Programación estruturada en Fortran
Exercicios clases interactivas

Semana 1

Instruccións para instalar o compilador Gfortran e o entorno Visual Studio Code no teu ordenador

Descarga o compilador **Gfortran** para Windows dende:

https://www.equation.com/ftpdir/gcc/gcc-13.2.0-32.exe

Executa o instalador e instala Gfortran no teu ordenador. Cando remate, reinicia o ordenador.
Para instalar o Gfortran nun ordenador Mac, instala **Homebrew** (https://brew.sh/gl) nunha terminal co comando:

```
/bin/bash -c ‘‘$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)’’
```

e logo instala o paquete **gcc** co comando: `brew install gcc@14`.
Descarga o entorno **Visual Studio Code** (VSCode) no teu ordenador dende este enlace (pulsa no botón **Windows**):

https://code.visualstudio.com/download (Pulsa no botón Windows(Windows 10,11))

Instala o VSCode executando o instalador. Executa o VSCode buscando “Visual Studio Code” no menú de inicio de Windows. Podes engadir a aplicación á barra de tarefas de Windows pulsando no botón dereito do rato no menú onde aparece VSCode e seleccionando “Anclar á barra de tarefas”. Así, poderás executalo a seguinte vez pulsando no seu botón da barra de tarefas no canto de abrir o menú de inicio. Vai á barra da esquerda, pulsa no botón **Extensións**, situado abaixo nesa barra, e busca e instala as extensións “**Modern Fortran**” e “**cppdebug**”.
Tes instruccións máis detalladas neste enlace:

https://github.com/fran-pena/met-num-for/blob/main/vscode_gfortran_windows/instalacion.md

Traballo en clase

Executa o VSCode indo ao menú de inicio do Windows, buscando “Visual” e pulsando no seu icono. Xa no VSCode, vai á barra da esquerda, pulsa no botón **Extensións**, situado abaixo nesa barra, e busca e instala as extensións “**Modern Fortran**” e “**cppdebug**”.
No VSCode, vai ao menú **File→Open Folder** e sitúate no **Escritorio**. Alí, crea unha carpeta chamada **fortran**, na cal almacenaremos tódolos programas.

1. **Variábeis. Expresións aritméticas. Entrada/saída básicas**. Crea un programa no VSCode no directorio **fortran** anterior dende o menú **File→New File...** do VSCode, ou pulsando en **New File...** na pestana **Welcome**, e chámalle `expresions.f90`. O programa debe ler un número real x por teclado e mostrar por pantalla $3x-1$, $x^2+\sqrt{x-2}$ e $(\operatorname{sen} x-3)/(\ln x+\mathrm{e}^x-1)$. Gárdao no directorio **fortran**. Para compilar o programa, vai ao menú **Terminal→New Terminal**. Ábrese así unha terminal na ventá de VSCode. Nela, executa o comando:

   ```
   gfortran expresions.f90
   ```

   Deste modo xérase o programa executable `a.exe`. Para executalo, na terminal teclea **a.exe** (isto require configurar o Path como se indicou antes) ou `.\a.exe`.

   ```
   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:

```
gfortran expresions.f90 -o expresions
```

Executa o programa na terminal co comando:

```
.\a.exe
```

Tamén o podes executar co comando:

```
a.exe
```

Para poder facelo desta segunda forma, debes configurar Windows de modo que podas executar programas que se atopan na carpeta actual. Vai ao menú de inicio, pulsa no botón **Configuración** e na ventá de configuración busca "**Editar variables de ambiente desta conta**". Selecciona **Path**, pulsa no botón **Editar** e engade o directorio **.** que representa ao directorio actual. Pulsa no botón **Aceptar**. Pecha o VSCode e vólveo a executar, para que a modificación do **Path** teña efecto. Logo, na terminal executa `a.exe`.

Unha vez executado o programa, introduce por teclado o valor 1 para a variábel x. Verás que a segunda expresión da **NaN**, que significa **Not a Number** (**https://es.wikipedia.org/wiki/NaN**), porque calcula a raíz cadrada dun número negativo. Logo execútao novamente usando o valor 2. Copia o programa `expresions.f90` á memoria flash, unidade **D:**

Para evitar ter que gardar o programa cada vez que o modifiques (Ctrl+S ou menú **File→Save**), pódese activar no VSCode a opción de autogardado (**Auto Save**). Para isto, vai ao menú **File→Auto Save**.

2. **Ecuación de 2^o grao. Sentenzas de selección**. Escribe un programa chamado `ecuacion.f90` que lea os coeficientes (reais) dunha ecuación de segundo grao $ax^2 + bx + c = 0$ e calcule as súas solucións segundo a fórmula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

Se $a = 0$, temos unha ecuación de primeiro grao. Neste caso, se $b = 0$ a ecuación redúcese a $c = 0$, sendo independente de x. Se o valor c introducido por teclado é $c = 0$, entón a ecuación cúmprese para todo $x \in \mathbb{R}$. Se, polo contrario, o c lido por teclado é $c \neq 0$, entón a ecuación non se cumpre para ningún x, de modo que a solución é o conxunto baleiro. Se $b \neq 0$, entón pódese resolver a ecuación de 1^o grao e $x = -c/b$.

Se $a \neq 0$, temos unha ecuación de segundo grao. Sexa $d = b^2 - 4ac$ o discriminante. Se $d = 0$, entón hai dúas solucións reais iguais $x = \frac{-b}{2a}$. 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 $i = \sqrt{-1}$. Finalmente, se $d > 0$ hai dúas solucións reais distintas $x = \frac{-b \pm \sqrt{d}}{2a}$.

Para comprobar que o programa funciona correctamente, proba cos exemplos da táboa 1:

a	b	c	N^o solucións	Solucións
0	0	0	∞	$x = \mathbb{R}$
0	0	1	0	$x = \varnothing$
0	1	-1	ec. 1^o grao	$x = 1$
1	1	1	2 complexas	$x = -0.5 \pm 0.866$i
1	0	-1	2 reais distintas	$x = \pm 1$
1	-2	1	2 reais iguais	$x = 1$

Cuadro 1: Valores dos coeficientes a, b, c e das solucións que debe obter o programa.

```
program ecuacion
print '("a,b,c? ",$)';read *, a,b,c
if(a==0) then
    if(b==0) then
        if(c==0) then
            print *, 'existen infinitas solucions reais'
        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(d==0) then
        print *,'2 solucions reais iguais: x=',u
    else
        print *,'2 solucions reais: x=',u+v,u-v
    endif
endif
end program ecuacion
```

3. **Sentenza de iteración definida. Acumulador. Constantes con nome**. Escribe un programa en Fortran chamado `multiplicatorio.f90` que lea por teclado un número enteiro positivo n e un valor real x e calcule o seguinte multiplicatorio:

$$2^{n-1}\prod_{k=1}^{n}\left[x-\cos\left(\frac{(2k-1)\pi}{2n}\right)\right] \tag{2}$$

Executa o programa usando $n=9$ e $x=2$. O programa debe mostrar 70225.9531 na terminal.

```
program multiplicatorio
real,parameter :: pi=3.141592
print '("n,x? ",$)'
read *,n,x
p=2**(n-1);t=pi/(2*n)
do k=1,n
    p=p*(x-cos((2*k-1)*t))
end do
print *,"p=",p
end program multiplicatorio
```

Exercicios propostos

1. Escribe un programa `plano.f90` 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, $|\mathbf{w}|=\sqrt{a^2+b^2+c^2}$. Usa a función `sqrt()` para calcular a raíz cadrada. O programa debe ler por teclado os valores a,b,c,d,x_0,y_0,z_0. Usa $a=b=c=x_0=y_0=z_0=1$ e debes obter d=2.0394.

```
program plano
print '("a,b,c,d? ",$)'
read *,a,b,c,d
print '("x0=(x,y,z)? ",$)'
read *,x,y,z
print *,'distancia=',abs(a*x+b*y+d*z+d)/sqrt(a**2+b**2+c**2)
end program plano
```

2. Escribe un programa `mediaprod.f90` que lea n números $x_1,\ldots,x_n$ por teclado e calcule a súa media $\frac{1}{n}\sum_{i=1}^{n}x_i$ e o seu producto $\prod_{i=1}^{n}x_i$ sen usar arrais. Usa n=5 e os números 1,2,3,4,5.

