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 $(\operatorname{sen} x - 3)/(\ln x + e^x - 1)$. 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
   ```

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. 1º 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.

```
        if(0 == c) then
            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
```

```fortran
call swgcbk(boton, calcula) ! conecta a pulsacion do boton
                            ! coa execucion dun 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
```

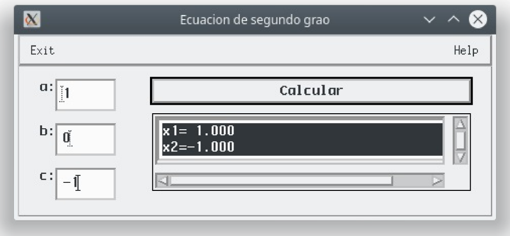

Figura 1: Programa en Fortran para resolver unha ecuación de 2^o grao usando unha interface gráfica creada coa libraría Dislin.

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 $T_9(2) = 70225{,}9531$.

```fortran
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 móbil 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

! v=[1,2,1]  ! inicializacion no programa (non necesita allocate)

print *,'v=',v  ! formato por defecto

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

! inicializacion no programa (non necesita allocate)
! a=reshape([1,2,3,4,5,6,7,8,9],shape(a))

print *,'a='  ! imprime matriz con formato por defecto
do i=1,n
    print *,a(i,:)
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 $\mathbf{p}$:

```
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 $i, j = 1, \ldots, n$.

```fortran
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-1
!    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-1
   m = m + sum(a(i,i+1:n))
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
```

A función externa `mcm` pódese vectorizar usando a función `any()` para ver se un factor de `y` é común e a función `findloc(vector,valor,1)` para obter o índice deste factor en `x`, aforrando así o bucle en `j`:

```
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)
```

```
    if(any(bx==k)) then
        j=findloc(bx,k,1)
        if(l>ex(j)) mcm=mcm*k**(l-ex(j))
    else
        mcm=mcm*k**l
    end if
end do
end function mcm
```

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 progreso2
real(8) :: t0,t1,dt,i=1,n=50000000.  !50000000.
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 progreso2
!----------------------------------
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-86400*d-3600*h-60*m)
     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)/3600)
     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*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)
    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.

$$\mathbf{v}'\mathbf{w} = \begin{bmatrix} v_1 \\ \ldots \\ v_n \end{bmatrix} [w_1 \ldots w_n] = \begin{bmatrix} v_1 w_1 & \ldots & v_1 w_n \\ \ldots & \ldots & \ldots \\ v_n w_1 & \ldots & v_n w_n \end{bmatrix}$$

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
```

```fortran
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 (o símbolo $ significa que tes que teclear na terminal):

   ```
   $ 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 (o símbolo ≫ significa que tes que teclealo no octave):

   ```
   >> 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 $y = f(x) = \dfrac{x^2 + x - 6}{x^2 - 4}$.

   ```fortran
   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-1);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 $\lim_{x \to 2} \dfrac{x^2 + x - 6}{x^2 - 4} = 1.25$.

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**.

   ```fortran
   program grafica3D
   use gnufor2
   integer,parameter :: n=100
   real(8) :: x(n),y(n),z(n,n)
   ```

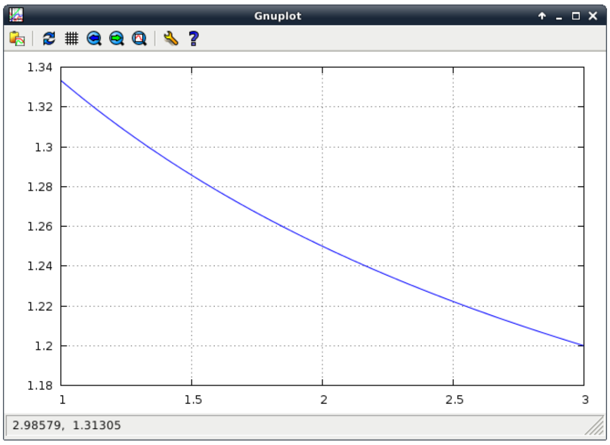

Figura 2: Representación gráfica dunha función dunha variábel usando gnufor2.

```
f(x,y)=sin(5*(x**2+y**2))*exp(-(x**2+y**2)/4)
a=-2;b=2;h=(b-a)/(n-1);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.

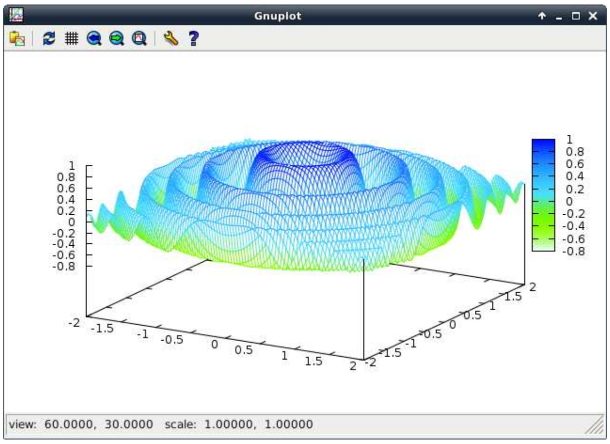

Figura 3: Representación gráfica da función $f(x,y) = \operatorname{sen} 5(x^2+y^2)\exp\left(-\dfrac{x^2+y^2}{4}\right)$ usando gnufor2.

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 $f(x)$.

