

Programación en Python

Exercicios clases interactivas

Sesión 1: uso do intérprete interactivo de Python

Traballo en clase

1. **Instalar o intérprete de Python en Linux:** instalar o paquete `python` para o interprete de Python utilizando as ferramentas de instalación de paquetes que manexedes. O intérprete mellorado é o paquete `ipython`.

Na práctica, para cálculo científico, necesítase funcionalidade que non está no interprete de `python`. Durante o curso utilizarase os seguintes paquetes (que hai que instalar ademais do interprete de `python`):

- `numpy` (NUMeric Python): matrices e alxebra lineal
<http://docs.scipy.org/doc/numpy/reference/>.
- `scipy` (SCientific Python): moitas rutinas numéricas (é como un `numpy` avanzado)
<http://docs.scipy.org/doc/scipy-0.14.0/reference/>.
- `matplotlib`: (PLOTting LIBrary): para gráficos
<http://matplotlib.org/index.html>.
- `sympy` (SYMbolic Python): cálculo simbólico
<http://www.sympy.org/en/index.html>.

2. **Arrancar o intérprete interactivo de Python.** Para arrancar o intérprete interactivo de Python executa nun terminal o comando `python` para arrancar o intérprete básico (aparece o símbolo `>>>` para comezar a introducir comandos) ou `ipython` para arrancar o intérprete mellorado (aparece o símbolo `In [1]:`). En ambos os dous casos, para saír do entorno interactivo pódese introducir a combinación de teclas **Control-D** ou o comando `quit()`. Lembrade que para os comentarios en Python utilízase o símbolo `#` (o intérprete de Python ignora todo o que está a continuación deste símbolo e ate o final da liña). En todo momento podes limpar a terminal tecleando **Ctrl-L** ou en `ipython` executando o comando `clear`.

Imos usar en adiante o `ipython`. Ao entrar, executa o comando:

```
%logstart -o sesion1.py append
```

para que `ipython` almacene no arquivo `sesion1.py` os comandos que executes e máis as súas saídas.

Podes consultar a documentación das funcións de Python accedendo co navegador á **documentación online**.

3. **Instalar un IDE (*Integrative Development Environment*) para programar en Python:** SPYDER (Scientific Python Development EnviRonment) probablemente sexa o máis coñecido e está integrado dentro da distribución de Python **Anaconda** que podes descargar de: **Anaconda**. Anaconda inclúe os paquetes mencionados (`numpy`, `scipy`, `matplotlib` e `sympy`) máis o IDE Spyder. Esta opción é a recomendada para as persoas que utilicedes o sistema operativo Windows. O SPYDER integra no mesmo programa un editor de texto e unha ventá co interprete mellorado de `python` (`ipython`), ademáis doutras utilidades. Utilizaremos a versión de Python 2.7.xx.
4. **Uso do intérprete de Python como unha calculadora.** A liña que comeza polo símbolo `>>>` é o comando de `python` e a seguinte liña representa o resultado da execución de do comando.

```
>>> 2+3 # suma
5
>>> 2 + 3 # suma
5
>>> 5 - 7 # resta
-2
```

```

>>> 5 * 3 # multiplicacion
15
>>> 8 / 3 # division enteira cando numerador e denominador son enteiros
2
>>> 8.0 / 3 # division en punto flotante
2.6666666666666665
>>> 8.0 // 3 # division enteira
2.0
>>> 3 ** 2 # exponenciacion
9
>>> 11 % 4 # modulo (resto da division enteira)
3
>>> 5 - 2 * 3 # operacions encadeadas
-1
>>> (5 - 2) * 3 # cos parentesis mudase a precedencia dos operadores
9

```

5. **Variables e tipos de datos básicos.** Igual que no apartado anterior, as liñas que comezas polo símbolo >>> son comandos python para executar e o resto das liñas son o resultado da execución do comando anterior.

```

>>> x = 5 # asigna o valor 5 a variable de nome x
>>> print x # visualiza no terminal o valor da variable x
5
>>> type(x) # devolve o tipo de variable x
<type 'int'>
>>> y = 2.3 # asignacion
>>> type(y) # tipo de dato
<type 'float'>
>>> z = x + y # asignar a variable z o valor da avaliacion dunha expresion
>>> print z
7.3
>>> type(z)
<type 'float'>
>>> cadea = "Ola a todas" # asigna a variable co nome cadea unha cadea de caracteres
>>> print cadea
Ola a todas
>>> type(cadea) # devolve de tipo string (cadea de caracteres)
<type 'str'>
>>> c = 4 + 2j # asigna a variable c un numero complexo (ollos: 1+1j, non vale 1+j)
>>> print c
(4+2j)
>>> type(c)
<type 'complex'>
>>> b = True # asignar a variable co nome b un booleano
>>> print b
True
>>> type(b)
<type 'bool'>
>>> c2 = 3 - 4j
>>> c + c2 # suma de numeros complexos
(7-2j)
>>> c * c2 # multiplicacion de numeros complexos
(20-10j)
>>> a, b, c = 5, False, "Ola" # asignacions multiples
>>> print a
5

```

```
>>> print b
False
>>> print c
Ola
```

6. **Funcións básicas.** Valor absoluto, tipo de dato, comparación de variables, conversión de tipos de datos.

```
>>> x = -3.4
>>> abs(x) # valor absoluto
3.4
>>> c = 4 - 3j # numero complexo
>>> abs(c) # valor absoluto
5.0
>>> type(x)
<type 'float'>
>>> y = int(x) # convirte x a un numero enteiro
>>> y
-3
>>> type(y)
<type 'int'>
>>> float(y) # convirte y a un numero real ou flotante
-3.0
>>> cmp(2, 3) # compara ambos elementos (2 > 3). Devolve False
-1
>>> cmp(3,2) # devolve True
1
```

7. **Funcións matemáticas.** Para executar as funcións matemáticas que normalmente existen nunha calculadora científica (funcións trigonométricas, logaritmos, etc.) hai que importar o módulo `math` da biblioteca estándar de Python. A partir dese momento, xa se pode utilizar as funcións matemáticas coma nunha calculadora.

```
>>> from math import * # importa todas as functions do modulo math
>>> pi # constante pi definida no modulo math
3.141592653589793
>>> e # constante e definida no modulo math
2.718281828459045
>>> sin(0)
0.0
>>> sin(pi/2)
1.0
>>> sqrt(3) # raiz cadrada de 3
1.7320508075688772
>>> log(e) # logaritmo neperiano
1.0
>>> exp(2) # funcion exponencial -- e^2
7.38905609893065
>>> fabs(-7.3) # valor absoluto dun numero real
7.3
```

8. **Manipulación de listas.** As listas son secuencias de elementos de calquer tipo separados por comas.

```
>>> # definicion dunha lista e asignacion a variable x
>>> x = [4, 7.4, True, 4 + 2j, "Ola", [2, 1]]
>>> type(x)
```

```

<type 'list'>
>>> len(x) # devolve a lonxitude (numero elementos) da lista
6
>>> x[0] # acceso ao primeiro elemento da lista (empeza no 0)
4
>>> x[1] # acceso ao segundo elemento da lista
7.4
>>> y = x[5] # asigna o sexto elemento da lista a variable y
>>> y
[2, 1]
>>> type(y) # pensa que os elementos da lista poden ser listas
<type 'list'>
>>> x[-1] # acceso o ultimo elemento da lista
[2, 1]
>>> x[-2] # acceso o penultimo elemento da lista
'0la'

```

9. Manipulación avanzada de lista. Faremos un exemplo cunha lista de números.

```

>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9] # definicion lista x
>>> n = len(x) # numero de elementos de x
>>> n
9
>>> x[3] = 20 # modificacion do cuarto elemento
>>> x
[1, 2, 3, 20, 5, 6, 7, 8, 9]
>>> x[3:5] # acceso a un subrango de elementos do vector x
[20, 5]
>>> # o operador : permite especificar un rango continuo inicio:final
>>> x[0:n:2] # elementos non contiguos separados 2 posiciones
[1, 3, 5, 7, 9]
>>> # para especificar rango non continuo inicio:final:incremento
x[4:] # acceso desde o quinto elemento ate o final
[5, 6, 7, 8, 9]
>>> x[:2] # acceso desde o primeiro elemento ate a posicion 2
[1, 2]
>>> x[2:-2] # desde o terceiro elemento ata o penultimo
[3, 20, 5, 6, 7]
>>> x
[1, 2, 3, 20, 5, 6, 7, 8, 9]
>>> x[2:4]
[3, 20]
>>> x[2:4] = [30, 30] # modifico unha subsecuencia ou subrango
>>> x
[1, 2, 30, 30, 5, 6, 7, 8, 9]
>>> x[4:7] = [] # borro elementos x[4] a x[6] na lista
>>> x
[1, 2, 30, 30, 8, 9]
>>> x[2:3] = [1,2,3,4,5] # modifico a lista (tamen en lonxitude)
>>> x
[1, 2, 1, 2, 3, 4, 5, 30, 8, 9]
>>> x[2]=[] #pon unha lista baleira na posicion 2
>>> x
[1, 2, [], 2, 3, 4, 5, 30, 8, 9]
>>> range(6) # funcion range([ini,]fin[,paso])
# xera unha lista desde ini ou 0 ate fin-1 utilizando un incremento de paso
[0, 1, 2, 3, 4]

```

```
>>> range(1,5)
[1, 2, 3, 4]
>>> range(1,10,3)
[1, 4, 7]
```

10. **Métodos sobre listas.** Internamente en Python, as listas son obxectos polo que se poden utilizar métodos para a súa manipulación.

```
>>> x = [2, 4, 1, 8, 2] # defino lista x
>>> len(x) # numero de elementos
5
>>> x.insert(3, 10) # inserto 10 na cuarta posicion
>>> x
[2, 4, 1, 10, 8, 2]
>>> x.append(5) # engado 5 o final da lista
>>> x
[2, 4, 1, 10, 8, 2, 5]
>>> x.reverse() # invirte a orde da lista
>>> x
[5, 2, 8, 10, 1, 4, 2]
>>> x.sort() # ordea a lista
>>> x
[1, 2, 2, 4, 5, 8, 10]
>>> x.count(2) # conta as ocorrencias do 2
2
>>> x.remove(8) # elimina a 1a ocorrencia do 8 da lista x
>>> x
[1, 2, 2, 4, 5, 10]
>>> x.index(5) # devolve a posicion do elemento 5 en x
4
>>> x.index(2) # devolve o indice onde aparece o primeiro 2 en x
1
>>> del x[4] # elimina o elemento da quinta posicion na lista x
>>> x
[1, 2, 2, 4, 10]
```

11. **Tuplas.** Son secuencias de elementos como as listas pero na que os seus elementos non se poden modificar.

```
>>> a = (1,2,3,4,5) # definicion dunha tupla
>>> type(a) # tipo de dato
<type 'tuple'>
>>> len(a) # numero de elementos
5
>>> a[1:3] # acceso a una subsecuencia da tupla
(2, 3)
>>> a[1] = 10 # da erro, unha tupla non se pode modificar
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Traballo a desenvolver pol@ alumn@

1. Define as variables $x = 4$, $y=5.6$, $z = 3$, $c = 4 + 5j$, $d = -3 + 2j$, $b = \text{True}$ e $m = \text{"Exemplo de python"}$ e realiza o seguinte:

a) Calcula $x + z$, x^z , yz , resto da división $\frac{x}{z}$, $(z - x)y^2$, cd e $\frac{c}{d}$.

- b) Comproba o tipo de dato de cada variable.
- c) Calcula $y + \frac{x}{z}$, a división enteira de $\frac{y}{x}$
- d) Comproba que número é maior (x ou z).
- e) Calcula o máximo e mínimo de x , y e z .
- f) Calcula o factorial de x .

2. Avalía as seguintes expresións: $\sin^2(\pi) \cos\left(\frac{4\pi}{3}\right)$, $e^3 \log 5$, $e^{\sqrt{2\pi}} 5$, $\sqrt{\frac{3\pi}{2}}$, $e^{7\pi/3}$, $8\pi^{50}$, $\sqrt{40\pi^5 + 6\pi + e^{7\pi}}$

3. Calcula o valor absoluto de $x=-8.3$ e $y=-4$

4. Calcula o valor máis próximo, valor por exceso, valor por defecto e parte enteira de $x=4.2$, $y=-3.7$ e $z=-3.6+4.2j$.

5. **Listas:** Define unha lista cos seguintes elementos 3, 4, 5, 6, 2, 6, 4, 1, 3, 4 e realiza as seguintes operacións:

- a) Calcula o número de elementos.
- b) Comproba o tipo de dato da variable na que está almacenada.
- c) Accede ó terceiro elemento, que será o 5.
- d) Accede ós primeiros 4 elementos.
- e) Accede ós últimos 3 elementos.
- f) Calcula o máximo e o mínimo dos elementos da lista.
- g) Suma os elementos da lista.
- h) Conta o número de veces que aparece o valor 4 na lista.
- i) Modifica o sexto elemento polo valor 20.
- j) Accede ós elementos entre a terceira e sétima posición na lista.
- k) Accede ós elementos situados nas posicións impares da lista (posicións 1, 3, 5, ...).
- l) Elimina os elementos nas posicións cuarta, quinta, sexta e sétima.
- m) Modifica os elementos das posicións segunda e terceira da lista polos valores 30, 40.
- n) Engade o elemento 15 ó principio da lista.
- ñ) Engade o elemento 50 ó final da lista.
- o) Engade o elemento 60 na posición 4 da lista (lembra que o primeiro elemento é a posición 0).
- p) Inverte a orde da lista.
- q) Ordea a lista de menor a maior.
- r) Calcula a posición do número 40 na lista e elimínalo.

Sesión 2 : cálculo científico con NumPy e representación gráfica con Matplotlib

Traballo en clase

O paquete Numpy ten funcionalidade para realizar operacións vectorizadas (aplicar unha operación a todo un vector ou matriz no canto dun escalar ó estilo do linguaxe de programación Matlab). O tipo de dato máis importante é `array` para representar vectores, matrices e vectores multidimensionais. O `array` é como unha lista de Python na que todos os seus elementos son do mesmo tipo (`Sn`, `bool`, `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`, `float`, `float32`, `float64`, `complex`, `complex128`). Nos seguintes apartados revísanse algunhas funcionalidades de Numpy executando comandos.

1. Probas no intérprete ipython:

```
>>> from numpy import *
>>> x = array([1,2,3,4,5,6]) # creacion dun array a partires dunha lista
>>> x.ndim # dimension do array
1
>>> x.size # numero de elementos
6
>>> x.itemsize # numero de bytes que ocupa cada elemento
8
>>> x.dtype # tipo de dato dos elementos
dtype('int64')
>>> z = arange(1,6, dtype='int8') # creacion dun array coa funcion arange()
>>> z
array([1, 2, 3, 4, 5])
>>> z.dtype
dtype('int8')
>>> z = arange(1,6,0.2, dtype='float') # creacion dun array
>>> z
array([ 1. ,  1.2,  1.4,  1.6,  1.8,  2. ,  2.2,  2.4,  2.6,  2.8,  3. ,
        3.2,  3.4,  3.6,  3.8,  4. ,  4.2,  4.4,  4.6,  4.8,  5. ,  5.2,
        5.4,  5.6,  5.8])
>>> z.size
25
>>> xl = linspace(1,8, 10) # crea un vector de elementos equiespaciados
# con linspace()
>>> xl
array([ 1.          ,  1.77777778,  2.55555556,  3.33333333,  4.11111111,
        4.88888889,  5.66666667,  6.44444444,  7.22222222,  8.          ])
>>> a = ones([2,3]) # creacion dun array de 2 filas e 3 columnas
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> a.ndim
2
>>> a.dtype
dtype('float64')
>>> b = zeros([3,4], dtype='int')
>>> b
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

```

>>> c=b # copia por referencia
>>> c[1][2]=5 # modifico elemento (1,2)
>>> c
array([[0, 0, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 0]])
>>> b # os cambios tamen se ven reflexados en b
array([[0, 0, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 0]])
>>> d =c.copy() # copia real da matriz c en d
>>> d
array([[0, 0, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 0]])
>>> d[0][0]=30 # modifico matriz d
>>> d
array([[30, 0, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 0]])
>>> c # a matriz c non se modifica
array([[0, 0, 0, 0],
       [0, 0, 5, 0],
       [0, 0, 0, 0]])
>>> x=array([1,2,3]) # creo vector x
array([1, 2, 3])
>>> x = insert(x, 1, 20) # inserto elemento 20 na posicion 1
>>> x
array([ 1, 20,  2,  3])
>>> x=append(x, 30) # engado o valor 30 o final do vector
>>> x
array([ 1, 20,  2,  3, 30])
>>> x=delete(x,2) # borro en x o elemento na posicion 2
>>> x
array([ 1, 20,  3, 30])
>>> y=array([[1,2,3],[4,5,6]]) # creo matriz y a partires dunha lista
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
>>> y.ndim # numero de dimensions
2
>>> y.T # matriz transposta
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> y.flatten() # convirto matriz en vector (conversion por filas)
array([1, 2, 3, 4, 5, 6])
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
>>> ravel(y) # igual que flatten pero funcion
array([1, 2, 3, 4, 5, 6])
>>> ravel(y, 'F') # conversion de matriz en vector por columnas
array([1, 4, 2, 5, 3, 6])
>>> concatenate((x,y)) # concatena dous vectores
array([ 1, 20,  3, 30,  1,  2,  3,  4,  5,  6])
>>> y=ones([2,3]) # creo matriz de uns

```



```

>>> y
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> z=zeros([2,3]) #creo matriz de ceros
>>> hstack((y,z)) # aplilo arrays dimensionais horizontalmente
array([[ 1.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.,  0.]])
>>> vstack((y,z)) # aplilo arrays dimensionais verticalmente
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> x
array([ 1, 20,  3, 30])
>>> x[0] # accedo o primeiro elemento do vector x
1
>>> x[1:2] # accedo a un rango de valores do vector
array([20])
>>> x[6] # erro por acceder a unha posicion que non existe
>>> y
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> y[0,1] # aceder o elemento da fila 0 e columna 1 da matriz y
1.0
>>> y=array([[1,2,3,4],[5,3,4,2]])
>>> y
array([[1, 2, 3, 4],
       [5, 3, 4, 2]])
>>> y[0,:] # devolve a fila 0 de y
array([1, 2, 3, 4])
>>> y[:,2] # devolve a columna 2 de y
array([3, 4])
>>> y[0:2, 1:3] # acceder a subrangos de matriz
array([[2, 3],
       [3, 4]])
>>> y.reshape(4,2) # modifica a dimensions da matriz
array([[1, 2],
       [3, 4],
       [5, 3],
       [4, 2]])
>>> xx=array(x, 'float') # modifico o tipo dun vector
>>> xx
array([ 1., 20.,  3., 30.])
>>> xx/2 # division vectorial
array([ 0.5, 10. ,  1.5, 15. ])
>>> x-4
array([-3, 16, -1, 26])
# operaciones aritmeticas vectoriales
>>> a=array([1,2,3,4])
>>> a
array([1, 2, 3, 4])
>>> xx
array([ 1., 20.,  3., 30.])
>>> a*xx # multiplica elemento a elemento dous vectores
array([ 1., 40.,  9., 120.])
# operaciones relacionais vectoriales
>>> x

```

```

array([ 1, 20,  3, 30])
>>> x > 10
array([False,  True, False,  True], dtype=bool)
>>> x >= 3
array([False,  True,  True,  True], dtype=bool)
>>> sin(x)
array([ 0.84147098,  0.91294525,  0.14112001, -0.98803162])
>>> round(sin(x)) # devuelve erro porque round() so se aplica a escalares
>>> round_(sin(x))
array([ 1.,  1.,  0., -1.])
>>> exp(x)
array([ 2.71828183e+00,  4.85165195e+08,  2.00855369e+01,
        1.06864746e+13])
>>> xf=x.astype('float') # modifiko tipo a un vector
>>> xf
array([ 1., 20.,  3., 30.])
>>> xf.dtype
dtype('float64')
>>> xf=float64(x) # modifiko tipo a un vector
array([ 1., 20.,  3., 30.])
>>> x
array([ 1, 20,  3, 30])
>>> sort(x) # ordear os elementos do vector de menor a maior
array([ 1,  3, 20, 30])
>>> argsort(x) # devuelve as posiciones
array([0, 2, 1, 3])
>>> a=array([[1,2,3],[4,5,6],[7,8,9]])
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a%2==1 # True en elementos impares
array([[ True, False,  True],
       [False,  True, False],
       [ True, False,  True]], dtype=bool)
>>> diagonal(a) # a.diagonal() fai o mesmo
array([1, 5, 9])
>>> a.prod(0) #producto elementos por columnas
array([28, 80,  0])
>>> a.sum(1) # suma dos elementos de a por filas
array([ 6,  9, 24])
>>> all(a)
True
>>> a[1,2]==0
>>> a
array([[1, 2, 3],
       [4, 5, 0],
       [7, 8, 9]])
>>> all(a)
False
>>> all(a,0) # all por columnas
array([ True,  True, False], dtype=bool)
>>> all(a,1) # all por filas
array([ True, False,  True], dtype=bool)
>>> all(a>=0) #True se todos los elementos son positivos
True
>>> any(a%2==0) #True se ten elementos pares

```

```

True
>>> from matplotlib.mlab import *
>>> find(a%2==0) #indices de elementos pares: require matplotlib.mlab
array([1, 3, 5, 7])
>>> a[a%2==0] #valores dos elementos pares
array([2, 4, 0, 8])
>>> extract(a%3==0,a) # elementos multiplos de tres
array([3, 0, 9])
>>> b=array([[1,-2,3],[-4,5,-6],[7,-8,9]])
>>> b
array([[ 1, -2,  3],
       [-4,  5, -6],
       [ 7, -8,  9]])
>>> allclose(a,b)
False
>>> array_equal(a,b) # True se son a e b son iguais
False
>>> greater(a,b)
array([[False,  True, False],
       [ True, False,  True],
       [False,  True, False]], dtype=bool)
>>> less(b,a)
array([[False,  True, False],
       [ True, False,  True],
       [False,  True, False]], dtype=bool)
>>> a==b # True onde a[i,j]==b[i,j]
array([[ True, False,  True],
       [False,  True, False],
       [ True, False,  True]], dtype=bool)
>>> equal(a,b) # igual que a==b
array([[ True, False,  True],
       [False,  True, False],
       [ True, False,  True]], dtype=bool)
>>> not_equal(a,b) # True onde a[i,j]!=b[i,j]
array([[False,  True, False],
       [ True, False,  True],
       [False,  True, False]], dtype=bool)
>>> logical_and(a,b) # a[i,j] AND b[i,j]
array([[ True,  True,  True],
       [ True,  True, False],
       [ True,  True,  True]], dtype=bool)
>>> logical_or([1,0,2,-1,1],[0,0,1,0,-1])
array([ True, False,  True,  True,  True], dtype=bool)
>>> logical_not([0,0,1,0,-1])
array([ True,  True, False,  True, False], dtype=bool)
>>> where(a>2,a,-1) # valor de a onde a>2 e no resto -1
array([[ -1, -1,  3],
       [ 4,  5, -1],
       [ 7,  8,  9]])
>>> where(logical_and(a>=2,a<=6),-1,0)
array([[ 0, -1, -1],
       [-1, -1,  0],
       [ 0,  0,  0]])
>>> argmax(a)
8
>>> argmax(a,0) # indice do valor maximo por columnas
array([2, 2, 2])

```

```

>>> argmax(a,1) # o mesmo por filas
array([2, 1, 2])
>>> # Estadística básica
>>> amin(a) # mínimo de a
-1
>>> amin(a, 0) # mínimo por columnas
array([-1, -1, -1])
>>> amin(a, 1) # mínimo por filas
array([-1, -1, 7])
>>> ptp(a) # rango de valores máximo - mínimo de a
10
>>> mean(a) # valor medio de a
3.6666666666666665
>>> median(a) # mediana de a
4.0
>>> std(a) # desviación típica dos elementos de a
3.7416573867739413

```

2. Entrada e saída a arquivos de texto usando NumPy.

```

>>> from numpy import *
>>> x=array([[1,2,3,4],[5,6,7,8],[3,4,5,6],[5,4,3,2]])
>>> savetxt('datos.txt', x) # garda array x en datos.txt
>>> savetxt('datos.txt', x, '%i') # garda array x en datos.txt como enteros
>>> y=loadtxt('datos.txt') # por defecto le datos tipo float
>>> y
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 3.,  4.,  5.,  6.],
       [ 5.,  4.,  3.,  2.]])
>>> y.dtype
dtype('float64')
>>> x[1,2]=400 # modifíco x[1,2]
>>> x
array([[ 1,  2,  3,  4],
       [ 5,  6, 400, 8],
       [ 3,  4,  5,  6],
       [ 5,  4,  3,  2]])
>>> savetxt('datos.txt', x, '%10i') #formato entero 10 cifras
>>> y=loadtxt('datos.txt')
>>> y
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., 400., 8.],
       [ 3.,  4.,  5.,  6.],
       [ 5.,  4.,  3.,  2.]])