```
program mediaprod
integer :: media=0,prod=1
print '("n? ",$)'
read *,n
do i=1,n
    print '(i0," x? ",$)',i
```

```
    read *,x
    media=media+x;prod=prod*x
end do
print '("media=",i0," prod=",i0)',media,prod
end program mediaprod
```

3. Escribe un programa `armonico.f90` que calcule a posición $x(t)$, velocidade $v(t)$ e aceleración $a(t)$ dun móvil en movemento armónico, dado por:

$$x(t) = b\,\mathrm{sen}(\omega t + \theta) \tag{4}$$
$$v(t) = b\omega\cos(\omega t + \theta) \tag{5}$$
$$a(t) = -b\omega^2\,\mathrm{sen}(\omega t + \theta) \tag{6}$$

sendo ω= 0.1 radiáns/s, $\theta = \pi/2$ radiáns, $b = 2.5\ m$ para tempos $0 \leq t \leq 100$ segs. separados 1 seg. entre si. Representa gráficamente x, v, a usando Octave.

```
program armonico
real,parameter :: pi=3.141592
n=100;w=0.1;theta=pi/2;b=2.5
open(1,file='armonico.txt',status='new')
do i=1,n
    x=b*sin(w*i+theta)
    v=b*w*cos(w*i+theta)
    a=-b*w**2*sin(w*i+theta)
    print *,x,v,a
    write (1,*) x,v,a
end do
close(1)
end program armonico
! en Octave:
! datos=load('armonico.txt');
! figure(1);plot(x)
! figure(2);plot(v)
! figure(3);plot(a)
```

Semana 2

Traballo en clase

1. **Números de Fibonacci**. Escribe un programa chamado `fibonacci.f90` que lea por teclado un número enteiro n maior que 1, verificando que é maior que 1, e calcule os números de Fibonacci F_i para $i = 0, \ldots, n$. Estes números están definidos como:

$$F_0 = 0, \quad F_1 = 1, \quad F_i = F_{i-1} + F_{i-2}, \text{ para } i \geq 2 \tag{7}$$

   Podes atopar máis información sobre estes números neste enlace:

   https://es.wikipedia.org/wiki/Sucesión_de_Fibonacci

   Executa o programa para n=9, e debes obter 0, 1, 1, 2, 3, 5, 8, 13, 21 e 34.

```
program fibonacci
integer :: f0,f1,f2
do  ! non sae do bucle mentres n<=1
    print '("n(>1)? ",$)'
    read *,n
    if(n>1) exit
end do
f0=0;f1=1
print '(a5," ",a10)','i','F(i)'
print '(i5," ",i10)',0,f0
print '(i5," ",i10)',1,f1
do i=2,n
    f2=f0+f1
    print '(i5," ",i10)',i,f2
    f0=f1;f1=f2
end do
end program fibonacci
```

2. **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 $\mathbf{w}$ con valores enteiros, 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
integer,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
```

3. **Cálculo iterativo de media e desviación típica**. Escribe un programa `media.f90` que defina o vector estático x=`[8 3 1 2 7 9 4]` con n=7 elementos e calcule iterativamente a súa media m e desviación típica d, definidas por:

$$m = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad d = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - m)^2} \tag{9}$$

Logo, volve a calcular m e d usando o **método de Welford**, definido polas seguintes fórmulas, onde m_i e d_i son a media e varianza de $x_1, \ldots, x_i$ para $i = 0, \ldots, n-1$, sendo $m_0 = 0$ e $w_0 = 0$:

$$m_{i+1} = m_i + \frac{x_{i+1} - m_i}{i+1}, \quad w_{i+1} = w_i + (x_{i+1} - m_i)(x_{i+1} - m_{i+1}), \quad i = 0, \ldots, n-1 \tag{10}$$

A desviación típica d dos números $x_1, \ldots, x_n$ é:

$$d = \sqrt{\frac{w_n}{n-1}} \tag{11}$$

Debes obter media=4.86 e desviación=3.13 con ambos métodos. Podes atopar máis información neste enlace:

https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Welford's_online_algorithm

```
program media
integer,parameter :: n=7
integer :: x(n)=[8,3,1,2,7,9,4]
real :: m,m0
! definicions-------------------------
m=sum(x)/real(n);d=sqrt(sum((x-m)**2)/(n-1))
print '("definicion: m=",f6.2," d=",f6.2)',m,d
! metodo de Welford-------------------
m=0;w=0
do i=0,n-1
    j=i+1;k=x(j);m0=m
    m=m+(k-m)/j
    w=w+(k-m0)*(k-m)
end do
d=sqrt(w/(n-1))
print '("Welford: m=",f6.2," d=",f6.2)',m,d
end program media
```

Exercicios propostos

1. Escribe un programa `escalar.f90` 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 producto escalar (ou interior) $\mathbf{v}^T\mathbf{w} = \sum_{i=1}^{n} v_i w_i$ de ambos. Usa n=5, $\mathbf{v}$=[1,2,3,4,5] e $\mathbf{w}$=[5,4,3,2,1]:

```
program escalar
integer,allocatable :: v(:),w(:)
integer :: p=0
print '("n? ",$)'
read *,n
allocate(v(n),w(n))
print '("v[]? ",$)'
read *,v ! introducir 1 2 3 4 5 na mesma linha
print '("w[]? ",$)'
read *,w
do i=1,n
    p=p+v(i)*w(i)
end do
print '("v*w=",i0)',p
! print '("v*w=",i0)',dot_product(v,w)
deallocate(v,w)
end program escalar
```

2. Escribe un programa `norma.f90` que lea un vector estático $\mathbf{v} = (v_1, \ldots, v_5)$ con 5 valores reais. Usa bf v=[1,2,3,4,5]. O programa debe calcula-la norma ou módulo $|\mathbf{v}| = \sqrt{\sum_{i=1}^{n} v_i^2}$ de $\mathbf{v}$. Usa a función **sqrt()** para calcular a raíz cadrada.

```
program norma
integer,parameter :: n=5
real :: v(n)
print '("v[]? ",$)'
read *,v ! usa 1 2 3 4 5
x=0
do i=1,n
    x=x+v(i)**2
end do
print '("norma=",f6.3)',sqrt(x)
end program norma
```

Semana 3

Traballo en clase

1. **Producto 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 producto

$$p = \mathbf{v}^T\mathbf{A}\mathbf{w} = [v_1 \dots v_n] \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix}$$

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,:)
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(:,:),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,:)
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,:)
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
```

2. **Búsqueda dos elementos comúns a dous vectores. Adición de elementos a un vector**. Escribe un programa `comun.f90` que defina dous vectores **x**=[1,8,2,9,0,3] e **y**=[5,1,8,2,7] e mostre por pantalla os elementos comúns a ambos.

```
program comun
integer :: x(6)=[1,8,2,9,0,3],y(5)=[5,1,8,2,7]
integer,allocatable :: z(:)
allocate(z(0))
do i=1,6
    if(any(x(i)==y)) z=[z,x(i)]
end do
print *,'elementos comuns=',z
end program comun
```

3. **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 **A** cadrada de orde n e calcule:

   - A súa traza (suma dos elementos da diagonal principal), definida pola ecuación:

$$tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} \tag{12}$$

   - A suma dos elementos do seu triángulo superior (sen a diagonal).
   - Determine se a matriz é simétrica, é decir, se $a_{ij} = a_{ji}$ para $i, j = 1, \ldots, n$.

```
program matriz
integer,allocatable :: a(:,:)
logical :: simetrica

print '("n? ",$)';read *, n
allocate(a(n,n))
print *,"a[;]?"
do i=1,n
    read *,a(i,:)
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 vectorizado con funcion sum()
m=0
do i=1,n-1
   m=m+sum(a(i,i+1:))
end do
print *,"sts=",s

! E a matriz simetrica
simetrica=.true.
filas: do i=1,n
    do j=i+1,n
        if(a(i,j)/=a(j,i)) then
            simetrica=.false.
            exit filas
        end if
    end do
end do filas
if(simetrica) then
    print *,'simetrica'
else
    print *,'non simetrica'
end if
```