```
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
```

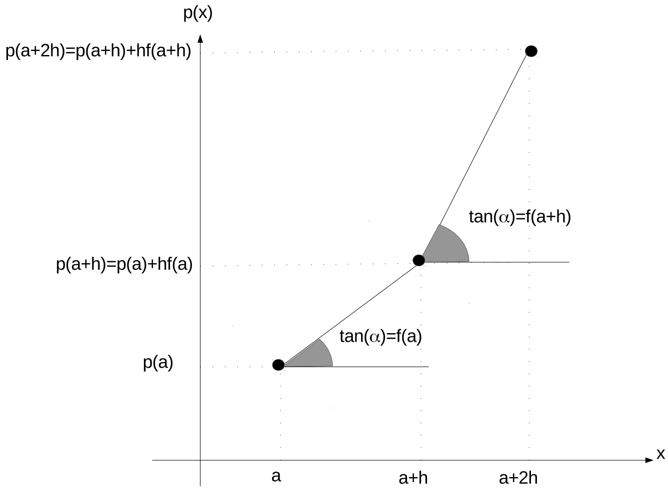

Figura 4: Aproximación numérica á primitiva $p(x)$ dunha función $f(x)$.

```
    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 $[a, b]$. Proba 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 $\mathbf{f} = (f_1, \dots, f_n)$, onde $f_i = f(x_i)$ con $x_i = a + h(i-1)$ con $i = 1 \dots n$ e $h = \dfrac{b-a}{n-1}$. 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 $\mathbf{d} = (d_1, \dots, d_{n-1})$, onde $d_i = \dfrac{f_{i+1} - f_i}{h}$, con $i = 1 \dots n-1$. É decir, a derivada calcúlase como a diferencia de dous valores consecutivos de f.
- A súa primitiva $p(x) = \int f(x)dx$ pode describirse polo vector $\mathbf{p} = (p_1, \dots, p_n)$, onde $p_1 = p(a)$ (prefixado por nós arbitrariamente, p.ex. $p(a) = 0$) e $p_{i+1} = p_i + hf_i$ con $i = 1 \dots n-1$. É decir, a primitiva p_i é a suma acumulativa dos valores f_i dende 1 ata i multiplicada por h.
- A súa integral definida $\int_a^b f(x)dx$ pode aproximarse pola suma $h \displaystyle\sum_{i=1}^{n} f_i$. É decir, é a suma de tódolos valores de f no intervalo $[a,b]$ multiplicada por h.

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 $x(t) = e^{-t/10} \operatorname{sen} 2t, y(t) = t^2, z(t) = \operatorname{sen} 3t$, con $t = 1 \dots 10$. Usa a subrutina `plot3d` da libraría `gnufor2`.
3. Escribe un programa que calcule límite $\lim_{x \to 0} e^{|x|/x}$.
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.

   Arquivo `determinante_orde3.dat`:

   ```
   3
   0 2 3
   4 5 6
   7 8 9
   ```

   Arquivo `determinante_orde4.dat`:

   ```
   4
   1 0 2 -1
   1 1 1 1
   3 2 0 1
   5 3 1 0
   ```

   Programa `determinante.f90`:

   ```fortran
   program determinante
   integer,allocatable :: a(:,:)
   integer :: det
   character(100) :: nf='determinante_orde3.dat'
   open(1,file=nf,status='old',err=1)
   read (1,*) n
   allocate(a(n,n))
   do i=1,n
       read (1,*) a(i,:)
   end do
   print *,'a:'
   call imprime(a,n)
   close(1)
   m=det(a,n)
   print '("det(a)=",i0)',m
   deallocate(a)
   stop
   1 print *,'erro open ',nf
   end program determinante
   !-----------------------------------------
   recursive integer function det(a,n) result(d)
   integer,intent(in) :: a(n,n),n
   integer,allocatable :: b(:,:)
   select case(n)
   case(1)
       d=a(1,1)
   case(2)
       d=a(1,1)*a(2,2)-a(1,2)*a(2,1)
   case default
       d=0;k=1;m=n-1;z=[(i,i=1,n)]
       do i=1,n
           b=a(2:n,[(j,j=1,i-1),(j,j=i+1,n)])
           print *,'--'; call imprime(b,m)
           d=d+k*a(1,i)*det(b,m);k=-k
       end do
   ```

```
end select
end function det
!-----------------------------------------------
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
end subroutine imprime
```

Versión pasando matrices e as súas dimensións, cunha interface para a función `le_matriz(...)`. Neste e nos seguintes casos, o número na primeira liña dos arquivos `determinante_orde3.dat` e `determinante_orde4.dat` deben ser borrados.

```
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)
do j=2,n
    do k=1,n
        if(k<i) then
            b(j-1,k)=a(j,k)
        else if(k>i) then
```

```
            b(j-1,k-1)=a(j,k)
        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 con pack:

```
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)
```

```fortran
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
  m=j-1;l=1
  do k=1,n
    if(k/=i) then
      b(m,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:

```fortran
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 compilar esta última versión do módulo, 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 =
```

```fortran
!    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 filas
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) :: 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)
```

```fortran
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.

```fortran
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 <u>enlace</u>. 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 <u>enlace</u>. 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
```

```fortran
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=.
   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
```

```fortran
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) $\mathrm{e}^{-x^2} = 0$ 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 x_0=0 e 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
```