```

3. Xerar números aleatorios. Necesítase cargar o submódulo random do paquete numpy (aínda que o numpy xa estea cargado).

```

>>> from numpy.random import * # cargar en memoria modulo random
>>> rand() # devuelve un numero aleatorio en [0,1)
0.3130237566728453
>>> rand(2,3) # matriz de numeros aleatorios
array([[ 0.60076503,  0.98189915,  0.15762115],
       [ 0.02760126,  0.33241119,  0.30466891]])
>>> random([2,3]) # matriz de numeros aleatorios

```

```

array([[ 0.46284244,  0.40791779,  0.53317039],
       [ 0.8805399 ,  0.5118586 ,  0.510961  ]])
>>> x=round_(10*rand(10)) # vector con numeros aleatorios enteireos en [0,10]
>>> x
array([ 8.,  5.,  1.,  7., 10.,  2.,  2.,  9.,  7.,  8.])
>>> shuffle(x) # baralla os elementos de x
>>> x
array([ 1., 10.,  7.,  5.,  2.,  8.,  9.,  2.,  8.,  7.])
>>> permutation(3) # realiza permutacions sobre range(3)
array([0, 2, 1])
>>> permutation([4,2,7,5])
array([5, 7, 4, 2])

```

Para a representación gráfica en Python utilízase o paquete externo **Matplotlib**. Na galería da documentación en liña (<http://matplotlib.org/gallery.html>) podes ver con exemplos todas as posibles gráficos que se poden facer con esta librería.

1. **Representación gráfica básica.** Para a representación gráfica podes: a) dende o intérprete python utilizar o subpaquete pylab (poñendo `from pylab import *`) do paquete matplotlib (non está por defecto no intérprete de Python e hai que instalalo); ou b) executar o intérprete ipython coa opción `-pylab`: `ipython --pylab`. Aquí tes un exemplo de gráfico de liñas en 2D con python:

```

>>> from pylab import *
>>> x = linspace(-pi,pi,100)
>>> y = sin(2*pi*x)
>>> plot(x, y, linewidth=1.0)
>>> xlabel('Tempo (s)')
>>> ylabel('Tension (mV)')
>>> title('Tension frente a tempo')
>>> grid(True)
>>> show(False) #crea a figura e retorna a consola

```

É moito mellor usar a consola interactiva `ipython --pylab`, porque non hai que facer o `show(False)` e ademáis calquer cambio que fagas xa se mostra na ventá do gráfico:

```

>>> # crea un vector con 256 elementos entre -pi e pi
>>> X = linspace(-pi, pi, 256,endpoint=True)
>>> C,S = cos(X), sin(X) # calcula o seno e coseno dos vectores anteriores
>>> plot(X,C) # grafico de linhas do vector C en funcion de X
>>> plot(X,S) # engade o grafico de linhas do vector S
>>> # grafico con linha continua vermella con ancho 2 pixels
>>> plot(X, C, color="red", linewidth=2.0, linestyle="-")
>>> figure(3) # crea unha nova figura
>>> # grafico do coseno cambiando color, linha, simbolo
>>> plot(X, S, color="green", linewidth=1.0, linestyle="--", marker="*")
>>> plot(X, C, color="red", linewidth=2.0, linestyle="-")
>>> xlim(X.min(),X.max()) # establece os limites no eixo x
>>> ylim(-1.0,1.0) # establece os limites no eixo y
>>> yticks([-1, 0, +1]) # ticks no eixo y
>>> xticks([-pi, -pi/2, 0, pi/2, np.pi]) # ticks no eixo x
>>> # pon etiquetas os ticks
>>> xticks([-pi, -pi/2, 0, np.pi/2, np.pi],
... [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
>>> clf() # limpa o contido dunha venta de graficos
>>> plot(X, C, color="red", linewidth=2.0, linestyle="-", label="coseno")
>>> plot(X, S, color="green", linewidth=1.0, linestyle=":", label="seno")
>>> legend(loc='upper left') # pon unha lenda no grafico para cada curva na posicion indicada

```

2. Representación gráfica. Gráficos de liñas, histogramas, gráficos de barras e gráficos de erros.

```
>>> x = [3,5,8,2,3,6,4,7,3,6,9,4,3,2,6] # lista de numeros
>>> hist(x) # histograma de x
>>> clf() # limpa o contido da figura
>>> hist? # podes ver a axuda da funcion hist no entorno interactivo
>>> x = linspace(0, 2*pi, 100) # vector con 100 elementos no intervalo [0,2pi]
>>> fill(x,sin(x), 'r') # grafico de liñas recheo
>>> fill(x,sin(3*x), 'b', alpha=0.5) # outro grafico de liñas
>>> grid() # pon enreixado o grafico
>>> grid(False) # quita o enreixado do grafico
>>> hold(False) # desactiva o mantemento do grafico
>>> # graficos de barras
>>> x = range(6) # lista con 6 elementos
>>> y = [3,4,2,6,4,8] # lista con 6 elementos
>>> bar(x,y) # grafico de barras vertical
>>> barh(x,y) # grafico de barras horizontal
>>> erroy = [0.2,0.3,0.5,0.1,0.2, 0.7] # lista de erros en y
>>> errox=[0.1,0.1,0.3,0.3,0.4, 0.1] # lista de erros en x
>>> errorbar(x, y, errox, erroy) # grafico de erros
>>> # grafico de erros establecendo cores, simbolos e grosor de linha
>>> errorbar(x, y,erroy, errox, fmt='go', ecolor='blue', elinewidth=1.25)
```

Traballo a desenvolver pol@ alumn@

Realiza por orde os seguintes exercicios:

1. Define unha lista **z** cos seguintes elementos: 1, 2, 3, 4, 3, 2, 5, 6, 7, 8, 4, 2, 1
2. Crea un vector (**array**) **x** a partires da lista **z**.
3. Sobre o array ou vector **x** calcula a seguinte información:
 - a) O número de dimensións e os seus valores.
 - b) O número de elementos.
 - c) O tipo de datos almacenados en **x**.
4. Crea unha matriz **a** de dimensións 3×4 a partir do vector **x**.
5. Realiza sobre a matriz **a** o cálculo de número de dimensións e os seus valores, número de elementos e tipo de datos dos seus elementos.
6. Crea un vector (**array**) **y** con 12 elementos equiespaciados no intervalo $[1, 5]$.
7. Crea un vector **p** con elementos no intervalo $[1, 5]$, utilizando un espaciado entre elemntos de 0.25.
8. Asigna o valor 20 ó cuarto elemento de **p**.
9. Asigna o valor 30 ó elemento que se sitúa na segunda fila e primeira columna da matriz **a**.
10. Copia o vector **x** nos vectores **xr** e **xv** utilizando unha copia por referencia e por valor respectivamente. Modifica un elemento en ambos vectores e describe que cambios ocorriron no vector **x**.
11. Converte o vector **y** nun vector de números enteiros utilizando varias opcións.
12. Inserta o valor 15 no vector **x** ó principio, final e na sexta posición do vector **x**.
13. Borra o cuarto elemento do vector **x** e asígnao o vector **v**.
14. Converte a matriz **a** nun vector, realizando a conversión por filas e columnas.

15. Une os vectores \mathbf{x} e \mathbf{y} no vector \mathbf{w} .
16. Define unha matriz de ceros, chamada \mathbf{b} , de orde 3×4 . Realiza a unión da matriz \mathbf{a} e \mathbf{b} por filas e por columnas.
17. Crea a lista `l1` cos elementos 1, 2 e 3. Engade esta lista como a primeira columna da matriz \mathbf{a} .
18. Suma o valor 1 a todos os elementos da matriz \mathbf{a} .
19. Divide por 10 todos os elementos do vector \mathbf{x} .
20. A partir da lista `[3.5, -2.3, -0.7, 4.2, 1.6]`, obtén vectores cos enteiros máis próximos por exceso e por defecto, a parte enteira e o enteiro máis próximo. Tamén a valor medio e a varianza.
21. Sea \mathbf{v} un vector definido coa seguinte lista de números `[4, 1, 6, 1, 2, 8, 1, 6]`, realiza as seguintes operacións:
 - a) Ordea o vector \mathbf{v} de menor a maior e obtén as posicións dos números ordeados.
 - b) Calcula o valor mínimo de \mathbf{v} e as posicións onde se atopa o mínimo.
 - c) Calcula os valores de \mathbf{v} que son superiores a 2, e as posicións deses valores no vector.
 - d) Devolve un vector cos elementos x_i de \mathbf{v} que cumpren a condición $0 < x_i < 5$.
 - e) Calcula un vector, das mesmas dimensións de \mathbf{v} , cos elementos de \mathbf{v} comprendidos no intervalo `[-1, 3]` e 5 no resto.
22. Converte o vector \mathbf{v} anterior nunha matriz \mathbf{m} de dimensións 4×2 e realiza as seguintes operacións:
 - a) Calcula un vector cos valores de \mathbf{a} que son superiores a 2.
 - b) Devolve un vector cos elementos x_i de \mathbf{a} que cumpren a condición $0 < x_i < 5$.
 - c) Calcula unha matriz \mathbf{c} cos elementos de \mathbf{a} comprendidos no intervalo `[-1, 3]` e 5 no resto.
 - d) Guarda a matriz \mathbf{c} no arquivo de texto `datos.txt`.
 - e) Le o contido arquivo `datos.txt` e almacénalo na variable \mathbf{d} .
23. Crea unha matriz \mathbf{a} de orde 3×4 con números enteiros no intervalo `[5,10]`.
24. Crea un vector \mathbf{x} de lonxitude 10 con elementos aleatorios en `[0,10)`.
25. Representa gráficamente $y = x^2 e^{-x/10} \sin 10x$ con $0 \leq x \leq 10$, rede (Grid), e títulos de eixos `Velocidade (m/s)` e `Aceleración (m/s2)`.
26. Representa a curva en forma paramétrica $x(t) = t \cos 10t, y(t) = t^2 \sin 10t$, con $t = 0, \dots, 10$ e 1000 puntos.
27. Xera un vector de números aleatorios co comando `rand` no paquete `numpy.random` e mostra un histograma con 50 intervalos.
28. Representa o vector $x_i = \arctan\left(\frac{i^2 - 1}{i^2 + 1}\right)$, con $i = -10, -9, \dots, 10$, e con barras de erro de cor vermella con alturas dadas por $e_i = \sin x_i / 10$.

Sesión 3 : Primeros programas

Traballo en clase

En Python tamén podemos executar programas (os programas de Python son arquivos de texto con extensión `.py`). Para crear un programa podes utilizar:

- SPYDER: utilizar o editor que trae incorporado. A aparencia de SPYDER é igual en calquer sistema operativo. Opción recomendada se se usa Windows ou Mac.
- Se usas Linux sen SPYDER, usa calquer editor de texto (preferiblemente que teña a sintaxe de Python resaltada como `gedit` ou `kate`).

1. **Programa básico con entrada e saída de datos.** Escribe o programa nun editor de texto (por exemplo `gedit`) e gárdao cun nome e extensión `.py`. Para os nomes dos arquivos utiliza só caracteres alfanuméricos sen espazos en branco nen símbolos raros. Escribe un programa chamado `radio.py` que pida ó usuario que introduza un radio e calcula o perímetro da circunferencia, a área do círculo e o volume da esfera.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import pi
radio = float(raw_input("Introduce o radio en metros: "))
perimetro = 2 * pi * radio
area = pi * radio * radio
volume = 4 * pi * radio ** 3/3
print "Perimetro ", perimetro, "m"
print "Area ", area, "m^2"
print "Volume ", volume, "m^3"
```

Para executar o programa, unha vez escrito, tes varias alternativas:

- a) Windows: desde un terminal co comando `python.exe radio.py`
- b) Linux:
 - 1) Dende o intérprete de Python tecleando `run radio.py` ou `execfile('radio.py')`.
 - 2) Dende a terminal no directorio do programa, executando `python radio.py`.
 - 3) Se lle das permisos de execución o arquivo `radio.py` co comando `chmod u+x radio.py`, podes executalo dende a terminal coa orde `./radio.py`. Para que esta alternativa funcione será necesario incluír a primeira liña no arquivo `radio.py` que indica o directorio onde está o intérprete de python. Nos sistemas linux normalmente está no directorio `/usr/bin`.
- c) SPYDER: independentemente do sistema operativo.
 - 1) Utilizando o botón `run`.
 - 2) Como en Linux desde a ventá do interprete.

Nós executaremos normalmente dende o intérprete porque así podemos ver os valores das variábeis en caso de erro, o que facilita a depuración. Os valores das variables do programa tamén se poden observar en SPYDER na ventá de exploración de variables. De todos modos, cando o programa xa está correcto, as execucións desde o terminal son máis rápidas e cómodas.

2. **Programa básico con sentenzas de selección.** Escribe un programa que pida ó usuario que introduza un número enteiro e devolve unha mensaxe informando se o número é par ou impar.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
x = int(raw_input("Introduce un numero enteiro: "))
resto = x % 2
```

```

if resto > 0:
    print x, "e impar"
else:
    print x, "e par"

```

Outra versión:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
x = int(input("Introduce un numero inteiro: "))
if x%2:
    print x, "e impar"
else:
    print x, "e par"

```

3. **Programa básico con sentenzas de iteración.** Escribe un programa que lea un número n por teclado e logo lea n números x_1, x_2, \dots, x_n e calcule a súa media e produto:

$$media = \frac{1}{n} \sum_{i=0}^{n-1} x_i, \quad produto = \prod_{i=1}^n x_i \quad (1)$$

Solución utilizando un bucle `while`:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
n = int(raw_input("Introduce numero de elementos: "))
i = 0
media = 0.0
produto = 1.0
while i < n:
    xi = float(raw_input("Introduce elemento: "))
    media = media + xi
    produto = produto * xi
    i = i + 1
media = media / n
print "Media=", media
print "Producto=", produto

```

Solución utilizando un bucle `for`:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
n = int(raw_input("Introduce numero de elementos: "))
media = 0.0
produto = 1.0
for i in range(n):
    xi = float(raw_input("Introduce elemento: "))
    media = media + xi
    produto = produto * xi
media = media / n
print "Media=", media
print "Producto=", produto

```

4. **Programa básico con argumentos en liña de comandos.** Este programa (que podes chamar `argumentos.py`) mostra por pantalla os argumentos cos que se executou o programa. Estos argumentos son as cadeas de caracteres que se escriben logo do nome do programa cando se executa dende a terminal de comandos, por exemplo:

```
python argumentos.py argumento1 10 5.3
```

ou

```
argumentos.py argumento1 10 5.3
```

(supoñendo que fixeches antes o `chmod u+x argumentos.py` para darlle permiso de execución). Se executas o programa dende `ipython`, podes executar `run argumentos.py argumento1 10 5.3`. Alternativamente:

```
import sys; sys.argv=['argumentos.py','argumento1',10,5.3]
execfile("argumentos.py")
```

O primeiro argumento que mostra sempre é o nome do programa. O programa debe mostrar as cadeas de caracteres `argumentos.py` e `argumento1`, e os números 10 e 5.3, ademais da súa suma.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from sys import *
n=len(argv)
print "Este programa ten", n, "argumentos en linha de comandos:"
for i in range(n):
    print "argumento", i+1, ":", argv[i]
n=int(argv[2]) # convirte de cadea a enteiro
x=float(argv[3]) # convirte de cadea a real
print n+x
```

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 $ax + by + cz + d = 0$. A distancia entre \mathbf{v} e π pódese calcular como:

$$D(\mathbf{v}, \pi) = \frac{|ax_0 + by_0 + cz_0 + d|}{\|\mathbf{w}\|} \quad (2)$$

O valor absoluto é coa función `abs(...)`; ademais, $\|\mathbf{w}\| = \sqrt{a^2 + b^2 + c^2}$. O programa debe ler por teclado os valores $a, b, c, d, x_0, y_0, z_0$.

2. **Ecuación de primeiro grao.** Escribe un programa que lea os coeficientes reais dunha ecuación de primeiro grao $ax + b = 0$ e calcule a solución segundo a fórmula:

$$x = \frac{-b}{a} \quad (3)$$

Debes ter en conta que so existe solución única cando a é distinto de 0 (que debes controlar cunha sentenza de selección). Se tes dificultades podes consultar a solución detallada no libro: *Introdución a programación en Python* de Andrés Marcial e Isabel Gracia.

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

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (4)$$

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 1º grao con solución $x = -c/b$. Se $a = b = c = 0$ todo $x \in \mathbf{R}$ é solución, e se $a = b = 0, c \neq 0$ non hai solución. Para comprobar que o programa funciona correctamente, proba os seguintes exemplos:

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	∞	$x = \mathbf{R}$
0	0	1	0	$x = \emptyset$

Se tes dificultades podes consultar a solución detallada no libro: *Introdución a programación en Python* de Andrés Marcial e Isabel Gracia.

Sesión 4 : Listas é sentenzas de iteración

Traballo en clase

1. Escribe un programa que lea do teclado dous vectores \mathbf{v} e \mathbf{w} n -dimensionais con valores reais e calcule o produto escalar (ou interior) de ambos dado pola ecuación:

$$\mathbf{vw} = \sum_{i=0}^{n-1} v_i w_i \quad (5)$$

Existen varias alternativas para a súa solución. Versión creando listas baleiras e engadindo elementos a lista en cada iteración:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
n = int(raw_input("Numero de elementos= "))
v = []; w = [] # crea listas baleiras
pescalar = 0
# ler listas do teclado e calculo
for i in range(n):
    vi = float(raw_input("Introduce elemento de v: "))
    wi = float(raw_input("Introduce elemento de w: "))
    v = v + [vi]
    w = w + [wi]
    pescalar = pescalar + v[i] * w[i]
print "Vector v=", v
print "Vector w=", w
print "Producto escalar=", pescalar
```

Versión creando unha lista do tamaño que necesitamos.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
n = int(raw_input("Numero de elementos= "))
v = [0] * n # crea lista de n elementos inicializada con 0
w = [0] * n
pescalar = 0
# ler listas do teclado e calculo
for i in range(n):
    v[i] = float(raw_input("Introduce elemento de v: "))
    w[i] = float(raw_input("Introduce elemento de w: "))
    pescalar = pescalar + v[i] * w[i]
print "Vector v=", v
print "Vector w=", w
print "Producto escalar=", pescalar
```

Utilizando a función `input()` no canto de `raw_input()`. Ten o inconveniente que non comproba os datos que introducimos pero para exercicios de proba é moi útil xa que permite ler unha lista de números nunha soa sentenza. Na execución do programa, hai que introducir a lista de números entre corchetes e separados os números por comas, e. d. `[1, 2, 6, 4]`.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
v = input("Introduce vector v: ")
w = input("Introduce vector w: ")
```

```

print "Vector v=", v
print "Vector w=", w
nv = len(v); nw = len(w)
if nv == nw:
    pescalar = 0
    for i in range(nv):
        pescalar = pescalar + v[i] * w[i]
    print "Producto escalar=", pescalar
else:
    print "Vectores de distintas dimensions"

```

Tamén se pode usar a función `dot(v,w)` no paquete `numpy`:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import dot
v = input("Introduce vector v: ")
w = input("Introduce vector w: ")
print "Vector v=", v
print "Vector w=", w
if len(v) == len(w):
    print "Producto escalar=", dot(v,w)
else:
    print "Vectores de distintas dimensions"

```

2. **Sumatorio dobre.** Escribe un programa que lea por teclado dous vectores n -dimensionais \mathbf{v} e \mathbf{w} e calcule:

$$\sum_{i=0}^{n-1} \sum_{j=0}^i v_i w_j \quad (6)$$

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
v = input("Introduce vector v: ")
w = input("Introduce vector w: ")
print "Vector v=", v
print "Vector w=", w
nv = len(v); nw = len(w)
if nv == nw:
    resul = 0
    for i in range(nv):
        aux = 0
        for j in range(i+1):
            aux = aux + w[j]
        print "aux= ", aux
        resul = resul + v[i] * aux
    print "Resultado=", resul
else:
    print "Vectores de distintas dimensions"

```

Unha alternativa vectorizada sería:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
v = input("Introduce vector v: ")
w = input("Introduce vector w: ")

```

```

nv = len(v); nw = len(w)
if nv == nw:
    resul = 0
    for i in range(nv):
        resul = resul + v[i] * sum(w[:i+1])
    print "Resultado=", resul
else:
    print "Vectores de distintas dimensions"

```

3. **Representación gráfica dende un programa.** Escribe un programa que calcule a posición $x(t)$, velocidade $v(t)$ e aceleración $a(t)$ dun móvil en movemento armónico, dado por:

$$x(t) = b \sin(\omega t + \theta) \quad (7)$$

$$v(t) = b\omega \cos(\omega t + \theta) \quad (8)$$

$$a(t) = -b\omega^2 \sin(\omega t + \theta) \quad (9)$$

sendo $\omega = \pi$ radiáns/s, $\theta = \pi/2$ radiáns, $b = 2.5 \text{ m/s}^2$ para tempos $0 \leq t \leq 100$ s. separados 1 s. entre si. Almacena os resultados en listas e representa gráficamente todas as listas. Garda a gráfica no arquivo `figuraarmonico.png`. Tes que obter a gráfica da figura 1

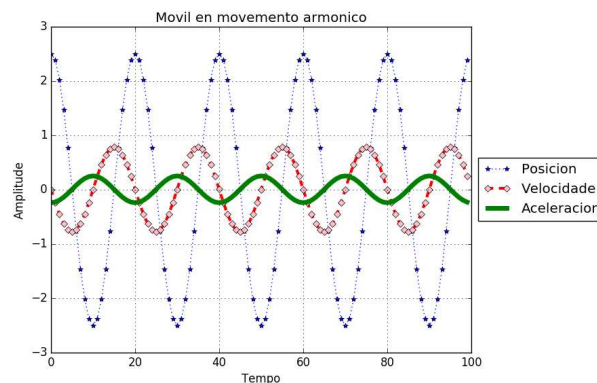


Figura 1: Movimiento armónico simple.