```
! mellor usar 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
```

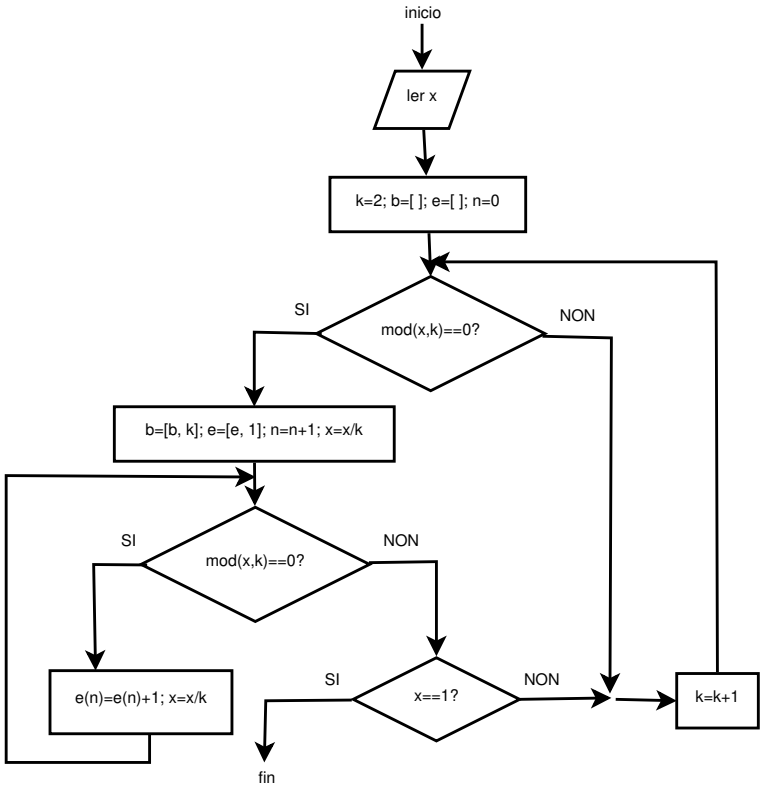

Figura 1: Diagrama de fluxo da factorización dun número enteiro.

4. **Máximo común divisor e mínimo común múltiplo de dous números enteiros usando o algoritmo de Euclides**. Escribe un programa en Fortran chamado `euclides.f90` que lea por teclado dous números enteiros n e m distintos (se $n = m$ entón mcd=mcm=n), e mostre por pantalla o seu máximo común divisor (mcd) e mínimo común múltiplo (mcm). Para calcular o mcd, usa o método de Euclides, descrito neste enlace:

   https://es.wikipedia.org/wiki/Algoritmo_de_Euclides

   Supoñemos que $n > m$. Se n fose menor que m, intercambia n e m. O método de Euclides baséase en: 1) que o mcd de n e m é o mesmo que o mcd(m,k) sendo k o resto de dividir n entre m; e 2) que o mcd(n,0)=n. Deste modo, o proceso repetirase e en cada paso substituimos n por k, ata que k sexa 0, que rematas dando como mcd o valor de n. Dado que o producto do mcd e mcm é o producto $n \cdot m$, coñecido o mcd o mcm pódese calcular como $n \cdot m$/mcd. Proba con n=252 e m=120, debes obter mcm=2520 e o mcd=12.

```
program euclides
print '("n,m? ",$)'
read *,n,m
if(n<m) then
     aux=n;n=m;m=aux
end if
i=n*m
do
     k=mod(n,m)
     n=m
     if(k==0) exit
     m=k
```

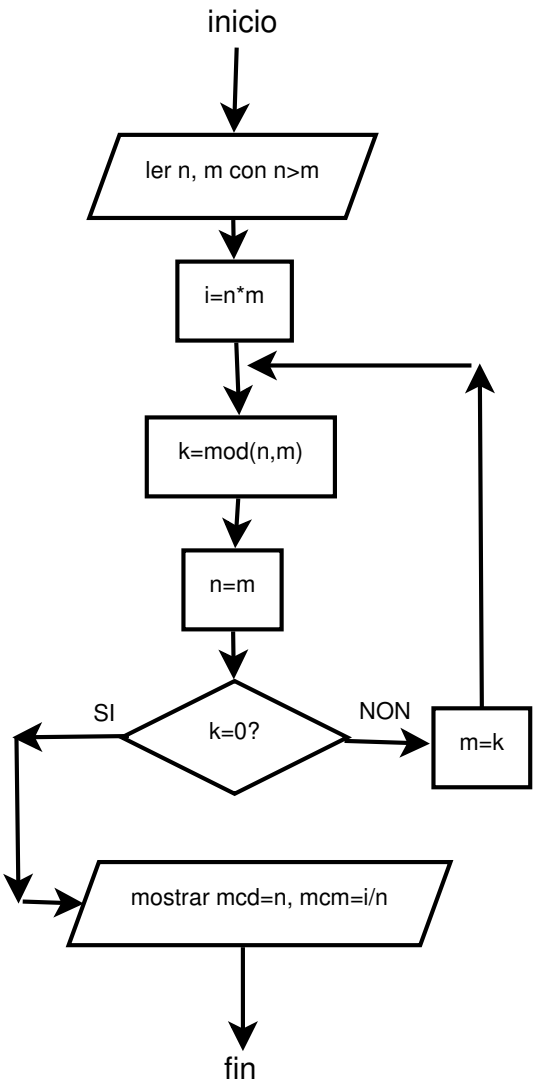

Figura 2: Pseudocódigo do cálculo de mcd e mcm usando o algoritmo de Euclides.

```
end do
print '("mcd=",i0," mcm=",i0)',n,i/n
end program euclides
```

Exercicios propostos

1. Escribe un programa `ocurrencia.f90` que lea por teclado un número n e un vector **v** enteiro de lonxitude n. Usa n=10 e **v**=[1,2,1,4,5,0,1,9,1,7]. O programa debe ler outro número enteiro m e mostrar por pantalla os índices das ocurrencias de m en **v**.

```
program ocurrencia
integer,allocatable :: v(:)
print '("n? ",$)'
read *,n ! n=10
allocate(v(n))
print '("v[]? ",$)'
read *,v ! usa v=1 2 1 4 5 0 1 9 1 7
print '("m? ",$)'
read *,m
print '("Ocurrencias de ",i0,": ",$)',m
k=0
do i=1,n
    if(v(i)==m) then
        print '(i0," ",$)',i
        k=k+1
```

```
    end if
end do
print '(": ",i0," ocurrencias")',k
deallocate(v)
end program ocurrencia
```

2. Escribe un programa `matvec.f90` que lea por teclado un número enteiro n, un vector **v** e unha matriz **A**, ambos de orde n. O programa debe calcula-lo resultado do producto matricial **Av** (sendo **v** un vector columna). Usa n=5, **v**=[1 2 3 4 5] e **a**=[1 2 3 4 5;6 7 8 9 8;7 6 5 4 3;2 1 2 3 4;5 6 7 8 9].

```
program matvec
integer,allocatable :: v(:),a(:,:),p(:)
print '("n? ",$)'
read *,n
allocate(v(n),a(n,n),p(n))
print '("v[]? ",$)'
read *,v
print *,'a[;]? '
do i=1,n
    read *,a(i,:)
end do
do i=1,n
    j=0
    do k=1,n
        j=j+a(i,k)*v(k)
    end do
    p(i)=j
end do
print *,'p=Av=',p
deallocate(v,a,p)
end program matvec
```

3. Escribe un programa `matricial.f90` que lea por teclado catro números enteiros n, m, p e q e dúas matrices A e B de orde $n \times m$ e $p \times q$ respectivamente, e calcule o producto matricial de ambas. O programa debe comprobar que son multiplicábeis. Usa n=2, m=3, p=3, q=2, **a**=[1 2 3;4 5 6] e **b**=[1 2;3 4;5 6], filas separadas por ";".

```
program matricial
integer,allocatable :: a(:,:),b(:,:),c(:,:)
integer :: p,q,s
print '("n,m,p,q? ",$)'
read *,n,m,p,q
if(m/=p) stop 'm debe ser igual a p'
allocate(a(n,m),b(p,q),c(n,q))
print *,'matriz a? '
do i=1,n
    read *,a(i,:)
end do
print *,'matriz b? '
do i=1,p
    read *,b(i,:)
end do
do i=1,n
    do j=1,q
        s=0
        do k=1,m
            s=s+a(i,k)*b(k,j)
        end do
        c(i,j)=s
    end do
end do
print *,'matriz a*b='
do i=1,n
    print *,c(i,:)
end do
deallocate(a,b,c)
end program matricial
```

Semana 4

Traballo en clase

1. **Aprendizaxe cooperativa na aula**. Este exercicio explicarase na clase interactiva.

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. **Valores únicos nun vector**. Escribe un programa **unico.f90** que defina o vector **x**=[3,1,2,0,2,0,1,9,1,9] e mostre por pantalla os seus elementos sen repeticións. Escribe outro programa **unico2.f90** que almacene os mesmos elementos, pero ordeados, nun vector **y**.

Programa **unico.f90**:

```
program unico
integer, parameter :: n=10
integer :: x(10)=[1,2,1,3,9,0,0,9,1,9],y
print *,'x=',x
print '("unico(x)=",$)'
do i=1,n
    y=x(i)
    if(all(x(i+1:n)/=y)) print '(i0," ",$)',y
end do
print *,''
end program unico
```

Programa **unico2.f90**:

```
program unico2
integer,parameter :: n=10
integer :: x(n)=[3,1,2,0,2,0,1,9,1,9],y(n),z
j=0
do i=1,n
    z=x(i)
    if(all(z/=x(:i-1))) then ! z e unico en vector x
        do k=1,j    ! busca primeiro valor maior que z en vector y
            if(y(k)>z) exit
        end do
        do l=j,k,-1 ! move y(l) con l>=k unha posicion a dereita
            y(l+1)=y(l)
        end do
        y(k)=z;j=j+1 ! inserta z en y(k)
    end if
end do
print *,'y=',y(:j)
end program unico2
```

4. **Cálculo de límite dunha función nun punto finito. Bucle indefinido**. Escribe un programa chamado `limite.f90` que calcule o límite:

$$\lim_{x\to 2} \frac{x^2+x-6}{x^2-4} \tag{13}$$

```
program limite
!------------------------
! version con variabel real: da warning por variable e paso non enteiros
! do x=1,3,0.05
!       print *,x,(x*x+x-6)/(x*x-4)
! end do
!------------------------
! version con bucle indefinido
x=1
do
    print *,x,(x*x+x-6)/(x*x-4)
    x=x+0.05
    if(x>3) exit ! para evitar pasar de x=3
end do
end program limite
```

Na terminal do VSCode, redirixe a saída do programa anterior a un arquivo executando o comando:

```
a.exe >limite.txt
```

E logo representa gráficamente esta saída co `Octave`. Para isto, vai ao menú de inicio de Windows e executa "Octave". Dentro do octave, vai á carpeta **fortran** e executa os seguintes comandos:

```
load limite.txt
plot(limite(:,1), limite(:,2))
grid on
```

Isto debe abrir unha ventá na que se mostra a gráfica de $f(x)$ para $x \in [1,3]$ tal como se mostra na figura 3. Finalmente, sae do Octave co comando `exit`.

Exercicios propostos

1. Escribe un programa `intervalo.f90` en Fortran que calcule para $x \in [-1,4]$ os valores da seguinte función definida por intervalos:

$$f(x) = \begin{cases} 1+x & x \le 0 \\ x & 0 < x < 1 \\ 2-x & 1 \le x \le 2 \\ 3x-x^2 & x > 2 \end{cases}$$

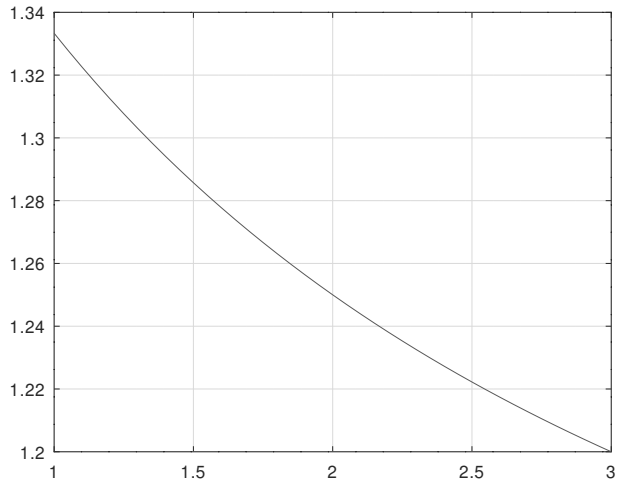

Figura 3: Representación gráfica dunha función dunha variábel usando Octave.

```
program intervalo
x=-1
do
    if(x<=0) then
        print *,x,1+x
    else if(x<1) then
        print *,x,x
    else if(x<=2) then
        print *,x,2-x
    else if(x<3) then
        print *,x,3*x-x*x
    else
        exit
    end if
    x=x+0.01
end do
end program intervalo
```

2. Escribe un programa `raices.f90` 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 $a_n, \ldots, a_0$ e calcule as raíces enteiras do polinomio, tendo en conta que as posíbeis raíces son divisores do termo independente a_0. Proba con $n = 5$ e o polinomio $x^5 + 3x^4 + 2x^3 - x^2 - 3x - 2$ que ten raíces enteiras $x = \pm 1$ e $x = -2$.

```
program raices
integer,allocatable :: a(:)
print '("n? ",$)'
read *,n
allocate(a(0:n))
do i=0,n
    print '("coeficiente de x^",i0,"? ")',n-i
    read *,a(n-i)
end do
print '("Polinomio: ",$)'
print '("(",i0,a,i0,$)',a(n),')x^',n
do i=n-1,0,-1
    print '("+(",i0,")x^",i0,$)',a(i),i
end do
print *,''
print '("Raices: ",$)'
m=abs(a(0))
do i=-m,m
    if(i==0) cycle
    if(mod(m,i)==0) then
        p=0
        do j=0,n
            p=p+a(j)*i**j
        end do
```

```
        if(p==0) print '(i0," ",$)',i
    end if
end do
print *,''
deallocate(a)
end program raices
```

3. Escribe un programa `distancia.f90` 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:

$$d = |\mathbf{x}-\mathbf{y}| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{14}$$

Proba con n=5, $\mathbf{x}$=[1,2,3,2,1] e $\mathbf{y}$=[5,2,1,3,4] e debes obter d=5.4772.

```
program distancia
real,allocatable :: x(:),y(:)
print '("n? ",$)'
read *,n
allocate(x(n),y(n))
print '("x[]? ",$)'
read *,x
print '("y[]? ",$)'
read *,y
print *,'distancia=',sqrt(sum((x-y)**2))
deallocate(x,y)
end program distancia
```

Semana 5

Traballo en clase

1. **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 producto 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. Os factores de 120 son $2^3 \cdot 3 \cdot 5$, e os factores de 252 son $2^2 \cdot 3^2 \cdot 7$. Os factores comúns son $2^3 \cdot 3^2 \cdot 5 \cdot 7$, e o mcm é 2520.

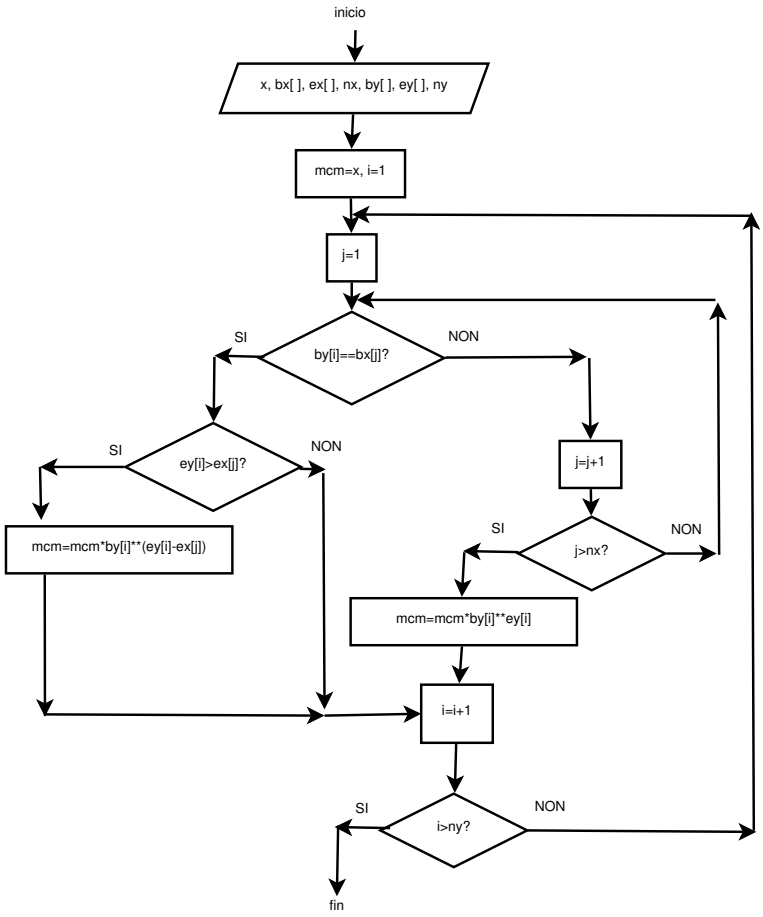

Figura 4: Diagrama de fluxo do cálculo do mcm de dous enteiros.

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

2. **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 $f(x) = \mathrm{e}^{-x}\,\mathrm{sen}\,2x$ 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.txt", 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(x)
        if(xh > b) exit
        x=xh;fx=fxh
    end do
    close(1)
    stop
    1 stop "derivada.txt xa existe"
    end program derivada
```

