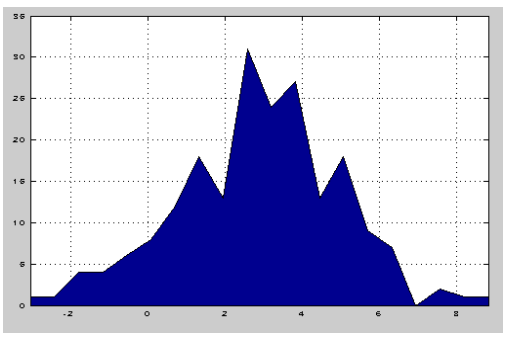
hist(y, x)



- O 2º vector debe ser crecente (se non, erro)
- Para que o histograma sexa cunha liña continua:

med=3;desv=2;n=20;
x=normrnd(med,desv,1,200);
t=linspace(min(x),max(x),n);
area(t,hist(x,n));grid on

Números aleatorios seguindo unha distribución gausiana

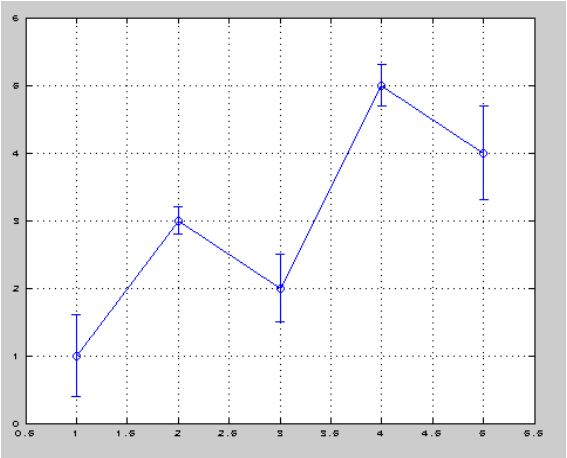


Gráficas con barras de erros

- Gráficas con barras de erros (p.e. para desviacións típicas de valores promedio):

errorbar(datos, erro, opcións)

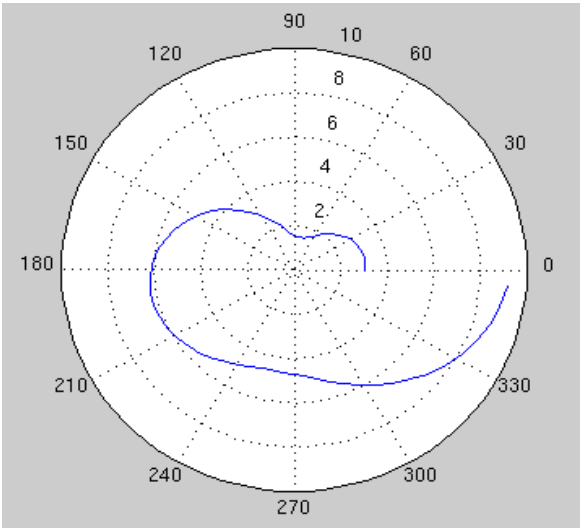
Ex: *x = [1 3 2 5 4];*
erro = [0.6 0.2 0.5 0.3 0.7];
errorbar(x, erro, 'o-')



Curvas en coordenadas polares

- Ecuación da curva en coordenadas polares: *radio = radio(ángulo)*: $\rho=\rho(\theta)$
- Ex: *polar(ángulo, radio)*

ángulo: vector cos valores do ángulo; *radio*: vector cos valores do radio




```
t=0:0.1:2*pi;
r=3*cos(t).^2+t;polar(t,r)
```

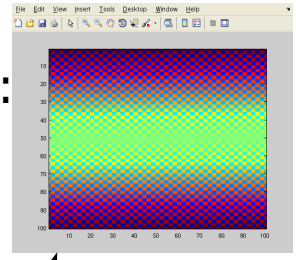
- *Función ezpolar($\rho(\theta)$,[θ_{min} θ_{max}])*

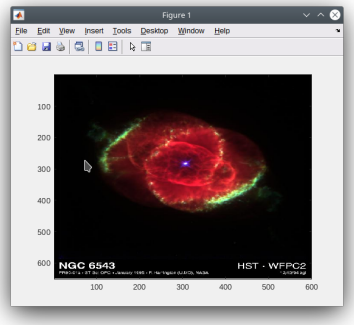
*ezpolar('3*cos(t)^2 + t', [0 2*pi])*

Máis funcións gráficas

- Gráficas para matrices:

  image(a): non escala
  imagesc(a): escala
  pcolor(a)



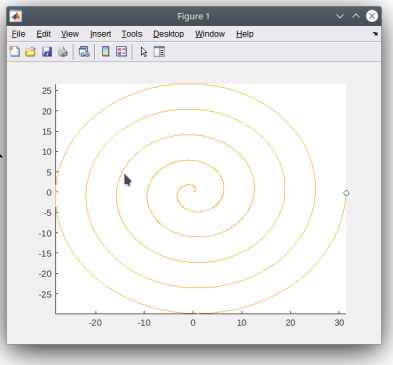


- Ex: *a=magic(10); image(a)*

  x=imread('ngc6543a.jpg');image(x)

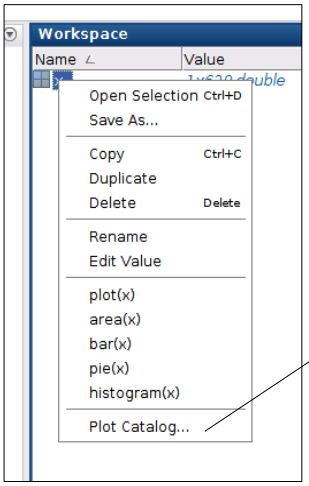
- Animacións: *comet(x), comet(x,y)*

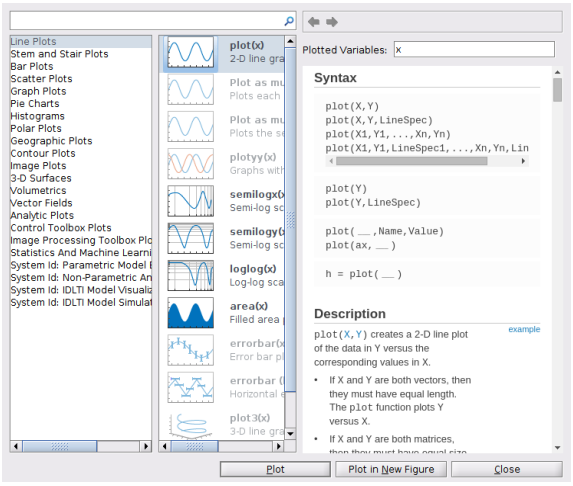
- Ex: *t=0:0.01:10*pi; x=t.*cos(t);y=t.*sin(t); comet(x,y)*



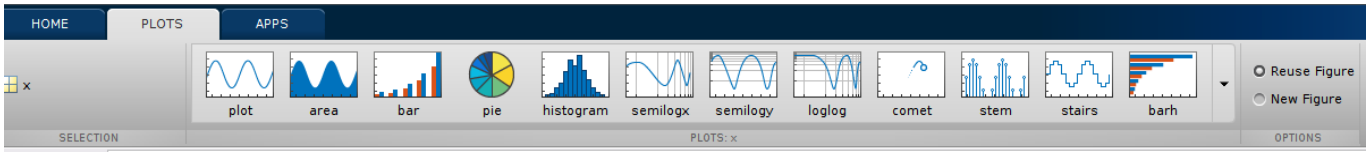
Asistente para gráficos

- Na ventá *workspace* pódese seleccionar un vector/matriz e o tipo de gráfico para representalo





- Tamén na barra de menú *Plots*



Exercicios

Representa as seguintes curvas e superficies:

1)Espiral de Arquímedes (polares): $r(\theta) = a\theta$

2)Bruxa de Agnesi: $y=\frac{8a^3}{x^2+4a^2}$

3)y=f(x) = tan x, en [-π/2, π/2] (usa `ezplot()`)

4)y=f(x) = $e^{-x/2}$ sen 20x, en [0, 10] (usa `fplot()`)

5) $y=\frac{x^2-x+1}{x^2+x+1}, x\in[-10,10]$

6)Representa os datos da táboa xunto coa función que os modela

$$y=320\left[\left(\frac{x}{210}\right)^{0.16}+1\right]$$

x	y
7E-5	345
2E-4	362
0.05	419
0.8	454
4.2	485
215	633
3500	831

Solucións aos exercicios

1) t = 0:0.1:pi; r = t; polar(t, r); (supoño a = 1)

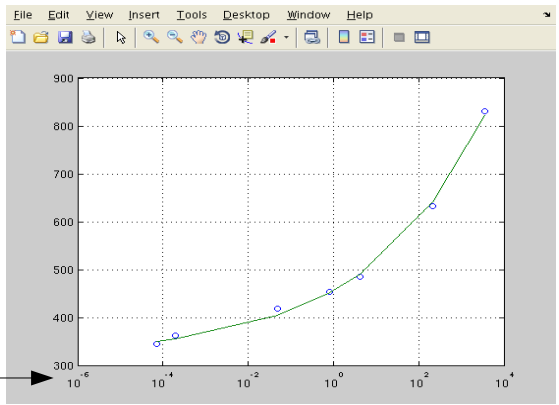
2) (supoño a = 1) fplot('8/(x^2 + 4)', [-10, 10])

3) ezplot('tan x', [-pi/2, pi/2])

*4) fplot('exp(-x/2)*sin(20*x)', [0, 10])*

5) fplot('(x^2 – x + 1)/(x^2 + x + 1)', [-10, 10])

6) a=load('datos.dat');
s = 320((a(:, 1)/210).^0.16 + 1);*
semilogx(a(:, 1), a(:, 2), 'bo-', a(:, 1), s, 'gs-')

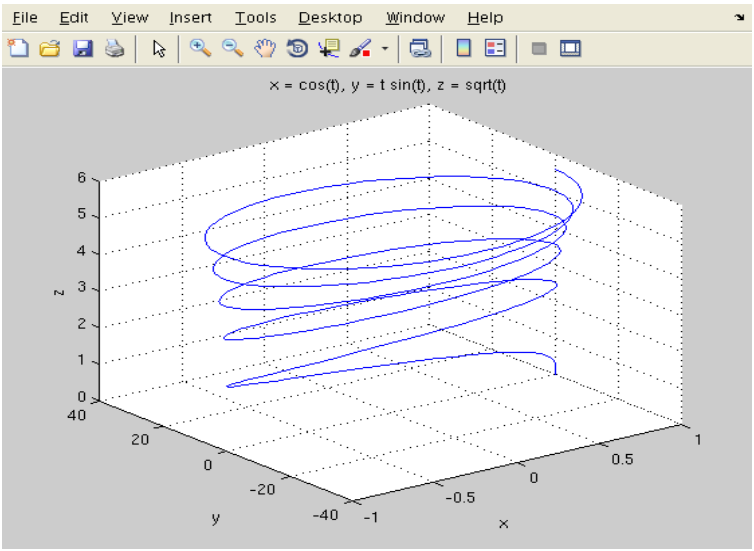


Curvas paramétricas en 3D: *ezplot3*

- Función *ezplot3('x(t)', 'y(t)', 'z(t)', [ini fin])*

  *ezplot3('cos(t)', 't*sin(t)','sqrt(t)', [0 10*pi])*

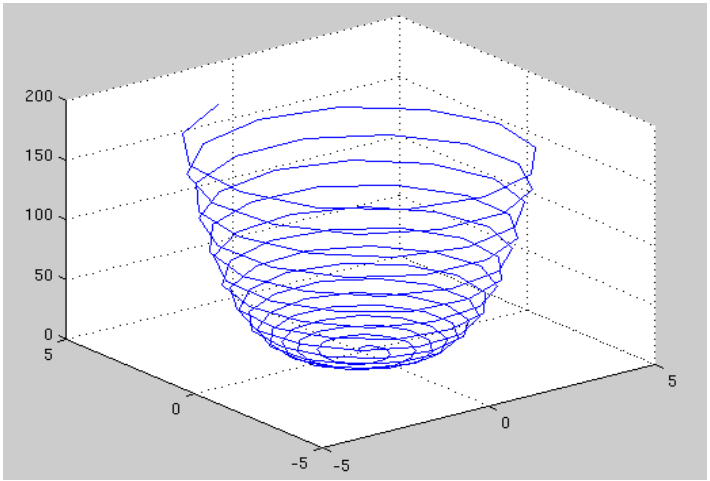
- Por defecto: rango *0..2*pi*



Curvas paramétricas en 3D: *plot3*

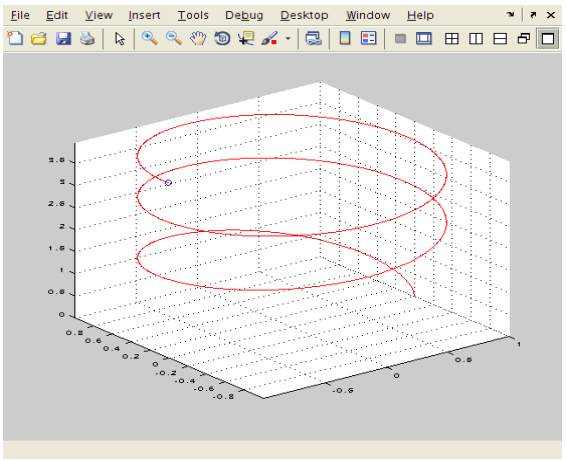
- Función *plot3(x,y,z,opcións)*
- *x*, *y*, *z*: vectores coas coordenadas dos puntos (ecuacións paramétricas); opcións: as mesmas que *plot*
- Exemplo:

```
t=0:0.1:6*pi;
x=sqrt(t).*sin(5*t);
y=sqrt(t).*cos(5*t);
z=t.^2./2;
plot3(x,y,z)
```



