PROGRAMACIÓN CIENTÍFICA

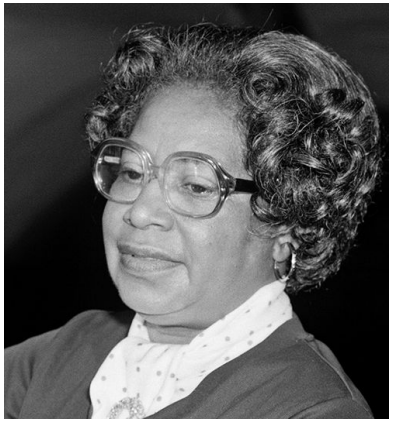
CO

MÓDULO NUMPY

Programadoras da NASA no programa espacial de viaxe á lúa (1969)

Katherine Johnson: **matemática** e programadora da NASA
Primeira civil en recibir a **Medalla de Ouro** do Congreso USA

Mary Jackson: enseñeira da NASA



Dorothy Vaughan: programadora en **Fortran** do primeiro ordenador da NASA

Película
“Figuras ocultas”
en 2016

NumPy

- NUMPY é o paquete fundamental para a computación científica en Python. Contén tipos de datos, clases, funcións e módulos que posibilitan o manexo de vectores (arrays) multidimensionais. Máis información no manual de referencia: http://docs.scipy.org/doc/numpy/reference
- O tipo de datos máis importante é **`array`** (ou **`ndarray`** para vectores multidimensionais).
- Pódese utilizar con tipos de datos da linguaxe Python.

NumPy: `array`

- Un **`array`** é como unha lista de python na que todos os seus elementos son do mesmo tipo. Os tipos posibles son:
  - Cadea de texto (string) de n-caracteres: **`|Sn`**
  - Booleano (True ou False) que se almacenan como 1 bit: **`bool`**
  - Enteiros con signo: **`int`** (int32 o int64, dependendo da plataforma), **`int8`** (-128 a 127) , **`int16`** (-32768 a 32767), **`int32`** (-2.147.483.648 a 2.147.483.647), **`int64`** (-2.147.483.648 a 2.147.483.647).
  - Enteiros sen signo: **`uint8`** (0 a 255), **`uint16`** (0 a 65535), **`uint32`** (0 a 4.294.967.295) e **`uint64`** (0 a 18.446.744.073.709.551.615).
  - Números reais: **`float`** ou **`float64`** (dobre precisión) e **`float32`** (precisión simple).
  - Números complexos: **`complex`** ou **`complex128`** (complexos de dobre precisión con tipo float64 para parte real e imaxinaria) e **`complex64`** (precisión simple, float32 real e imaxinaria).

NumPy: `array`

- Os tipos referencianse como cadea (**`'int'`**) ou como constante numpy (**`numpy.int64`**).
- Un **`array`** é un obxecto que ten as seguintes propiedades (entre outras):
  - **`ndarray.shape`**: tupla coas dimensións.
  - **`ndarray.ndim`**: número de dimensións.
  - **`ndarray.size`**: número de elementos.
  - **`ndarray.itemsize`**: tamaño dos elementos en bytes.
  - **`ndarray.nbytes`**: tamaño total ocupado polos elementos.
  - **`ndarray.dtype`**: tipo de dato dos elementos.

NumPy: `array`: Creación de arrays

- A partires de secuencias (listas ou tuplas) de Python:
  - **`numpy.array(secuencia, tipo dato)`**
- Usando funcións de NumPy:
  - **`numpy.arange([inicio], fin[, incremento], dtype=None)`**
  - Equivalente a función **range(inicio, fin, incremento)** de Python, pero permite utilizar incrementos decimais. O fin non se inclúe.
  - **`numpy.linspace(inicio, fin, num=50, endpoint=True, retstep=False):`** devolve un vector con **num** puntos equi-espaciados entre inicio e fin. Por defecto o punto final non se inclúe.

NumPy: `array`

- Os tipos referencianse como cadea (**`'int'`**) ou como constante numpy (**`numpy.int64`**).
- Un **`array`** é un obxecto que ten as seguintes propiedades (entre outras):
  - **`ndarray.shape`**: tupla coas dimensións.
  - **`ndarray.ndim`**: número de dimensións.
  - **`ndarray.size`**: número de elementos.
  - **`ndarray.itemsize`**: tamaño dos elementos en bytes.
  - **`ndarray.nbytes`**: tamaño total ocupado polos elementos.
  - **`ndarray.dtype`**: tipo de dato dos elementos.