Para representar a función e a derivada, executa o **Octave**, sitúate no directorio **fortran** e teclea os seguintes comandos:

```
x=load('derivada.txt');
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)
```

Exercicios propostos

1. Escribe un programa chamado `vida.f90` que codifique o **algoritmo da vida**. Mediante unha matriz cadrada de orde n=10 representarase unha poboación aleatoria inicial de individuos. Un "1" nunha compoñente da matriz representará a existencia dun individuo nesa posición, mentres ca un "_" representará a non existencia de individuo nesa posición. O número de veciños dun individuo é o que determina o seu destino na seguinte xeración. O programa debe pedir por teclado un número enteiro m entre 1 e 100 (usa o 54). Entón debe crear a matriz inicial, inicializando o xerador de números aleatorios co valor m, e xerando n^2 veces un número x real aleatorio en $[0, 1]$ de modo que cada elemento da matriz sexa "1" se $x > 0{,}5$ ou "_" en caso contrario. As regras que gobernan a evolución das sucesivas xeracións dunha poboación inicial son as seguintes:

   - Un individuo con mais de 3 veciños nas posicións máis próximas morre por superpoboación.
   - Un individuo con menos de 2 veciños máis próximos morre por aillamento.
   - Aparece un individuo en calquer posición baleira que ten exactamente 3 veciños próximos.

   Estas regras aplícanse sobre a poboación inicial para determina-la seguinte xeración, e así sucesivamente, determinando a evolución das seguintes xeracións. O programa deberá presentar no monitor a poboación inicial e as sucesivas xeracións obtidas aplicando as regras anteriores. Para visualiza-la seguinte xeración será necesario que o usuario pulse unha tecla.

```
program vida
integer,parameter :: n=10
integer :: xeracion,v,p,q    !v=num. vecinhos
character(1) :: a(n,n),b(n,n),aleatorio,c
call inicializa(a,n)
print *,'Matriz inicial: '
call mostra(a,n)
xeracion=1; c='s'
do while(c=='s')
    do i=1,n
        do j=1,n
            v=0
            do k=-1,1
                p=i+k
                do l=-1,1
                    q=j+l
                    if(p>=1.and.p<=n.and.q>=1.and.q<=n) then
                        if(a(p,q)=='1'.and.(k/=0.and.l/=0)) v=v+1
                    end if
                end do
            end do
            if(a(i,j)=='1') then
                if(v>3.or.v<2) then
                    b(i,j)='_'
                else
                    b(i,j)=a(i,j)
                end if
```

```
            else
                if(v==3) then
                    b(i,j)='1'
                else
                    b(i,j)=a(i,j)
                end if
            end if
        end do
    end do
    a=b;xeracion=xeracion+1    ! copia de b -> a
    print '("xeracion ",i0,":")',xeracion
    call mostra(a,n)
    print *, 'continuar? (s/n)'
    read *, c
end do
end program vida
!-----------------------------------------
subroutine mostra(a,n)
character(1),intent(in) :: a(n,n)
integer,intent(in) :: n
do i=1,n
  print *,a(i,:)
end do
end subroutine mostra
!-----------------------------------------
subroutine inicializa(a,n)
character(1),intent(out) :: a(n,n)
integer,intent(in) :: n
print *, 'Introduce un numero entre 1 e 100:'
read *, i
call srand(i)
do i=1,n
  do j=1,n
    x=rand()
    if(x>0.5) then
      a(i,j)='1'
    else
      a(i,j)='_'
    endif
  end do
end do
end subroutine inicializa
```

Semana 6

Traballo en clase

1. **Determinante dunha matriz cadrada de orde n. Subprogramas recursivos. Paso de matrices a subprogramas. Arquivos**. 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 `matriz3.txt` que conteña a matriz [0 2 3; 4 5 6; 7 8 9] (filas separadas por ;) con determinante 3. Proba logo con outro arquivo `matriz4.txt` coa matriz [1 0 2 -1; 1 1 1 1; 3 2 0 1; 5 3 1 0], que ten determinante 4.

   Arquivo `matriz3.txt`:

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

   Arquivo `matriz4.txt`:

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

   Programa `determinante.f90`:

   ```
   program determinante
   integer,allocatable :: a(:,:)
   integer :: det
   character(100) :: nf='matriz3.txt'
   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
   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(:0)
       print *,'erro: matriz de orde <=0';stop
   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
       do i=1,n
           b=a(2:n,[(j,j=1,i-1),(j,j=i+1,n)])
           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
       print *,a(i,:)
   end do
   ```

```
print *,'-------------'
end subroutine imprime
```

2. **Derivada dun polinomio**. Escribe un programa chamado `polider.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 `polider.txt` (inicialmente baleiro). Logo, debe chamar n veces a un subprograma `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:

$$p^{(k)}(x) = \sum_{i=k}^{n} \left[\prod_{j=0}^{k-1} (i-j) \right] a_i x^{i-k}, \qquad k = 1, \ldots, n \tag{16}$$

Deste modo, o coeficiente b_{i-k} de x^{i-k} en $p^{(k)}(x)$ para $i = k, \ldots, n$, está dado por:

$$b_{i-k} = a_i \prod_{j=0}^{k-1} (i-j), \quad i = k..n \tag{17}$$

O subprograma `derivada(...)` anterior debe engadir ao arquivo `polider.txt` os coeficientes dos polinomios derivados (un polinomio en cada liña do arquivo).

EXEMPLO: dado o polinomio $p(x) = x^4 + x^3 + x^2 + x + 1$, resulta que $n = 4$ e as derivadas do polinomio son:

$$p'(x) = 4x^3 + 3x^2 + 2x + 1$$
$$p''(x) = 12x^2 + 6x + 2$$
$$p^{(3)}(x) = 24x + 6$$
$$p^{(4)}(x) = 24$$

e polo tanto o arquivo `polinomio.txt`, logo de executa-lo programa, debe almacena-lo seguinte contido:

```
4 3 2 1
12 6 2
24 6
24
```

```
program polider
    integer,allocatable :: a(:)
    print '("n? ",$)';read *,n
    allocate(a(0:n))
    print '("a[0:n]? ",$)';read *,a
    print '("p(x): ",$)'
    call imprime(a,n)
    open(1,file='polider.txt')
    do k=1,n
        call derivada(a,n,k)
    end do
    close(1)
end program polider
!----------------------------------------
subroutine derivada(a,n,k)
    integer,intent(in) :: a(0:n),n,k
    integer,allocatable :: b(:)
```

```
    integer :: p
    p=n-k
    allocate(b(0:p))
    do i=k,n
        ! m=a(i)
        ! do j=0,k-1
        !     m=m*(i-j)
        ! end do
        ! b(i-k)=m
        b(i-k)=a(i)*product([(i-j,j=0,k-1)])
    end do
    print '("derivada ",i0,": ",$)',k
    call imprime(b,p)
    do i=p,0,-1
        write (1,'(i0," ",$)') b(i)
    end do
    write (1,*)
    deallocate(b)
end subroutine derivada
!---------------------------------------
subroutine imprime(a,n)
    integer,intent(in) :: a(0:n),n
    do i=n,0,-1
        if(i==0) then
            print '(i0)',a(0)
        else if(i==1) then
            if(a(1)==1) then
                print '("x + ",$)'
            else
                print '(i0," x + ",$)',a(1)
            end if
        else
            if(a(i)==1) then
                print '(" x^",i0," + ",$)',i
            else
                print '(i0," x^",i0," + ",$)',a(i),i
            end if
        end if
    end do
end subroutine imprime
```

3. **Módulo, clase, herdanza, polimorfismo e sobrecarga de operador aritmético**. Escribe un programa `obxecto.f90` que defina o módulo **obxecto** cunha clase **punto** con dúas coordenadas x e y e un subprograma **mostra()** que mostre por pantalla x e y. No mesmo módulo, define unha clase **masa** derivada de **punto** que incorpore o valor m da masa e redefina o subprograma **mostra()** de modo que mostre por pantalla x, y e m. Sobrecarga o operador + cunha función **suma()** que retorne un punto onde x e y sexan a suma dos valores de x e y, respectivamente, dos dous sumandos. Sobrecarga tamén o operador > cunha función **maior()** que retorne **.true.** se o primeiro operando ten unha norma cadrada $x^2 + y^2$ superior ao segundo operando. Dende o programa principal:

   a) Declara tres obxectos p, q e r da clase **punto**.