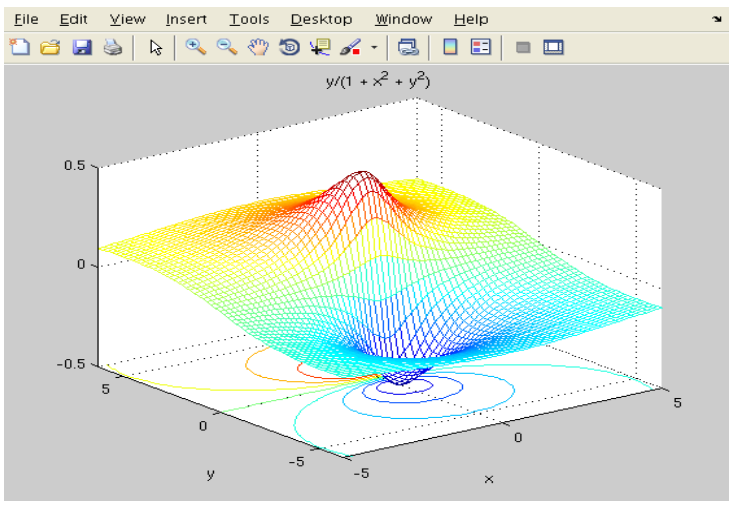
Curva animada en 3D: comet3

- Sintaxe: *comet3(x,y,z)*
- Vectores *x*,*y*,*z* da mesma dimensión
- Lonxitude elevada para que a animación sexa lenta
- Exemplo: *t=0:0.0001:10*pi; comet3(sin(t),cos(t),sqrt(t))*
- Moi útil para visualizar movementos en 3D



Superficies en 3D: *ezmesh, ezsurf*

- Función *ezsurf('f(x,y)',[xmin,xmax,ymin,ymax])*, e *ezmesh*
- Ex: *ezsurf('y/(1 + x^2 + y^2)',[-5,5,-2*pi,2*pi])*
- Tamén funcións *ezmesh(...)* e *ezmeshc(...)*, con contornos no plano XY



Superficies en 3D: *mesh*

Ecuación normal (explícita): *z = f(x, y)*

1. *[X Y]=meshgrid(x, y)* ou *[X Y]=meshgrid(x)*
   - *x,y*: vectores cos puntos en coordenadas. Ex: se os intervalos de representación son [0,1] para x e [-1,1] para y, podemos ter: *x=0:0.1:1* e *y=-1:0.1:1;*
   - *X,Y*: matrices coas coordenadas de tódolos puntos do plano XY para os cales se calcula *z = f(x,y)*

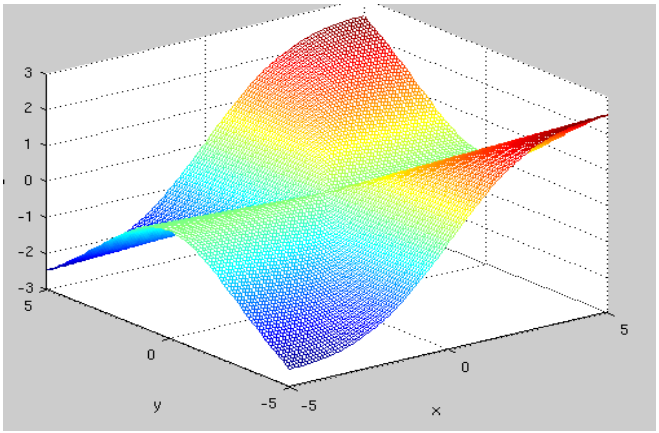
2. Cálculo de Z: *Z = f(X, Y)*. A expresión *f* debe estar vectorizada (operacións .* ./ .^)

3. Representación: *mesh(X, Y, Z)*

Superficies en 3D: *mesh*

- Exemplo: función $z=\frac{xy^2}{x^2+y^2}$ en [-5,5]x[-5,5]

```
x=[-5:0.1:5];y=[-5:0.1:5];
[X Y] = meshgrid(x, y);
Z = X.*Y.^2./(X.^2 + Y.^2 + eps);
mesh(X, Y, Z)
xlabel('x');ylabel('y');
zlabel('z')
```

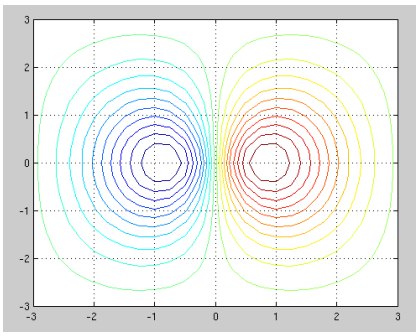


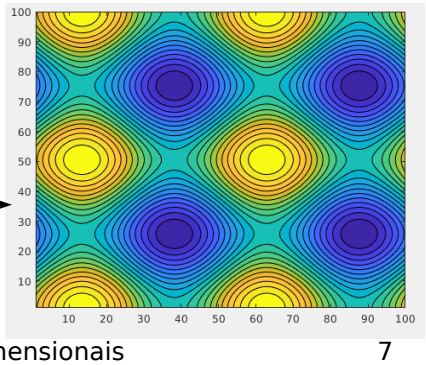
Superficies en 3D: *surf*, *contour*

- Tamén se pode empregar-la función *surf(X,Y,Z)*, que rechea a superficie
- Contorno: *contour(X,Y,Z,n)*

  contorno da superficie 3D sobre o plano XY; n=nº niveis do contorno (opcional)

- Mapa de calor: *contourf(Z,n)*



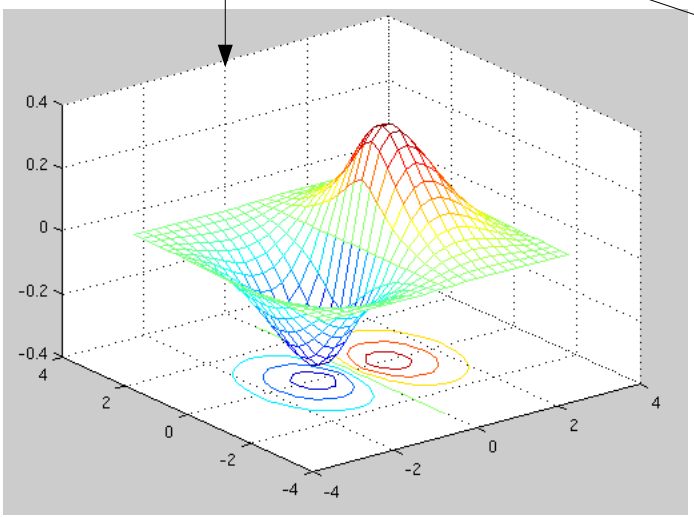


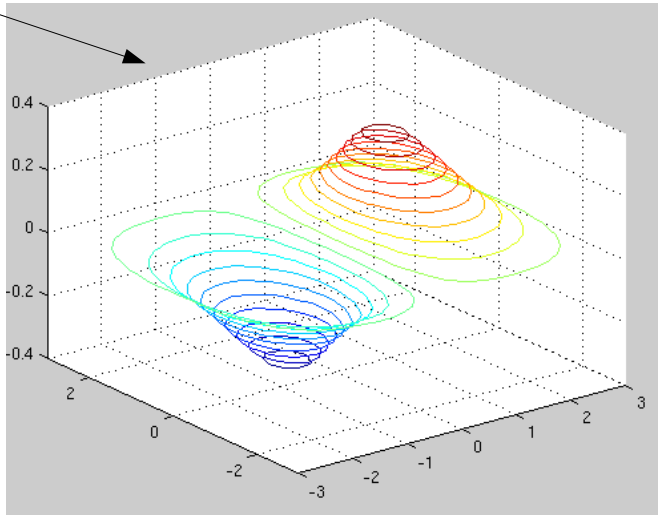
```
x = linspace(-2*pi,2*pi);
[X,Y] = meshgrid(x);
Z = sin(X)+cos(Y);
contourf(Z)
```


Superficies en 3D

$$z = f(x,y) = 1.8^{-1.5\sqrt{x^2+y^2}} \cdot \sin x \cdot \cos y$$

```
x=-3:0.25:3
[X Y] = meshgrid(x);
Z=1.8.^(-1.5*sqrt(X.^2+Y.^2)).*cos(0.5*Y).*sin(X);
meshc(X,Y,Z); contour3(X,Y,Z,20)
```

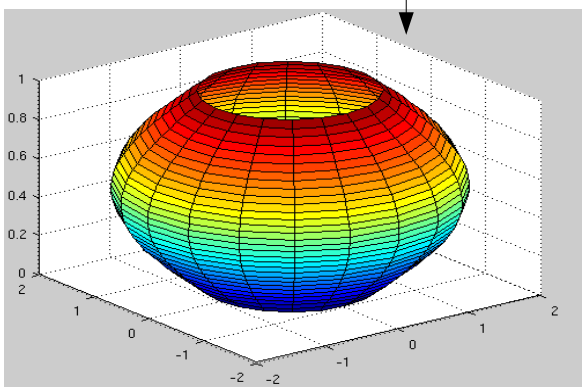


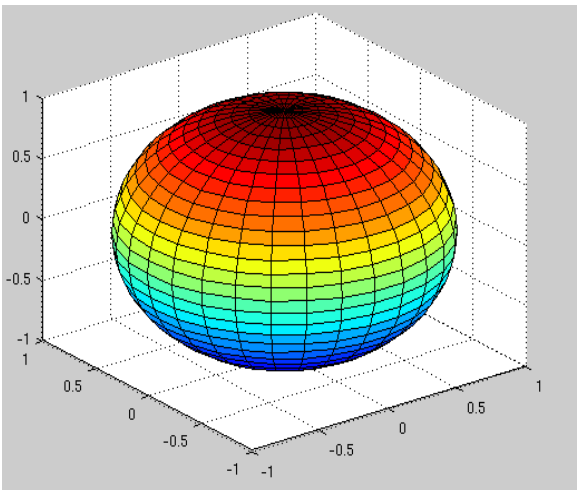


Outras gráficas

- Esfera: *sphere(50)* % nº de puntos
- Cilindro: *cylinder(r)* %r= vector cos radios

t=0:0.1:pi;
r=1+sin(t);
cylinder(r);