Solución utilizando listas de python.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import *
from matplotlib.pyplot import *
THETA = pi / 2 # constante
A = 2.5 # constante
OMEGA = pi / 10
n = 100
x = [0.0] * n; v = [0.0] * n; a = [0.0] * n
t = range(100)
for i in t:
    x[i] = A * sin(OMEGA*i + THETA)
    v[i] = A * OMEGA * cos(OMEGA*i + THETA)
    a[i] = - A * OMEGA * OMEGA * sin(OMEGA*i + THETA)
clf()
plot(t, x, 'b*:', label="Posicion")
plot(t, v, color="red", linestyle="--", linewidth=3.0, markersize=5, \
markerfacecolor="pink", marker="D", label="Velocidade")

```

```

plot(t, a, color="green", linestyle="-", linewidth=5.0, \
label="Aceleracion")
xlabel("Tempo")
ylabel("Amplitude")
title("Movil en movimiento armonico")
grid()
# para lenda dentro da caixa de debuxado
# legend(loc='upper center', shadow=True)
# para situar a lenda fora da caixa do grafico
lgd=legend(loc='center left', bbox_to_anchor=(1, 0.5))
savefig('figuraarmonico.png', bbox_extra_artists=(lgd,), \
bbox_inches='tight')
show() # visualiza a figura

```

A barra \ que hai en algunhas liñas significa que a liña de código non rematou. A 'u' diante dalgunhas cadeas de caracteres é para que se consideren como cadeas unicode e podan ter tildes (no texto non as levan por problemas co Latex, pero no programa poden levalas).

Se utilizas o módulo `numpy` simplifícase a:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import *
from matplotlib.pyplot import *
from numpy import *
THETA = pi / 2 # constante
A = 2.5 # constante
OMEGA = pi / 10
t=arange(1,101,1)
x = A * sin(OMEGA*t + THETA)
v = A * OMEGA * cos(OMEGA*t + THETA)
a = - A * OMEGA * OMEGA * sin(OMEGA*t + THETA)
clf()
plot(t, x, 'b*:', label="Posicion")
plot(t, v, color="red", linestyle="--", linewidth=3.0, markersize=5, \
markerfacecolor="pink", marker="D", label="Velocidade")
plot(t, a, color="green", linestyle="-", linewidth=5.0, \
label="Aceleracion")
xlabel("Tempo")
ylabel("Amplitude")
title("Movil en movimiento armonico")
grid()
legend(loc='upper center', shadow=True)
show() # visualiza a figura

```

Se queres poñelo en tres gráficas distintas, sería co `subplot()`:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import *
from matplotlib.pyplot import *
from numpy import *
theta= pi/2; b=2.5; w=pi/10; n=100
t=arange(1,101,1)
x = A * sin(OMEGA*t + THETA)
v = A * OMEGA * cos(OMEGA*t + THETA)
a = - A * OMEGA * OMEGA * sin(OMEGA*t + THETA)
clf()

```

```

subplots_adjust(hspace=0.4)
subplot(311)
plot(t, x, 'b*:')
xlabel("Tempo (s)"); ylabel("Amplitude (m)"); grid(True)
subplot(312)
plot(t, v, color="red", linestyle="--", linewidth=3.0, markersize=5, \
markerfacecolor="pink", marker="D")
xlabel("Tempo (s)"); ylabel("Velocidade (m/s)"); grid(True)
subplot(313)
plot(t, a, color="green", linestyle="-", linewidth=5.0)
xlabel("Tempo (s)"); ylabel("Aceleracion (m/s^2)"); grid(True)

```

4. **Medida do tempo de execución dun programa:** escribe un programa que lea por teclado un número enteiro n e defina unha lista x de n elementos con valores $x_i = i^2, i = 1, \dots, n$, medindo o tempo de execución para valores altos de n . Esta lista x pódese crear das seguintes formas:

- Usando un bucle `while` e engadindo elementos á lista con `x=x+[i*i]`.
- Usando un bucle `while` e engadindo elementos á lista co método `append()`.
- Usando `while` e creando a lista ao comezo do bucle con `x=[0]*n`.
- Usando un bucle `for` e creando a lista do mesmo xeito.

Usa, por exemplo, $n = 50000$ iteracións. Os valores concretos de tempo dependen de cada ordenador e diferentes execucións, pero, seguramente, a primeira opción é a máis lenta e a última a máis rápida.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from time import *
n = int(raw_input("Numero iteracions: "))
#version 1 -----
inicio = clock(); x=[]; i=0
while i < n:
    x=x + [i*i]
    i=i+1
fin = clock()
print "version 1: tempo: %f" % (fin - inicio)
#version 2 -----
inicio = clock(); x=[]; i=0
while i < n:
    x.append(i*i)
    i=i+1
fin = clock()
print "version 2: tempo: %f" % (fin - inicio)
#version 3 -----
inicio = clock(); x=[0]*n; i=0
while i < n:
    x[i]=[i*i]
    i=i+1
fin = clock()
print "version 3: tempo: %f" % (fin - inicio)
#version 4 -----
inicio = clock(); x=[0]*n
for i in range(n):
    x[i]=[i*i]
fin = clock()
print "version 4: tempo: %f" % (fin - inicio)

```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa que lea número enteiro n e un vector \mathbf{v} de dimensión n . O programa debe calcular a norma (módulo) de \mathbf{v} :

$$\|\mathbf{v}\| = \sqrt{\sum_{i=0}^{n-1} v_i^2} \quad (10)$$

2. Escribe un programa que lea dous vectores \mathbf{x} e \mathbf{y} de dimensión n por teclado e calculen a súa distancia $\|\mathbf{x} - \mathbf{y}\|$ definida como:

$$\|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=0}^{n-1} (x_i - y_i)^2} \quad (11)$$

3. Escribe un programa que lea por teclado un número enteiro n e un vector \mathbf{v} de dimensión n , e calcule o vector transformado \mathbf{w} , tamén de dimensión n , definido por:

$$w_i = \sum_{j=0}^{i-1} v_j, \quad i = 0, \dots, n-1 \quad (12)$$

Proba con $n = 5$, $\mathbf{v} = (1, 2, 3, 4, 5)$ e tes que obter $\mathbf{w} = (1, 3, 6, 10, 15)$.

4. Escribe un programa que simule a altura $h(t)$, velocidade $v(t)$ e aceleración $a(t)$ do salto feito por Felix Baumgartner dende unha altura de 39045m, a unha velocidade superior á do son (detalles en este [enlace](#)). Nesta web supoñen gravidade constante $g=9.8\text{m/s}^2$, $m=120\text{ kg}$ (paracaidista e traxe), unha forza de rozamento contra o aire $F_r(t)$, altura $h(t + \Delta t)$, velocidade $v(t + \Delta t)$ e aceleración $a(t + \Delta t)$ dados polas ecuacións seguintes (nas que $\Delta t = 1\text{ seg.}$):

$$k(t) = \frac{AC(t)\rho(t)}{2} \quad (13)$$

$$F_r(t) = -k(t)v(t)^2 \quad (14)$$

$$a(t+1) = g - \frac{F_r(t)}{m} \quad (15)$$

$$v(t+1) = v(t) + \frac{a(t+1) + a(t)}{2} \quad (16)$$

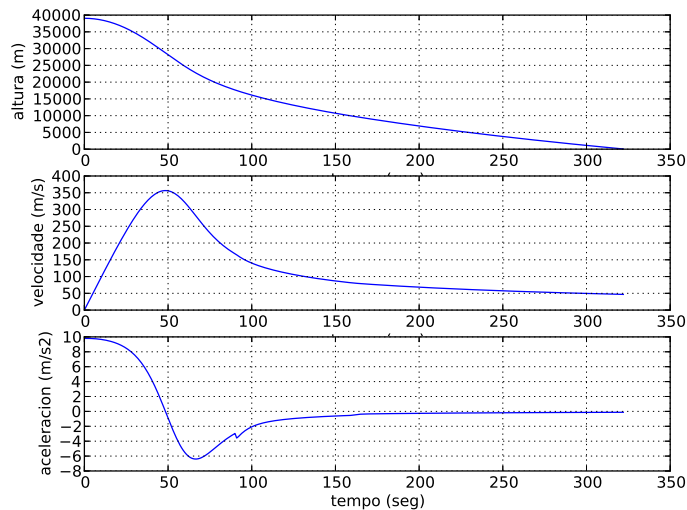
$$h(t+1) = h(t) - \frac{v(t+1) + v(t)}{2} \quad (17)$$

O parámetro $C(t)$ toma o valor $C=0.69$ para $t < 90\text{ seg.}$ e $C=0.73$ máis adiante. Debes supor a seguinte densidade $\rho[h(t)]$ do aire:

$$\rho[h(t)] = \begin{cases} Ae^{-h(t)/Z_1} & h(t) < h_0 \\ B \exp\left(-\frac{h(t) - h_0}{Z_2}\right) & h(t) \geq h_0 \end{cases}$$

Onde $A=1.225\text{ kg/m}^3$, $B=0.4136\text{ kg/m}^3$, $Z_1=9208\text{ m}$, $Z_2=6494\text{ m}$ e $h_0=10000\text{m}$. Tes que usar $h(0)=39045\text{m}$, $v(0)=0\text{ m/s}$ e $a(0) = g\text{ m/s}^2$. O programa debe representar $h(t), v(t), a(t)$ para $t = 1, 2, 3, \dots\text{ seg.}$ mentres que $h > 0$.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from matplotlib.pyplot import *
A=1.225; B= 0.4136; Z1=9208; Z2=6494; h0=10000; g=9.8; m=120
```



```

n=1000; h=[0]*n; v=[0]*n; a=[0]*n; Fr=[0]*n;
h[0]=39045; v[0]=0; a[0]=g; t=0
for t in range(n):
    if h[t] < h0:
        rho=A*exp(-h[t]/Z1)
    else:
        rho=B*exp(-(h[t]-h0)/Z2)
    if t<90:
        C=0.69
    else:
        C=0.73
    k=rho*A*C/2
    Fr[t]=k*v[t]**2
    a[t+1]=g-Fr[t]/m
    v[t+1]=v[t]+(a[t+1]+a[t])/2
    h[t+1]=h[t] - (v[t+1] + v[t])/2
    if h[t+1] < 0:
        break
    t=t+1
u=range(t)
clf()
subplot(3,1,1)
plot(u,h[0:t],'b-'); xlabel("tempo (seg)");
ylabel("altura (m)"); grid(); show(False)
subplot(3,1,2)
plot(u,v[0:t],'b-'); xlabel("tempo (seg)");
ylabel("velocidade (m/s)"); grid(); show(False)
subplot(3,1,3)
plot(u,a[0:t],'b-'); xlabel("tempo (seg)");
ylabel("aceleracion (m/s2)"); grid(); show(False)

```

5. Realiza un indicador de progreso da porcentaxe de programa que se executou.

```

from sys import *
n=1000000
for i in range(n):
    print '%5.1f%%\r' % (100.*i/n),

```

```
stdout.flush()
```

Sesión 5 : Matrices. Interpolación. Axuste a funcións

Traballo en clase

1. Dada a matriz:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & -1 & 0 \\ 1 & 0 & 2 \end{bmatrix}$$

calcula, usando NumPy, o seu determinante, inversa, autovalores e autovectores.

```
>>> from numpy.linalg import *
>>> a=array([[1,2,3],[2,-1,0],[1,0,2]])
>>> det(a)
-7.0000000000000009
>>> inv(a)
array([[ 0.28571429,  0.57142857, -0.42857143],
       [ 0.57142857,  0.14285714, -0.85714286],
       [-0.14285714, -0.28571429,  0.71428571]])
>>> eig(a)
(array([ 3.6624648 , -2.44437482,  0.78191002]), array([[-0.80430281, -0.58045714, -0.58383444],
              [-0.34501185,  0.80374863, -0.6552906 ],
              [-0.48380141,  0.1306049 ,  0.47930321]]))
```

2. **Operacións con matrices.** Escribe un programa que lea por teclado unha matriz cadrada a (comproba que sexa cadrada) e calcule:

- a) A suma dos elementos de a dados pola expresión:

$$sa = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} \quad (18)$$

- b) A traza da matriz a dada por:

$$tr = \sum_{i=0}^{n-1} a_{ii} \quad (19)$$

- c) Sumas dos elementos do triángulo superior dados pola expresión:

$$sts = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} a_{ij} \quad (20)$$

- d) Determine se a matriz é simétrica: $a_{ij} = a_{ji}, i, j = 0, \dots, n-1$.

Visualiza os cálculos na pantalla. Lee a matriz desde un arquivo.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
# para ler dun arquivo a matriz
#nome=raw_input("Introduce nome arquivo: ");
#a=loadtxt(nome);
# para ler do teclado a matriz
a=array(input('Introduce matriz como unha lista: '))
```

```

# Exemplo: [[1, 2, 3], [3, 2, 1], [3, 8, 1]]
nf, nc = a.shape
print "Matriz a: "
print a
if nf == nc:
    print "Suma elementos de a : ", sum(a)
    print "Traza da matriz ", trace(a)
    print "Suma do triangulo superior: ", sum(triu(a))-trace(a)
    print "Suma do triangulo superior: ", sum(a-tril(a))
    print "Suma do triangulo superior: ", sum(triu(a)-diag(diag(a)))
    if all(a == a.T):
        print "Matriz simetrica"
    else:
        print "Matriz non simetrica"
else:
    print "Matriz non cadrada"

```

3. **Resolución de sistema de ecuacións lineares mediante Eliminación Gaussiana.** Escribe un programa que resolve un sistema de n ecuacións lineais con n incógnitas empregando o Método de Eliminación Gaussiana. Dado o sistema seguinte:

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= a_{1(n+1)} \\
 &\dots \\
 a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= a_{n(n+1)}
 \end{aligned}$$

O método de eliminación consiste en empregar as seguintes transformacións:

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

para transformar este sistema no seguinte:

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= a_{1(n+1)} \\
 a_{22}x_2 + \dots + a_{2n}x_n &= a_{2(n+1)} \\
 &\dots \\
 a_{nn}x_n &= a_{n(n+1)}
 \end{aligned} \tag{21}$$

onde os a_{ij} **non** son os iniciais. Concretamente, para cada incógnita $x_i, i = 1, \dots, n$, restamos as ecuacións $j = i+1, \dots, n$ menos a ecuación i multiplicada por $l = a_{ji}/a_{ii}$. Deste modo, o coeficiente da incógnita x_i na ecuación j ($j = i+1, \dots, n$) pasa a ser nulo. No sistema transformado podemos despexar directamente x_n , substituír na $(n-1)$ -ésima ecuación e despexar x_{n-1} e así sucesivamente ata calcula-las n incógnitas, mediante a fórmula.

$$x_i = \frac{1}{a_{ii}} \left(a_{i(n+1)} - \sum_{j=i+1}^n a_{ij}x_j \right) \quad i = n, \dots, 1 \tag{22}$$

Isto vale se $a_{ii} \neq 0, i = 1, \dots, n$, pero se $\exists i$ tal que $a_{ii} = 0$, faise o denominado “pivote”, consistente en intercambiar a ecuación i -ésima por aquela ecuación p que ten o coeficiente a_{pi} máximo. Proba co seguinte sistema:

$$\begin{aligned}
 2y - z &= 1 \\
 x - y + z &= 0 \\
 2x - z &= 3
 \end{aligned}$$

A solución debe ser $x = 1, y = 0, z = -1$:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import * # para arrays
from numpy.linalg import * # para determinante
#a=array([[0,2,-1],[1,-1,1],[2,0,-2]], 'float') # coef. sist. ecs.
#b=array([[1],[0],[3]], 'float') # termo independiente
a=array([[1,2,1],[-3,-2, 9],[4,9,6]], 'float') # coef. sist. ecs.
b=array([[4],[40],[24]], 'float') # termo independiente

ao=a.copy() # copio sistema
bo=b.copy()

ra=matrix_rank(a) # rango de a
ab=hstack([a, b]) # crea matriz ampliada
rab=matrix_rank(ab) # rango matriz ampliada
nf,nc=ab.shape # devuelve numero filas e columnas

if ra == rab: # sistema compatible
    print "Sistema compatible"
    print ab
    print "-----"
    for i in range(nf-1):
        # se a(i,i)= 0 pivotar: intercambia ec. i pola ec. que ten o coef.
        # da variable i maximo
        # para i=n-1 (ultima ec. xa que python comeza a numerar por 0)
        # non fai falla pivotar
        if 0 == a[i,i] and i < nf-1:
            print "0 na ec. ", i
            imax=argmax(fabs(ab[i+1:nf, i])) # posicion do maximo na columna i
            pos=i + imax + 1 # posicion absoluta (imax e unha posicion
            # relativa a i)
            print "imax= ",imax, "pos= ", pos
            # pivote: intercambia ec. i por ec. p
            aux=ab[i].copy(); ab[i]=ab[pos].copy(); ab[pos]=aux.copy()
            print ab
            print "-----"

            if ab[i,i] != 0:
                for j in range(i+1, nf):
                    l=ab[j,i]/ab[i,i]
                    ab[j]=ab[j]-l*ab[i]
            print ab
            print "-----"

    if ra == nf:
        print 'Sistema compatible determinado'
        x=zeros(nf, 'float') # crea vector de ceros
        x[-1]=ab[-1,-1]/ab[-1, -2]
        for i in range(nf-2, -1, -1):
            x[i]=(ab[i,-1]-sum(ab[i, i+1:nf]*x[i+1:nf]))/ab[i,i]
        print x
    else:
        print "Sistema compatible indeterminado"
        print "Solucion de dimension ", nf- ra
```

```
else:
    print "sistema incompatible"
```

Outra solución máis compacta utilizando a descomposición LU que está no paquete `scipy.linalg`.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import array
from scipy.linalg import lu
a=array([[0,2,-1],[1,-1,1],[2,0,-2]], 'float') # coef. sist. ecs.
b=array([[1],[0],[3]], 'float') # termo independente
# a=array([[1,2,1],[-3,-2,9],[4,9,6]], 'float') # coef. sist. ecs.
# b=array([[4],[40],[24]], 'float') # termo independente
ab=hstack([a, b]) # crea matriz ampliada
print ab
pl, u = lu(ab, permute_l=True) # descomposicion LU
print "Matriz triangular: "
print u
nf,nc=u.shape # devolve numero filas e columnas
x=zeros(nf, 'float')
x[-1]=u[-1,-1]/u[-1,-2]
for i in range(nf-2, -1, -1):
    x[i]=(u[i,-1]-sum(u[i, i+1:nf]*x[i+1:nf]))/u[i,i]
print x
```

4. **Axuste de puntos a polinomios:** dado un conxunto de puntos $\{(x_i, y_i), i = 1 \dots m\}$ almacenados no arquivo `puntos.dat` (na primeira liña os x_i e na segunda liña os y_i), atopar o polinomio $p(x)$ de grao n que mellor se axusta ós puntos (minimiza a suma dos erros cadráticos entre os puntos (x_i, y_i) e os valores $(x_i, p(x_i))$ do polinomio segundo a expresión:

$$E = \sum_{i=1}^{i=m} (y_i - p(x_i))^2 \quad (23)$$

O programa debe pedir o usuario a orde do polinomio n , facer o axuste e visualizar gráficamente os puntos e o polinomio axustado. Proba cos seguintes datos:

```
0.0 1.0 2.0 3.0 4.0 5.0
0.0 0.8 0.9 0.1 -0.8 -1.0
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
a=loadtxt('datos.dat') # ler arquivo
x=a[0] # primeira linha
y=a[1] # segunda linha
n=0 # orde do polinomio
while n < 1:
    n=int(raw_input('Introduce orde do polinomio: '))
z = polyfit(x, y, n) # axusta a un polinomio de orde n
print "Polinomio axustado: "
for i in range(n):
    print z[0], " x **", i, "+",
print z[n], " x **", n
# representar graficamente puntos e a curva axustada
from matplotlib.pyplot import *
xp = linspace(min(x), max(x), 100)
```

```

p=polyval(z, xp) # calcula os valores do polinomio axustado en xp
# os valores de xp tamen se poden calcular
# p1=poly1d(z) # defino un obxecto poly1z
# p=p1(xp)
plot(x, y, '.', xp, p, '--')
grid(True)
title(u"Axuste a polinomio de orde %d" % (n))
xlabel('x'); ylabel('y'); show()

```

5. **Interpolación de funcións:** dados m puntos $\{(x_j, y_j), j = 1 \dots m\}$ dunha función $f(x)$ descoñecida, quérese calcular $y_i = f(x_i)$ con $x_i \neq x_j, j = 1 \dots m$. Os valores y_i chámaseles valores interpolados. Realiza un programa que a partir da función $f(x) = \text{seno}(x)$ en 10 valores equidistantes no intervalo $[0, 2\pi]$, calcule os valores interpolados en 50 valores equidistantes no mesmo intervalo. Utiliza interpolación lineal e cúbica.
-

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
# x son 10 mostras equiespaciadas en [0, 2pi]
# se probas con 5 mostras a diferenza entre interpolacion
# lineal e cubica e maior
x = linspace(0, 2*pi, 10)
y = sin(x)
xvals= linspace(0, 2*pi, 50)
yinterp = interp(xvals, x, y) # interpolacion lineal
# representacion grafica
from matplotlib.pyplot import *
plot(x, y, 'ob', label='Puntos reais') # puntos reais
plot(xvals, yinterp, '-xg', label='Interp. lineal') # puntos interpolados
title('seno(x)'); xlabel('x'); ylabel('y')
# outros tipos de interpolacion no paquete scipy
from scipy.interpolate import *
f=interp1d(x, y, 'cubic')
yinterpc=f(xvals)
plot(xvals, yinterpc, '-dm', label='Interp. cubica')
legend(loc='upper right')
grid(True); show()

```

Traballo a desenvolver pol@ alumn@

1. Escribe un programa lea a matriz cadrada **a** de orde n dende o arquivo **datos.dat**. Proba coa seguinte matriz:

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

A partir desta matriz, o programa debe calcular un vector **v**, de dimensión $2n-1$, cos seus elementos v_k , con $k = 0, \dots, 2n-2$, dados pola seguinte expresión:

$$v_k = \sum_{i=l}^{m-1} a_{i(k-n+i+1)}, \quad k = 0, \dots, 2n-2 \quad (24)$$

onde $l = \max(0, n - k - 1)$ e $m = \min(2n - k - 1, n)$. **NOTA:** Probando coa matriz de arriba, debes obter $\mathbf{v} = [11, 28, 41, 50, 65, 54, 37, 24, 15]$.

NOTA: o elemento v_k é a suma da k -ésima diagonal da matriz \mathbf{a} . Esta k -ésima diagonal está formada polos elementos a_{ij} que cumpren a condición $i - j = n - 1 - k$, ou ben $j = k - n + i + 1$, con $k = 0, \dots, 2n - 2$. Como debe ser $j \geq 0$, entón $k - n + i + 1 \geq 0$ e $i \geq \max(0, n - k - 1)$. Ademais, como debe ser $j < n$, isto implica que $k - n + i + 1 < n$ e $i < \min(2n - k - 1, n)$. Para $k = 0$ temos a primeira diagonal, integrada polo elemento $a_{(n-1)0}$. Para $k = n - 1$ temos a diagonal principal, integrada polos elementos a_{ii} , que verifican $i - j = 0$. Para $k = 2n - 2$ temos a última diagonal, integrada polo elemento $a_{0(n-1)}$. En total hai $2n - 1$ diagonais para unha matriz cadrada de orde n .

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
a=loadtxt('matriz.dat');n=size(a,0)
p=2*n-1;v=array([0]*p)
for k in range(p):
    s=0;l=max(0,n-k-1);m=min(2*n-k-1,n)
    for i in range(l,m):
        s=s+a[i,k-n+i+1]
    v[k]=s
print 'v=',v
```

- Dados os puntos con coordenadas $x=[0,1,2,3,4,5]$, $y=[1,-0.6242,-1.4707,3.2406,-0.7366,-6.3717]$, axústaos a un polinomio de orde 5 e representa gráficamente os puntos e o polinomio.
- Dados os puntos con coordenadas $x=[1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$ e $y=[8\ 6\ 9\ 4\ 7\ 2\ 6\ 1\ 2\ 3]$, axústaos a unha función potencial da forma $y = ax^b$.

```
>>> from numpy import *
>>> x=[1,2,3,4,5,6,7,8,9,10];y=[8,6,9,4,7,2,6,1,2,3];
>>> p=polyfit(log(x),log(y),1)
>>> p
array([-0.65511141,  2.35723692])
>>> a=exp(p[1]);b=p[0]
>>> from pylab import *
>>> clf();x2=linspace(1,10,100)
>>> y2=a*x2**b
>>> plot(x,y,'sb')
>>> plot(x2,y2,'r-')
```

- Dados os puntos anteriores, axústaos a unha función exponencial da forma $y = ae^{bx}$:

```
>>> polyfit(x,log(y),1)
array([-0.16691054,  2.28573762])
```

- Axusta os puntos anteriores a unha función logarítmica da forma $y = a \cdot \log(x) + b$:

```
>>> polyfit(log(x),y,1)
array([-2.68882221,  8.86130799])
```

- Axusta os puntos a unha función recíproca da forma $y = \frac{1}{ax + b}$:

```
>>> polyfit(x,1./array(y),1)
array([ 0.05309043,  0.03756614])
```

Sesión 6: Entrada/Saída desde archivos con manejo de excepciones.