   b) Inicialíza p a un obxecto da clase **punto** e chama ao seu subprograma **mostra()**. Comproba que se executa **mostra()** da clase **punto**.

   c) Inicialíza q a un obxecto da clase **masa** e chama ao seu subprograma **mostra()**. Comproba que se executa o **mostra()** de **masa**, e non da clase **punto**.

   d) Suma p e q e almacena o resultado en r. Chama ao subprograma **mostra()** de r e comproba que r é a suma de p e q.

   e) Mostra "p>q" se $p > q$ é certo e "p≤q" en caso contrario, usando o operador > sobrecargado.

```
module obxecto
    type :: punto
        integer :: x,y
    contains
        procedure :: mostra
    end type punto
```

```
    !-------------------------------------
    type,extends(punto) :: masa
        integer :: m
    contains
        procedure :: mostra=>mostra_masa
    end type masa
    !-------------------------------------
    interface operator(+)
        procedure suma
    end interface operator(+)
    !-------------------------------------
    interface operator(>)
        procedure maior
    end interface operator(>)
    !-------------------------------------
contains
    !-------------------------------------
    subroutine mostra(a)
        class(punto),intent(in) :: a
        print *,'punto: x=',a%x,' y=',a%y
    end subroutine mostra
    !-------------------------------------
    subroutine mostra_masa(a)
        class(masa),intent(in) :: a
        print *,'masa: x=',a%x,' y=',a%y,' m=',a%m
    end subroutine mostra_masa
    !-------------------------------------
    type(punto) function suma(a,b)
        class(punto),intent(in) :: a,b
        suma=punto(a%x+b%x,a%y+b%y)
    end function suma
    !-------------------------------------
    logical function maior(a,b)
        class(punto),intent(in) :: a,b
        maior=(a%x**2+a%y**2 > b%x**2+b%y**2)
    end function maior
end module obxecto
!-----------------------------------------
program principal
use obxecto
class(punto),allocatable :: p,q,r
!-----------------------------------------------
print *,'polimorfismo'
p=punto(1,3)
print *,'p:'
call p%mostra
q=masa(4,5,1)
print *,'q:'
call q%mostra
!-----------------------------------------------
print *,'sobrecarga de operador (+)'
r=p+q
print *,'r:'
call r%mostra
!-----------------------------------------------
print *,'sobrecarga de operador (>)'
if(p>q) then
    print *,'p>q'
else
    print *,'p<=q'
end if
end program principal
```

Exercicios propostos

1. **Produto escalar e matricial usando funcións intrínsecas**. Crea o programa `escalar.f90`, que defina un vector **v** e unha matriz cadrada **a**, ambos de orde 3, e calcule o producto escalar $\mathbf{v}^T\mathbf{v}$ e o producto matricial **aa**.

```
program exemplos_funcions
integer :: v(3) = [1,2,3]
integer :: a(3,3) = reshape([1,2,3,4,5,6,7,8,9],shape(a)), b(3,3)
interface
  subroutine imprime_matriz(a)
    integer,intent(in) :: a(:,:)
  end subroutine imprime_matriz
end interface
print '("v=",3(i0," "))', v
print *, "a="
call imprime_matriz(a)
print '("dot(v,v)=",i0)', dot_product(v,v)
b = matmul(a,a)
print *, "a*a="
call imprime_matriz(b)
b = transpose(a)
print *, "a^T="
call imprime_matriz(b)
end program exemplos_funcions
!---------------------------------
subroutine imprime_matriz(a)
integer,intent(in) :: a(:,:)
n=size(a,1)
do i=1,n
  do j=1,n
    print '(i0," ",$)',a(i,j)
  end do
  print *,''
end do
end subroutine imprime_matriz
```

2. Escribe un programa `vector.f90` que lea por teclado un número enteiro n, un vector **v** e unha matriz **A**, ambos de orde n. O programa principal debe chamar a un subprograma `producto(...)` (debes decidir o seu tipo e argumentos) que calcule o resultado do producto matricial **vA** (sendo **v** un vector fila). Proba con n=5, **v**=[1 2 3 4 5] e **a**=[1 2 3 4 5;6 7 8 9 8;7 6 5 4 3;2 1 2 3 4;5 6 7 8 9], filas separadas por ";".

```
program vector
integer,allocatable :: v(:),a(:,:),p(:)
print '("n? ",$)'
read *,n
allocate(v(n),a(n,n),p(n))
print '("v[]? ",$)'
read *,v
print *,'matriz a? '
do i=1,n
    read *,a(i,:)
end do
call producto(v,a,p,n)
print *,'producto v*a: ',p
deallocate(v,a,p)
end program vector
!---------------------------------------
subroutine producto(v,a,p,n)
integer,intent(in) :: v(n),a(n,n),n
integer,intent(out) :: p(n)
integer :: s
do i=1,n
    s=0
    do j=1,n
        s=s+v(j)*a(j,i)
    end do
    p(i)=s
end do
end subroutine producto
```

3. Escribe un programa `externa.f90` que xeralice `integral.f90` para calcular integrais definidas de modo que, usando funcións `external`, poida calcular a integral de calquer función (definida como función externa no programa).

```
program externa
real :: integral
external :: f,h
print *,integral(f,0.,1.)
print *,integral(h,-1.,1.)
end program externa
!-----------------------------------
real function integral(g,a,b)
real,intent(in) :: a,b
integral=0;x=a;h=0.001
do
    integral=integral+g(x)
    x=x+h
    if(x>b) exit
end do
integral=integral*h
end function integral
!-------------------------------------
real function f(t)
real,intent(in) :: t
f=sin(t)
return
end function f
!-------------------------------------
real function h(t)
real,intent(in) :: t
h=cos(t)
return
end function h
```

4. Escribe un programa chamado `covarianza.f90` que calcule a matriz de covarianza $\mathbf{\Sigma}$ dun conxunto de n vectores m-dimensionais $\{\mathbf{x}_i, i = 1, \ldots, n\}$, sendo $\mathbf{x}_i = (x_{i1}, \ldots, x_{im})$. O elemento ij da matriz de covarianza $\mathbf{\Sigma}$ (cadrada de orde m) defínese como:

$$\Sigma_{ij} = \frac{1}{n}\sum_{k=1}^{n}(x_{ki} - \langle x_i\rangle)(x_{kj} - \langle x_j\rangle), \quad j = 1..m \tag{18}$$

Onde $\langle x_i\rangle$ é o valor medio da compoñente i dos vectores $\mathbf{x}_k$:

$$\langle x_i\rangle = \frac{1}{n}\sum_{k=1}^{n} x_{ki} \tag{19}$$

O programa debe, dados $n = 12$ e $m = 5$, 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 $\langle x_i\rangle, i = 1, \ldots, m$. Por ltimo, debe chamar a un subprograma que calcule cada elemento $\Sigma_{ij}, i, j = 1, \ldots, m$, mediante a fórmula 3. Finalmente, debe chamar a outro subprograma que imprima a matriz Σ (fila a fila). Emprega o seguinte arquivo **vectores.txt** ($n = 12$, $m = 5$).

```
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
```

Programa `covarianza.f90`:

```
program covarianza
integer,parameter :: n=12,m=5
real :: x(n,m),c(n,m),media(m)
call le_vectores(x,n,m)
print *,'datos='
call imprime(x,n,m)
do i=1,m
    media(i)=sum(x(i,:))/n
end do
do i=1,m
  do j=1,m
    s=0
    do k=1,n
      s=s+(x(k,i)-media(i))*(x(k,j)-media(j))
    end do
    c(i,j)=s/n
  end do
end do
print *, 'matriz de covarianza='
call imprime(c,m,m)
end program covarianza
!-------------------------------------------------------------
subroutine le_vectores(x,n,m)
real,intent(out) :: x(n,m)
integer,intent(in) :: n,m
open(1,file='vectores.txt',status='old',err=1)
do i=1,n
  read (1,*) x(i,:)
end do
close(1)
return
1 stop 'erro en open vectores.txt'
end subroutine le_vectores
!-------------------------------------------------------------
subroutine imprime(x,n,m)
real,intent(in) :: x(n,m)
integer,intent(in) :: n,m
do i=1,n
  do j=1,m
    print '(f10.3," ",$)',x(i,j)
  end do
  print *,''
end do
end subroutine imprime
```