NumPy: `array`: Creación de arrays

- Usando funcións de NumPy para crear vectores n-dimensionais:
  - **`ones(shape,dtype=None):`** `crea unha matriz de 1's.` **shape**, é a forma do array de saída (entero ou lista/tupla). Ex: **ones(5)**: vector de 5 uns; **ones([2,4])**: matriz 2x4 con uns. **dtype**, cualquera dos tipos de datos de NUMPY.
  - **`zeros(shape,dtype=float)`**: igual comportamento que ones( ) pero crea arrays contendo ceros.

NumPy: `array`: Creación de arrays

- Copiando outro array.
  - **Copia por referencia**: Se tes un array **a** e o asignas a **b** (**b = a**). Isto é perigoso, xa que non se fai unha copia real e os dous obxectos comparten a mesma memoria (se modifico a tamén se modifica b).
  - **Copia por valor**: a función copy(...) (**b=numpy.copy(a)**) ou o método copy(...) (**b=a.copy()**) realizan unha copia real do espacio de memoria que ocupa o obxecto **a**.

NumPy: `array`: Indexación

- Indexación é o acceso a elementos concretos do array.
- Aspectos que son como as listas de Python:
  - Acceso a un elemento: **`nome_array[posicion]`**
  - Acceso a un rango de elementos consecutivos: **`nome_array[inicio:fin]`**
  - O primeiro índice é o 0, e non colle o elemento **fin**.
  - Se os índices son negativos comeza polo final do array como en Python.
  - Se o índice é maior ou igual ó número de elementos do array lanzará unha excepción (IndexError).
- Acceso a elementos nun array: **`nome_array[[p1,p2,...,pn]]`**, onde [p1,p2,...,pn] é unha lista ou array.

NumPy: **`ndarray`**

- Para acceder a un elemento faise referencia con tantos índices como dimensións:
  - **`nome_array[indiceDim1, indiceDim2,...,indiceDimN]`**
- O operador ":" sustitue a todo o rango de índices posibles nesa dimensión. Por exemplo, se **`a`** é unha matriz, **`a[0,:]`** devolve un array coa primeira fila. Tamén podes utilizar o operador ":" para especificar subrangos en cada dimensión.
- O método **reshape(newshape)** permite modificar as dimensión dos arrays n-dimensionais, sempre que a dimensión global coincida.Ex: **a=array([1,2,3,4,5,6]; a.reshape([2,3])**

NumPy: `array`

- Outras funcións para modificar un array:
  - **`insert(arr, pos, values, axis=None)`** : inserta values no array arr, nas posicións **pos**.
  - **`append(arr,values, axis=None)`**: inserta ó final do array os valores values (escalar ou secuencia).
  - **`delete(arr,pos, axis=None)`**: devolve un array no que se borraron os elementos das posicións de **pos**.
  - **`nome_array.flatten(order="C")`**: método que permite transformar un array n-dimensional nun vector (array 1D). Por defecto, a conversión é por filas (order="C"). Para columnas, order="F".
  - **`ravel(arr, order="C")`**: función que permite transformar un array n-dimensional nun vector (array 1D).

NumPy: `array`

- Outras funcións para modificar un array:
  - **`concatenate((a1, a2, ...), axis=0)`**: función que permite concatenar a secuencia de arrays “(a1, a2, ...)”. A shape dos array ten que coincidir e únense na dimensión dada por axis.
  - **`nome_array.T`** : método que realiza a transposta do array.
  - **`hstack(seq)`**: apila os arrays da secuencia “seq” horizontalmente (agrega columnas). Ex: a=[[1],[2],[3]];b=[[1,2],[3,4],[5,6]];hstack([a,b])
  - **`vstack(seq)`**: apila os arrays da secuencia “seq” verticalmente (agrega filas). Ex: a=[1,2,3]; b=[4,3,5]; vstack([[a],[b]])

NumPy: `array`
Operacións aritméticas

- Operacións aritméticas de +, -, * e / dun array cun número. Sea a un array e x un número, a operación a*x realiza a multiplicación de x por todos os elementos do array.
  - `>>> from numpy import *`
  - `>>> a=array([1,2,3], dtype="float")` # devolve array([1., 2., 3.])
  - `>>> a/2` # devolve array([0.5, 1. , 1.5])
  - `>>> a*2` # devolve array([2., 4., 6.])
  - `>>> a-2` # devolve array([-1., 0., 1.])
- Para array con dimensións idénticas as operacións aritméticas realízanse elemento a elemento:
  - `>>> b=array([3,1,0], dtype="float")` # devolve array([3., 1., 0.])
  - `>>> a*b` # devolve array([3., 2., 0.])