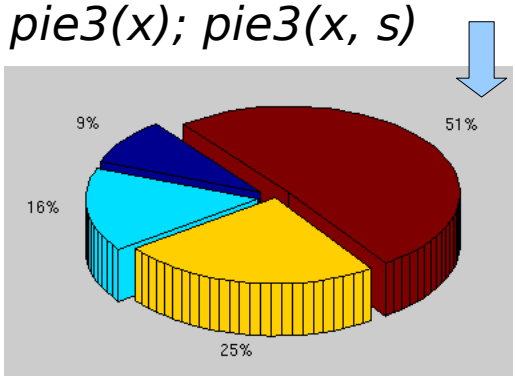
Outros diagramas 3D

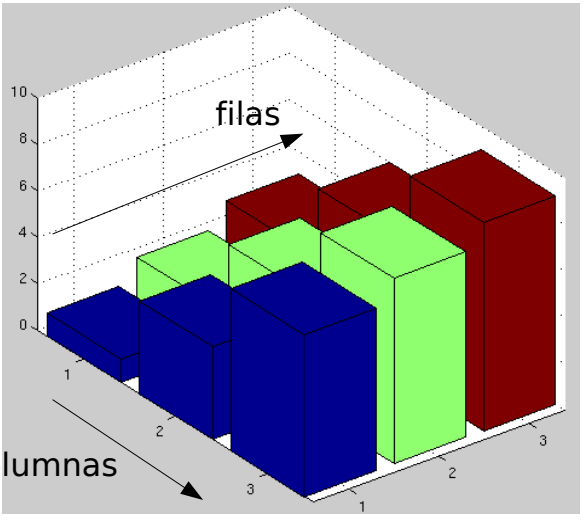
- Diagrama de barras 3D

  y=[1 2 3; 4 5 6; 7 8 9]; bar3(y)

- Tarta 3D (sentido antihorario): *x* e *s* deben ter igual lonxitude

  x = [5 9 14 29];
  s=[1 2 0 1]; %separación
  pie3(x); pie3(x, s)





Exercicios

Representa as seguintes curvas e superficies 3D:

1) $f(x,y)=\frac{sen\left(\sqrt{x^2+y^2}\right)}{\sqrt{x^2+y^2}}$ $x,y\in[-20,20]$

2) $x(t) = \cos t - t \operatorname{sen} t;\ y(t) = \operatorname{sen} t - t \cos t;\ z(t) = t^2$

3) $x(\theta) = 1 + \cos\theta;\ y(\theta) = 1 + \operatorname{sen}\theta;\ z(\theta)=4\theta$

4) $z=f(x,y)=\frac{1}{\pi}\sum_{n=1}^{10}\sin[(2n-1)\pi\, x]\sinh[(2n-1)\pi\, y],\ x,y\in[-0.1,0.1]$

5) $z=-x^2/4-y^2/4$, $x,y\in[-4,4]$ (fai o contorno 3D)

6) $z=(y+3)^2+1.5x^2-x^2y$, $x,y\in[-3,3]$ (ídem)

7) $x(t)=2\cos(2\pi t)$, $y(t)=2\operatorname{sen}(2\pi t)$, $z(t)=t$

8) $z=\frac{x}{x^2+y^2}$, $x,y\in[-0.1,0.1]$

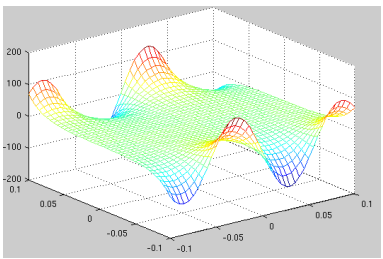
Solucións aos exercicios (I)

1) *ezmesh('sin(x^2 + y^2)/(x^2 + y^2)')*; ou ben: *[X Y] = meshgrid(-20:0.1:20); Z = sin(X.^2 + Y.^2)./(X.^2 + Y.^2) mesh(X, Y, Z)*

2) *ezplot3('cos(t)-t*sin(t)','sin(t)-t*cos(t)','t^2',[0 10*pi]);* ou ben: *t = 0:0.1:10*pi; x= cos(t) – t.*sin(t); y = sin(t) – t.*cos(t); z= t.*t; plot3(x, y, z)*

3) *ezplot3('1+cos(t)','1+sin(t)','4*t',[0 10*pi])* ou ben: *t = 0:0.1:10*pi; x= 1+cos(t); y=1+sin(t); z=4*t; plot3(x, y, z)*

4) *[X Y] = meshgrid(-0.1:0.005:0.1); Z = f(X, Y); mesh(X, Y, Z)*



```
function z = f(x, y)
z = 0;
for n = 1:10
    z = z + sin((2*n - 1)*pi*x).*sinh((2*n - 1)*pi*y);
end
z = z/pi;
end
```

Solucións aos exercicios (II)

5) ezmesh('-x^2/4 – y^2/4',[-4 4])
ou ben: [X Y] = meshgrid(-4:0.1:4); Z=-X.^2/4 -Y.^2/4; meshc(X, Y, Z)

*6) ezmesh('(y+3)^3 + 1.5x^2*y',[-3 3])*
ou ben: *[X Y] = meshgrid(-3:0.1:3); Z = (Y + 3).^2 + 1.5*X.^2 – X.^2.*Y; meshc(X, Y, Z)*

*7) ezplot3('2*cos(2*pi*t)','2*sin(2*pi*t)','t',[0 pi])*
ou ben: *t = 0:0.1:2*pi; x = 2*cos(2*pi*t); y = 2*sin(2*pi*t); z = t; plot3(x, y, z)*

8) ezmesh('x/(x^2 + y^2)',[-1 1])
ou ben: *[X Y] = meshgrid(-1:0.1:1); Z = X./(X.^2 + Y.^2); mesh(X, Y, Z)*

Polinomios

- Definición dun polinomio:
  - A partir dos seus coeficientes: $p(x) = x^5+6x^2+7x+3$: *p = [1 0 0 6 7 3];*
    - Ollo: comezar polo monomio de máis alto grao e poñer 0 nos coeficientes dos monomios ausentes
  - A partir das raíces: *p(x)*=polinomio con raíces 1, 1, 0, -1: *poly([1 1 0 -1])*: da un polinomio con coeficiente 1 para o monomio de máis algo grao
- Valor dun polinomio nun punto x_0: *polyval(p, x0)*. Ex: *polyval(p, 1)*
- Raíces dun polinomio: *roots(p)*: retorna un vector coas raíces

Representación gráfica e operacións con polinomios

- Representación gráfica do polinomio:
  x = -1:0.1:1;p=[1 2 3 1 0];
  y=polyval(p,x);
  plot(x,y,'o-')
- Suma / resta de polinomios (p, q da mesma orde): *s = p+q;* *r=s-q;* Se *length(p)≠length(q): s = p + [0 0 q]*
- Produto de polinomios: *prod=conv(p, q)*
- Cociente de polinomios: *[c r]=deconv(p, q)*
- Derivada dun polinomio: *d = polyder(p);*
- Derivada dun produto de pols: *d=polyder(p, q)*
- Integral indef. dun polinomio: *ip = polyint(p);*

Operacións con polinomios

- Derivada dun cociente de pols: *[n d]=polyder(p,q); n* e *d* son os vectores de coeficientes dos polinomios do numerador e denominador da derivada do cociente
- Descomposición dun cociente de polinomios en suma de fraccións:

$$\frac{n(x)}{d(x)}=\sum_{i=1}^{N}\frac{r_i}{(x-p_i)^{m_i}}+di(x)$$

Depende das N raíces (polos) de $d(x)$, p_i con multiplicidade m_i. Termo directo: $di(x)$: polinomio

[r p di] = residue(n, d);

$r = r_i$: vector cos residuos, $p = p_i$ (vector cos polos), di = vector cos coeficientes do termo directo (polinomio)

Axuste de funcións a polinomios (I)

- Problema: dado un conxunto de puntos $\{(x_i, y_i), i=1...m\}$, atopa-lo polinomio $p(x)$ de grao n que mellor se axusta aos puntos (minimiza a suma dos erros cadráticos entre os puntos (x_i,y_i) e os valores $(x_i,p(x_i))$ do polinomio).

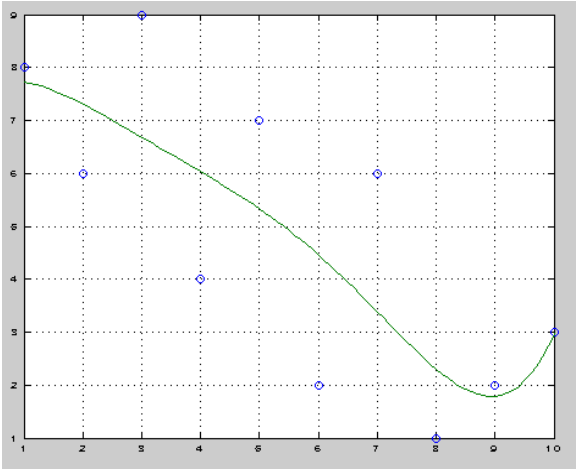
$$E = \sum_{i=1}^{m} (y_i - p(x_i))^2$$

- Función *polyfit(x, y, n)*
- *x, y* = vectores con coordenadas X, Y; *n* = orde do polinomio buscado
- Retorna o vector de coeficientes do polinomio que minimiza este erro

Axuste de funcións a polinomios (II)

• Exemplo: axuste a un polinomio de orde 5:

```
x=1:10;y=[8 6 9 4 7 2 6 1 2 3];
p=polyfit(x,y,5);
x2=1:0.1:10;
plot(x,y,'o',x2,polyval(p,x2),'-')
grid
```



• Permite axustar a funcións non polinómicas.

Axuste a función potencial

- **Potencial**: $y = ax^b$: Tomando logaritmos: $log(y) = log(a) + b \cdot log(x)$: $log(y)$ é un polinomio de orde 1 con variábel $log(x)$:

$$p = polyfit(log(x),log(y),1);$$

Como axusto a un polinomio de grado 1, $length(p)=2$, e entón

$$\left.\begin{array}{l} log(y)=b\cdot log(x)+log(a) \\ log(y)=p(1)*log(x)+p(2) \end{array}\right\} \; b=p(1),\ p(2)=log(a) \rightarrow a=e^{p(2)}$$

E a función potencial é: $\boxed{y=e^{p(2)}x^{p(1)}}$

```
x=[1 2 3 4 5 6 7 8 9 10];y=[8 6 9 4 7 2 6 1 2 3];
p=polyfit(log(x),log(y),1);
plot(x,y,'o',x,exp(p(2))*x.^p(1),'-')
```


Axuste a función exponencial

- **Exponencial**: $y = ae^{bx}$ (ou ben $y = a10^{bx}$): tomando logaritmos: $log(y) = log(a) + b \cdot x$: $log(y)$ é un polinomio de orde 1 con variábel x:

$$p = polyfit(x,log(y),1);$$

Como axusto a un polinomio de grado 1, $length(p)=2$, e entón:

$$\left.\begin{array}{l} log(y)=b\cdot x+log(a) \\ log(y)=p(1)*x+p(2) \end{array}\right\} \; b=p(1),\ p(2)=log(a) \rightarrow a=e^{p(2)}$$

E a función exponencial é: $y=e^{p(2)}e^{p(1)x} \rightarrow \boxed{y=e^{p(2)+p(1)x}}$

```
x=[1 2 3 4 5 6 7 8 9 10];y=[8 6 9 4 7 2 6 1 2 3];
p=polyfit(x,log(y),1);
plot(x,y,'o',x,exp(p(2)+p(1)*x),'-')
```

Axuste a función logarítmica

- **Logarítmica**: $y = a \cdot log(x)+b$ (ou ben $y = a \cdot log_{10}x+b$): y é un polinomio de orde 1 en $log(x)$:

$$p = polyfit(log(x),y,1)$$

Temos que:

$$\left.\begin{array}{l} y=a \cdot log(x)+b \\ y=p(1) \cdot log(x)+p(2) \end{array}\right\} a=p(1), b=p(2)$$

E a función logarítmica é: $y=p(1) \cdot log(x)+p(2)$

```
x=[1 2 3 4 5 6 7 8 9 10];y=[8 6 9 4 7 2 6 1 2 3];
p=polyfit(log(x),y,1);
plot(x,y,'o',x,p(1)*log(x)+p(2),'-')
```


Axuste a función recíproca

- **Recíproca**: $y=\frac{1}{ax+b}$

Invertindo temos: $1/y=ax+b$, e $1/y$ é un polinomio de orde 1 en x:

$$p = polyfit(x,1./y,1)$$

Temos que:

$$\left.\begin{array}{l} 1/y=a \cdot x+b \\ 1/y=p(1) \cdot x+p(2) \end{array}\right\} a=p(1), b=p(2)$$

E a función recíproca é: $y=\frac{1}{p(1) \cdot x+p(2)}$

```
x=[1 2 3 4 5 6 7 8 9 10];y=[8 6 9 4 7 2 6 1 2 3];
p=polyfit(x,1./y,1);
plot(x,y,'o',x,1./(p(1)*x+p(2)),'-')
```