Trabajo en clase

Cando se traballe con arquivos numéricos moi regulares (todas as filas teñen igual número de columnas) será moito máis simple utilizar a funcionalidade do paquete `Numpy`. Para o resto dos casos, utilizaremos a funcionalidade da linguaxe `python`. En ambos casos, pode existir un erro durante o proceso de lectura eou escritura (arquivo que non existe, ruta incorrecta, etc) que deberíamos xestionar para que o programa non aborte. A xestión destes erros realízase en Python utilizando **excepcións**.

As excepcións son erros detectados por Python durante a execución dun programa. Cando o programa atopa unha situación anómala xera ou lanza un evento (obxeto chamado excepción) informando ó usuario que hai un problema. Posibles excepcións son: división por cero, arquivo que non existe, etc. Python utiliza a construción `try/except` para capturar e tratar as excepcións. O bloque `try` (intentar) define o fragmento de código no que cremos que se produce unha excepción. O bloque `except` (excepción) permite indicar o que se fará se se produce dita excepción. Python ten definidas varios tipos de excepcións que se poden consultar en: <https://docs.python.org/2/library/exceptions.html>.

A lectura/escritura xenérica en arquivos sempre se realiza con cadeas de caracteres ou listas de cadeas, polo tanto, se queres ler ou escribir números será necesario convertilos a cadeas de caracteres (usando, por exemplo, as funcións `str()`, `float()`, `int()` ...).

1. **Escritura en arquivo:** escribe un programa que pida ó usuario un número enteiro positivo n e escriba no arquivo `datos.txt` o seguinte: x , x^2 e \sqrt{x} para $x = 0 \dots n$ (unha liña para cada x).

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import *
n=abs(int(raw_input('Introduce un numero: ')))
outfile = open("datos.txt", 'w')
outfile.write('%5s %8s %10s\n' % ('x', 'x * x', 'sqrt(x)'))
for x in range(n + 1):
    print x, x*x, sqrt(x)
    # version basica que funciona
    #outfile.write(str(x)+' '+str(x*x)+' '+str(sqrt(float(x)))+'\n')
    # version optima utilizando formatos
    outfile.write('%5d %8d %10.3f\n' % (x, x*x, sqrt(x)))
outfile.close()
```

2. **Lectura de archivos como texto:** le o arquivo anterior e visualizao en pantalla.

- a) Lectura do arquivo de texto nunha sentenza xestionando os erros con excepcións.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
try:
    infile = open("datos.txt", 'r')
    aux=infile.read()
    print aux
    infile.close()
except IOError:
    print 'Erro abrindo o arquivo datos.txt'
```

- b) Igual que o anterior pero proporcionando o nome do arquivo en liña de comandos.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```

from sys import * # para a funcion exit()
if 2!=len(argv):
    exit("uso: arquivos2.py nome_arquivo") # aborta o programa
try:
    infile = open(argv[1], 'r') # argv[1] debe ser datos.txt
    aux=infile.read()
    print aux
    infile.close()
except IOError:
    print "erro: o arquivo", argv[1], "non existe"

```

c) Lectura do arquivo de texto liña a liña utilizando iteración.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
try:
    infile = open("datos.txt", 'r')
    for linha in infile:
        print linha
    infile.close()
except IOError:
    print 'Erro abrindo datos.txt'

```

3. Lectura de arquivos con números: Le o arquivo datos.txt e garda ó final do arquivo a suma e o produto dos tres números para cada liña do arquivo orixinal (e dicir, dous valores en cada liña: $x + x^2 + \sqrt{x}$ e $x \times x^2 \times \sqrt{x}$ para $x = 0 \dots n$).

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
try:
    infile = open("datos.txt", 'r+')
    infile.readline() # le a primeira linha
    output=[] # inicializo unha cade baleira
    output.append('%10s %10s\n' % ('suma', 'producto'))
    for linha in infile:
        print linha # obxecto tipo string
        aux=linha.rsplitt()
        print aux
        x1=float(aux[0])
        x2=float(aux[1])
        x3=float(aux[2])
        suma = x1 + x2 + x3
        producto = x1 * x2 * x3
        output.append('%10.2f %10.2f\n' % (suma, producto))
    # escribo o final do arquivo
    for l in output:
        infile.write(l)
    # forma alternativa para escribir todo nunha sentenza
    #infile.writelines(output)
    infile.close()
except IOError:
    print 'Erro abrindo datos.txt'

```

4. Crea un arquivo de texto chamado datos.dat co seguinte contido:

```

15 7 -10 5
23 3 8

```

```
45
1 4 7 14
20 9
18
```

Escribe un programa que pida ó usuario o nome dun arquivo, lea números mentres que a súa suma sexa inferior a 100, almacene os números nun vector **x** e mostre por pantalla o vector.

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
from numpy import *
nome=raw_input('Nome arquivo: ')
# version con for
try:
    suma=0
    f=open(nome, 'r')
    rematar=False;x=[]
    for linha in f:
        v=linha.rsplitt()
        for i in v:
            t=float(i);suma += t; x.append(t)
            if suma > 100:
                rematar=True
                break
    if rematar:
        break
    print 'Suma = ', suma
    f.close()
except IOError:
    print 'Erro no arquivo %s' % nome
#-----
# alternativa con while (menos estable se
# se acada o final do arquivo antes de sumar 100
#try:
#    suma=0
#    f=open(nome, 'r')
#    rematar=False; x=[]
#    while suma < 100:
#        linha=f.readline();v=linha.rsplitt()
#        for i in v:
#            suma += float(i)
#            x.append(float(i))
#            if suma > 100:
#                break
#    f.close()
#except IOError:
#    print 'Erro en open ou fin de arquivo '
#-----
print 'Vector x: ', x
```

Traballo a desenvolver pol@ alumn@

1. Dado o arquivo de texto coa táboa periódica (descarga o arquivo **taboaperiodica.txt** con datos dos elementos químicos ordeados por número atómico. Significado das columnas: 1) abreviatura do elemento, 2) nome do elemento e 3) peso atómico en gramos por mol.

Escribir un programa que pida repetidamente ó usuario a abreviatura dun elemento químico visualice na pantalla o seu nome completo e peso atómico (ten que atopar a información no arquivo anterior). O programa ten que rematar cando o usuario introduza unha X.

2. Escribe un programa que lea por teclado unha cadea de caracteres coa fórmula química dun composto e calcule o peso molecular do mesmo. Para simplificar considera:
 - Usa como abreviaturas dos elementos químicos os nomes que figuran no arquivo da táboa periódica, mantendo as convencións de primeira letra en maiúscula e segunda ou terceira en minúscula.
 - Despois de cada elemento un número especificando o número de elementos no composto. Por exemplo: C102 (para CO_2), H2O1 (para H_2O), C1O3Ca1 (para CO_3Ca), etc.
 - O peso atómico de cada elemento químico obtérase do arquivo **taboaperiodica.txt** do exercicio anterior.

Nota aclaratoria: sea x unha cadea de caracteres, o método `x.isdigit()` devolve true se x é un número e false en caso contrario (se x é unha letra). O método `isdigit()` tamén pode ser aplicado sobre un elemento da cadea (`x[0].isdigit()` devolve true se o primeiro elemento da cadea é un número).

Sesión 7 : Definición de funcións. Creación de módulos.

Traballo en clase

1. Escribe unha función, chamada `fact()`, que calcule o factorial dun número enteiro. Escribe un programa que pida ó usuario un número n e calcule o factorial de todos os enteiros no intervalo $[0, n]$. Para comprobar o resultado podes utilizar a función `factorial()` do módulo `math` que calcula o factorial dun número enteiro.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
n = int(raw_input("Introduce n: "))
# inicio da definicion de funcion
def fact(x):
    if x < 2:
        return 1
    else:
        f = 1
        for i in range(2,x+1):
            f = f * i
        return f
# fin da definicion de funcion
for i in range(n+1):
    print "Factorial de ", i, " e ",fact(i)
# utilizando o modulo math
from math import *
for i in range(n+1):
    print "Math: factorial de ", i, " e ", factorial(i)
```

2. Escribe un programa que use dúas funcións `calcula_mcd` e `calcula_mcm` que calculen respectivamente o máximo común divisor e o mínimo común múltiplo de dous números enteiros positivos x e y . Proba con $x = 120 = 2^3 \cdot 3 \cdot 5$, $y = 36 = 2^2 \cdot 3^2$, tes que obter $mcd = 12$, $mcm = 360$.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
print "calcula o mcd e mcm de dous enteiros positivos x e y: "
x= int(raw_input("x? "))
y= int(raw_input("y? "))

#descomposicion en factores primos
def factores_primos(a):
    x=a; b=[]; e=[]; i=2
    while x>1:
        if x%i == 0:
            x=x/i; b.append(i); e.append(1)
            while x%i == 0:
                e[-1]=e[-1]+1; x=x/i
            i=i+1
    print "descomposicion de",a,":",
    for i in range(len(b)):
        print b[i],"^",e[i],"*",
    print
    return [b,e]

[bx,ex] = factores_primos(x)
```

```

[by,ey] = factores_primos(y)

def calcula_mcm(bx,ex,by,ey): #minimo comun multiplo
    b=bx[:]; e=ex[:] #copia bases e exponentes de factores de x
    nx=len(b); ny=len(by)
    for i in range(ny): # percorre factores de y
        comun=False
        for j in range(nx): # compara con factores de x
            if b[j]==by[i]:
                comun=True
                if ey[i]>e[j]:
                    e[j]=ey[i] #colle o meirande exponhente
                break
        if not comun:
            b.append(by[i]); e.append(ey[i]) #engade factor non comun

    mcm=1
    print "descomposicion do mcm:",
    for i in range(len(b)):
        print b[i],"^",e[i],"*",
        mcm=mcm*b[i]**e[i]
    print
    print "mcm=",mcm
    return mcm

mcm=calcula_mcm(bx,ex,by,ey)

def calcula_mcd(bx,ex,by,ey): #maximo comun divisor
    b=[]; e=[]; nx=len(bx); ny=len(by)
    for i in range(nx):
        z=bx[i]
        for j in range(ny):
            if z==by[j]:
                b.append(z)
                e.append(min(ex[i],ey[j])) #colle o menor exponhente
                break

    mcd=1
    print "descomposicion do mcd:",
    for i in range(len(b)):
        print b[i],"^",e[i],"*",
        mcd=mcd*b[i]**e[i]
    print
    print "mcd=",mcd
    return mcd

mcd=calcula_mcd(bx,ex,by,ey)

```

3. **Creación dun módulo de usuario:** crear un módulo chamado `mcdmcm.py` que conteña as funcións para calcular o mínimo común múltiplo e o máximo común divisor de dous números enteiros. Para isto só é necesario definir as funcións nun arquivo `.py`, e o nome do arquivo será o nome do módulo. O uso de módulos definidos polo usuario segue a mesma sintaxe que os módulos de Python. Desde o programa principal `modulosPropios.py` lee dous números desde o teclado e calcula o mínimo común múltiplo e o máximo común divisor utilizando o módulo que acabas de crear. As funcións que contén o módulo `mcdmcm.py` son:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-

```

```

#descomposicion en factores primos
def factores_primos(a):
    x=a; b=[]; e=[]; i=2
    while x >1:
        if x %i == 0:
            x=x/i; b.append(i); e.append(1)
            while x %i == 0:
                e[-1]=e[-1]+1; x=x/i
            i=i+1
    return [b,e]

def calcula_mcm(x,y): # minimo comun multiplo
    [bx ,ex] = factores_primos(x)
    [by ,ey] = factores_primos(y)
    b=bx[:]; e=ex[:] # copia bases exponhentes a mcm
    nx=len(b); ny=len(by)
    for i in range(ny): # percorre factores de y
        comun =False
        for j in range(nx): # compara con factores de x
            if b[j]== by[i]:
                comun =True
                if ey[i]>e[j]:
                    e[j]= ey[i] # colle meirande expenhente
                break
        if not comun:
            b.append(by[i]); e.append(ey[i]) # engade factores non comuns
    mcm =1
    for i in range(len(b)):
        mcm=mcm*b[i]**e[i]
    return mcm

def calcula_mcd(x, y): #maximo comun divisor
    [bx ,ex] = factores_primos(x)
    [by ,ey] = factores_primos(y)
    b=[]; e=[]; nx=len(bx); ny=len(by)
    for i in range(nx):
        z=bx[i]
        for j in range(ny):
            if z== by[j]:
                b.append(z)
                e.append(min(ex[i],ey[j])) # colle o menor expenhente
    mcd =1
    for i in range(len(b)):
        mcd=mcd *b[i]**e[i]
    return mcd

```

e o arquivo `modulosPropios.py` contén:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from mcdmcm import *
print " calcula o mcd e mcm de dous enteiros positivos x e y: "
x= int(raw_input ("x? "))
y= int(raw_input ("y? "))
print 'Minimo comun multiplo: ', calcula_mcm(x,y)
print 'Maximo comun divisor: ', calcula_mcd(x,y)

```

Pola outra banda, no paquete `fractions` existe a función `gcd(x,y)` (greatest common divider) que calcula o máximo común divisor (mcd) de dous números `x` e `y`. Logo, usando que $mcm(x,y)mcd(x,y) = xy$, temos que $mcm(x,y) = \frac{xy}{mcd(x,y)}$:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from fractions import gcd
print "calcula o mcd e mcm de dous enteiros positivos x e y"
x= int(raw_input("x? "))
y= int(raw_input("y? "))
print "mcd de",x,"e",y,"=",gcd(x,y)
print "mcm de",x,"e",y,"=",x*y/gcd(x,y)
```

4. **Funcións lambda:** o operador `lambda` sirve para crear funcións anónimas en línea. As funcións `lambda` (ou anónimas) constrúense co operador `lambda`, a lista de argumentos separados por comas e SEN paréntases, dous puntos (:) e o código da función (unha única expresión). En tempo de execución son igual as funcións normais pero, as funcións `lambda` resultan uteis cando o corpo da función contén só unha expresión. Define unha función que calcule se os elementos dunha lista son pares ou impares. Tamén define unha función `lambda` que sume tres números.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
# usando funciones
def es_par(n):
    return (n % 2 == 0)

l = array([1, 2, 3])
print "Usando funciones: ",es_par(l)
# usando funciones lambda con un argumento
f=lambda n: n % 2 == 0
print "Usando funciones lambda: ", f(l)
# usando a funcion lambda con varios argumentos
g=lambda n1, n2, n3: n1+n2+n3
print "Sumando numeros 2+4+5: ", g(2,4,5)
```

Traballo a desenvolver pol@ alumn@

1. **Función definida por intervalos.** Escribe unha función `funcionIntervalos(...)` que reciba como argumentos os límites superior e inferior dun intervalo e retorne unha lista cos valores da función $f(x)$ seguinte neste intervalo:

$$f(x) = \begin{cases} 4e^{x+2} & -6 \leq x < -2 \\ x^2 & -2 \leq x < 2 \\ (x+6)^{1/3} & 2 \leq x < 6 \end{cases}$$

Escribe tamén un programa que represente gráficamente $f(x)$ no intervalo $-10 \leq x \leq 10$.

2. Escribe unha función que calcule o máximo e o mínimo de tres números. Escribe un programa que pida ó usuario a introducción de tres números por teclado, chame a función que acabas de definir e visualice o máximo e mínimo dos números.
3. **Persistencia dun número enteiro:** Escribe un programa que lea por teclado un número enteiro e calcule a súa persistencia. Con este fin, o programa deberá separar o número nas súas cifras e multiplicalas entre si. O resultado tamén se dividirá 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 descomponse nas súas cifras (7,1 e 5), multiplícanse as cifras entre si e resulta 35, repítese o proceso e resulta 15, repítese o proceso e resulta 5 (715- > 35- > 15- > 5). A persistencia é o número de veces que se repite a descomposición, polo tanto, 3 para 715. NOTA: para descompoñer un número nas súas cifras podes dividir sucesivamente por 10 e quedar co resto da división.

4. Calcula o valor da función $f(x) = x^4 - 10x^2 + 9$ nos puntos $x=0$ e $x=3.4$ utilizando dunha función lambda.
5. Realiza un programa en python que calcule o valor da función:

$$f(x, y) = \frac{x^2 + y}{\sqrt{x^2 + y^2}} \quad (25)$$

nos puntos $(x, y) = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$ utilizando unha función lambda para definir a función $f(x, y)$.

Sesión 8 : Regresión lineal. Creación de módulos.

Traballo en clase

A ciencia experimental intenta relacionar unha ou máis magnitudes mediante unha función matemática. A análise de regresión proporciona un método estadístico para avaliar cal é a función que describe, de xeito máis exacto, a relación entre as magnitudes.

- **Regresión lineal:** supoñamos que dúas magnitudes x e y verifican a función lineal $y = a + bx$. Queremos determinar, a partir dun conxunto de N pares de valores x_i, y_i , $i = 0, \dots, N-1$, os valores de a e b , así como as súas incertezas e o correspondente intervalo de confianza. O **método de mínimos cadrados** permite determinar os valores de a e b minimizando a suma dos produtos do peso estadístico de cada punto polos cadrados das desviacións dos datos respecto ós valores preditos pola función a determinar. Os valores de a e b veñen dados polas expresións:

$$b = \frac{N \sum_i x_i y_i - \left(\sum_i x_i \right) \left(\sum_i y_i \right)}{N \sum_i x_i^2 - \left(\sum_i x_i \right)^2} \quad i = 0, 1, \dots, N-1 \quad (26)$$

$$a = \frac{\left(\sum_i x_i^2 \right) \left(\sum_i y_i \right) - \left(\sum_i x_i \right) \left(\sum_i x_i y_i \right)}{N \sum_i x_i^2 - \left(\sum_i x_i \right)^2} \quad i = 0, 1, \dots, N-1 \quad (27)$$

A **desviación típica do axuste** para a mostra ven dada pola expresión:

$$s = \sqrt{\frac{\sum_i (y_i - a - bx_i)^2}{N-2}} \quad i = 0, 1, \dots, N-1 \quad (28)$$

As **incertezas dos parámetros a e b** polas expresións:

$$s(a) = s \sqrt{\frac{\sum_i x_i^2}{N \sum_i x_i^2 - \left(\sum_i x_i \right)^2}} \quad i = 0, 1, \dots, N-1 \quad (29)$$

$$s(b) = s \sqrt{\frac{N}{N \sum_i x_i^2 - \left(\sum_i x_i \right)^2}} \quad i = 0, 1, \dots, N-1 \quad (30)$$

e o **coeficiente de regresión lineal (r)**, relacionada coa exactitude da ecuación da recta na representación dos datos (r será máis preciso cando máis se achegue a 1 o seu valor absoluto ($|r|$)), pola expresión:

$$r = \frac{N \sum_i x_i y_i - \sum_i x_i \sum_i y_i}{\sqrt{\left(N \sum_i x_i^2 - \left(\sum_i x_i \right)^2 \right) \left(N \sum_i y_i^2 - \left(\sum_i y_i \right)^2 \right)}} \quad i = 0, 1, \dots, N-1 \quad (31)$$

- **Regresión lineal ponderada:** nela debe minimizarse a suma ponderada dos cadrados das desviacións, co que as ecuaciones 26 a 31 transformaríanse en:

$$a = \frac{\sum_i w_i y_i \sum_i w_i x_i^2 - \sum_i w_i x_i \sum_i w_i x_i y_i}{\Delta} \quad i = 0, 1, \dots, N-1 \quad (32)$$

$$b = \frac{\sum_i w_i \sum_i w_i x_i y_i - \sum_i w_i x_i \sum_i w_i y_i}{\Delta} \quad i = 0, 1, \dots, N-1 \quad (33)$$

onde $w_i = \frac{1}{s(y_i)^2}$ é o peso estadístico, $s(y_i)$ é a incerteza na medida y_i e Δ é o determinante da matriz:

$$\begin{pmatrix} \sum_i w_i & \sum_i w_i x_i \\ \sum_i w_i x_i & \sum_i w_i x_i^2 \end{pmatrix}$$

As incertezas de a e b serán:

$$s(a) = \sqrt{\frac{\sum_i w_i x_i^2}{\Delta}} \quad \text{e} \quad s(b) = \sqrt{\frac{\sum_i w_i}{\Delta}} \quad i = 0, 1, \dots, N-1 \quad (34)$$

A desviación típica do axuste sería:

$$s = \sqrt{\frac{N}{(N-2) \sum_i w_i} \sum_i w_i (y_i - a - bx_i)^2} \quad (35)$$

e o coeficiente de regresión para un axuste ponderado será:

$$r = \frac{\sum_i w_i \sum_i w_i x_i y_i - \sum_i w_i x_i \sum_i w_i y_i}{\sqrt{\left(\sum_i w_i \sum_i w_i x_i^2 - \left(\sum_i w_i x_i \right)^2 \right) \left(\sum_i w_i \sum_i w_i y_i^2 - \left(\sum_i w_i y_i \right)^2 \right)}} \quad i = 0, 1, \dots, N-1 \quad (36)$$

1. Realiza un módulo en python chamado **regresion.py** que conteña a funcionalidade para calcular a función matemática $y = a + bx$ utilizando regresión simple ou ponderada (para o cal debe calcular os valores de a e b . Tamén debe calcular as incertezas de a e b ($s(a)$ e $s(b)$) e o coeficiente de regresión r .

