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_octave.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 (arquivo **progreso_matlab.m**):

```
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 = numel(x); 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;
pkg load symbolic  % so para octave
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 (lido por teclado), o programa debe buscar un punto b no cal $f(a)f(b) < 0$ e $f(a) \neq f(b)$, e calcular

$$x = \frac{af(b) - bf(a)}{f(b) - f(a)} \tag{11}$$

(ver **figura**). Notemos que $a < x < b$. Se $f(a)f(x) < 0$, o programa debe repeti-lo proceso con a e x. Se $f(x)f(b) < 0$, o programa debe repeti-lo proceso con x e b. O programa debe rematar cando $|b - x| < 10^{-5}$.

```
clear all
s=input('f(x)? ','s');   % usa f(x)=x*exp(-x)-0.2
```

```
a=input('a? ');  % usa a=0
f=str2func(sprintf('@(x) %s',s));
h=0.1;b=a+h;i=0;n=10;
while f(a)*f(b)>0 && f(b)~=f(a)
    b=b+h;
end
printf('%10s %10s %10s %10s\n','iteracion','a','x','b')
while i<n
    i=i+1;x=(a*f(b)-b*f(a))/(f(b)-f(a));
    printf('%10i %10f %10f %10f\n',i,a,x,b)
    if abs(b-x)<1e-5
        printf('converxeu a x=%f en %i iteracions\n',x,i)
        return
    end
    if f(a)*f(x)<0
        b=x;
    end
    if f(x)*f(b)<0
        a=x;
    end
end
printf('non converxeu en %i iteracions\n',n)
```

6. **Método do punto fixo** para a resolución de ecuacións non lineares. 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:

$$x_{i+1} = g(x_i), \quad i = 0, \ldots \tag{12}$$

   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 x_0=0 e ε =0.001. O programa debe executar a operación 12 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.

```
clear all
pkg load symbolic
syms x
f=input('f(x)? ','s');    % usa f(x)=x+x*exp(x)+1
x0=input('x0? ');n=100;   % usa x0=0
g=str2func(sprintf('@(x) x-(%s)',f));
dg=matlabFunction(diff(g,x));flag=0;
for i=1:n
    x1=g(x0);
    if abs(x1-x0)<1e-5
        flag=1;break
    end
    printf('%10i %10g\n',i,x1)
    x0=x1;
end
if flag
    printf('x=%i , %i iteracions\n',x1,i)
else
    printf('non converxeu\n')
end
```

7. **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:

$$x^2 + y^2 - 1 = 0$$
$$\text{sen}\left(\frac{\pi x}{2}\right) + y^3 = 0$$

  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:

$$1 - x^2 - y^2 = 0$$
$$x^2 - y^2 = 0$$

  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:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \tag{13}$$

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:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{14}$$

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:

$$f(x_i) + f'(x_i)\Delta x_i = 0 \Rightarrow \Delta x_i = -\frac{f(x_i)}{f'(x_i)} \tag{15}$$

(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:

$$\mathbf{f}(\mathbf{x}_i) + \mathbf{J^f}(\mathbf{x}_i)\Delta\mathbf{x}_i = \mathbf{0} \Rightarrow \Delta\mathbf{x}_i = -[\mathbf{J^f}(\mathbf{x}_i)]^{-1}\mathbf{f}(\mathbf{x}_i) \tag{16}$$

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);
```

8. 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
```

9. 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
```