Interpolación de funcións

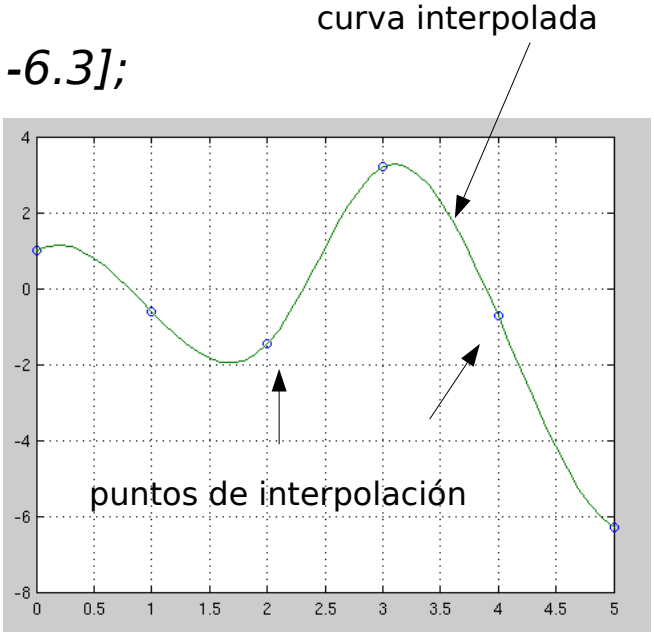
- Problema: temos m puntos $\{(x_j,y_j),j=1...m)\}$ dunha función $f(x)$ descoñecida, e queremos coñecer $f(x_i)$ con $x_i \neq x_j$, $j=1...m$. Usamos a función *interp1* de Matlab:

$$y2 = interp1(x, y, x2, 'método')$$

- x e y = vector cos puntos x_j e y_j
- *x2*: puntos nos que queremos calcular *f(x2)*
- *y2*: valores interpolados (estimados) para *f(x2)*
- método: *nearest* (retorna *y* para o punto *x* máis cercano a *xi*), *linear* (usando interpolación *spline* lineal), *spline* (usando interp. *spline* cúbica) e *pchip* (usando interp. cúbica de Hermite)

Exemplo de interpolación

```
x = 0:1:5;
y = [1 -0.62 -1.47 3.2 -0.73 -6.3];
x2=0:0.1:5;
y2=interp1(x,y,x2,'spline');
plot(x,y,'o',x2,y2,'-')
```



Función *spline* para interpolación con splines cúbicas

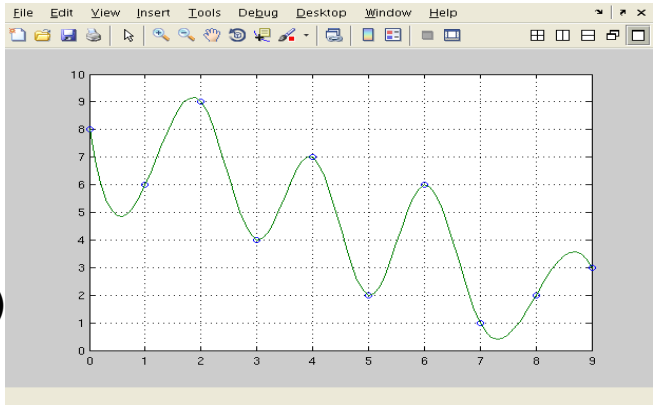
- Funcións polinómicas a cachos

y2=spline(x,y,x2)

x = coordenadas X dos puntos
y = coordenadas Y dos puntos
x2 = puntos nos que se calcula o polinomio
y2 = valor da *spline* en *x2*

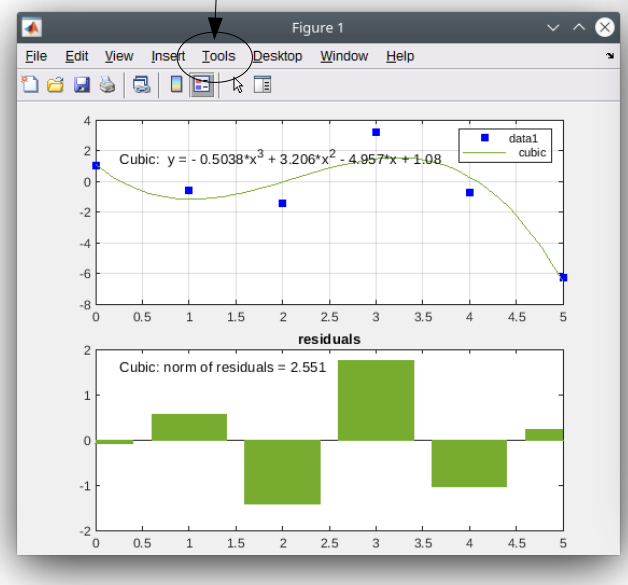
- Exemplo:

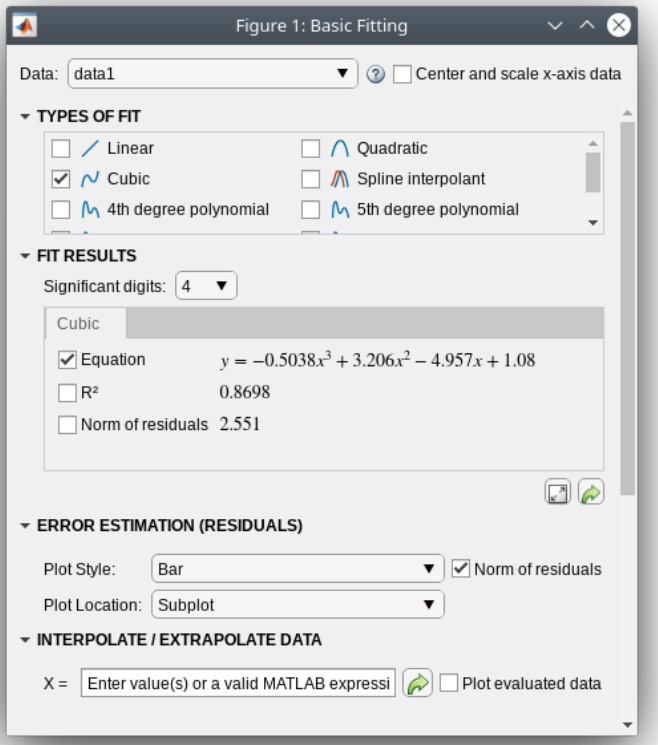
```
x=[0 1 2 3 4 5 6 7 8 9];
y=[8 6 9 4 7 2 6 1 2 3];
x2 = 0:0.1:9;
plot(x,y,'o',t,spline(x,y,x2))
grid
```



Asistente para interpolación

- Menú *Tools* -> *Basic Fitting* en ventá de figura





Exercicios

x	y
0	1
1	-0,6242
2	-1,4707
3	3,2406
4	-0,7366
5	-6,3717

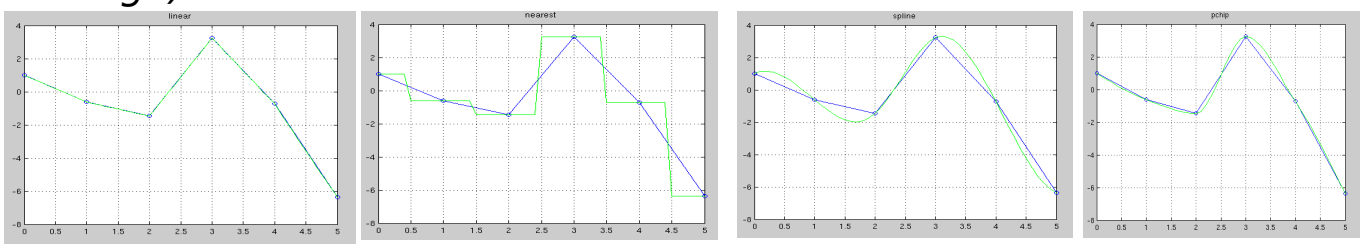
- Dado o polinomio $p(x) = x^5 - 12x^4 + 40x^3 - 17x^2 - 72x + 36$, calcula as súas raices, o seu valor no punto x = 1 e $p'(x)$
- Considera o polinomio $q(x) = (x-1)^2(x-2)(x-10)$. Calcula $p+q$ (p é o do exercicio anterior), $p \cdot q$ e a súa derivada, o cociente p/q e a súa derivada. Calcula os residuos, polos e termo directo de p/q
- Os puntos seguintes corresponden á función $f(x)=1.5^x cos2x$. Interpólaos usando os métodos *linear*, *spline*, *pchip* e representa gráficamente os puntos, a función *f* e as funcións interpoladas

Solucións aos exercicios

1) `p = [1 -12 40 -17 -72 36];roots(p); polyval(p, 1); polyder(p)`

2) `q = poly([1 1 2 10]); p+[0 q]; conv(p, q); polyder(p, q); deconv(p, q);polyder(deconv(p, q)); [r po d] = residue(p, q)`
-> r =[27.1852 2.0000 -6.1852 -2.6667], po =[10.0000 2.0000 1.0000 1.0000], d =[1 2] (polinomio de orde 1)

3) `x = 0:5; y = [1 -0,6242 -1,4707 3,2406 -0,7366 -6,3717]; plot(x, y)`
`xi=0:0.1:5; yi=interp1(x, y, xi, 'linear'); plot(x, y, 'bo-', xi, yi, 'g-')`



Programación estruturada en Matlab
Exercicios clases interactivas

Semana 10

Traballo en clase

1. **Entrada/Saída básica. Vectores. Bucles definidos. Vectorización**. Escribe un programa chamado `sumatorio.m` que lea por teclado dous vectores **v** e **w**. Debes comprobar que teñen a mesma lonxitude n. O programa debe calcular:

$$s = \sum_{i=1}^{n} \sum_{j=1}^{i} v_i w_j \tag{1}$$

   Proba con $\mathbf{v} = (1, 2, 1)$ e $\mathbf{w} = (-1, 0, 1)$ e tes que obter $s = -3$.

```
clear
while 1
    v=input('vector v: ');n=numel(v);  % introducir vector entre corchetes
    w=input('vector w: ');
    if n==numel(w); break; end
end

% como en fortran
r = 0;
for i = 1:n
    t = 0;
    for j = 1:i
        t = t + w(j);
    end
    r = r + v(i)*t;
end
fprintf('r = %g\n', r);

% alternativa simple
r = 0;
for i = 1:n
    r = r + v(i)*sum(w(1:i));
end
fprintf('r = %g\n', r);

% alternativa mais curta
r = 0; t = 0;
for i = 1:n
    t = t + w(i); r = r + v(i)*t;
end
fprintf('r = %g\n', r);
```

2. **Matrices**. Escribe un programa chamado `matriz.m` que lea por teclado unha matriz **A**, cadrada de orde n, e calcule:

   - A súa traza (suma dos elementos da diagonal principal), definida pola ecuación:

$$tr(A) = \sum_{i=1}^{n} a_{ii} \tag{2}$$

   - A suma do seu triángulo superior.
   - Determine se a matriz é simétrica: $A_{ij} = A_{ji}, i, j = 1, \ldots, n$.

```
clear
% introducir elementos entre corchetes, filas separadas por punto e coma
while 1
  a=input('matriz a [;]? ');[n,m]=size(a);
  if n==m; break; end
end
disp('a='); disp(a)

% traza
tr = trace(a);
%tr = sum(diag(a));
fprintf('tr = %g\n', tr);

% suma do triangulo superior
% sts = sum(sum(triu(a)) - diag(diag(a)));
sts = sum(sum(a - tril(a)));
fprintf('sts = %g\n', sts);

% matriz simetrica / non simetrica
if all(all(a == a'))
    disp('matriz simetrica');
else
    disp('matriz non simetrica');
end
```