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# paquete que contén funcionalidade para realizar regresions lineais
from numpy import *
from numpy.linalg import * # para determinante matriz

def regresionSimple(x,y):
    """Axusta os datos dos vectores x e y a unha recta dada
    pola ec. y=a + bx
    Parametros:
    x vector con medidas da magnitud x
    y vector con medidas da magnitud y
    Devolve:
    a coeficiente a
    b coeficiente b
```

```

    sa incerteza de a
    sb incerteza de b
    r coeficiente de regresion lineal """
n=len(x)
sx=sum(x); sy=sum(y); xx=dot(x,x); yy=dot(y,y); xy=dot(x,y);
denom=(n*xx - sx**2)
b=(n*xy - sx*sy)/denom
a=(xx*sy - sx*xy)/denom
s=sqrt(sum((y-a-b*x)**2)/(n-2))
sa=s*sqrt(xx/(n*xx-sx**2))
sb=s*sqrt(n/(n*xx-sx**2))
r=(n*xy-sx*sy)/sqrt((n*xx-sx**2)*(n*yy-sy**2))
return [a,b, sa, sb, r]

def regresionPonderada(x,y,s):
    """Axusta os datos dos vectores x e y a unha recta dada pola
    ec.  $y=a + bx$  utilizando regresion ponderada
    con incertazas s na medida y
    Parametros:
    x vector con medidas da magnitud x
    y vector con medidas da magnitud y
    s vector coas incertezas na medida y
    Devolve:
    a coeficiente a
    b coeficiente b
    sa incerteza de a
    sb incerteza de b
    r coeficiente de regresion lineal """
w=1.0/(s*s) # pesos estadisticos
wy=sum(w*y); wx=sum(w*x);
wxx=sum(w*x*x); wxy=sum(w*x*y); wyy=sum(w*y*y)
sw=sum(w)
d=det(array([[sw, wx],[wx, wxx]]))
a=(wy*wxx-wx*wxy)/d
b=(sw*wxy-wx*wy)/d
sa=sqrt(wxx/d); sb=sqrt(sw/d)
r=(sw*wxy-wx*wy)/sqrt((sw*wxx-wx**2)*(sw*wyy-wy**2))
return [a, b, sa, sb, r]

```

2. Realiza un programa que, utilizando o módulo `regresion.py`, calcula a recta $y = a + bx$ que mellor axuste os datos experimentais do seguinte arquivo de texto `datos.txt` supoñendo que se realiza unha regresión simple:

```

1 3 4 6 8 11 12 15
1 7 12 17 25 34 36 45

```

onde a primeira fila son as medidas x_i e a segunda fila as medidas y_i . Visualiza as incertezas de a e b e o coeficiente de regresión na pantalla. Amplia o programa anterior para que realice unha regresión ponderada supoñendo que as incertezas de x_i son despreziables e as incertezas de y_i son: $s(y_i) = 0,1y_i$. Representa gráficamente os puntos e as rectas axustadas.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
# proba o modulo regresion.py lendo dun arquivo os datos a axustar.
from numpy import *
from sys import exit
from regresion import *

```

```

from matplotlib.pyplot import *
arquivo=raw_input("Arquivo de dados: ")
try:
    a=loadtxt(arquivo)
    (nf,nc)=a.shape
    if nf == 2:
        x=a[0]; y=a[1]
    elif nf == 3:
        x=a[0]; y=a[1]; s=a[2]
    else:
        raise nf
except IOError:
    exit("Arquivo %s non existe" % (arquivo))
except:
    exit("Arquivo non adecuado")
clf()
subplots_adjust(hspace=0.4)
subplot(211)
plot(x, y, 'b*')
n=len(x); p=[0, n-1]
a,b, sa, sb, r=regresionSimple(x,y)
print "RS. Ec. da recta: ", 'y= %g + %g x' % (a, b)
print "a= %g b= %g sa=%g sb=%g r=%g" % (a,b, sa, sb, r)
yr=a + b * x
plot(x[p], yr[p], 'g-', label='RS')
b2, sb2, r2=regresionSimpleSenTermoIndependente(x,y)
print "RSSTI. Ec. da recta: ", 'y= %g x' % (b2)
print "b= %g sb=%g r=%g" % (b2, sb2, r2)
yr2=b2 * x
plot(x[p], yr2[p], 'r-', label='RSSTI')
legend(loc='upper left')
xlabel('$x_i$'); ylabel('$y_i$')
title('Regresion simple')
subplot(212)
plot(x,y,'b*')
if nf == 2:
    print "Utiliza pesos por defecto w=sqrt(0.1 yi)"
    a,b, sa, sb, r=regresionPonderada(x,y, 0.1*y)
    print "RP. Ec. da recta: ", 'y= %g + %g x' % (a, b)
    print "a= %g b= %g sa=%g sb=%g r=%g" % (a,b, sa, sb, r)
    b2, sb2, r2=regresionPonderadaSenTermoIndependente(x,y, 0.1*y)
    print "RPSTI. Ec. da recta: ", 'y= %g x' % (b2)
    print "b= %g sb=%g r=%g" % (b2, sb2, r2)
else:
    a,b, sa, sb, r=regresionPonderada(x,y, s)
    print "RP. Ec. da recta: ", 'y= %g + %g x' % (a, b)
    print "a= %g b= %g sa=%g sb=%g r=%g" % (a,b, sa, sb, r)
    b2, sb2, r2=regresionPonderadaSenTermoIndependente(x,y, s)
    print "RPSTI. Ec. da recta: ", 'y= %g x' % (b2)
    print "b= %g sb=%g r=%g" % (b2, sb2, r2)
yr= a + b*x
plot(x[p], yr[p], 'g-', label='RP')
yr2= b2*x
plot(x[p], yr2[p], 'r-', label='RSSTI')
legend(loc='upper left')
xlabel('$x_i$'); ylabel('$y_i$')
title('Regresion ponderada')

```

```
savefig('regresion.eps')
show(False)
```

3. Proba o programa anterior cos datos simulados:

```
1 2 3 4 5 6 7 8
9.9 13.5 16.1 19.9 22.5 25.2 29.2 32
```

onde a primeira fila son as medidas x_i e a segunda fila as medidas y_i . Na regresión ponderada supoñ que as incertezas de x_i son despreziables e as incertezas de y_i son o 10 % da medida.

Traballo a desenvolver pol@ alumn@

1. **Regresión lineal simple sen termo independente.** Amplía o módulo `regresion.py` para que calcule o axuste a unha recta utilizando regresión lineal simple sen termo independente. Se a recta de axuste na regresión lineal pasa pola orixe de coordenadas, a función matemática sería $y = bx$ e as ecuacións 26 a 31 tranfórmanse nas seguintes expresións:

$$b = \frac{\sum_i x_i y_i}{\sum_i x_i^2} \quad i = 0, 1, \dots, N-1 \quad (37)$$

$$s = \sqrt{\frac{\sum_i (y_i - bx_i)^2}{N-1}} \quad i = 0, 1, \dots, N-1 \quad (38)$$

$$s(b) = \frac{s}{\sqrt{\sum_i x_i^2}} \quad i = 0, 1, \dots, N-1 \quad (39)$$

$$r = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}} \quad i = 0, 1, \dots, N-1 \quad (40)$$

2. **Regresión lineal ponderada sen termo independente.** Amplía o módulo `regresion.py` para que calcule o axuste a unha recta utilizando regresión lineal ponderada sen termo independente. Se a recta de axuste na regresión lineal pasa pola orixe de coordenadas, a función matemática sería $y = bx$ e as ecuacións 32 a 36 tranfórmanse nas seguintes expresións:

$$b = \frac{\sum_i w_i x_i y_i}{\sum_i w_i x_i^2} \quad i = 0, 1, \dots, N-1 \quad (41)$$

$$s = \sqrt{\frac{N}{(N-1) \sum_i w_i} \sum_i w_i (y_i - bx_i)^2} \quad i = 0, 1, \dots, N-1 \quad (42)$$

$$s(b) = \frac{1}{\sqrt{\sum_i w_i x_i^2}} \quad i = 0, 1, \dots, N-1 \quad (43)$$

$$r = \frac{\sum_i w_i x_i y_i}{\sum_i w_i x_i^2 \sum_i w_i y_i^2} \quad i = 0, 1, \dots, N-1 \quad (44)$$

3. Realiza un programa que lea dun arquivo de texto `datos.txt` os datos experimentais a axustar e, utilizando o módulo `regresion.py`, calcule a ecuación da recta utilizando os dous tipos de regresión sen termo independente. No caso dos datos simulados, deberías obter os seguintes calculos:

Regresión simple. Ec. da recta: $y = 6.94286 + 3.13214 x$
 $a = 6.94286$ $b = 3.13214$ $sa = 0.278293$ $sb = 0.0551102$ $r = 0.999073$
 Regresión simple sen termo independente. Ec. da recta: $y = 4.35735 x$
 $b = 4.35735$ $sb = 0.236926$ $r = 0.98981$
 Utiliza pesos por defecto $w = 0.1$ y_i
 Regresión ponderada. Ec. da recta: $y = 6.88012 + 3.14575 x$
 $a = 6.88012$ $b = 3.14575$ $sa = 1.02997$ $sb = 0.297306$ $r = 0.999093$
 Regresión ponderada sen termo independente. Ec. da recta: $y = 4.75988 x$
 $b = 4.75988$ $sb = 0.17321$ $r = 2.18172$

Deberás obter a figura 2.

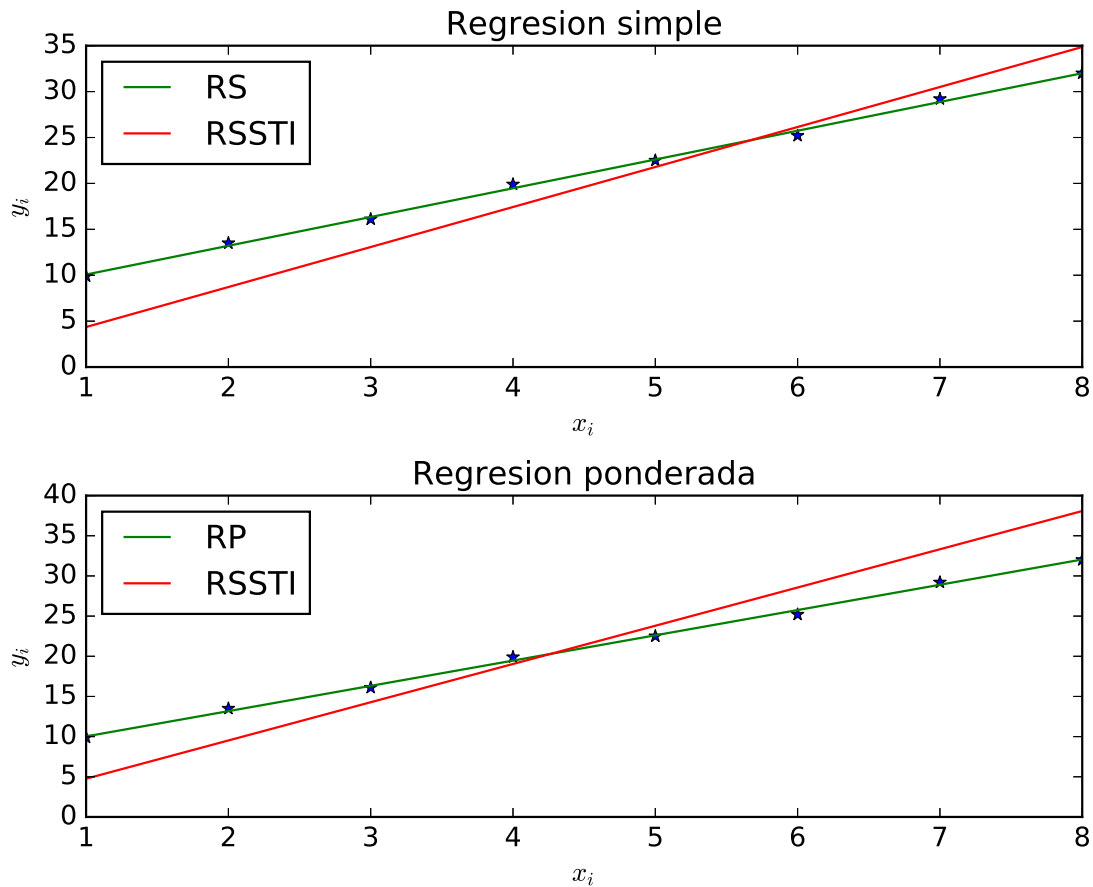


Figura 2: Gráfico que mostra a recta que mellor axusta os datos simulados utilizando regresión lineal simple e ponderada.

Sesión 9: suma de series, derivadas e integrais numéricas

Traballo en clase

1. **Suma dunha serie.** Escribe un programa chamado `serie.py` que calcule o valor aproximado da serie numérica infinita $\sum_{n=1}^{\infty} \frac{1}{n^2}$. Para isto, debes sumar os termos $1/n^2$ mentres que o seu valor sexa maior que 10^{-5} . O programa debe representar gráficamente: a) os valores dos sumandos fronte a n ; b) o valor da suma en función de n .

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from matplotlib.pyplot import *
suma=0;sumando=1;n=1;umbral=1e-5
x=[];y=[];z=[]
while sumando > umbral:
    sumando=1./n**2
    suma=suma+sumando
    x.append(n);y=y.append(sumando);z.append(suma)
    n=n+1
subplot(211);semilogy(x,y)
xlabel('n');ylabel('sumando');grid(True)
xlim(1,n)
subplot(212);plot(x,z)
xlabel('n');ylabel('suma');grid(True)
xlim(1,n)
print 'umbral= %g' % umbral
print 'valor calculado= %.10f' % suma
# calculo do valor exacto da serie
from sympy import *
n=symbols('n')
vex=Sum(1/n**2,(n,1,oo)).evalf()
print 'valor exacto= %.10f' % vex # 1.64493406684823
print 'diferencia= %.10f' % abs(vex - suma)
```

2. **Derivada dunha función dados os seus valores nun intervalo.** Escribe unha función chamada `derivada(...)` que reciba como argumento unha lista `f` cos valores dunha función nun intervalo. A función `derivada()` debe retornar outra lista `df` cos valores da derivada da función nese intervalo. Considera que se $\{f(x_0), \dots, f(x_{n-1})\}$ é a lista de valores da función no intervalo, entón $x_{i+1} - x_i \approx h, i = 0, \dots, n-2$, de modo que, supoñendo $h \simeq 0$:

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h} = f(x_{i+1}) - f(x_i), i = 0, \dots, n-2 \quad (45)$$

Descarga o arquivo `posicion.dat`. Dende o programa principal, le os datos do arquivo a unha lista e chama á función `derivada(...)` para que calcule a velocidade como a derivada da posición. Finalmente, usa a función `ediff1d(...)` de `numpy`, que fai o mesmo que a función `derivada(...)`.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from matplotlib.pyplot import *
from numpy import ediff1d
def derivada(f):
    n=len(p) - 1;df=[0]*n
```

```

    for i in range(n):
        df[i]=f[i+1] - f[i]
    return df

p=[]
try:
    f=open("posicion.dat","r")
    for linha in f:
        p=p+[float(linha)]
    f.close()
    v=derivada(p)
    #v=ediff1d(p) # alternativa
    subplot(2,1,1); plot(p);
    xlabel("tempo (s)"); ylabel("posicion (m)"); grid(True)
    subplot(2,1,2); plot(v);
    xlabel("tempo (s)"); ylabel("velocidade (m)"); grid(True)
    show(False)
except IOError:
    print "erro abrindo posicion.dat"

```

3. **Integral indefinida dunha función dados os seus valores nun intervalo.** Escribe unha función chamada `integral(...)`, pasándolle como argumento os valores dunha función $f(x)$ nun certo intervalo (os límites deste intervalo non se deben pasar como argumentos). Esta función debe calcular a integral indefinida $\int f(x)dx$, usando que se $p(x) = \int f(x)dx$ é unha primitiva de $f(x)$, entón $\frac{dp(x)}{dx} = f(x)$. Considerando a definición de derivada, temos que:

$$\frac{dp(x)}{dx} = \lim_{h \rightarrow 0} \frac{p(x+h) - p(x)}{h} \approx \frac{p(x+h) - p(x)}{h} = f(x) \Rightarrow \quad (46)$$

$$f(x) \approx \frac{p(x+h) - p(x)}{h} \Rightarrow p(x+h) \approx p(x) + hf(x) \quad (47)$$

Supoñendo que no punto inicial do intervalo a primitiva $p(x)$ é coñecida (por exemplo, que vale 0), o cal é equivalente a fixar a constante de integración C , podemos calcular os valores $p(x+ih)$, $i = 1, \dots, n$, sendo $n = 1 + \frac{b-a}{h}$ o número de valores de $f(x)$, calculando así unha primitiva de $f(x)$. Usa $h=0.01$. A función `integral(...)` debe retornar unha lista cos valores de $p(x)$ calculados. Logo, descarga o arquivo **forza.dat**, que contén os valores dunha forza $F(x)$ en función da posición x . Escribe un programa que lea dende este arquivo a forza $F(x)$, e chame á función `integral(...)` anterior, pasándolle como argumento unha lista cos valores $F(x)$ lidos dende o arquivo. Os valores retornados serán os valores do potencial $V(x) = -\int F(x)dx$. O programa principal debe representar gráficamente os valores da forza $F(x)$ e do potencial $V(x)$ en función de x .

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from matplotlib.pyplot import *
from numpy import *
def integralf(f):
    n=len(f); g=[0.0]*n
    for i in range(1,n):
        p[i]=p[i-1]-0.01*f[i-1] # signo - porque V=-int F(x) dx
    return p
# programa principal
try:
    fz=loadtxt('forza.dat')

```

```

v=integralf(fz)
x=range(len(fz))
subplot(2,1,1); plot(x,fz); xlabel(u'x (metros)')
ylabel(u'Forza (Nw)'); grid(True)
subplot(2,1,2); plot(x,v); xlabel(u'x (metros)')
ylabel(u'Potencial (Nw*m)'); grid(True)
show()
except IOError:
    print 'Erro abrindo forza.dat'

```

Traballo a desenvolver pol@ alumn@

1. **Partícula nun potencial.** Consideremos un obxecto (que podes considerar que é unha partícula) con masa $m = 1$ kg que se atopa nun campo de forzas radial dado polo potencial:

$$V(r) = V_0 \ln \left(1 + \frac{b}{r^2} \right) \quad (48)$$

Onde r é o radio (distancia ao orixe do campo) en metros, $V_0=1$ Nw·m e $b=1$ m². No instante inicial $t = 0$ s. a partícula atópase en $r_0=0.01$ m con velocidade $v_0=1$ m/s. Calcula e representa gráficamente para $r_0 \leq r \leq r_1 = 1.5$ m. con paso $\Delta r=0.01$ m:

- a) O potencial $V(r)$.
- b) A forza $F(r)$, dada por $F = -\frac{dV}{dr}$.
- c) A velocidade $v(r)$. Para isto, debes ter en conta que $F = ma$, $a = \frac{dv}{dt}$, $v = \frac{dr}{dt}$, de modo que:

$$a = \frac{F(r)}{m} = \frac{dv}{dt} = \frac{dv}{dr} \frac{dr}{dt} = \frac{dv}{dr} v \rightarrow a dr = v dv \rightarrow \int_{r_0}^r \frac{F(s)}{m} ds = \int_{v_0}^v w dw$$

e finalmente:

$$\frac{v^2 - v_0^2}{2} = \frac{1}{m} \int_{r_0}^r F(s) ds \rightarrow v(r) = \sqrt{v_0^2 + \frac{2}{m} \int_{r_0}^r F(s) ds} \quad (49)$$

Ademáis, nota que $\int_{r_0}^r F(s) ds \simeq \Delta r \sum_{i < n} F(r_i)$, con $r_0 \leq r_i \leq r_1$, sendo $r_i = r_0 + i\Delta r$ onde

$$\Delta r = \frac{r_1 - r_0}{n - 1} \text{ e } i = 0, \dots, n - 1.$$

- d) A posición $p(t)$ para $0 \leq t \leq 0.5$ s. con $\Delta t=0.001$ s. Para isto, considera que $p(t=0) = r_0=0.01$ m. e que $\frac{dp}{dt} \simeq \frac{\Delta p}{\Delta t} = v[p(t)]$, de modo que $p(t + \Delta t) = p(t) + v[p(t)]\Delta t$. Para calcular $v[p(t)]$ busca o valor $v(r)$ para o valor de r máis cercano a $p(t)$.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from pylab import *

clf()

V0,b,m=1,1,1;h=0.01
r=arange(0.01,1.5,h);n=len(r);

```

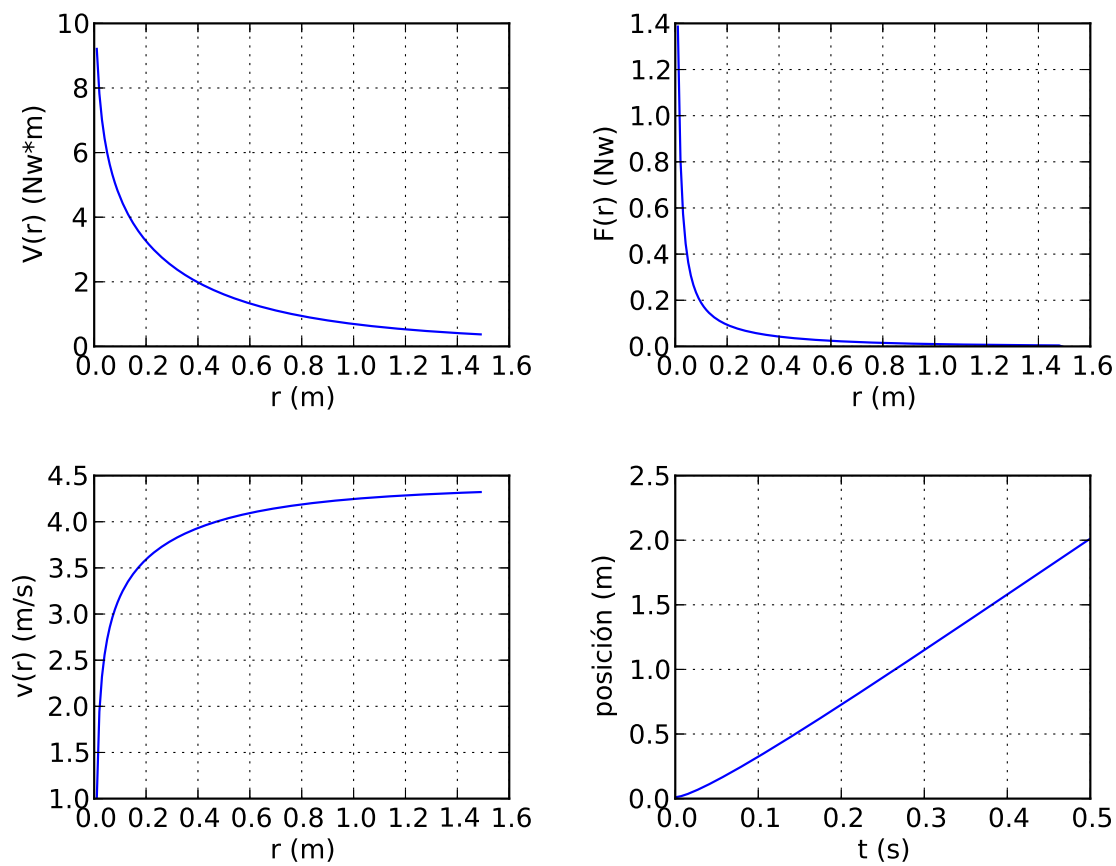


Figura 3: Potencial $V(r)$, forza $F(r)$, velocidade $v(r)$ e posição $p(t)$ no exercício do potencial.

```
subplots_adjust(hspace=0.4, wspace=0.4)

V=V0*log(1+b/r**2) # potencial
subplot(221)
plot(r,V);grid(True);xlabel('r (m)');ylabel('V(r) (Nw*m)')

F=zeros(n) #forza
for i in range(n-1):
    F[i]=(-V[i+1]+V[i])/h #F=-dV/dr ~ -(V(r+h)-V(r))/h
subplot(222)
plot(r,F);grid(True);xlabel('r (m)');ylabel('F(r) (Nw)')

v0=1;v=zeros(n) #velocidade
for i in range(n): # v=sqrt(v0^2+int(F(x)dx)/m)
    v[i]=sqrt(v0**2+2*h*sum(F[0:i])/m)
subplot(223)
plot(r,v);grid(True);xlabel('r (m)');ylabel('v(r) (m/s)')

#posicion
dt=1e-3;t=arange(0,0.5,dt);n=len(t);p=zeros(n);p[0]=0.01
for i in range(1,n):
```

```

    #atopa k tal que r[k] e o mais cercano a p[i-1]
    k=argmin(fabs(r-p[i-1]))
    p[i]=p[i-1]+v[k]*dt
subplot(224)
plot(t,p);grid(True);xlabel('t (s)');ylabel(u'posicion (m)')

show(False)
savefig('potencial.eps')

```

Tes que obter a gráfica da figura 3:

Sesión 10 : Representación gráfica

Trabajo en clase

1. **Curvas en escala logarítmica.** Representa gráficamente as curvas: 1) $f(x) = e^{-x/50}$ con escala Y logarítmica; 2) $f(x) = \sin(2\pi x)$ con escala X logarítmica; 3) $f(x) = 20e^{-t/10}$ con escalas X e Y logarítmicas. Usa x dende 0.01 ata 20 con paso 0.01.

```
from numpy import *
from matplotlib.pyplot import *

subplots_adjust(hspace=0.4)
t = arange(0.01, 20.0, 0.01)

# log y axis
subplot(311)
semilogy(t, exp(-t/5.0))
title('semilogy')
grid(True)

# log x axis
subplot(312)
semilogx(t, sin(2*pi*t))
title('semilogx')
grid(True)

# log x and y axis
subplot(313)
loglog(t, 20*exp(-t/10.0), basex=2)
grid(True)
title('loglog base 4 on x')

show(False)
```

2. Representa gráficamente a **curva 2D animada** $f(x) = e^{-x^2} \sin(tx/2)$, con $x \in [-\pi/2, \pi/2]$ e $t = 0 \dots 200$

```
from numpy import *
from matplotlib.pyplot import *
from matplotlib.animation import *

fig = figure()
ax = fig.add_subplot(111)

x = arange(-pi/2, pi/2, 0.01)
line, = ax.plot(x, exp(-x**2)*sin(x/2))
axis([-pi/2, pi/2, -1, 1])
grid(True)

def animate(t):
    line.set_ydata(exp(-x**2)*sin(t*x/2))
    return line,

def init():
    line.set_ydata(ma.array(x, mask=True))
```

```

    return line,

ani = FuncAnimation(fig, animate, arange(1, 200),
    init_func=init, interval=25, blit=True)

show(False)

```

3. Curva en coordenadas polares en 2D. Son curvas da forma $\varrho = \varrho(\theta)$. Representa a curva $\varrho = 2\pi\theta$ con $\theta = 0 \dots 3$ con paso 0.01.
-

```

from matplotlib.pyplot import *
from numpy import *

r = arange(0, 3.0, 0.01); theta = 2*pi*r

# fixo figura cadrada
ancho, alto = matplotlib.rcParams['figure.figsize']
tam = min(ancho, alto); fig = figure(figsize=(tam, tam))
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], polar=True)

ax.plot(theta, r)
grid(True)

ax.set_title("Espiral rho=2*pi*theta", fontsize=15)
show(False)

```

4. Escribe un programa que represente gráficamente en 3D a función $f(x, y) = \sin(\sqrt{x^2 + y^2})$ con $x, y \in [-4, 4]$ con paso 0.25. Código en (grafico3d.py). O resultado sería a figura 4.