5. **Conversión entre tipos de datos** Escribe un programa `conversion.f90` que defina n=5, x=1.23 e unha cadea de caracteres c=`'456'`. O programa debe convertir e mostrar por pantalla: 1) n a real; 2) x a enteiro; 3) redondear x a enteiro por exceso; 3) redondear x a enteiro por defecto; 4) redondear x ao enteiro máis cercano; 5) c a enteiro; 6) c a real; n a carácter; 7) x a carácter.

```
program conversion
character(10) :: c='456'
n=5;x=1.23
print *,'n->real: ',real(n)
print *,'x->enteiro: ',int(x)
print *,'x->enteiro defecto: ',floor(x)
print *,'x->enteiro exceso: ',ceiling(x)
print *,'x->enteiro mais cercano: ',nint(x)
read (c,*) i
print *,'c->enteiro: ',i
read (c,*) y
print *,'c->real: ',y
write (c,'(i0)') n
print *,'n->caracter: <',c,'>'
write (c,'(f5.2)') x
print *,'x->caracter: <',c,'>'
end program conversion
```

6. Escribir un programa en Fortran chamado `gauss.f90` que resuelva un sistema de n ecuacións lineais con n incógnitas empregando o Método de Eliminación Gaussiana. Dado o sistema seguinte:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \tag{20}$$
$$\ldots \tag{21}$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \tag{22}$$

O método de eliminación transforma este sistema no seguinte:

$$x_1 + a'_{12}x_2 + \ldots + a'_{1n}x_n = b'_1 \tag{23}$$
$$0 + x_2 + \ldots + a'_{1n}x_n = b'_1 \tag{24}$$
$$\ldots \tag{25}$$
$$0 + 0 + \ldots + x_n = b'_n \tag{26}$$

Onde se pode despexar directamente x_n, sustituir na $(n-1)$-ésima ecuación e despexar x_{n-1} e así sucesivamente ata calcula-las n incógnitas. As únicas transformacións permitidas son:

- Dividir tódolos elementos dunha fila polo mesmo número.
- Sumar a tódolos elementos dunha fila o producto dun escalar polo elemento correspondente doutra fila

Arquivo **sistema.txt**:

```
3
1 0 2 1
-1 1 2 -2
0 1 1 3
```

O programa `gauss.f90` é o seguinte:

```
! x +   + 2z = 1
!-x + y + 2z = -2
!     y + z  = 3
! x = 5, y = 5, z = -2
program gauss
real,allocatable  :: a(:,:),x(:)
print *, "sistema inicial:"
open(1,file='sistema.txt',status='old')
read (1,*) n
m=n+1
allocate(a(n,m),x(n))
do i=1,n
  read (1,*) a(i,:)
  print *,a(i,:)
end do
close(1)
call verifica(a,n)
do i=1,n
  do j=i,n
    if(a(j,i)/=0) then
      t=a(j,i)
      do k=1,m
        a(j,k)=a(j,k)/t
      end do
    end if
  end do
  do j=i+1,n
    if(a(j,i)/=0) then
            a(j,:)=a(j,:)-a(i,:)
    end if
  end do
  print *, "pasada", i, "-esima"
```

```
  do k=1,n
    print *,a(k,:)
  end do
end do
do i=n,1,-1
  x(i)=a(i,m)
  do j=i+1,n
    x(i)=x(i)-a(i,j)*x(j)
  end do
  print *, "x(", i, ") = ", x(i)
end do
stop
end program gauss
!-----------------------------------------------
subroutine verifica(a, n)
real,intent(inout) :: a(n,n)
integer, intent(in) :: n
m=n+1
do i=1,n
  if(a(i,i)==0) then
    do j=1,n
      if(a(j,i)/=0) exit
    end do
    if(j==m) then
      print *, "incognita", i, "ten todolos coeficientes nulos"
      stop
    end if
    a(i,:)=a(i,:)+a(j,:)
  end if
    print *,a(i,:)
end do
return
end subroutine verifica
!-----------------------------------------------
subroutine le_sistema(a,n)
real,intent(out) :: a(10,11)
integer,intent(in) :: n
print *, "sistema inicial:"
open(1,file='sistema.txt',status='old')
do i=1,n
  read (1,*) (a(i,j),j=1,n+1)
  print *,(a(i,j),j=1,n+1)
end do
close(1)
return
end subroutine le_sistema
```

Semana 7

Traballo en clase

1. **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:

$$f(x) = p'(x) = \lim_{h \to 0} \frac{p(x+h) - p(x)}{h} \tag{27}$$

Se tomamos $h \simeq 0^+$ podemos aproximar:

$$f(x) \simeq \frac{p(x+h) - p(x)}{h} \quad \rightarrow \quad p(x+h) \simeq p(x) + hf(x) \tag{28}$$

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 $p(x)$ para $x > a$. Este valor inicial $p(a)$ prefixado é equivalente á constante C que se lle pode sumar á función primitiva $p(x)$. Polo tanto, $p(a)$ pode ter calquera valor simplemente cambiando C, de modo que podemos decidir o valor $p(a)$. A fórmula 28 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 5. O valor de h debe verificar que para $x \in [a, b]$ a función $f(x)$ pode aproximarse entre x e $x+h$ por unha liña recta con pendente $f(x)$. No programa, calcula $p(x) = \int_a^x f(t)dt$ no intervalo $[a, b]$ usando $a = 0, b = 1, f(t) = t$, $p(a) = 0$. Repite o cálculo para $a = 0, b = \pi, f(t) = \operatorname{sen} t, p(a) = 0$. Se queres calcular outra integral indefinida, so tes que cambiar a, b, $p(a)$ e $f(x)$.

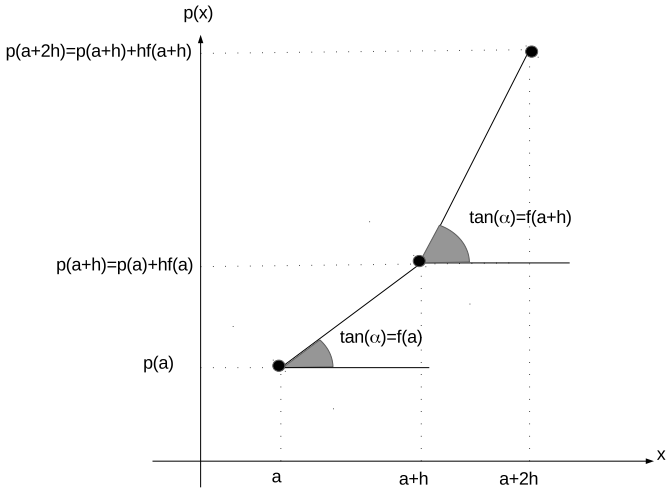

Figura 5: Aproximación numérica á primitiva $p(x)$ dunha función $f(x)$.

```
program primitiva
f(x)=x;a=0;b=1;px=0
!f(x)=sin(x);a=0;b=3.141592;px=0
open(1, file="integral.txt", status="new", err=1)
x=a;h=0.01
do
     fx=f(x)
     write (1, *) x,fx,px
     x=x+h;px=px+h*fx
     if(x > b) exit
end do
close(1)
stop
```

```
1 stop "integral.txt 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.txt`, que podes descargar dende este **enlace**.

```
program primitiva2
open(1, file="integral.txt", status="new", err=1)
open(2, file="valores_funcion.txt", 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.txt xa existe"
2 stop "valores_funcion.txt non existe"
end program primitiva2
```

2. **Cálculo dunha integral definida. Reais de dobre precisión**. Escribe un programa chamado `integral.f90` que calcule a integral definida funha función $f(x)$ no intervalo $[a, b]$. Prueba con $\int_{-1}^{1} \frac{\arccos x}{1+x^2} dx$. Usa reais de dobre precisión.

```
program integral
real(8) :: a=-1,b=1,h=1d-004,s=0,x,f  ! modificar a,b,h para cada caso
f(x)=acos(x)/(1+x*x)   ! modificar para cada caso
x=a
do
    s=s+f(x);x=x+h
    if(x > b) exit
end do
s=h*s
print *,'h=',h
print *,'integral=', s
print *,'valor correcto= 2.46740110027234'
print *,'diferencia=',abs(s-2.46740110027234)
end program integral
```

Compara o resultado co proporcionado polo Octave. Para isto, executa:

```
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 programas `derivada.f90`, `primitiva.f90` e `integral.f90`, consideremos unha función $f(x)$ definida no intervalo $[a, b]$ por un vector de n valores $\mathbf{f} = (f_1, \ldots, f_n)$, onde $f_i = f(x_i)$ con $x_i = a + h(i-1)$ con $i = 1 \ldots n$ e $h = \frac{b-a}{n-1}$. Consideraremos que o número n de puntos é suficientemente elevado como para que a función $f(x)$ poda aproximarse con precisión por unha recta entre x_i e x_{i+1} ou, equivalentemente, que h é suficientemente pequeno. Entón temos que:

- A súa derivada $d(x) = f'(x)$ pode describirse polo vector $\mathbf{d} = (d_1, \ldots, d_{n-1})$, onde $d_i = \frac{f_{i+1} - f_i}{h}$, con $i = 1 \ldots n-1$. 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 $\mathbf{p} = (p_1, \ldots, p_n)$, onde $p_1 = p(a)$ (prefixado por nós arbitrariamente, p.ex. $p(a) = 0$) e $p_{i+1} = p_i + hf_i$ con $i = 1 \ldots n-1$. 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 $h\sum_{i=1}^{n} f_i$. É decir, é a suma de tódolos valores de f no intervalo $[a,b]$ multiplicada por h.