3. **Números aleatorios**. Escribe un programa chamado `aleatorio.m` que defina $n = 10$ e cree un vector $\mathbf{x}$ con n números enteiros aleatorios entre 0 e 100. Queremos poñer estos valores nos triángulos superior e inferior dunha matriz cadrada $\mathbf{a}$. É decir, queremos poñer tódolos elementos de $\mathbf{x}$ no triángulo superior da matriz, e queremos poñer tódolos elementos de $\mathbf{x}$ tamén no triángulo inferior, de modo que $a_{ij} = a_{ji}$, con $j \neq i$. Polo tanto, a orde m da matriz $\mathbf{a}$ debe verificar que $\frac{m^2-m}{2}$, que é o número de elementos do triángulo superior ou inferior, sexa o mínimo número enteiro igual ou maior que n. Tomando a igualdade temos que $m^2 - m - 2n = 0$. Polo tanto, o programa debe crear unha matriz cadrada de orde $m = \left\lceil \frac{1+\sqrt{1+8n}}{2} \right\rceil$. No caso $n = 10$ temos que $m = 5$. Esta matriz debe ter valores nulos na diagonal ($a_{ii} = 0, i = 1 \ldots, m$) e os elementos do vector $\mathbf{x}$ nos triángulos superior e inferior. É dicir:

$$a_{12} = a_{21} = x_1, a_{13} = a_{31} = x_2, \ldots, a_{1m} = a_{m1} = x_m,$$
$$a_{23} = a_{32} = x_{m+1}, \ldots, a_{(m-1)m} = a_{m(m-1)} = x_n$$

```
clear
% para inicializar de forma distinta o xerador de numeros aleatorios
%  en cada execucion: rng('shuffle')
% para inicializar sempre da mesma forma: rng('default')
n=10;x=randi([0  100],1,n)
disp('x='); disp(x)
m=ceil((1+sqrt(1+8*n))/2);a=zeros(m);k=1;
for i=1:m
    for j=i+1:m
        t=x(k);a(i,j)=t;a(j,i)=t;k=k+1;
        if k>n; break; end
    end
    if k>n; break; end
end
disp('a='); disp(a)
```

Alternativa más corta usando `return`:

```
clear
n=10;x=round(100*rand(1,n));
disp('x='); disp(x)
m=floor((1+sqrt(1+8*n))/2);a=zeros(m);k=1;
```

```
for i=1:m
    for j=i+1:m
        t=x(k);a(i,j)=t;a(j,i)=t;k=k+1;
        if k>n
            disp('a=');disp(a);return
        end
    end
end
```

4. **Medida de tempos. Bucle indefinido**. Descarga o programa `tempo.m` desde este **enlace**. Este programa pide por teclado a introducción dun número natural n, comprobando que o número é positivo, e define un vector **x** con n elementos onde $x_i = i$, $i = 1 \ldots n$. Proba distintos xeitos de crear o vector **x** e calcula o tempo computacional que necesita cos comandos de Matlab `tic` e `toc` para distintos valores de n. Usa $n = 10^5$.

```
% Eficiencia computacional
clear;n=0;
while n <= 0  % Ler o valor de n positivo
    n=round(input('Numero de iteracions (> 0): '));  % Usa n=1e5
end
%-------------------------------------------------
tic;x=[]; % inicia o reloxio e crea un vector baleiro
for i=1:n
    x=[x i];
end
% tempo transcorrido desde que se executou tic
fprintf('Tempo agregando elementos o vector=%g s.\n',toc)
%-------------------------------------------------
tic
for i=1:n
    y(i)=i;
end
fprintf('Tempo con vector baleiro= %g s.\n',toc)
%-------------------------------------------------
tic; z=zeros(1,n);  % con reserva de memoria
for i=1:n % declarando un vector de tamanho n
    z(i)=i;
end
fprintf('Tempo con reserva de memoria= %g s.\n',toc)
%-------------------------------------------------
tic; t=1:n;  % sen utilizar bucle for
fprintf('Tempo utilizando o operador= %g s.\n',toc)
```

5. **Progreso dun programa**. Descarga o programa `progreso.m` desde este **enlace**. Este programa imprime o seu progreso (en %).

```
n=1000000;
for i=1:n
    fprintf('%10.1f%%\r',100*i/n)
end
```

Este programa funciona en Octave ou executando Matlab dende a terminal de comandos (neste caso, debes engadir un comando `exit` ao final do programa. Se é dentro do entorno de Matlab, a secuencia de escape `r` non funciona e hai que facer o seguinte:

```
n=100000;
fprintf(repmat(' ',1,7));
for i=1:n
  fprintf(repmat('\b',1,7));fprintf('%6.2f%%',100*i/n);
end
fprintf('\n');
```

Descarga o programa `progreso2.m` desde este **enlace**. Este programa emplía o anterior para que mostre, en cada iteración, o tempo estimado que queda de execución. Usa para isto unha función chamada `strtime(t)` que transforma un tempo `t` numérico nunha cadea de caracteres da forma `Y y MM m DD d HH h MM m SS s`.

```
n=1000000;t=tic;
for i=1:n
    t2=toc(t);t3=t2*(n-i)/i;
    fprintf('%10.1f%% %40s\r',100*i/n,strtime(t))
end

function str=strtime(t)
    if t<60  % seconds in a minute
        str=sprintf('%2d s',floor(t));
    elseif t<3600  % seconds in an hour
        m=floor(t/60);s=floor(t-60*m);
        str=sprintf('%2d m %2d s',m,s);
    elseif t<86400  % seconds in a day
        h=floor(t/3600);m=floor((t-3600*h)/60);s=floor(t-3600*h-60*m);
        str=sprintf('%2d h %2d m %2d s',h,m,s);
    elseif t<2592000  % seconds in a month
        d=floor(t/86400);h=floor((t-86400*d)/3600);m=floor((t-86400*d-3600*h)/60);
        s=floor(t-86400*d-3600*h-60*m);str=sprintf('%2d d %2d h %2d m %2d s',h,m,s);
    elseif t<31536000  % seconds in a year
        month=floor(t/2592000);d=floor((t-2592000*month)/86400);
        h=floor((t-2592000*month-86400*d)/3600);
        m=floor((t-2592000*month-86400*d-3600*h)/60);
        s=floor(t-2592000*month-86400*d-3600*h-60*m);
        str=sprintf('%2d month %2d d %2d h %2d m %2d s',h,m,s);
    else
        y=floor(t/31536000);month=floor((t-31536000*y)/2592000);
        d=floor((t-31536000*y-2592000*month)/86400);
        h=floor((t-31536000*y-2592000*month-86400*d)/3600);
        m=floor((t-31536000*y-2592000*month-86400*d-3600*h)/60);
        s=floor(t-31536000*y-2592000*month-86400*d-3600*h-60*m);
        str=sprintf('%1d y %2d month %2d d %2d h %2d m %2d s',h,m,s);
    end
end
```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa chamado que lea por teclado dous vectores **v** e **w** e unha matriz **A**, todos eles da mesma orde n, e calcule o produto:

$$\mathbf{vAw}' = [v_1 \dots v_n] \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix}$$

```
clear
v = input('vector v: '); % introducir vector entre corchetes
w = input('vector w: '); % introducir vector entre corchetes
a = input('matriz a: '); % elementos entre corchetes, filas separadas por punto e coma
p = v*a*w';
fprintf('prod = %g\n', p);
```

2. Escribe un programa que imprima os n primeros termos da serie de Fibonacci, dados por $a_0 = 1; a_1 = 1; a_i = a_{i-1} + a_{i-2}, i = 2, ...n$

3. Escribe un programa que defina unha matriz **a** 5×8 onde cada elemento con fila e columna par sexa a raíz cadrada da suma dos índices do elemento. Nos restantes elementos, o valor debe se-lo cadrado da suma dos índices.

4. Escribe un programa en Matlab que calcule a suma dos m primeiros termos da serie:

$$\sum_{n=0}^{m} \frac{(-1)^n}{2n+1} \tag{3}$$

Proba con $m = 10$ e $m = 100$, e compara o resultado con $\pi/4$ (suma da serie infinita).

Semana 11

Traballo en clase

1. **Funcións propias. Vectorización**. Escribe un programa chamado `intervalo.m` cunha función `funcionf(x)` en Matlab que calcule $f(x)$ definida por:

$$f(x) = \begin{cases} 4e^{x+2} & -6 \le x < -2 \\ x^2 & -2 \le x < 2 \\ (x+6.5)^{1/3} & 2 \le x < 6 \end{cases}$$

   O programa debe calcular os valores de $f(x)$ para $x \in [-10, 10]$ e representalos gráficamente.

```
clear
x = -10:0.1:10; n = length(x); y = zeros(1,n);
for i = 1:n
    y(i) = funcionf(x(i));
end
plot(x, y)
grid on
%-------------------------------
function y = funcionf(x)
  if x < -6
    y = 0;
  elseif x < -2
    y = 4*exp(x+2);
  elseif x <= 2
    y = x*x;
  elseif x < 6
    y = (x+6.5)^(1/3);
  else
    y = 0;
  end
end
```

   Versión vectorizada:

```
x = -10:0.1:10;
y = funciony(x);
plot(x,y)
%-------------------------------
function y = funcionf(x)
n = size(x, 2); y = zeros(1, n);
t = find(x >= -6 & x < -2); y(t) = 4*exp(x(t)+2);
t = find(x >= -2 & x < 2); y(t) = x(t).^2;
t = find(x >= 2 & x <= 6); y(t) = nthroot(x(t) + 6.5, 3);
```

2. **Chamada a funcións anónimas con `feval()`. Gráficas simultáneas con lendas**. Escribe un programa chamado `grafica.m` que represente gráficamente na mesma gráfica e no intervalo $[-\pi, \pi]$, con 100 puntos, as seguintes funcións: $f_1(x) = \sin x \cos x$, como función inline (en cor azul); $f_2(x) = \sin\cos 2x$ como función anónima (en cor vermella); $f_3(x) = \sin x$ como referencia a función con `str2func` (en cor verde); e $f_4(x) = \cos x$ como referencia a función con `@` (en cor negra). Engade etiquetas de texto coas expresións das distintas curvas.

```
clear
f{1}=inline('sin(x).*cos(x)'); % funcion inline
f{2}=@(x) sin(cos(2*x)); % funcion anonima
f{3}=str2func('@(x) sin(x)'); % referencia a funcion con str2func
f{4}=@cos;  % referencia con @
n=numel(f);m=100;x=linspace(-pi,pi,m);y=zeros(n,m);
c={'b','r','g','k'};clf;hold on
for i=1:n
```

```
    y=feval(f{i},x);plot(x,y,c{i})
%   plot(x,f{i}(x),c{i})   % alternativa
end
label={'sin(x)cos(x)'};  % func2str non funciona con inline
for i=2:n; label{i}=func2str(f{i}); end
legend(label,'location','bestoutside');grid on
```

3. **Resolución de sistema de ecuacións lineares mediante Eliminación Gaussiana**. Consideremos un sistema de n ecuacións lineais con n incógnitas:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= a_{1(n+1)} \\ &\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= a_{n(n+1)} \end{aligned}$$

O método de eliminación gausiana consiste en empregar as seguintes transformacións:

- Dividir tódolos elementos dunha fila polo mesmo número.
- Sumar a tódolos elementos dunha fila o produto dun escalar polo elemento correspondente doutra fila

para transformar este sistema no seguinte:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= a_{1(n+1)} \\ a_{22}x_2 + \ldots + a_{2n}x_n &= a_{2(n+1)} \\ &\ldots \\ a_{nn}x_n &= a_{n(n+1)} \end{aligned} \tag{4}$$

onde os a_{ij} **non** son os iniciais. Concretamente, para cada incógnita x_i, con $i = 1, \ldots, n-1$ (coa última incógnita non hai que facer nada), substitúe a ecuación j, con $j = i+1, \ldots, n$, pola ecuación j menos a ecuación i multiplicada por $l = a_{ji}/a_{ii}$. Deste modo, o coeficiente da incógnita x_i na ecuación j (con $j = i+1, \ldots, n$) pasa a ser nulo. Dito doutra forma, a columna i da matriz ampliada pasa a valer 0 por debaixo da diagonal. No sistema transformado (ecuación 5) podemos despexar directamente x_n, sustituir na $(n-1)$-ésima ecuación e despexar x_{n-1} e así sucesivamente ata calcula-las n incógnitas, mediante a fórmula.

$$x_i = \frac{1}{a_{ii}} \left(a_{i(n+1)} - \sum_{j=i+1}^{n} a_{ij}x_j \right) \quad i = n, \ldots, 1 \tag{5}$$