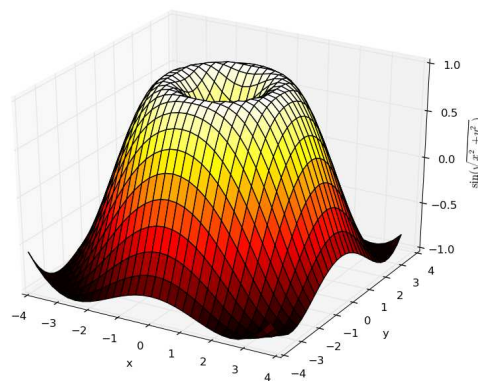


Figura 4: Gráfico 3D da superficie $z = \sin(\sqrt{x^2 + y^2})$.

```

from pylab import *
from mpl_toolkits.mplot3d import Axes3D

fig = figure()
ax = Axes3D(fig) # crear eixos en 3D
X = np.arange(-4, 4, 0.25) # crea vector no eixo x
Y = np.arange(-4, 4, 0.25) # crea vector no eixo y
X, Y = np.meshgrid(X, Y) # xera duas matrices no plano xy
R = np.sqrt(X**2 + Y**2) # calcula o valor da funcion nos puntos do plano
Z = np.sin(R)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot') # representa
ax.set_xlabel('x')

```

```
ax.set_ylabel('y')
ax.set_zlabel(r'\sin(\sqrt{x^2+y^2})')
savefig('grafico3d.png')
show()
```

5. Representa gráficamente a curva en 3D dada polas ecuacións paramétricas $x(t) = \sqrt{t^2 + 1}\sin(t)$, $y(t) = \sqrt{t^2 + 1}\cos(t)$, $z(t) = t$, con $t = -4\pi \dots 4\pi$.
-

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from matplotlib.pyplot import *
from mpl_toolkits.mplot3d import Axes3D
from numpy import *

fig = figure()
ax = fig.gca(projection='3d')

# ecuacions de x(t), y(t), z(t): cambiar para outras ecuacions
t = linspace(-4*pi, 4*pi, 100); r = t**2 + 1
x = r * sin(t)
y = r * cos(t)
z = t

ax.plot(x, y, z, label=u'curva parametrica')
ax.legend()

show(False)
```

6. Representa a animación 3D dada por $z = f(x, y, t) = (1 - \sqrt{x^2 + y^2})\cos(2\pi x + t)$, sendo $x, y \in [-1, 1]$ (50 valores para x e y respectivamente) e $t \in [0, 100]$ (100 valores).
-

```
from mpl_toolkits.mplot3d import axes3d
from matplotlib.pyplot import *
from numpy import *

def f(X, Y, t):
    R = 1 - sqrt(X**2 + Y**2)
    return cos(2 * pi * X + t) * R

ion()
fig = figure()
ax = fig.add_subplot(111, projection='3d')

xs = linspace(-1, 1, 50)
ys = linspace(-1, 1, 50)
X, Y = meshgrid(xs, ys)
Z = f(X, Y, 0.0)

wframe = None
for t in linspace(0, 100, 100):

    oldcol = wframe

    Z = f(X, Y, t) #funcion z=f(x, y, t)
    wframe = ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)

    # Remove old line collection before drawing
```



```

if oldcol is not None:
    ax.collections.remove(oldcol)

draw()

```

7. Escribe un programa que lea dende o arquivo `temperaturas.txt` os seguintes datos de temperaturas diarias nas cidades de Ourense (primeira liña) e Vigo (segunda liña) n mes de agosto:

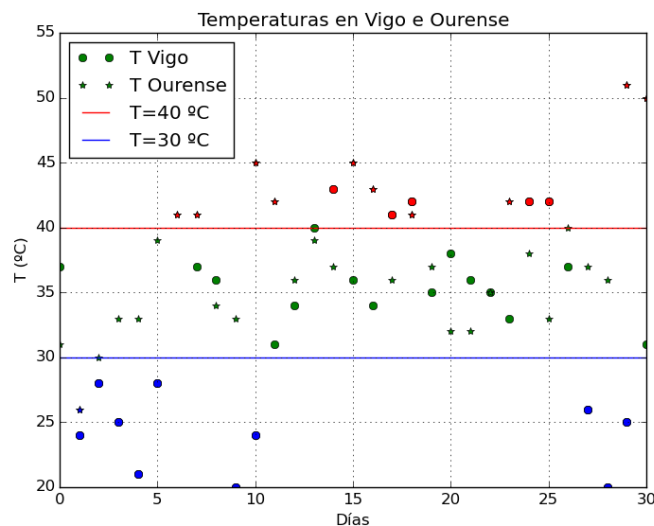
```

31 26 30 33 33 39 41 41 34 33 45 42 36 39 37 45 43 36 41 37 32 32 35 42 38 33 40 37 36 51 50
37 24 28 25 21 28 46 37 36 20 24 31 34 40 43 36 34 41 42 35 38 36 35 33 42 42 37 26 20 25 31

```

O programa debe calcular, para cada cidade: a temperatura media de cada cidade, os días nos que a temperatura foi maior e menor a media, os días nos que a temperatura de Ourense foi superior, inferior e igual a de Vigo e cando se acadou unha temperatura superior a 40 graos Celsius. Representa gráficamente a temperaturas de Ourense en azul (vector x), poñendo en vermello as temperaturas maiores á temperatura media e en negro as temperaturas inferiores a 30 graos. Noutro gráfico, visualiza as temperaturas de Ourense e Vigo x e y , mostrando en vermello as temperaturas superiores a 40 graos, en azul as menores de 30 graos e en verde as que están no intervalo $[30,40]$. O gráfico final na figura 5.

Figura 5: .



```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from sys import * # para abortar o programa se hai erros no arquivo
name=raw_input('Introduce o nome arquivo: ')
try:
    dat=loadtxt(name)
except IOError:
    print 'Erro lendo arquivo: ', name
    exit()
x=dat[0]; y=dat[1] # vectores x e y
mx=mean(x); my=mean(y) # valores medios
print 'Temperatura media en Ourense: ', mx
print 'Temperatura media en Vigo: ', my
print 'Temperaturas superiores a media en Ourense: ', x[x>mx]
print 'Temperaturas inferiores a media en Ourense: ', x[x<mx]

```

```

print 'Temperaturas de Ourense superiores a Vigo: ', x[x>y]
print 'Temperaturas iguais: ', x[x==y]
print 'Temperaturas de Ourense superiores a 40: ', x[x > 40]

from matplotlib.pyplot import *
t=arange(x.size)
figure(1)
plot(t, x, 'b*', label=u'T (C)')
tmx=where(x > mx)[0]
plot(tmx, x[tmx], 'r*', label=u'$T > \mu$')
t30=where(x<30)[0]
v30=30*ones(x.size)
plot(t, v30, 'g-', label=u'T=30')
plot(t30, x[t30], 'ko', label=u'T<30')
legend(loc='upper left')
grid(True)
show(1)
figure(2)
v40=40*ones(x.size)
plot(t, y, 'go', label=u'T Vigo')
plot(t, x, 'g*', label=u'T Ourense')
plot(t, v40, 'r-', label=u'T=40 C')
plot(t, v30, 'b-', label=u'T=30 C')
tt=where(x >40)[0] # posiciones onde x maior a 40
plot(tt, x[tt], 'r*')
tt=where(y>40)[0]
plot(tt, y[tt], 'ro')
tt=where(x <30)[0]
plot(tt, x[tt], 'b*')
tt=where(y<30)[0]
plot(tt, y[tt], 'bo')
grid(True)
legend(loc='upper left')
xlabel(u'Dias')
ylabel(u'T (C)')
title('Temperaturas en Vigo e Ourense')
savefig('graficoanalise.png')
show(2)

```

Traballo a desenvolver pol@ alumn@

1. Representa a curva $f(x) = x^2 e^{-x^2}$ con escala Y logarítmica.
2. Representa a animación 2D $f(x, t) = e^{-xt/10} \sin(10xt)$ con $x \in [0, 1]$ e $t = 1 \dots 100$ s.
3. Representa gráficamente a curva polar $\varrho = \sin(4\theta)$ con $\theta = 0 \dots 2\pi$.
4. Representa a curva paramétrica 3D dada polas ecuacións $x(t) = \cos(t)$, $y(t) = \tan(t)$, $z(t) = \sin(t)$, con $t = 0 \dots 10$ s.
5. Representa a superficie 3D animada $z = f(x, y, t) = \exp(-x^2 - y^2) \sin(t(x^2 + y^2))$ con $x, y \in [-1, 1]$ e $t = 0 \dots 10$ s.
6. Quérese comparar gráficamente a variación do vento de dúas vilas A e B durante a estación do outono. Supón que a estación contén 90 días e que as medidas da velocidade do vento simúlanse xerando aleatoriamente números no rango $[10, 100]$ Km/h. Realiza un programa que:
 - Xere as velocidades do vento e visualice na pantalla a velocidade máxima, mínima e media para ambas vilas.

- Representa nun gráfico de liñas a velocidade do vento nos distintos días para ambas vilas, utilizando o verde para a vila A e o azul para a vila B. Representa como asteriscos a velocidade máxima e con rombos a velocidade mínima para ambas vilas. Pon título, texto nos eixos e lendas axeitadas no gráfico.
- Pon en vermello os días nos que a velocidade do vento na vila A é maior que na vila B.
- Traza unha liña horizontal para a velocidade media do vento en cada vila.

Sesión 11: problemas de termodinámica, óptica e mecánica

Traballo en clase

1. **Ecuación de estado de Van der Waals dun gas non ideal.** A ecuación de estado de Van der Waals (aquí tes un enlace á súa descripción) dun gas non ideal é:

$$\left(P + \frac{a}{V^2}\right)(V - b) = RT \quad (50)$$

onde P é a presión (en bares), V é o volume molar (en l/mol), a (en l² bar/mol²) e b (en l/mol) son dúas constantes propias de cada gas, $R=0.083145$ l · bar/mol · °K é a constante universal dos gases ideais, e T é a temperatura absoluta en °K. Consideremos o gas cloro Cl_2 , para o cal $a=6.579$ l² bar/mol², $b=0.05622$ l/mol. Da ecuación anterior, como T ten que ser positivo, debe ser $V > b$. Por outro lado, despregando P e considerando que debe ser positiva, temos que:

$$P = \frac{RT}{V-b} - \frac{a}{V^2} > 0 \rightarrow T > \frac{a(V-b)}{RV^2} \quad (51)$$

$$\frac{d}{dV} \left[\frac{a(V-b)}{RV^2} \right] = \frac{2abRV - aRV^2}{R^2V^4} = 0 \rightarrow V = 0, V = 2b \quad (52)$$

Esta función é crecente para $V < 2b$ e decrecente para $V > 2b$, polo tanto usaremos $V > V_{min} = 2b$ e $T > T_{min} = a(V_{min}-b)/RV_{min}^2$. No caso do gas Cl_2 temos que $V_{min}=0.11244$ l/mol, e $T_{min}=351.86$ °K. Representa gráficamente: 1) a presión P fronte a V no rango $V_{min} \leq V \leq 0.5$ l/mol (100 valores), unha curva para cada valor de T entre T_{min} e $T_{min} + 50$ °K (5 valores); 2) T como función de V (entre V_{min} e 0.5 l/mol) e P (entre 0 e 80 bar); 2) as liñas isotermas; e 3) a temperatura T como un mapa de calor. Tes que obter as gráficas da figura 6:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from pylab import *

R=0.083145
a,b=6.579,0.05622 # gas cloro Cl2

Vmin=2*b;Tmin=a*(Vmin-b)/(R*Vmin**2)

nV=100;V=linspace(Vmin,0.5,nV)
nT=5;T=linspace(Tmin,Tmin+50,nT)
P=zeros([nT,nV])

figure(1);clf()
for t in range(nT):
    P[t]=R*T[t]/(V-b)-a/V**2
    s=u'%.2f K' % T[t]
    plot(V,P[t],label=s)
grid(True);xlabel('Volume molar (l/mol)')
ylabel(u'Presion (bar)')
title(u'Ecuacion de Van der Waals - Gas Cloro')
legend(loc='lower right');show(False)
savefig('vanderwaals1.png')

# temperatura funcion de V e P
```

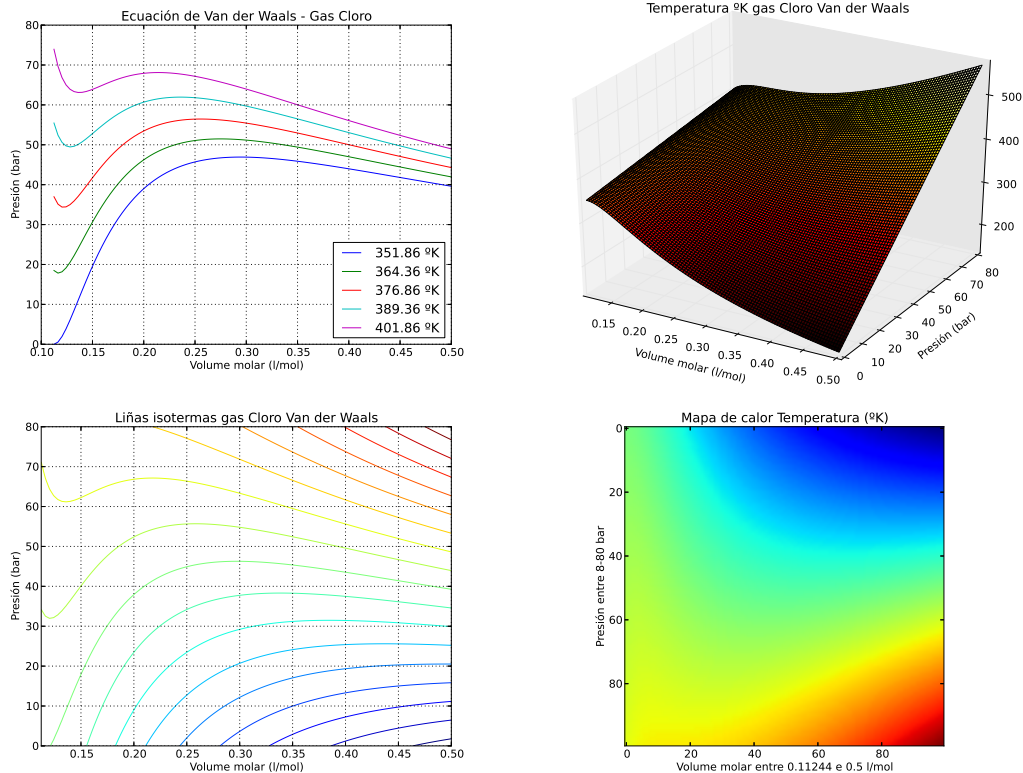


Figura 6: Presión frente a volume, temperatura frente a presión e volume, líneas isotermas e mapa de calor da temperatura frente a presión e volume dadas polas ecuacións de Van der Waals do gas Cl_2 .

```
def superficie3d(X,Y,Z,id,xlab,ylab,titulo):
    from mpl_toolkits.mplot3d import Axes3D
    fig = figure(id);clf();ax = Axes3D(fig)
    xlabel(xlab);ylabel(ylab);title(titulo)
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')
    show(False)

P=linspace(0,80,100);
V2,P2=meshgrid(V,P)
T2=(P2+a/V2**2)*(V2-b)/R
superficie3d(V2,P2,T2,2,'Volume molar (l/mol)',u'Presion (bar)',\
    u'Temperatura K gas Cloro Van der Waals')
savefig('vanderwaals2.png')

# contorno linhas isotermas
def contorno(X,Y,Z,id,nl,xlab,ylab,titulo):
    figure(id);clf();contour(X,Y,Z,nl)
    xlabel(xlab);ylabel(ylab);title(titulo);grid(True)
    show(False)

contorno(V2,P2,T2,3,20,'Volume molar (l/mol)',u'Presion (bar)',\
    u'Linhas isotermas gas Cloro Van der Waals')
savefig('vanderwaals3.png')

# mapa de calor
def mapa_calor(Z,id,xlab,ylab,titulo):
    figure(id);clf();imshow(Z);xlabel(xlab);ylabel(ylab)
```

```

title(titulo);show(False)

mapa_calor(T2,4,'Volume molar entre %g e 0.5 l/mol' % Vmin,\
u'Presion entre 0 e 80 bar',u'Mapa de calor Temperatura (K)')
savefig('vanderwaals4.png')

```

2. **Mecánica: traballo realizado por unha forza nunha traxectoria.** Consideremos a forza $\mathbf{F}(\mathbf{r})$ e o traballo W que xera sobre unha traxectoria $\mathbf{r}(t)$ entre dous puntos \mathbf{r}_1 e \mathbf{r}_2 :

$$W = \int_{\mathbf{r}_1}^{\mathbf{r}_2} \mathbf{F}(\mathbf{r}) \cdot d\mathbf{r} \quad (53)$$

Consideremos unha traxectoria $\mathbf{r}(t) = x(t)\mathbf{i} + y(t)\mathbf{j}$, e unha forza $\mathbf{F} = F_x(x, y)\mathbf{i} + F_y(x, y)\mathbf{j}$, onde x e y son as coordenadas dun punto. Aproximando a integral como unha suma discreta, e considerando os vectores $\mathbf{x} = (x_0, \dots, x_{n-1})$ e $\mathbf{y} = (y_0, \dots, y_{n-1})$ coas coordenadas da traxectoria, o traballo pódese escribir da seguinte forma:

$$W = \sum_{i=1}^{n-1} F_x(x_i, y_i)(x_i - x_{i-1}) + F_y(x_i, y_i)(y_i - y_{i-1}) \quad (54)$$

onde $d\mathbf{r} = dx\mathbf{i} + dy\mathbf{j} \simeq \Delta x\mathbf{i} + \Delta y\mathbf{j} = (x_i - x_{i-1})\mathbf{i} + (y_i - y_{i-1})\mathbf{j}$. Escribe un programa que calcule o traballo realizado pola forza $\mathbf{F}(\mathbf{r}) = x^2y\mathbf{i} + xy^2\mathbf{j}$ Nw. ao longo da traxectoria dada polas ecuacións paramétricas $x(t) = \rho \cos t$, $y(t) = \rho \sin t$, con $\rho = 1$ m., $t = 0 \dots 2\pi$ s. e $\Delta t = 0.1$ s. O programa tamén debe mostrar a traxectoria $\{x_i, y_i\}_i$ en 2D e representar o módulo $|\mathbf{F}| = \sqrt{F_x^2 + F_y^2}$ da forza en 3D e como mapa de cor en 2D. Debes obter as gráficas da figura 7.