3. **Medida do tempo consumido por un programa en Fortran**. Descarga o programa `tempo.f90` dende este **enlace**. Este programa executa un bucle de 10^8 iteracións e mostra o tempo consumido:

```
program tempo
real(8) :: inicio, fin, n, i
n=1e8; i=0
print '("medindo tempo consumido por ",d8.1," iteracions ...")',n
call cpu_time(inicio)
do
    i=i+1
    if(i>n) exit
end do
call cpu_time(fin)
print '("n=",d8.1, " tempo= ",f10.4," s.")',n,fin-inicio
end program tempo
```

4. **Xerador de números aleatorios**. Descarga o programa `aleatorio.f90` dende este **enlace**. Este programa imprime por pantalla 10 números aleatorios no intervalo $[a,b]$ usando a función `rand()` de `gfortran`. Usa $a=1, b=10$. O programa é aleatorio reproducible, da os mesmos números aleatorios en tódalas execucións). Para ser non reproducible (diferentes números en distintas execucións) descomenta a liña do `system_clock` para que inicialice o xerador de números aleatorios co reloxio do ordenador.

```
program aleatorio
integer,parameter :: n=10
real :: x(n)
integer :: y(n)
call system_clock(count=i);call srand(i)  ! non reproducible
print '(a,$)', "introduce a,b: "
read *,a,b
rango=b-a
do i=1,n
  x(i)=rango*rand()+a
end do
print '("real: ",10f8.4)', x
print '("enteiro: ",10i5)', nint(x)
stop
end program aleatorio
```

5. **Exemplo de exame de Fortran**. Escribe un programa `exame.f90` que defina unha constante enteira n=10 e dous vectores **x** e **y** de lonxitude n con elementos $x_i = 1 + i^2(n-i)$ e $y_i = 1 + (i-1)(n-i+1)$ para $i = 1..n$. Mostra **x** e **y** por pantalla, cada un nunha liña. Chama ao subprograma **subprog(...)** que calcule a suma m dos elementos y_i de **y** que cumpren ao mesmo tempo que: 1) son distintos de tódolos elementos de **x**; e 2) existe en **x** algún elemento maior que y_i. Mostra m por pantalla no programa principal. Calcula e imprime unha matriz enteira **a** de orde n. Para calcular o elemento a_{ij} con $i,j = 1..n$, suma os elementos x_p dende p=1 ata que a suma k supere o umbral $u_{ij} = i^2 + j$, de modo que $a_{ij} = i + j^2 + p$ onde p é o índice do último x_p sumado. Se no cálculo da suma $p > n$ entón fai $p = 1$. Almacena no arquivo **exame.txt** a matriz **a**, cada fila nunha liña con formato enteiro de ancho mínimo.

```
program cli6
integer,parameter :: n=10
integer :: x(n),y(n),subprog,a(n,n),p
do i=1,n
  x(i)=1+(n-i)*i**2; y(i)=1+(i-1)*(n-i+1)
end do
print *,'x=',x
print *,'y=',y
m=subprog(x,y,n)
print *,'m=',m
open(1,file='exame6.txt')
write(1,*) 'a='
do i=1,n
  do j=1,n
    k=0;p=1;u=i**2+j
```

```
    do while(k<u)
      k=k+x(p)
      p=p+1
      if(p>n) p=1
    end do
    a(i,j)=i+j**2+p
    write(1,'(i0," ",$)') a(i,j)
  end do
  write (1,*)
end do
close(1)
end program cli6
!---------------------------------------
integer function subprog(x,y,n) result(m)
integer,intent(in) :: x(n),y(n),n
m=0
do i=1,n
  j=y(i)
  if(all(j/=x).and.any(x>j)) m=m+j
end do
end function subprog
```

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
```

Exercicios propostos

1. **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 na terminal do VSCode os comandos:

   ```
   gfortran -c media.f90 mediana.f90 desviacion.f90 ordea.f90
   ar qv libstat.a media.o desviacion.o mediana.o ordea.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:

   ```
   gfortran -L. principal.f90 -lstat
   ```

   b) **Libraría dinámica**. Para crear unha libraría dinámica `libstat.so`, executa os comandos:

   ```
   gfortran -fpic -c media.f90 desviacion.f90 mediana.f90 ordea.f90
   gfortran -shared -o libstat.so media.o desviacion.o mediana.o ordea.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:

   ```
   gfortran -L. principal.f90 -lstat
   ```

   Executa o programa co comando **a.exe**.

   c) **Compilación separada**. Tamén se pode compilar separadamente, sen crear ningunha libraría, cos comandos:

   ```
   gfortran -c media.f90 mediana.f90 desviacion.f90 ordea.f90
   gfortran principal.f90 *.o
   ```

2. Codificar un programa `minmax.f90` que lea dende un arquivo **minmax.txt** un número enteiro n e unha matriz **A** enteira cadrada de orde n e calcule, usando subprogramas: 1) cal das súas filas ten menor valor medio; 2) o valor máximo da dita fila. Usa n=5 e a matriz [15 19 12 19 13; 18 19 3 5 5;8 3 8 6 14; 13 11 8 15 15; 3 8 15 19 16].

```
program minmax
integer,allocatable :: a(:,:)
open(1,file='minmax.txt',status='old',err=1)
read (1,*) n
allocate(a(n,n))
do i=1,n
    read (1,*) a(i,:)
end do
close(1)
print *,'a:'
do i=1,n
    print *,a(i,:)
end do
call fila_menor_media(a,n,i,j)
```

```
print '("fila con menor media=",i0," valor maximo=",i0)',i,j
deallocate(a)
stop
1 stop 'erro open minmax.txt'
end program minmax
!-----------------------------------------------
subroutine fila_menor_media(a,n,i,j)
integer,intent(in) :: a(n,n),n
integer,intent(out) :: i,j
real(8),parameter :: inf=huge(dbl_prec_var)
s_min=inf
do k=1,n
    s=sum(a(k,:))/n
    if(s<s_min) then
        s_min=s;i=k;j=a(k,1)
        do l=2,n
            if(a(k,l)>j) j=a(k,l)
        end do
    end if
end do
end subroutine fila_menor_media
```

3. Escribe un programa `transformado.f90` 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:

$$w_i = \sum_{j=1}^{i} v_j, \quad i = 1, \ldots, n \tag{29}$$

Usa n=10 e $\mathbf{v}$=[1,2,3,4,5,6,7,8,9,10].

```
program transformado
integer,allocatable :: v(:),w(:)
print '("n? ",$)'
read *,n
allocate(v(n),w(n))
print '("v[]? ",$)'
read *,v
w(1)=v(1)
do i=2,n
    w(i)=w(i-1)+v(i)
end do
print *,'w= ',w
deallocate(v,w)
end program transformado
```

4. Escribe un programa `exterior.f90` 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_producto_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. Usa n=5, $\mathbf{v}$=[1,2,3,4,5] e $\mathbf{w}$=[5 4 3 2 1].

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

```
program exterior
integer,allocatable :: v(:),w(:),a(:,:)
print '("n? ",$)'
read *,n
allocate(v(n),w(n),a(n,n))
print '("v[]? ",$)'
read *,v
print '("w[]? ",$)'
read *,w
call calcula_producto_exterior(v,w,a,n)
print *,'producto exterior v*w='
```

```
do i=1,n
    print *,a(i,:)
end do
deallocate(v,w,a)
end program exterior
!------------------------------
subroutine calcula_producto_exterior(v,w,a,n)
integer,intent(in) :: v(n),w(n),n
integer,intent(out) :: a(n,n)
do i=1,n
    do j=1,n
        a(i,j)=v(i)*w(j)
    end do
end do
end subroutine calcula_producto_exterior
```