Isto vale se $a_{ii} \neq 0$ para todo $i = 1, \ldots, n$. Pero se $\exists i$ tal que $a_{ii} = 0$, non se pode dividir por a_{ii} e resulta necesario realizar o denominado "pivote", consistente en intercambiar a ecuación i-ésima por aquela ecuación p posterior (é decir, $p > i$) que ten o coeficiente a_{pi} con valor absoluto máximo, coa intención de que $a_{pi} \neq 0$. Este cambio permitirá facer que a_{ii} pase a ser non nulo, a non ser que se trate dun sistema compatíbel indeterminado, porque neste caso unha ou varias das derradeiras ecuacións serán nulas e non será posíbel intercambiar filas para que $a_{ii} \neq 0$. Escribe un programa chamado `gauss.m` que implemente este método de resolución de sistemas de ecuacións lineares. Proba cos seguintes sistemas:

a) **Sistema compatíbel determinado sen pivote**. Solución: $x = 3, y = -2, z = 5$. Arquivo `sistema_compatibel_determinado.dat`:

```
1  2  1  4
-3 -2  9  40
4  9  6  24
```

b) **Sistema compatíbel determinado con pivote** (coeficiente nulo na diagonal). Solución: $x = 1, y = 0, z = -1$. Arquivo `sistema_compatibel_determinado_pivote.dat`:

```
0  2 -1  1
1 -1  1  0
2  0 -1  3
```

c) **Sistema incompatíbel**. Arquivo `sistema_incompatibel.dat`:

$$\begin{matrix} 1 & 2 & 3 & 0 \\ 2 & 4 & 6 & 5 \\ 3 & 6 & 9 & 2 \end{matrix}$$

d) **Sistema compatíbel indeterminado**. Ecuacións do subespazo vectorial solución (unha recta de dimensión 1): $x + 3y = 4, z = -11$. Arquivo `sistema_compatibel_indeterminado.dat` seguinte:

$$\begin{matrix} 1 & 3 & 0 & 4 \\ -1 & -3 & 0 & -4 \\ 2 & 6 & 1 & -3 \end{matrix}$$

Emprega o depurador de Matlab para:

a) Deter a execución do programa nunha determinada sentenza (establecendo un punto de ruptura ou *breakpoint* co menú *Debug-Set/Clear breakpoint* ou coa tecla F12).

b) Executar as seguintes sentenzas paso a paso (menú *Debug-Step* ou tecla F10).

c) Entrar nunha función (menú *Debug-Step in* ou tecla F5).

d) Ver os valores das variábeis do programa (escribindo o seu nome na ventá de comandos, poñendo o rato enriba da variábel ou mirando o seu valor na ventá *Workspace*).

e) Continuar a execución (menú *Debug-Continue* ou coa tecla F5).

f) Ou deter a execución (menú *Debug-Exit debug mode* ou tecla MAY+F5), saíndo do modo de depuración.

```
clear
% sist_comp_det.dat, sist_comp_det_pivote, sist_comp_indet, sist_incomp
nf=input('arquivo? ','s');a=load(nf);[n m]=size(a); %a=matriz ampliada
if (n+1)~=m
    fprintf('#ecuacions~=#incognitas');return
end
disp('a=');disp(a);disp('--------------------');
c=a(:,1:n);b=a(:,m);rmc=rank(a(:,1:n));rma=rank(a);
if rmc~=rma
    fprintf('sistema incompatibel: rmc =% i rma =% i\n',rmc,rma);x=pinv(c)*b;
    fprintf('solucion de norma minima: ');disp(x);fprintf('|ax-b|=%g\n',norm(a*x-b))
    return
end
for i=1:n-1
    % se a(i,i)==0, pivote: intercambia ec. i coa que ten a(i,i) maximo
    if 0==a(i,i)
        % suma i a j porque p=1 para ec. i+1
        [~,j]=max(abs(a(i+1:n,i)));  p=i+j;
        % pivote: intercambia ecuacion i por p
        aux=a(i,:);a(i,:)=a(p,:);a(p,:)=aux;
    end
    if 0~=a(i,i)  % se o sistema e indeterminado, pode ser a(i,i)==0
        t=a(i,i);u=a(i,:);
        for j=i+1:n
            l=a(j,i)/t;a(j,:)=a(j,:)-l*u;
        end
    end
    disp(a); disp('--------')
end
if ra<n
    fprintf('sistema compatibel indeterminado\n')
    d=n-rmc;fprintf('solucion de dimension % i:\n',d)
    fprintf('ecuacion implicita da solucion:\n');disp(a(1:rmc,:))
    fprintf('ecuacion parametrica da solucion:\nx=\n')
    c=a(:,1:n);b=a(:,end);x0=pinv(c)*b;k=null(c);v=1:size(k,2);
    fprintf('%4.2g ',x0(1));fprintf('+ c%i(%4.2g)',v,k(1,:));fprintf('\n')
    for i=2:n
        fprintf('%4.2g ',x0(i));fprintf('     (%4.2g)',k(i,:));fprintf('\n')
    end
```

```
    return
end
x=zeros(1,n);
for i=n:-1:1
    j=i+1:n;x(i)=(a(i,m)-a(i,j)*x(j)')/a(i,i);
end
fprintf('sistema compatibel determinado\n')
fprintf('x  =');disp(x)
fprintf('a\b='); disp((c\b)')
```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa en Matlab que lea dende `datos.dat` os seguintes datos:

   31 26 30 33 33 39 41 41 34 33 45 42 36 39 37 45 43 36 41 37 32 32 35 42 38 33 40 37 50
   37 24 28 25 21 28 46 37 36 20 24 31 34 40 43 36 34 41 42 35 38 36 35 33 42 42 37 26 31

   O programa debe calcular, para cada fila: o valor medio, os valores por riba e por baixo da media, os valores dunha fila superiores ao seu correspondente da outra fila, os valores que coinciden co valor da outra fila na mesma posición, e os valores inferiores a 40.

2. Escribe un programa que realice o escrutinio de apostas de lotería primitiva. O programa debe pedir por teclado o nome dun arquivo de texto, e ler neste arquivo as apostas (en cada fila, unha aposta, integrada por 6 números enteiros entre 1 e 49). Logo debe pedir por teclado o nome dun arquivo coa aposta ganhadora (unha liña con 6 números enteiros entre 1 e 49). Por último, debe almacenar nun arquivo (de nome introducido por teclado) tódalas apostas con 3 ou máis acertos: o n^o de aposta, o n^o de acertos e a súa combinación de números asociada. Empregar os seguintes arquivos de mostra:

   ```
   apostas.dat
   3  5 19 16 33 41
   21  9  1 12 44 20
   12 24  1 23 47 28
    3  5 41 25 19  1
   18 12 11  1  8  9
   11 33 21 45  1 12
   ganhadora.dat
   3 5 19 16 33 41
   ```

   SOLUCIÓN:

   ```
   clear
   fap = input('arquivo coas apostas? ', 's');
   fgan = input('arquivo coa aposta ganhadora? ', 's');
   fres = input('arquivo cos resultados? ', 's');
   ap = load(fap); [nap n]=size(ap); g = load(fgan);
   fid = fopen(fres, 'w');
   if -1 == fid
       error(sprintf('fopen %s\n', fres));
   end
   for i = 1:nap
       acertos = 0;
       for j = 1:n
           acertos=acertos+any(ap(i,j)==g);
       end
       fprintf('aposta %i: %i acertos\n', i, acertos);
       if acertos > 3
           fprintf(fid, 'aposta %i: %i acertos: ', i, acertos);
           fprintf(fid, '%i ', ap(i, :)); fprintf(fid, '\n');
       end
   end
   fclose(fid);
   ```

3. **Función con varios valores retornados**. Escribe un programa que lea por teclado un número enteiro $n > 0$ e cree unha matriz **a** cadrada de orde n con valores enteiros aleatorios no conxunto $\{0, \ldots, n\}$. Logo, este programa debe chamar a unha función de Matlab `calcula(...)` (debes decidir os seus argumentos), que retorne unha matriz **b** cadrada e un vector **x**, ambos de orde n, e un escalar y. A matriz **b** retornada debe ter tódolos elementos nulos agás os das diagonais principal e secundaria, que deben coincidir cos da matriz **a**. O vector **x** retornado debe verificar que x_i, con $i = 1, \ldots, n$, sexa a suma dos elementos iguais a i na columna i da matriz **a**. Pola súa banda, o escalar y retornado debe ser o número de elementos nulos na matriz **a**. Finalmente, o programa principal debe pedir por teclado o nome dun arquivo e almacenar neste arquivo as matrices **a** e **b**, o vector **x** e o escalar y.

   SOLUCIÓN:

```
clear
n=0;
while n<=0
    n=round(input('n? '));
end
a=round(n*rand(n));
[b v ceros]= calcula(a);
% gardar en arquivo
arquivo=input('Introduce nome arquivo ', 's');
f=fopen(arquivo, 'w');
if f<0
    error('fopen %s\n', arquivo);
end
fprintf(f, 'Matriz a: \n');
for i=1:n
    fprintf(f, '%d ', a(i,:));fprintf(f, '\n');
end
fprintf(f, 'Matriz b: \n');
for i=1:n
    fprintf(f, '%d ', b(i,:));fprintf(f, '\n');
end
fprintf(f, 'Vector v: ');fprintf(f, '%d ', v);
fprintf(f, '\nCeros= %d \n', ceros);
fclose(f);
%-------------------------------------------------------------
function [b, x, y]=calcula(a)
% calcula: fai calculos sobre unha matriz de enteiros a
% b e a matriz a con ceros fora da diagonal principal e secundaria
% xi e a suma dos elementos de a que son igual a i
% c e o numero de elementos de a que son igual a 0
    n=size(a,2); m = n + 1;
    b=diag(diag(a));x=zeros(1,n);
    for i=1:n
        b(i, m-i)=a(i, m-i);
        x(i) = i*sum(a(:,i)==i);
    end
    y=sum(sum(a==0));
end
```

4. **Mínimos cuadrados**. Escribe un programa chamado `regresion.m` que calcule as constantes a e b que axustan os vectores **x** e **y** (ambos do mesmo tamaño) á recta de regresión $y = ax + b$. Os coeficientes a e b calcúlanse coas seguintes expresións:

$$a = \frac{N\sum_{i=1}^{N} x_i y_i - \left(\sum_{i=1}^{N} x_i\right)\left(\sum_{i=1}^{N} y_i\right)}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \qquad b = \frac{\left(\sum_{i=1}^{N} x_i^2\right)\left(\sum_{i=1}^{N} y_i\right) - \left(\sum_{i=1}^{N} x_i\right)\left(\sum_{i=1}^{N} x_i y_i\right)}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \tag{6}$$

   sendo N o tamaño dos vectores **x** e **y**. Representa gráficamente os vectores **x** e **y** xunto cos valores de y calculados segundo a ecuación da recta. Podes probar con **x**=[1 2 3 4 5] e **y**=[3 5.1 7 16.1 22.8].