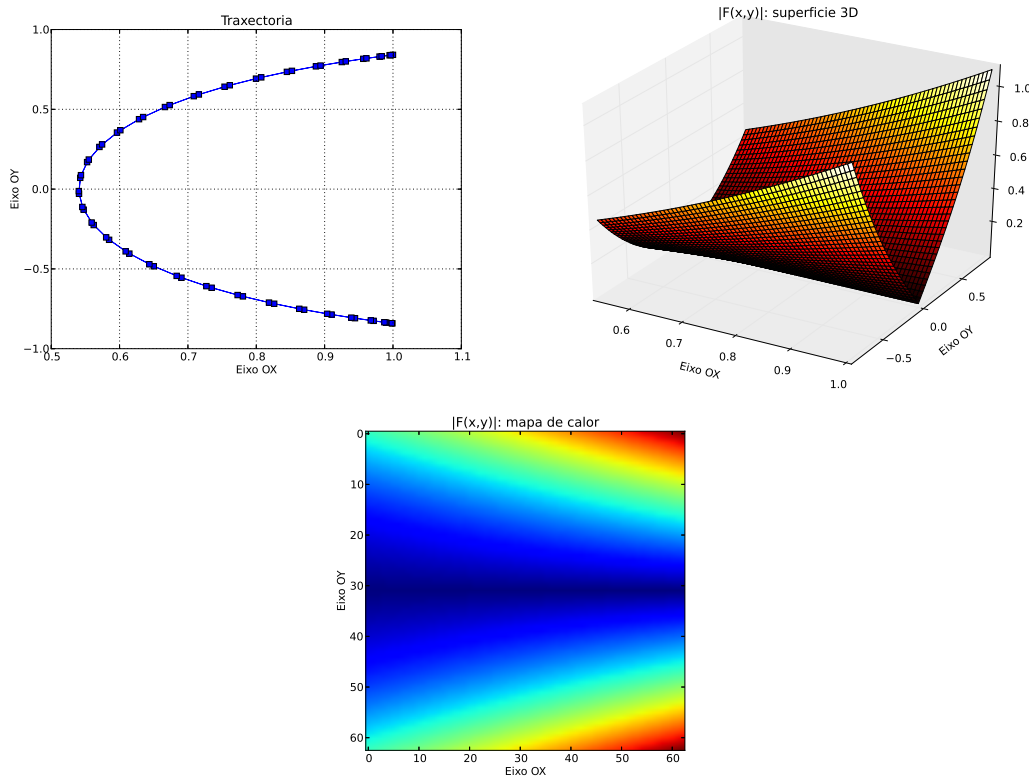


Figura 7: Traxectoria $\mathbf{r}(t)$, módulo da forza $|F(x, y)|$ como superficie en 3D e como mapa de calor no exercicio do traballo.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from pylab import *
from mpl_toolkits.mplot3d import Axes3D

def forza(x,y):
    f=[x**2*y,x*y**2]
    return f

def modulo_forza(x,y):
    f=forza(x,y);mf=sqrt(f[0]*f[0]+f[1]*f[1])
    return mf

def traxectoria(t0,t1,dt):
    rho=1;t=arange(t0,t1,dt)
    x=rho*cos(sin(t));y=rho*sin(cos(t))
    return [x,y]

r=traxectoria(0,2*pi,0.1)
f=forza(r[0],r[1])

w=0;n=shape(r)[1];x=r[0];y=r[1]
for i in range(1,n):
    w=w+f[0][i]*(x[i]-x[i-1])+f[1][i]*(y[i]-y[i-1])
print 'W= %g xulios' % w

figure(1);clf();plot(x,y,'s-')
xlabel('Eixo OX');ylabel('Eixo OY')
title('Traxectoria');grid(True)
show(False);savefig('trabalho1.eps')

def superficie3d(x,y,z,id):
    fig=figure(id);clf();ax=Axes3D(fig)
    xlabel('Eixo OX');ylabel('Eixo OY')
    title('F(x,y): superficie 3D')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')
    show(False)

xx=linspace(min(x),max(x),n);yy=linspace(min(y),max(y),n)
X,Y=meshgrid(xx,yy);Z=modulo_forza(X,Y);superficie3d(X,Y,Z,2)
savefig('trabalho2.eps')

figure(3);clf();imshow(Z);xlabel('Eixo OX')
ylabel('Eixo OY');title('F(x,y): mapa de calor');show(False)
savefig('trabalho3.eps')
```

Trabalho a desenvolver pol@ alumn@

1. **Termodinámica: cálculo de enerxía interna dun sistema.** Un sistema gaseoso dun mol cunha soa compoñente experimenta un proceso adiabático (é dicir, sen transferencia de calor) mediante o cal recibe un traballo dado por $dW = \frac{dV}{V} + V^{5/3}dP$, onde o volume V e a presión P mídense en litros e bares respectivamente. Calcula e representa gráficamente, como superficie e como mapa de calor (usa unha función de Python distinta para xerar cada gráfica), a enerxía interna $U(P,V)$ do sistema para $1 \leq P \leq 50$ bar con $\Delta P=2$ bar e $1 \leq V \leq 10$ l. con $\Delta V=0.5$ l.

Posteriormente, realízase unha transferencia de calor a volume constante dada por $\frac{dQ}{dt} = e^{-t}$, ao tempo que se segue aportando o traballo dW do apartado anterior, e un axitador aumenta a presión a volume constante sobre o sistema de modo que $\frac{dP}{dt} = (2t - t^2)e^{-t}$. Calcula e representa gráficamente a enerxía interna U en función do tempo t para $V = 1$ l. e $0.1 \leq t \leq 10$ s. con paso $\Delta t = 0.5$ s.

SOLUCIÓN. Polo primeiro principio da termodinámica, temos que $dU = dQ + dW$. No proceso adiabático, $dQ = 0$ de modo que $U = \int dU = \int dW$. Temos que:

$$U = \int dU = \int dW = \int \frac{dV}{V} + \int V^{5/3} dP = \ln V + PV^{5/3} = U(P, V) \quad (55)$$

No segundo proceso, séguese aportando traballo dW , pero agora a volume constante $V = \text{cte} = 1$ l. e polo tanto $dV = 0$, e como se aporta calor dQ , traballo dW polo tanto:

$$dQ = e^{-t} dt, \quad dW = dP = (2t - t^2)e^{-t} dt \quad (56)$$

$$U = \int dQ + \int dW = \int e^{-t} dt - \int (2t - t^2)e^{-t} dt = -e^{-t} + t^2 e^{-t} = (t^2 - 1)e^{-t} = U(t) \quad (57)$$

O programa fica así:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from pylab import *

# U funcion de V e P
def superficie3d(X,Y,Z,id,xlab,ylab,titulo):
    from mpl_toolkits.mplot3d import Axes3D
    fig = figure(id);clf();ax = Axes3D(fig)
    xlabel(xlab);ylabel(ylab);title(titulo)
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')
    show(False)

p=arange(1,50,2);v=arange(1,10,0.5)
V,P=meshgrid(v,p);U=log(V)+P**2*V**(5/3)/2

superficie3d(V,P,U,1,'Volume V (l)',u'Presion P (bar)',\
    u'Enerxia interna U (Xulios)')
savefig('enerxia1.eps')

def mapa_calor(Z,id,xlab,ylab,titulo):
    figure();clf();imshow(Z);xlabel(xlab);ylabel(ylab)
    title(titulo);show(False)

mapa_calor(U,2,'Volume (l)',u'Presion (bar)',\
    'Mapa de calor U(V,P)')
savefig('enerxia2.eps')

figure();clf() # representa U(t)
t=arange(0.1,10,0.5);U=(t**2-1)*exp(-t);plot(t,U,'sb-')
xlabel('tempo (s.)');ylabel('U(t) (Xulios)')
title(u'Enerxia interna U(t)');grid(True);show(False)
savefig('enerxia3.eps')
```

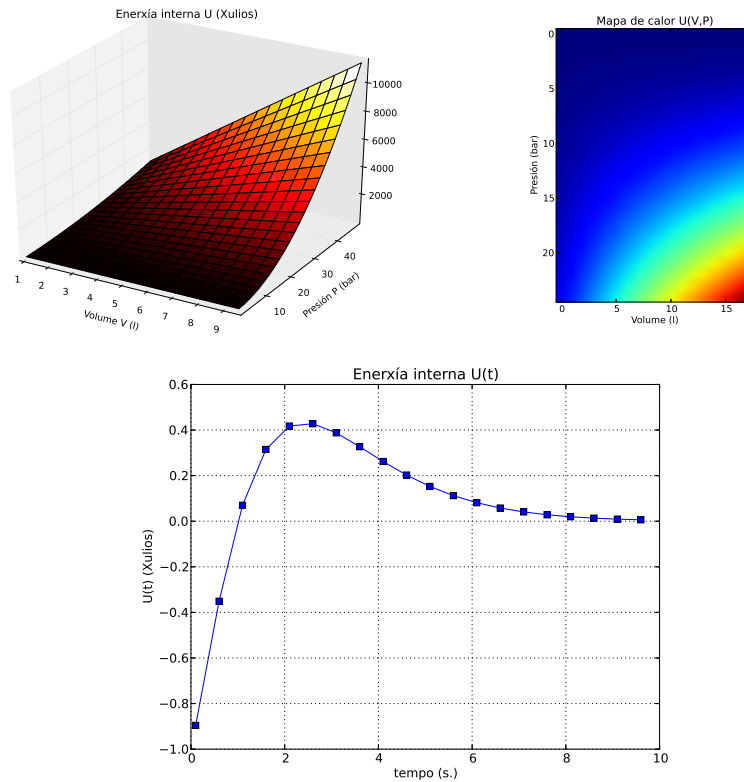


Figura 8: Enerxía interna U en función da presión e volume en 3D e como mapa de calor, e enerxía interna como función do tempo.

Tes que obter as gráficas na figura 8:

2. **Óptica: cálculo e representación gráfica das perturbacións provocadas por pulsos cortos de luz láser.** Esta perturbación ξ ten a seguinte expresión (ver R.W. Ditchburn, Óptica, páxs. 719-720):

$$\xi(t, z) = \sqrt{\frac{a^2 \operatorname{sen}^2\left(\frac{\pi N c \Delta t}{2L}\right)}{\operatorname{sen}^2\left(\frac{\pi c \Delta t}{2L}\right)}}, \quad \Delta t = t - \frac{nz}{c} \quad (58)$$

onde $a=1$ é a amplitude do pulso, $N=49$ é o número de modos menos 1, $c=300000$ km/s é a velocidade da luz, $L=100$ mm é a lonxitude do láser, $n=1.4$ é o índice de refracción do medio e z é a coordenada espacial. Representa gráficamente nun mapa de calor $\xi(t, z)$ con 50 valores de t e z entre 1 e 10 s., e entre 0 e 100 mm, respectivamente. Tes que obter a gráfica da figura 9:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from pylab import *

a=1; #amplitude da luz
N=49 #N=(no. de modos)-1
nm=1.4 #indice de refraccion do medio
c=3E8 # velocidade da luz
Lt=10; nt=50 #intervalo temporal e no. de valores temporais
Lz=100; L2=2*Lz; nz=50 # lonxitude do laser e no. de puntos
```

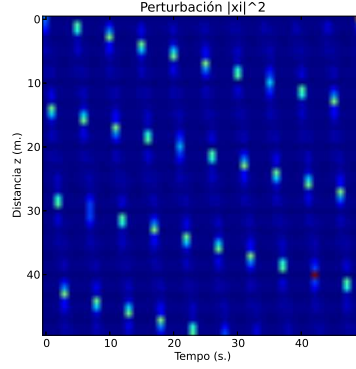


Figura 9: Perturbación $\xi(t, z)$ en función do tempo e da distancia z .

```
t=linspace(1,Lt,nt);z=linspace(0,Lz,nz);
T,Z=meshgrid(t,z);dt=T-nm*Z/c
xi=sqrt(a**2*sin(pi*N*c*dt/L2)**2/sin(pi*c*dt/L2)**2)

clf();imshow(xi);xlabel('Tempo (s.)');ylabel('Distancia z (m.)');
title(u'Perturbacion |xi|^2');show(False);savefig('laser.eps')
```

3. **Centro de masas dun sistema de partículas.** O centro de masas (ou de gravidade) \mathbf{c} dun sistema de partículas (ou dunha distribución de masa) está dado, para unha distribución discreta (é dicir, un sistema de n partículas de masas $\{m_i\}_1^n$ e posicións $\{\mathbf{r}_i\}_1^n$), e para unha distribución de masa continua m , polas seguintes expresións:

$$\mathbf{c} = \frac{\sum_{i=1}^n m_i \mathbf{r}_i}{\sum_{i=1}^n m_i} = \frac{1}{M} \sum_{i=1}^n m_i \mathbf{r}_i, \quad \mathbf{c} = \frac{\int \mathbf{r} dm}{\int dm} \quad (59)$$

No caso continuo, dado que $dm = \rho(\mathbf{r})dV$, temos que:

$$\mathbf{c} = \frac{\int_V \mathbf{r} \rho(\mathbf{r}) dV}{\int_V \rho(\mathbf{r}) dV} \quad (60)$$

Nos casos uni-, bi- e tri-dimensional, temos:

$$\mathbf{c} = \frac{\int x \rho(x) dx}{\int \rho(x) dx}, \quad \mathbf{c} = \frac{\iint_S (x\mathbf{i} + y\mathbf{j}) \rho(x, y) dx dy}{\iint_S \rho(x, y) dx dy}, \quad \mathbf{c} = \frac{\iiint_V (x\mathbf{i} + y\mathbf{j} + z\mathbf{k}) \rho(x, y, z) dx dy dz}{\iiint_V \rho(x, y, z) dx dy dz} \quad (61)$$

A aproximación discreta a estas integrais son:

$$\mathbf{c} = \frac{\sum_{i < n} x_i \rho_i}{\sum_{i < n} \rho_i}, \quad \mathbf{c} = \frac{\sum_{i < n} \sum_{j < n} (x_i \mathbf{i} + y_j \mathbf{j}) \rho_{ij}}{\sum_{i < n} \sum_{j < n} \rho_{ij}}, \quad \mathbf{c} = \frac{\sum_{i < n} \sum_{j < n} \sum_{k < n} (x_i \mathbf{i} + y_j \mathbf{j} + z_k \mathbf{k}) \rho_{ijk}}{\sum_{i < n} \sum_{j < n} \sum_{k < n} \rho_{ijk}} \quad (62)$$

Escribe un programa que calcule o centro de masas de:

- a) Lea un número enteiro n por teclado (proba con $n = 3$) e calcule aleatoriamente coordenadas $\{x_i, y_i, z_i\}_1^n$ con valores entre -10 e $+10$, para un sistema de n partículas con masas $\{m_i\}_1^n$ con $1 \leq m_i \leq 10$ kg.
- b) Calcule o centro de masas c para este sistema discreto considerando so as coordenadas $\{x_i\}_1^n$ (caso unidimensional).
- c) Calcule o centro de masas $\mathbf{c} = (c_x, c_y)$ para este sistema considerando as coordenadas $\{x_i, y_i\}_1^n$ (caso discreto bidimensional).
- d) Calcule o centro de masas $\mathbf{c} = (c_x, c_y, c_z)$ para este sistema considerando as coordenadas $\{x_i, y_i, z_i\}_1^n$ (caso discreto tridimensional).
- e) Unha distribución continua unidimensional dada por $\rho(x) = \frac{\arccos(x)}{1+x^2}$ con $-1 \leq x, y \leq 1$ e $\rho(x, y) = 0$ noutro caso (debes obter $c = -0.207$).
- f) Unha distribución continua bidimensional dada por $\rho(x, y) = \exp(-(x-1)^2 - (y-1)^2)$ con $-1 \leq x \leq 3$ (debes obter $\mathbf{c} = (1, 1)$).
- g) Unha distribución continua tridimensional dada por $\rho(x, y, z) = e^{-r^2}(1 + \sin 10r)$, sendo $r = \sqrt{x^2 + y^2 + z^2}$, con $-2 < x, y, z < 2$

Tes que obter as gráficas da figura 10

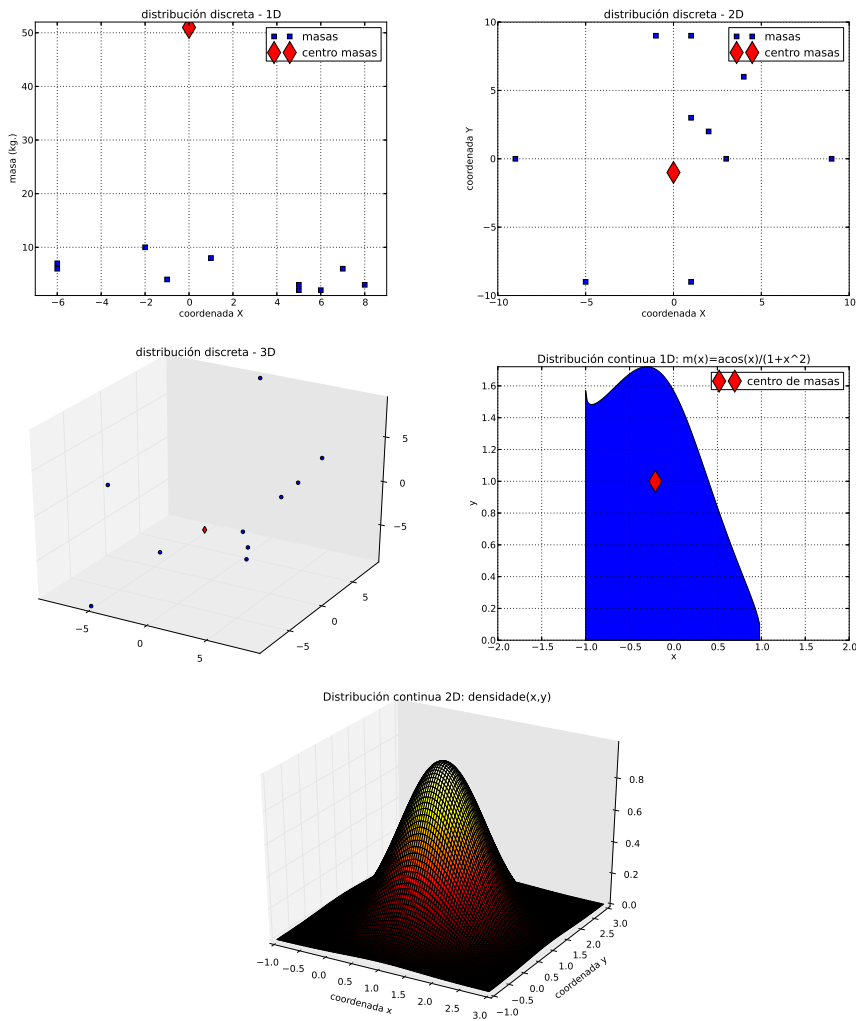


Figura 10: Centro de masas dunha distribución discreta 1D, 2D e 3D, e continua 1D e 2D.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from pylab import *
from mpl_toolkits.mplot3d import Axes3D

n=input('numero de particulas? ')
x=zeros(n);y=zeros(n);z=zeros(n);m=zeros(n)
for i in range(n):
    x[i]=int(-10+20*random());y[i]=int(-10+20*random())
    z[i]=int(-10+20*random());m[i]=int(1+10*random())
M=sum(m)

#-- caso unidimensional -----
c=sum(x*m)/M
print 'discreta 1D: c=',c
figure(1);clf()
plot(x,m,'sb',label='masas')
plot(c,M,'dr',markersize=20,label='centro masas')
axis([min(x)-1,max(x)+1,min(m)-1,sum(m)+1])
xlabel('coordenada X');ylabel('masa (kg)')
legend(loc='upper right')
title(u'distribucion discreta - 1D');grid(True);show(False)
savefig('centro_masas_discreto_1d.eps')

#-- caso bidimensional -----
c=[sum(x*m),sum(y*m)]/M
print 'discreta 2D: c=',c
figure(2);clf();plot(x,y,'sb',label='masas')
plot(c[0],c[1],'dr',markersize=20,label='centro masas')
tx=hstack([x,c[0]]);ty=hstack([y,c[1]])
axis([min(tx)-1,max(tx)+1,min(ty)-1,max(ty)+1])
xlabel('coordenada X');ylabel('coordenada Y')
legend(loc='upper right')
title(u'distribucion discreta - 2D');grid(True);show(False)
savefig('centro_masas_discreto_2d.eps')

#-- caso tridimensional -----
c=[sum(x*m),sum(y*m),sum(z*m)]/M
print 'discreta 3D: c=',c
f=figure(3);ax=Axes3D(f)
ax.scatter(x,y,z)
ax.scatter(c[0],c[1],c[2],s=40,c='r',marker='d')
legend(loc='upper right')
title(u'distribucion discreta - 3D');show(False)
savefig('centro_masas_discreto_3d.eps')

#distribucion continua 1-D-----
a=-1.;b=1.;n=100;h=(b-a)/n;x=[];y=[];t=a
numer,denom=0,0 #n=numero de puntos no intervalo
while t<b:
    u=arccos(t)/(1+t**2);
    x.append(t);y.append(u)
    numer+=t*u;denom+=u;t=t+h
c=numer/denom
print 'continua 1D: c=',c
```

```

figure(4);clf();
fill_between(x,y)
plot(c,1,'dr',markersize=20,label='centro de masas')
axis([-2,2,0,max(y)])
xlabel('x');ylabel('y')
title(u'Distribucion continua 1D:  $m(x)=\cos(x)/(1+x^2)$ ')
legend(loc='upper right')
grid(True);show(False)
savefig('centro_masas_continuo_1d.eps')

#distribucion continua 2-D-----
a=-1.;b=3.;n=100;x=linspace(a,b,n);y=x.copy();c=zeros(2)
X,Y=meshgrid(x,y);M=exp(-(X-1)**2-(Y-1)**2);denom=0;px=0;py=0
for i in range(n):
    tx=x[i]
    for j in range(n):
        ty=y[j];t=exp(-(tx-1)**2-(ty-1)**2)
        px+=tx*t;py+=ty*t;denom+=t
c=[px,py]/denom
print 'continua 2D: c=',c
f=figure(5);clf();ax=Axes3D(f)
ax.plot_surface(X,Y,M, rstride=1, cstride=1, cmap='hot')
ax.scatter(c[0],c[1],0,s=40,c='r',marker='s')
xlabel('coordenada x');ylabel('coordenada y')
title(u'Distribucion continua 2D: densidade(x,y)')
show(False);savefig('centro_masas_continuo_2d.eps')

#distribucion continua 3-D-----
denom=0;px=0;py=0;pz=0
a=-2.;b=2.;n=100;x=linspace(a,b,n)
y=x.copy();z=x.copy();c=zeros(3)
for i in range(n):
    tx=x[i]
    for j in range(n):
        ty=y[j];
        for k in range(n):
            tz=z[k];r=sqrt(tx**2+ty**2+tz**2)
            t=exp(-r**2)*(1+sin(10*r))
            px+=tx*t;py+=ty*t;pz+=tz*t;denom+=t
c=[px,py,pz]/denom
print 'continua 3D: c=',c

```

Sesión 12: Paquetes SymPy para cálculo simbólico. Métodos numéricos.