Solución 1: un programa que codifique as expresións:

```
clear
x=[];y=0
while length(x)~=length(y)
    fprintf('introduce vectores x, y coa mesma lonxitude\n')
    x=input('x[]? ');
    y=input('y[]? ');
end
N = length(x);
Sx  = sum(x); Sy  = sum(y);
Sxy = sum(x.*y); Sx2 = sum(x.^2);
a = (N*Sxy -  Sx* Sy) / (N*Sx2-Sx^2);
b = (Sx2* Sy - Sx * Sxy) / (N * Sx2 - Sx^2);
fprintf('y= %g x + %g\n', a, b);

%representacion grafica
x_recta = [min(x) max(x)]; y_recta = a * x_recta + b;
figure(1)
hold on
plot(x, y, 'r*')
plot(x_recta, y_recta, 'g-');
hold off
% debuxar as duas graficas xuntas
figure(2)
plot(x, y, 'r*', x, y_recta, 'g-');
```

Solución 2: utilizando as funcións *polyfit()* e *polyval()* definidas en MATLAB:

```
clear
x=input('introduce vector x (entre []): ');
y=input('introduce vector y (entre []): ');
% coeficientes a=p(1) e b=p(2): y = a*x + b
p=polyfit(x, y, 1)
fprintf('y = %g x + %g\n', p(1), p(2));

%representacion grafica
y_recta=polyval(p,x)
plot(x, y, 'r*', x, y_recta, 'g-');
```

5. **Bucles indefinidos**. Escribe un programa chamado `valida.m` que lea por teclado un úmero enteiro n no rango $10 \leq n \leq 20$ e cre un vector **x** de lonxitude n. Logo, o programa debe ler por teclado números enteiros no rango $0 \leq x_i \leq n$, non introducindo en **x** aqueles valores fora de rango. Fai tamén este exercicio en Fortran.

```
clear
n = 0;
while n < 10 || n > 20
    n=input('introduce n no rango [10,20]: ');
end
x=zeros(1,n); i=0;
fprintf('introduce numeros enteiros no intervalo [0,%i]\n', n);
while i <= n
    t=round(input(sprintf('x(%i)? ', i+1)));
    if t >= 0 && t <= n
        i=i+1; x(i)=t;
    end
end
disp(x)
```

Semana 12

Traballo en clase

1. **Lectura dende arquivo en formato R con fscanf**. Escribe un programa chamado `le_arquivo_R.m` que lea os datos (numéricos e caracteres) dende `arquivo_R.dat` seguinte, que tamén se pode ler en R coa función `read.table(...)`:

	E1	E2	E3	E4	Saida
1	0.25	0.33	1.23	−0.51	Branco
2	−0.34	1.3E5	0.22	4.3	Negro

```
clear
f=fopen('arquivo_R.dat');
if -1==f
  error('arquivo_R.dat non existe')
end
s=strsplit(fgetl(f));s(1)=[];m=numel(s)-1;n=0;
while ~feof(f)
    fgetl(f);n=n+1;
end
frewind(f);x=zeros(n,m);y=cell(1,n);fgetl(f);
for i=1:n
  fscanf(f,'%i',1);x(i,:)=fscanf(f,'%g',m);y{i}=fscanf(f,'%s',1);
end
fclose(f);
%-------------------------------
fprintf('%6s',s{:})
for i=1:nf
  fprintf('%6g ',x(i,:));fprintf('%6s\n',y{i})
end
```

2. **Resolución de ecuacións non lineares polo Método de Newton. Bucle híbrido definido-indefinido. Derivación simbólica**. O método de Newton resolve unha ecuación non linear da forma $f(x) = 0$ (con f non linear). Para isto, partimos dun punto x_0 e trazamos a recta tanxente á curva $f(x)$ en $x = x_0$. Esta recta tanxente ten a ecuación $y = mx + n$, onde $m = f'(x_0)$ por ser a recta tanxente a $f(x)$ en $x = x_0$. Pola outra banda, o feito de que a curva $f(x)$ e a recta $y = xf'(x_0) + n$ intersecten no punto $(x_0, f(x_0))$ fai que este punto cumpra a ecuación da recta, de modo que temos que $f(x_0) = x_0 f'(x_0) + n \Rightarrow n = f(x_0) - x_0 f'(x_0)$, e a ecuación da recta fica así:

$$y(x) = f(x_0) + f'(x_0)(x - x_0) \tag{7}$$

Para atopar o punto x_1 de corte desta recta co eixo horizontal, abonda con impor a condición $y(x_1) = 0$ e atopamos:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{8}$$

Repetindo o proceso iterativamente, de modo que a partir dun punto x_i calculamos o novo punto x_{i+1}, aproximámonos a unha solución x^* que verifica $f(x^*) = 0$ (se existe), a non ser que para algún valor x_i teñamos $f'(x_i) = 0$. Podes atopar unha ilustración animada do proceso de búsqueda iterativa da solución neste **enlace**. Escribe un programa chamado `newton.m` que lea por teclado a expresión analítica da función $f(x)$, o punto inicial x_0 e o número máximo n de iteracións e calcule os puntos sucesivos x_i empregando a seguinte fórmula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{9}$$

Esta operación iterativa deberá executarse ata que $|x_i - x_{i-1}| < 10^{-5}$. Antes de executar esta expresión, o programa debe avaliar que $f'(x_i) \neq 0$, e en caso contrario debe rematar cun erro. Como é posíbel que o proceso interativo non converxa, o programa debe rematar cando se supere o número máximo n de iteracións permitidas (usa $n = 100$). O programa debe almacenar os valores $(x_i, f(x_i)), i = 1, \ldots$ no ficheiro `newton.dat` (un par en cada liña).

EXEMPLO 1: dada a ecuación $x^3 - x^2 - x + 1 = 0$ ten raíces en $x = 1$ (dobre) e $x = -1$ (simple). Proba con $x_0 = 2$ (para calcular a raíz $x = 1$) e con $x_0 = -3$ (para calcular a raíz $x = -1$).

EXEMPLO 2: probando coa mesma ecuación e $x_0 = 1$ ten que dar derivada nula en x_0.

EXEMPLO 3: a ecuación $xe^{-x} - 0.2 = 0$ ten solución x =2.54264 (proba con $x_0 = 2$).

EXEMPLO 4: a ecuación $x^2 + 1 = 0$ non ten solución, así que o programa esgotará o número máximo de iteracións sen converxer a unha solución.

NOTA: a expresión `expr`, que é de tipo `char`, transfórmase en función anónima `f` usando `str2func()`, para poder chamar a `f()`. A función `diff()` emprégase para derivar `f` a respeito de `x` (variábel simbólica), e o resultado transfórmase tamén en función anónima `df` usando `matlabFunction()`, para poder chamar a `df()`.

Versión usando bucle híbrido `while+and` lóxico:

```
clear;syms x;dif=inf;i=0;n=100;
g=input('f(x)? ','s');xi=input('x0? ');
f=str2func(sprintf('@(x) %s',g));
df=matlabFunction(diff(f,x));
while dif>1e-5 && i<=n
    dfx=df(xi);
    if dfx==0; error('f''(%g)=0: proba con x0 distinto',xi); end
    xi1=xi-f(xi)/dfx;
    fprintf('%i: %g\n',i,xi1)
    dif=abs(xi1-xi);xi=xi1;i=i+1;
end
if i<n
    fprintf('x=%g\n',xi)
else
    fprintf('non hai solucion\n')
end
```

Versión usando bucle híbrido `for+return`:

```
clear;clc;niter=100;tol=1e-5;
if exist('OCTAVE_VERSION','builtin'); pkg load symbolic; end
syms x;expr=input('f(x)? ', 's');xi=input('x0? ');
f=str2func(sprintf('@(x) %s',expr));df=matlabFunction(diff(f,x));
for i=1:n
    dfx=df(xi);
    if abs(dfx)<1e-10; fprintf('f''(%g)=0: usa outro x0\n', xi); end
    xi1 = xi - f(xi)/dfx;
    fprintf('i =%i x =%g\n', i, xi1);
    if abs(xi1-xi)<tol; fprintf('x=%g\n',xi1);return; end
    xi = xi1;
end
fprintf('non hai solucion\n')
```

Traballo a desenvolver pol@ alumn@

1. **Lectura con `fscanf` detectando fin de arquivo**. Escribe un programa chamado `le_arquivo_eof.m` que lea tódolos datos de `arquivo_eof.dat` seguinte:

```
1 2 3 4
5 6 7
8
```

```
clear
f=fopen('arquivo_eof.dat','r');
if -1==f
    error('arquivo_eof.dat non existe')
end
% x=fscanf(f,'%i',inf)';  % le todolos datos a vector x
while ~feof(f)
    x=fscanf(f,'%i',1);  % le dato a dato e mostra por pantalla
    fprintf('%i ',x);
end
fprintf('\n')
fclose(f);
```

2. Escribe unha función en Matlab chamada `maxoumin` que calcule o valor máximo ou mínimo dunha función cuadrática $f(x) = ax^2 + bx + c$. A función en Matlab debe te-la forma `[x y w] = maxoumin(a,b,c)`, onde `x` é o valor de x que maximiza/minimiza $f(x)$, `y` é o valor máximo/mínimo de $f(x)$ e `w` sexa 1 se é un máximo e 2 se é un mínimo.

3. Escribe unha función de Matlab `[a n] = axusta_funcion_potencial(x, y)` que reciba como argumentos dous vectores `x` e `y` e axuste os puntos $\{(x_i, y_i)\}_{i=1}^n$ a unha curva da forma $y = ax^n$. A función debe retorna-los valores a e n. Escribe un programa que lea os seguintes datos dende o arquivo `datos.dat` e realice o axuste chamando á función:

x	0.5	1.9	3.3	4.7	6.1	7.5
y	0.8	10.1	25.7	59.2	105	122

   Fai un gráfico que represente os puntos e a función.

4. **Serie de Fourier dunha función. Cálculo simbólico e numérico de integrais definidas**. Dada unha función $f(x)$, a súa serie de Fourier $F(x)$ está dada por:

$$F(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos nx + b_n \operatorname{sen} nx), \quad a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx dx, \quad b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\operatorname{sen} nx dx \tag{10}$$

   Escribe un programa en Matlab chamado `fourier.m` que lea por teclado a expresión analítica dunha función (proba con x^2 e con x^3) chame á función `coef(...)`, pasándolle os argumentos axeitados, que calcule os coeficientes $a_n, n = 0, \ldots, m$ e $b_n, n = 1, \ldots, m$ para m=20. Calcula as integrais definidas usando os comandos `int` e `eval` de cálculo simbólico. Logo, o programa principal debe calcular o valor da función $f(x)$ e máis da súa serie de Fourier $F(x)$ para 100 valores de x no intervalo $[-\pi, \pi]$, representar gráficamente ambas e mostrar a media dos valores absolutos das diferencias e máis o tempo empregado.

```
clear;tic
s=input('f(x)?  ','s');
f=str2func(sprintf('@(x) %s',s));
m=20;n=100;j=1:m;
[a,b]=coef(s,m);
x=linspace(-pi,pi,n);
y=zeros(1,n);Y=zeros(1,n);
for i=1:n
    z=x(i);y(i)=f(z);u=j*z;
    Y(i)=a(1)+sum(a(2:end).*cos(u)+b.*sin(u));
end
clf;plot(x,y,'b-',x,Y,'r-')
grid;legend({'f(x)','F(x)'})
fprintf('erro=%g  tempo=%g  s\n',mean(abs(y-Y)),toc)
%-------------------------------------------------
function [a,b]=coef(s,m)
a=zeros(1,m+1);b=zeros(1,m);
syms x;f=str2sym(s);
a(1)=eval(int(f,x,-pi,pi))/2;
for n=1:m
    a(n+1)=eval(int(f*cos(n*x),x,-pi,pi));
    b(n)=eval(int(f*sin(n*x),x,-pi,pi));
end
a=a/pi;b=b/pi;
end
```

5. Escribe un programa que resolva unha ecuación non linear da forma $f(x) = 0$, con $f(x)$ non linear, polo método **Regula Falsi**. A partir dun punto inicial a_1 (lido por teclado), o programa debe buscar un punto b_1 con $f(a_1)f(b_1) < 0$, e calcular

$$b_2 = \frac{a_1 f(b_1) - b_1 f(a_1)}{f(b_1) - f(a_1)} \tag{11}$$

   (ver **figura**). Se $f(a_1)f(b_2) < 0$, o programa debe repeti-lo proceso con a_1 e b_2. Se $f(b_2)f(b_1) < 0$, o programa debe repeti-lo proceso con b_2 e b_1. O programa debe rematar cando $|b_2 - b_1| < 10^{-5}$

6. **Método de Newton para resolver sistemas de ecuacións non lineares:** realiza un programa chamado `newton_sistema.m` que permita resolver un sistema de ecuacións non lineares utilizando o método de Newton. O

programa pedirá por teclado as n ecuacións do sistema como cadeas de caracteres, o punto inicial $\mathbf{x}_0$ (vector de dimension n), a tolerancia para o test de converxencia e o número máximo de iteracións. O programa devolve a solución $\mathbf{x}$ do sistema de ecuacións, se se acadou a converxencia. Proba o programa cos seguintes sistemas de ecuacións non lineares:

- Co punto inicial $\mathbf{x}_0 = [1, 1]$, este sistema de ecuacións:

$$\begin{aligned} x^2 + y^2 - 1 &= 0 \\ \text{sen}\left(\frac{\pi x}{2}\right) + y^3 &= 0 \end{aligned}$$

  debe converxer á solución $\mathbf{x}$=[0.47, -0.87].

- Co punto inicial $\mathbf{x}_0 = \left[\frac{1}{2}, 1\right]$, este sistema de ecuacións:

$$\begin{aligned} 1 - x^2 - y^2 &= 0 \\ x^2 - y^2 &= 0 \end{aligned}$$

  debe converxer á solución $\mathbf{x} = \left[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right] = [0.7071, 0.7071]$.

Notas sobre o método de Newton: dado un sistema de ecuacións da forma:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \tag{12}$$

onde $f_1, \ldots, f_n$ son funcións non lineares. Se consideramos $\mathbf{f} = (f_1, \ldots, f_n)^T$ e $\mathbf{x} = (x_1, \ldots, x_n)^T$, o sistema anterior pódese escribir como:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{13}$$

Para aplicar o método de Newton a resolución dun sistema de ecuacións hai que substituir a primeira derivada $f'(x)$ da función escalar $f(x)$ na ecuación do método de Newton uni-dimensional:

$$f(x_i) + f'(x_i)\Delta x_i = 0 \Rightarrow \Delta x_i = -\frac{f(x_i)}{f'(x_i)} \tag{14}$$

(onde $\Delta x_i = x_{i+1} - x_i$, e i é o n^o de iteración) pola *matriz Xacobiana* $\mathbf{J^f}$ da función vectorial $\mathbf{f}$ cuxas compoñentes son $J^{\mathbf{f}}_{ij} = \dfrac{\partial f_i}{\partial x_j} \quad i, j = 1, \ldots, n$. Tendo isto en conta, o método de Newton aplícase do seguinte xeito: dado un valor inicial $\mathbf{x}^0 \in \mathbb{R}^n$ para $i = 0, 1, \ldots$, iteracións ate a converxencia, hai que calcular $\Delta\mathbf{x}_i$ na ecuación:

$$\mathbf{f}(\mathbf{x}_i) + \mathbf{J^f}(\mathbf{x}_i)\Delta\mathbf{x}_i = \mathbf{0} \Rightarrow \Delta\mathbf{x}_i = -[\mathbf{J^f}(\mathbf{x}_i)]^{-1}\mathbf{f}(\mathbf{x}_i) \tag{15}$$

e logo facer $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i$.

SOLUCION: dividimos o programa en varios bloques: programa principal `newton_sistemas.m` e unha función que aplique propiamente dito o método de Newton (función `newtonsys`). A continuación o código de ambos bloques:

- Programa principal `newton_sistema.m`:

```
% Metodo de newton para a solucion dun sistema
%  de ecuacions non lineal do tipo f(x)=0
%  con f un vector de funcions e x un vector.
clear
tic; % para medir o tempo computacional combinado con toc
% sistema de ecuacions
f = {'1-x^2-y^2', 'x^2-y^2'};
n=2; % numero de ecuacions
```

```
% Para simplificar utilizase estas letras para as variables
var={'x' 'y' 'z' 't' 'u' 'v' 'w'};
% Conversion de expresions en funcions
F=cell(1,n);
for i=1:n
    F{i}=inline(char(f{i}), var{1:n});
end
% Calculo da matriz xacobiana
J=cell(n);
for i=1:n
    for j=1:n
        J{i,j}=inline(char(diff(char(f{i}), sym(var{j}))), var{1:n});
    end
end
x0=[1/2 1]; % punto inicial
epsilon=1e-5; % tolerancia
k_max=100; % numero maximo de iteracions
% Chamada o metodo de newton
[x F2 iter]=newtonsys(F, J, x0, epsilon, k_max);
fprintf('A solucion e o punto x: %e \n', x);
fprintf('Tempo transcorrido: %g segundos \n', toc);
```

- Función `newtonsys`: ó codificar esta función dámonos conta de que necesitamos calcular $\mathbf{f}(\mathbf{x}_i)$ e $\mathbf{J}^{\mathbf{f}}(\mathbf{x}_i)$ en cada iteración i ate a converxencia. Como $\mathbf{f}$ e $\mathbf{J}$ son un vector e unha matriz de función de dimensión variable e $\mathbf{x}_i$ tamén é un vector de dimensión variable, o proceso non é directo e codifícase dunhas funcións `evalF` e `evalJ` para calcular, respectivamente, o valor das funcións $\mathbf{f}$ e $\mathbf{J}$ no punto $\mathbf{x}_i$.

```
% newtonsys: metodo de Newton para sistemas non lineares
% Calcula un cero dun sistema non lineal. Os argumentos
%    de saida son o vector cero x dun sistema non lineal F
%    con matriz xacobiana J mais proximo o vector inicial x0
%    para unha tolerancia tol. iter e o numero de iteracions
%    e F o residuo.
function [x Fx iter]=newtonsys(F, J, xo, tol, nmax)
  iter=0; error=tol+1; x=xo;
  while error > tol && iter <= nmax
      Jx=evalJ(J, x);
      Fx=evalF(F, x);
      if det(Jx)==0
          fprintf('Determinante da matriz xacobiana vale 0 \n');
          fprintf('Non se pode aplicar o metodo de Newton.\n');
          break
      end
      delta= -Jx\Fx';
      x = x + delta';
      error=norm(delta); % norma dun vector
      iter=iter +1;
  end
  Fx=norm(evalF(F, x));
  if iter >= nmax
      fprintf('Non converxe no numero maximo de iteracions\n');
      fprintf('O iterante devolto ten o residuo relativo %e\n', Fx);
  else
      fprintf('O metodo converxe na iteracion %i cun residuo %g\n', iter, Fx);
  end
  return
end
```

- Función `evalF`:

```
% evalF: evaluacion do sistema de ecuacions lineais F no punto x
% Datos de saida: vector F(x)
function Fx=evalF(F, x)
```

```
    n=length(x);
    Fx=zeros(1,n);
    var=num2cell(x);
    for i=1:n
        Fx(i)=feval(F{i},var{:});
    end
end
```

- Función `evalJ`:

```
% evalJ: evaluacion da matriz xacobiana J no punto x
% Datos de saida: vector J(x)
function Jx=evalJ(J, x)
  n=length(x);
  Jx=zeros(n);
  var=num2cell(x); % paso de vector de numeros a vector de celdas
  for i=1:n
      for j=1:n
          Jx(i, j)=feval(J{i,j}, var{:});
      end
  end
end
```

Se queremos xeralizar para máis ecuacións e incógnitas, e usar unha interfaz gráfica para introducir os datos de entrada, podemos usar o seguinte programa `newton_sistema2.m` ampliado:

```
% Metodo de newton para a solucion dun
%   sistema de ecuacions non lineal do tipo f(x)=0
%   con f un vector de funcions e x un vector.
clear
tic; % para medir o tempo computacional combinado con toc
% Entrada do sist. de ecs. cunha interface grafica
titulo = 'Metodo de Newton sist. de ecs. non lineares.
  Utiliza as variables x, y, z, t, u, v, w';
n=input('Numero de ecuacions (< 7): ');
prompt=cell(n, 1)
for i=1:n
    prompt{i} = sprintf('Ecuacion %d: ', i);
end
lines = 1;
datos = inputdlg(prompt,titulo,lines);
while length(datos)< n
    datos = inputdlg(prompt,titulo,lines);
end
assignin('base','f',datos);
% Para simplificar utilizase estas letras para as variables
var={'x' 'y' 'z' 't' 'u' 'v' 'w'};
% Conversion de expresions en funcions
F=cell(1,n);
for i=1:n
    F{i}=inline(char(f{i}), var{1:n});
end
J=cell(n);
% Calculo da matriz xacobiana
for i=1:n
    for j=1:n
        J{i,j}=inline(char(diff(char(f{i}), sym(var{j}))), var{1:n});
    end
end

textos={'Valor inicial []: ','Numero maximo de iteracions: ',
   'Tolerancia do test de convergencia: '};
dato = inputdlg(textos,titulo,lines);
```

```
while length(datos)< n
    dato = inputdlg(textos,titulo,lines, def);
end
assignin('base','x0',str2num(char(dato{1})));
assignin('base','k_max',str2num(char(dato{2})));
assignin('base','epsilon',str2num(char(dato{3})));
if length(xo) ~= n
    fprintf('Tamanho do valor inicial ten que coincidir co numero de ecs.\n');
    break;
end

% Chamada o metodo de newton
[x F2 iter]=newtonsys(F, J, x0, epsilon, k_max);
fprintf('A solucion e o punto x: %e \n', x);
fprintf('Tempo transcorrido: % g segundos\n', toc);
```

7. Descarga o seguinte arquivo de texto cos datos dos elementos químicos da táboa periódica ordeados por número atómico (arquivo **taboaperiodica.txt**). As columnas deste arquivo teñen os seguintes significados: 1) abreviatura do elemento, 2) nome do elemento e 3) peso atómico en gramos por mol. Escribe un programa en Matlab que pida repetidamente por teclado a abreviatura dun elemento químico e visualice na pantalla o seu nome completo e peso atómico (ten que atopar a información no arquivo anterior). O programa ten que rematar cando se introduza por teclado unha X. Usa a función `upper()`, que convirte unha cadea de caracteres en maiúsculas.

SOLUCIÓN:

```
% Escribe un programa que lea do arquivo taboaperiodica.txt
% que conten a seguinte informacion:
% columna 1 a abreviatura dos elementos quimicos
% columna 2 o nome dos elementos
% columna 3 o peso atomico
% O programa pide o usuario a abreviatura de elementos quimicos
% e visualiza o nome do elemento e o seu peso atomico
% mentres que o usuario non introduce unha x
clear all
fid=fopen('taboaperiodica.txt', 'r');
if -1 == fid
  error('Erro abrindo taboaperiodica.txt');
end
taboa=textscan(fid, '% s % s % f');
% funcion upper() convirte a maiusculas
abrev=upper(taboa{1}); % primeira columna (vector de celdas)
elementos=taboa{2}; % segunda columna
pa=taboa{3}; % terceira columna (vector de numeros)

el='' % cadea baleira

while ~strcmp(el, 'X')
  el=upper(input('Introduce elemento: ', 's'));
  pos=strmatch(el, abrev, 'exact');
  if length(pos) > 0
      fprintf('Elemento: % s con peso atomico %f\n', elementos{pos(1)}, pa(pos(1)));
  else
      disp('Elemento non existe');
  end
end
```

8. Escribe un programa que lea por teclado unha cadea de caracteres coa fórmula química dun composto e calcule o peso molecular do mesmo. Para simplificar:

   - Usa como abreviaturas dos elementos químicos os nomes que figuran no arquivo **taboaperiodica.txt** anterior.

- Despois de cada elemento químico, debe haber un díxito especificando o número de átomos dese elemento na molécula de composto (ou de moles do elemento nun mol de composto). Por exemplo: C102 (para CO_2), H2O1 (para H_2O), C1O3Ca1 (para CO_3Ca), etc.
- O peso atómico de cada elemento químico obterase do arquivo **taboaperiodica.txt** do exercicio anterior.

SOLUCIÓN:

```
% Escribe un programa que lea do arquivo taboaperiodica.txt
% que conten a seguinte informacion:
% columna 1 a abreviatura dos elementos quimicos
% columna 2 o nome dos elementos
% columna 3 o peso atomico
% O programa pide o usuario a formula dun composto quimico
% e visualiza  o seu peso molecular
% a formula ten que ser elementoNumero como por exemplo Cl1Na1, H2O1
% mentres que o usuario non introduce unha x
clear all
fid=fopen('taboaperiodica.txt', 'r');
if -1 == fid
  error('Erro abrindo taboaperiodica.txt');
end
taboa=textscan(fid, '%s %s %f');
% funcion upper() convirte a maiusculas
abrev=upper(taboa{1}); % primeira columna (vector de celdas)
elementos=taboa{2}; % segunda columna
pa=taboa{3}; % terceira columna (vector de numeros)

el='' % cadea baleira

while ~strcmp(el, 'X')
  el=upper(input('Introduce molecula: ', 's'));
  d=isletter(el);  % vector con 1 na posicion de letra e 0 na posicion de numero
  pm=0; % peso molecular
  inicio=1;
  n=length(d);
  while inicio < n
      for j=inicio:n
          if d(j) == 0 % chegase a un numero
              break;
          end
      end
      elem=el(inicio:j-1);
      pos=strmatch(elem, abrev, 'exact'); % posicion elemento na taboa periorica
      inicio=j
      % calcula o numero de elementos
      for j=inicio:n
          if d(j)== 1 % chegase a unha letra
              fin=j-1;
              break;
          else
              fin=j;
          end
      end
      nel=str2num(el(inicio:fin))
      inicio=j;
      pm = pm + nel*pa(pos);
  end
end
```