Traballo en clase

1. Realización de cálculos simbólicos usando SymPy. Podes consultar a documentación de SymPy en <http://docs.sympy.org>:

```
>>> from sympy import *
>>> x,a,n = symbols('x a n')
>>> init_printing(use_unicode=True) # mostra as respostas en 2D.
>>> limit(sin(a*x)/x, x, 0) # limite en x=0 de sin(ax)/x
a
>>> limit(n**2/(n**2+1),n,oo) # limite dunha sucesion: n^2/(n^2+1)
1
>>> diff(exp(x**2), x) # deriva exp(x^2)
2*x*exp(x**2)
>>> integrate(exp(-x**2), x) # integra exp(-x^2)
pi**(1/2)*erf(x)/2
>>> integrate(exp(-x**2), (x, 0, oo)) #integral definida de exp(-x^2) en [0,inf)
pi**(1/2)/2
>>> solve(a**2*x**2/(a**2 + x**2)-1,x) # resolve a ecuacion a^2 x^2/(a^2 + x^2) = 1
[-(a**2/((a - 1)*(a + 1)))**(1/2), (a**2/((a - 1)*(a + 1)))**(1/2)]
# se solve() retorna [] ou activa unha excepcion NotImplementedError
# isto non significa que non existan soluciones, simplemente non as atopou.
>>> b,c=symbols('b c')
>>> solve(a*x**2+b*x+c, x)
[(-b + sqrt(-4*a*c + b**2))/(2*a), -(b + sqrt(-4*a*c + b**2))/(2*a)]
>>> s=solve(a*x**2+b*x+c, x)
>>> s[0]
(-b + sqrt(-4*a*c + b**2))/(2*a)
>>> s[1]
-(b + sqrt(-4*a*c + b**2))/(2*a)
>>> s[0].subs([(a,1), (b,1), (c,1)])
-1/2 + sqrt(3)*I/2
>>> s[0].subs([(a,1), (b,1), (c,1)]).evalf()
-0.5 + 0.866025403784439*I
>>> expr = exp(sin(x))
>>> expr.series(x, 0, 10) #expande sin(x) en serie de Taylor de orde 10 a en torno a 0
1 + x + x**2/2 - x**4/8 - x**5/15 - x**6/240
+ x**7/90 + 31*x**8/5760 + x**9/5670 + O(x**10)
>>> series(exp(sin(x)), x, 0, 10)
1 + x + x**2/2 - x**4/8 - x**5/15 - x**6/240
+ x**7/90 + 31*x**8/5760 + x**9/5670 + O(x**10)
>>> Sum(1/n**2, (n,1,oo)) # deixa indicada a suma dunha serie
Sum(n**(-2), (n, 1, oo))
>>> Sum(1/n**2, (n,1,oo)).doit() # calcula a suma da serie
pi**2/6
>>> Sum(1/n**2,(n,1,oo)).evalf() # calcula sum_{n=1}^inf (1/n^2) con numeros decimais
1.64493406684823
>>> M = Matrix([[1, 0, 1, 3], [2, 3, 4, 7], [-1, -3, -3, -4]]) # define unha matriz
>>> M
[ 1,  0,  1,  3]
[ 2,  3,  4,  7]
```

```

[-1, -3, -3, -4]
>>> M.rref() # convierte en triangular superior como no metodo de eliminacion gaussiana
([1, 0, 1, 3]
 [0, 1, 2/3, 1/3]
 [0, 0, 0, 0], [0, 1])
>>> M = Matrix([[3, -2, 4, -2], [5, 3, -3, -2], [5, -2, 2, -2], [5, -2, -3, 3]])
>>> M.eigenvals() # calcula autovalores
{-2: 1, 3: 1, 5: 2} # autovalor -2 con multiplicidade 1, ...
>>> M.eigenvects()
[(3, 1, [[1] # autovector [1,1,1,1] asociado a autovalor 3 con multiplicidade 1
 [1]
 [1]
 [1]])],
 (-2, 1, [[0]
 [1]
 [1]
 [1]])],
 (5, 2, [[1]
 [1]
 [1]
 [0], [ 0]
 [-1]
 [ 0]
 [ 1]]])]
>>> P, D = M.diagonalize() # diagonaliza M, calculando as matrices P, D que verifican  $M=PDP^{-1}$ 
>>> P # matriz de permutacions
[1, 0, 1, 0]
[1, 1, 1, -1]
[1, 1, 1, 0]
[1, 1, 0, 1]
>>> D # matriz diagonal
[3, 0, 0, 0]
[0, -2, 0, 0]
[0, 0, 5, 0]
[0, 0, 0, 5]
>>> P*D*P**-1 == M # comprueba que  $PDP^{-1} = M$ 
True
>>> p=M.charpoly(x)
>>> p # polinomio caracteristico de M
Poly(x**4 - 11*x**3 + 29*x**2 + 35*x - 150, x, domain='ZZ')
>>> factor(p) # factoriza p en factores primos
(x - 5)**2*(x - 3)*(x + 2)
>>> roots(p) # raices dun polinomio como
{-2: 1, 3: 1, 5: 2}
>>> roots(x**4-4)
{-sqrt(2)*I: 1, sqrt(2): 1, -sqrt(2): 1, sqrt(2)*I: 1}
>>> zeros(3) # matriz de ceros
>>> Matrix([
[0, 0, 0],
[0, 0, 0],
[0, 0, 0]])
>>> zeros(2,4)
>>> Matrix([
[0, 0, 0, 0],
[0, 0, 0, 0]])
>>> ones(2,3) # matriz de uns
>>> Matrix([

```

```

[1, 1, 1],
[1, 1, 1]])
>>> eye(3) # matriz identidad
>>> Matrix([
[1, 0, 0],
[0, 1, 0],
[0, 0, 1]])
>>> f = lambdify(x, x**2) # convierte unha expresion nunha funcion lambda
>>> f(2)
4
>>> f = lambdify(x, sqrt(x))
>>> f(4)
2.0
>>> f = lambdify((x, y), sin(x*y)**2)
>>> f(0, 5)
0.0

```

2. **Uso de sympy dentro de programas:** Escribe un programa que pida ó usuario a expresión dunha función $f(x)$ e unha lista de valores (vector) x e calcula o valor de $f(x)$ e $f'(x)$ en todos os puntos de x utilizando o módulo sympy.

```

#!/usr/bin/python
#-*- coding: utf-8 -*-
# calcula valor de f(x) and d(f(x))/dx nos puntos dunha lista

import numpy as np # importar numpy con un alias
from sympy import *
expresion=raw_input('f(x)= ')
p=input('Lista con puntos: ')
# convierte string a obxecto sympy
fx=sympify(expresion)
# convierte a funcion lambda
x=symbols('x')
f=lambdify(x, fx)
y=np.array(p)
print "f(", p, ")= ", f(y)
# calcula a derivada
df=diff(fx,x)
n=len(p)
z=np.zeros(n)
for i in range(n):
    # calcula derivada nun punto
    z[i]=df.subs(x, p[i])

```

Necesítase cargar en memoria o módulo numpy con un alias para determinar se se quere utilizar a función zeros() de numpy ou a do módulo sympy. Outra alternativa sería:

```

#!/usr/bin/python
#-*- coding: utf-8 -*-
# calcula valor de f(x) and d(f(x))/dx nos puntos dunha lista
import numpy as np # importar numpy con un alias
from sympy import *
expresion=raw_input('f(x)= ')
p=input('Lista con puntos: ')
# convierte string a funcion lambda
f=eval('lambda x: '+expresion)
y=np.array(p)
print "f(", p, ")= ", f(y)

```

```
# convirte string a obxeto sympy
fx=sympify(expresion)
# calcula a derivada
x=symbols('x')
df=diff(fx,x)
dfx=lambdify(x, df)
print "df(", p, ")=", dfx(y)
```

A función `lambdify()` convirte unha expresión de `sympy` nunha función `lambda`. O ler a expresión do teclado, non é un obxeto de `sympy` senon unha simple cadea de caracteres. Polo tanto, hai que convertila a función `lambda` utilizando a función `eval()`.

3. **Resolución de ecuacións non lineares polo Método de Newton. Bucle híbrido definido-indefinido. Derivación simbólica.** O método de Newton resolta unha ecuación non linear da forma $f(x) = 0$ (con f non linear). Para isto, partimos dun punto x_0 e trazamos a recta tanxente á curva $f(x)$ en $x = x_0$. Esta recta tanxente ten a ecuación $y = mx + n$, onde $m = f'(x_0)$ por ser a recta tanxente a $f(x)$ en $x = x_0$. Pola outra banda, o feito de que a curva $f(x)$ e a recta $y = x f'(x) + n$ intersecten no punto $(x_0, f(x_0))$ fai que este punto cumpra a ecuación da recta, de modo que temos que $f(x_0) = x_0 f'(x_0) + n \Rightarrow n = f(x_0) - x_0 f'(x_0)$, e a ecuación da recta fica así:

$$y(x) = f(x_0) + f'(x_0)(x - x_0) \quad (63)$$

Para atopar o punto x_1 de corte desta recta co eixo horizontal, abonda con impor a condición $y(x_1) = 0$ e atopamos:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (64)$$

Repetindo o proceso iterativamente, de modo que a partir dun punto x_i calculamos o novo punto x_{i+1} , aproximámonos a unha solución x^* que verifica $f(x^*) = 0$ (se existe), a non ser que para algún valor x_i teñamos $f'(x_i) = 0$. Podes atopar unha ilustración animada do proceso de búsqueda iterativa da solución neste **enlace**. Escribe un programa que, partindo dun punto inicial x_0 , calcule os puntos sucesivos x_i empregando a seguinte fórmula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (65)$$

Esta operación iterativa deberá executarse ata que $|x_i - x_{i-1}| < 10^{-5}$. Almacena os valores $(x_i, f(x_i))$, $i = 1, \dots$ no ficheiro `datos.dat` (un par en cada liña).

EXEMPLO 1: dada a ecuación $x^3 - x^2 - x + 1 = 0$ ten raíces en $x = 1$ (dobre) e $x = -1$ (simple). Neste caso, $n = 3$. Probar con $x_0 = 2$ (para calcular a raíz $x = 1$) e con $x_0 = -3$ (para calcular a raíz $x = -1$).

EXEMPLO 2: a ecuación $x e^{-x} - 0.2 = 0$ ten solución $x = 2.54264$ (proba con $x_0 = 2$).

EXEMPLO 3: a ecuación $x^2 = 0$ con $x_0 = 0$ da derivada nula e non converge.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from sys import exit
from sympy import *
from math import *
expresion=raw_input('f(x)= ')
xi=input('x0= ')
niter=input('No. maximo iteracions: ')
x=symbols('x')
f=lambdify(x, sympify(expr))
# alternativa
# f=eval('lambda x: '+expresion)
```

```

df=lambdify(x, diff(expresion))
for i in range(niter):
    fx=f(xi)
    dfx=df(xi)
    if dfx == 0:
        exit('Erro: f\'(%g)=0: proba cun x0 distinto ' % xi)
    xi1= xi - float(fx)/dfx
    print 'i= %d xi1= %.6f xi=%.6f ' % (i, xi1,xi)
    if fabs(xi1-xi) < 1e-5:
        break
    xi=xi1

if i < niter:
    print 'Converxeu: x=%g f=%g en %d iteracions ' % (xi, fx, i)
else:
    print 'O metodo de newton non converxeu '

```

O módulo `scipy.optimize` contén a función `newton` para calcular o cero dunha función utilizando o método de newton. O código sería:

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import *
from scipy.optimize import *
expresion=raw_input('f(x)= ')
xo=input('x0= ')
f=eval('lambda x :'+expresion)
print 'x= ', newton(f, xo)

```

4. Resolución numérica dun sistema de ecuacións diferenciais. **Movimento dunha partícula nun campo de forzas no plano.** Consideremos unha partícula de masa 1 kg. que se atopa no campo de forzas $\mathbf{F}(x,y) = (x \sin(r-1), y \sin(r-1))$, onde $r = \sqrt{x^2 + y^2}$. A súa posición $\mathbf{r}(t) = (x(t), y(t))$ e a súa velocidade $\mathbf{v}(t) = (v_x(t), v_y(t))$ están dadas polas seguintes ecuacións diferenciais:

$$\frac{dx}{dt} = v_x, \quad \frac{dy}{dt} = v_y, \quad \frac{dv_x}{dt} = x \sin(r-1), \quad \frac{dv_y}{dt} = y \sin(r-1) \quad (66)$$

A súa posición e velocidade iniciais están dadas por $\mathbf{r}(0) = (0, 1)$ m. e $\mathbf{v}(0) = (-1, 0)$ m/s para $t = 0$ s. Representa gráficamente a posición $\mathbf{r}(t)$ para $0 \leq t \leq 17$ s. (50 valores): 1) usando integración directa; 2) usando o método de Runge-Kutta; e 3) usando a función `odeint` do paquete `scipy`.

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
from matplotlib.pyplot import *
from scipy.integrate import *
from time import clock

def derivada(x):
    n = x.size #x[0]=coord. x; x[1]=coord. y; x[2]=v_x; x[3]=v_y
    if (n!=4):
        print 'erro: o numero de dimensions non coincide'
        quit()
    r=sqrt(x[0]*x[0]+x[1]*x[1])
    #fx=-k*x[0]/r**3; fy=-k*x[1]/r**3

```

```

    fx=sin(r-1)*x[0];fy=sin(r-1)*x[1]
    der=array([x[2],x[3],fx,fy])
    return der #retorna o vector de derivadas: [v_x, y-y, f_x, f-y]

# metodo 1: iterador simple con incremento de eps
def metodo1(f,x,t0,eps):
    fder = f(x)
    y = x+ fder*eps
    return y,t0+eps

## metodo 2: iterador de Runge Kutta de cuarto orde
def metodo2(f,x,t0,eps):
    y0 = x
    fder1 = f(y0); y1 = y0 +0.5*eps*fder1
    fder2 = f(y1); y2 = y0 +0.5*eps*fder2
    fder3 = f(y2); y3 = y0 + eps*fder3
    fder4 = f(y3); y=y0 + eps/6.*(fder1+2.*fder2+2.*fder3+fder4)
    return y,t0+eps

#-----
# valores iniciais
eps=1e-3;t0=0.;t1=17.;x0=array([0.,1.,-1.,0.]);x=x0;niter=50;

# metodo 1: xevol1 e unha matriz coa evolucion completa do sistema en
# fotogramas de 1 a niter fotogramas para non requerer moita memoria
tini=clock();xevol1=x0;i=0;t=t0
while t<t1:
    y,t=metodo1(derivada,x,t,eps)
    x=y;i=i+1
    if (i%niter == 0):
        xevol1 = vstack((xevol1,x))

xt1=xevol1.transpose()[0]
yt1=xevol1.transpose()[1]
print 'tempo metodo1= %.3f s.' % (clock()-tini)
figure(1);clf()
title(u'Integracion simple')
xlabel('x');ylabel('y')
plot(xt1,yt1,'r-');axis('equal');grid(True);show(False)

# metodo 2: Runge-Kutta
tini=clock();xevol2=x0;niter=50;i=0;t=t0
while t< t1:
    y,t=metodo2(derivada,x,t,eps)
    x=y;i=i+1
    if (i%niter == 0):
        xevol2 = vstack((xevol1,x))

xt2=xevol2.transpose()[0]
yt2=xevol2.transpose()[1]
print 'tempo metodo2= %.3f s.' % (clock()-tini)
figure(2);clf()
title('Runge-Kutta')
xlabel('x');ylabel('y');axis('equal')
plot(xt1,yt1,'r-');grid(True);show(False)

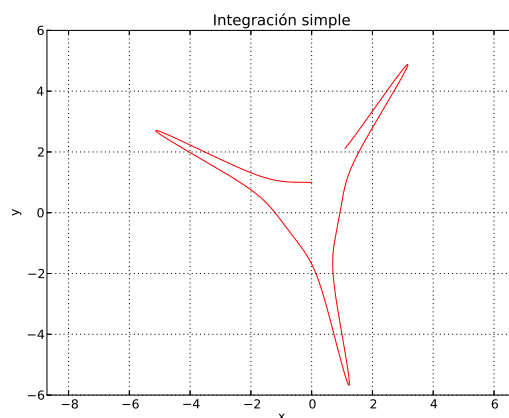
```

```

# metodo 3: usando paquete scipy; nt=numero de tempos usados
# cos metodos anteriores
tini=clock();nt = xevol2.size/x0.size;tt = linspace(t0,t1,nt)
xevol3 = odeint(derivada,x0,tt)
xt3=xevol3.transpose()[0]
yt3=xevol3.transpose()[1]
print 'tempo metodo3= %.3f s.' % (clock()-tini)
figure(3);clf()
title(u'funcion odeint de scipy')
xlabel('x');ylabel('y');axis('equal')
plot(xt3,yt3,'r-');grid(True);show(False)

```

Tes que obter a gráfica da figura 4:



Traxectoria de partícula sometida a forza $F(x, y)$.

Traballo a desenvolver pol@ alumn@

1. Dadas as seguintes matrices:

$$\mathbf{MI} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

calcula:

- a) A matriz inversa de $\mathbf{MI}-\mathbf{A}$
- b) $(\mathbf{MI}-\mathbf{A})^4$
- c) $\frac{\mathbf{MI}+\mathbf{A}}{\mathbf{MI}-\mathbf{A}}$

2. Diagonaliza, calcula os autovalores, autovectores, determinante e matriz inversa de:

$$\begin{bmatrix} -1 & -3 & -6 \\ 3 & 5 & 6 \\ -3 & -3 & -4 \end{bmatrix}$$

3. $\lim_{x \rightarrow \infty} \left(1 + \frac{\pi}{x}\right)^x$
4. $\frac{d}{dx} \left(\frac{x^2 - x + 1}{x^2 + x + 1} \right)$

5. $\int \frac{x-5}{(x-1)(x+1)^2} dx$
6. $\int_0^\pi e^x \sin x dx$: calcúla o seu valor simbólico e en punto flotante, co número de decimais por defecto e con 5 decimais.
7. $\int_{-\infty}^0 \frac{x^2+1}{x^4+1} dx$.
8. $\int_0^2 \frac{1}{x^2-4x+3} dx$.
9. Calcula o polinomio de Taylor de orde 10 de $f(x) = \sin(\tan(x)) - \tan(\sin(x))$ en torno a $x = 0$.
10. Resolve o seguinte sistema de ecuacións:

$$\begin{aligned} 2x + 3y + z - t &= 1 \\ x - 3y - 2z + t &= 6 \\ 3x + 2y - 3z + 2t &= 9 \\ x - y - z + 2t &= 10 \end{aligned}$$

11. Factoriza o polinomio $x^4 + x^3 - 4x^2 - 4x$.
12. Resolve a ecuación $x^3 - 5ax^2 + x - 1 = 0$ (en función de a).
13. **Resolución de ecuacións non lineares polo Método de Bisección** (ver [enlace](#)). Este método permite buscar solucións dunha ecuación non lineal da forma $f(x) = 0$, nun intervalo $[a, b]$ tal que $f(a)f(b) < 0$, sendo $f(x)$ unha función continua. Partindo de $x_1 = a$ e $x_2 = b$, calcula $x_m = \frac{a+b}{2}$ e avalía o signo de $f(a)f(x_m)$ e $f(x_m)f(b)$. Inicialmente comproba se $f(a)f(b) < 0$, en caso afirmativo podes aplicar o método e, en caso contrario, o programa remata cunha mensaxe de erro. Posteriormente, se $f(a)f(x_m) < 0$, repite o proceso con a e $x_2 = x_m$, e se $f(x_m)f(b) < 0$, repite o proceso con $x_1 = x_m$ e b . Así vas reducindo o intervalo de búsqueda ata que $f(x_m) = 0$, momento no que o proceso remata, sendo x_m a solución. Como é difícil que se produza a igualdade con 0, podes rematar cando $|f(x_m)| < 10^{-5}$. Proba con $f(x) = xe^{-x} - 0,2$ (solución= 0.2592) con $a = 0$ e $b = 1$.
14. Escribe un programa que resolva unha ecuación da forma $f(x) = 0$ polo método **Regula Falsi**. A partir dun punto inicial a_1 (lido por teclado), o programa debe buscar un punto b_1 con $f(a_1)f(b_1) < 0$, e calcular

$$b_2 = \frac{a_1 f(b_1) - b_1 f(a_1)}{f(b_1) - f(a_1)} \quad (67)$$

(ver figura). Se $f(a_1)f(b_2) < 0$, o programa debe repeti-lo proceso con a_1 e b_2 . Se $f(b_2)f(b_1) < 0$, o programa debe repeti-lo proceso con b_2 e b_1 . O programa debe rematar cando $|b_2 - b_1| < 10^{-5}$

Sesión 13: Outros exercicios

Traballo en clase

1. **Función con varios valores retornados.** Escribe un programa que lea por teclado un número enteiro n , e cree unha matriz **a** cadrada de orde n con valores enteiros aleatorios no conxunto $\{0, \dots, n\}$. Logo, este programa debe chamar a unha función `calcula(...)` (debes decidir os seus argumentos), que retorne unha matriz **b** cadrada e un vector **x**, ambos de orde n , e un escalar y . A matriz **b** retornada debe ter tódolos elementos nulos agás os das diagonais principal e secundaria, que deben coincidir cos da matriz **a**. O vector **x** retornado debe verificar que x_i , con $i = 1, \dots, n$, sexa a suma dos elementos iguais a i na columna i da matriz **a**. Pola súa banda, o escalar y retornado debe ser o número de elementos nulos na matriz **a**. Finalmente, o programa principal debe pedir por teclado o nome dun arquivo e almacenar neste arquivo as matrices **a** e **b**, o vector **x** e o escalar y .

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy.random import *
from numpy import *
n=-1
while n < 1:
    n=int(input('Introduce n positivo: '))
a=randint(0, n+1, [n,n])
#*****
def calcula(a1):
    [nf, nc]=a.shape
    b1=diag(diag(a1))
    for i in range(nc):
        b1[i, -i-1]=a1[i, -i-1]
    x1=zeros(nc)
    for i in range(nc):
        l=i+1
        x1[i]=sum(a[:,i] == l)*l
    y=sum(a == 0)
    return [b1, x1, y]
#*****
def transforma(a):
    linhas=[]
    if a.ndim == 2:
        [nf, nc]=a.shape
        for i in range(nf):
            l=''
            for j in range(nc):
                l=l+('%5d' % a[i,j])
            l=l+('\n')
            linhas.append(l)
    if a.ndim == 1:
        nc=len(a)
        l=''
        for j in range(nc):
            l=l+('%5d' % a[j])
        l=l+('\n')
        linhas.append(l)
    return linhas
#*****
```

```
[b, x, y]=calcula(a)
arquivo=raw_input('Nome arquivo: ')
try:
    out=open(arquivo, 'w')
    out.write('Matriz a: \n')
    lin=transforma(a)
    out.writelines(lin)
    out.write('Matriz b: \n')
    lin=transforma(b)
    out.writelines(lin)
    out.write('Vector x: \n')
    lin=transforma(x)
    out.writelines(lin)
    out.write('y= %d \n'% y)
    out.close()
except IOError:
    print 'Erro no escribindo no arquivo %s' % arquivo
```

2. Crea un arquivo de texto chamado `matriz.dat` co seguinte contido:

```
1 2 3 4 5
6 7 8 9 8
7 6 5 4 3
2 3 4 5 1
3 6 5 1 2
```

Escribe un programa que lea o arquivo `matriz.dat` e almacene o seu contido na matriz **a**. O programa debe chamar a unha función `calcula(...)`, pasándolle os argumentos axeitados, que retorne: 1) un vector **x** cos elementos de **a** que dan resto 3 cando se dividen entre catro; e 2) unha matriz **b** das mesmas dimensión de **a** onde:

$$b_{ij} = \begin{cases} a_{ij}a_{ji} & a_{ij} < a_{ji} \\ \frac{a_{ij}}{a_{ji}} & a_{ij} > a_{ji} \\ a_{ji}^3 & a_{ij} = a_{ji} \end{cases}$$

Logo, o programa debe dividir o vector **x** (é decir, cada elemento de **x**) por 2 mentres a suma dos elementos do vector **x** sexa maior que 10, contando o número de veces que se fixo a división. Finalmente, o programa debe mostrar o vector **x** resultante, a matriz **b** e o número de veces que se dividiu o vector **x** na pantalla.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from numpy import *
try:
    a=loadtxt('matriz.dat')
except IOError:
    print 'Erro lendo arquivo matriz.dat'
#*****
def calcula(a1):
    x1=extract(a1%4 == 3, a1)
    b1=a1.copy()
    [nf, nc]=a1.shape
    for i in range(nf):
        for j in range(nc):
            if a1[i,j] < a1[j,i]:
                b1[i,j]=a1[i,j]*a1[j,i]
```

```

        elif a1[i,j] > a1[j,i]:
            b1[i,j]=a1[i,j]/a1[j,i]
        else:
            b1[i,j]=a1[i,j]**3
    return [x1,b1]
#*****
[x,b]=calcula(a)
n=0;
while sum(x) > 10:
    x=x/2
    n= n+1
print 'Vector x= ', x
print 'Numero de veces que se divide x= ', n
print 'Matriz b= ', b

```

Traballo a desenvolver pol@ alumn@

1. Escribe unha función chamada `sumaSerie(...)`, cos argumentos axeitados, que aproxime a serie

$$\sum_{n=1}^{\infty} \frac{3n}{n^p + 6} \quad (68)$$

usando só os m primeiros sumandos con valores superiores a 10^{-2} . Os valores dos sumandos deben almacenarse no vector $\mathbf{y} = (y_1, \dots, y_m)$, de modo que $y_n = \frac{3n}{n^p + 1}, n = 1, \dots, m$. Escribe un programa que chame á función `sumaSerie()` con $p = 3$, e represente gráficamente o vector \mathbf{y} cos valores dos sumandos. Este programa debe construír unha matriz a cadrada de orde m con elementos a_{ij} para $i, j = 1, \dots, m$, dados por:

$$a_{ij} = \begin{cases} y_i y_j & i < j \\ \frac{1}{y_i} & i > j \\ y_i & i = j \end{cases}$$

Finalmente, o programa debe gardar no arquivo `datossaida.dat` a suma da serie (con 5 díxitos decimais) e a matriz a (unha fila en cada liña). **NOTA:** o número de sumandos é $m = 17$.

2. Crea un arquivo de texto `datos3.txt` co seguinte contido:

```

1 2 3 4 5
6 7 8
9 0
1 2 3 4
5 6

```

Escribe un programa que lea números dende `datos3.txt` mentres que sexan non nulos e os almacene no vector \mathbf{x} . Este programa debe chamar á función `sumaPrimos(...)`, cos argumentos axeitados, que retorne o número n de elementos primos do vector \mathbf{x} e a suma s destes elementos primos. Tamén debe retornar unha matriz \mathbf{a} de orde n con elementos $a_{ij}, i, j = 1, \dots, n$, dados por:

$$a_{ij} = \begin{cases} x_i x_j & i + j \text{ par} \\ x_i + x_j & i + j \text{ impar} \end{cases}$$

Finalmente, o programa debe mostrar por pantalla o vector \mathbf{x} e a matriz \mathbf{a}