NumPy: `array`
Operacións relacionais

- Son as que comparan un array con un dato simple ou arrays entre si.
- O resultado é un array de valores booleanos (True/False).
- Operadores: **>**(maior que), **>=**(maior ou igual que), **<**(menor que), **<=**(menor ou igual que), **==**(igual que), **!=**(distinto que).
  - `>>> from numpy import *`
  - `>>> a=array([1,2,3], dtype="float")` # devolve array([1, 2, 3])
  - `>>> a > 2` # devolve array([False, False, True], dtype=bool)
  - `>>> a >= 2` # devolve array([False, True, True], dtype=bool)
  - `>>> b=numpy.array([1,4,5], dtype="int")`
  - `>>> a == a` # devolve array([True, True, True], dtype=bool)
  - `>>> a != b` # devolve array([False, True, True], dtype=bool)

NumPy: `array`
Operacións lóxicos

- Son as que se dan entre datos (simples ou arrays) de tipo booleano.
- O resultado é un valor ou array de tipo booleano (True/False).
- As operacións realízanse elemento a elemento.
- Operadores: **&**(“Y/AND lóxico”), **|**(“O/OR lóxico”), **~**(“Non/NOT lóxico”, operador unario que só necesita un operando).
  - `>>> from numpy import *`
  - `>>> a = arange (4)` # devolve array([0, 1, 2, 3])
  - `>>> ab = a > 1` # devolve array([False False True True])
  - `>>> b = ones (4 , dtype = numpy.bool )`
  - `>>> print b` # devolve [True True True True]
  - `>>> print ab & b` # devolve [False False True True]
  - `>>> print ~ ab` # devolve [True True False False]
  - `>>> print ab | b` # devolve [True True True True]

NumPy: `array`
Funcións aritméticas

- NUMPY contén varias funcións aritméticas que toman como argumentos vectores e aplícanse a todos os elementos do vector (o seu nome coincide en moitos casos coas do módulo math).
- Trigonométricas: sin(x), cos(x), tan(x), arcsin(x), arccos(x), arctan(x), degrees(x), radians(x), deg2rad(x), rad2deg(x)
- Hiperbólicas: sinh(x), cosh(x), tanh(x), arcsinh(x), arctan(x)
- Redondeo: round_(a[,decimais]), rint(x), fix(x), floor(x), ceil(x), trunc(x)
- Exponenciais e logarítmicas: exp(x), exp1m(x), exp2(x), log(x), log10(x), log2(x), log1p(x) (=log(1+x))
- Sumas, producto e diferencia: prod(a[, axis]), sum(a[, axis]), nansum(a[, axis])), cumprod(a[, axis]), cumsum(a[, axis]), gradient(f, *varargs), cross(a, b)
- Miscelánea: sqrt(x), power(x), sign(x), nan_to_num(x)
- Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.math.html

NumPy: `array`
Funcións relacionais e lóxicas

- **`all(a[,axis])`**: comproba se todos os elementos do eixo “axis” do array son True ou non nulos (nese caso devolve True).
- **`any(a[,axis])`**: comproba se algún elemento do eixo “axis” do array é True ou non nulo (nese caso devolve True).
- **`allclose(a,b[,rtol,atol])`**: devolve True se os dous arrays a e b teñen os seus elementos iguais dentro dun intervalo de tolerancia.
- **`array_equal(a1, a2)`**: True se os dous arrays teñen a mesma forma e elementos. False no caso contrario.
- **`greater(x1,x2)`**: devolve o valor de comparación (**`x1>x2`**) elemento a elemento.
- **`greater_equal(x1,x2)`**: devolve o valor de comparación (**`x1>=x2`**) elemento a elemento.

NumPy: `array`

Funcións relacionais e lóxicas

- **`less(x1,x2)`**: devolve o valor de comparación (**`x1<x2`**) elemento a elemento.
- **`less_equal(x1,x2)`**: devolve o valor de comparación (**`x1<=x2`**) elemento a elemento.
- **`equal(x1,x2)`**: devolve (**`x1 == x2`**) elemento a elemento.
- **`not_equal(x1,x2)`**: devolve (**`x1 != x2`**) elemento a elemento.
- **`logical_and(x1,x2):`** determina o valor de x1 AND x2 elemento a elemento.
- **`logical_or(x1,x2)`**: determina o valor de x1 OR x2 elemento a elemento.
- **`logical_not(x1)`**: determina o valor de NOT x1elemento a elemento.

NumPy: `array`
Casting do tipo de datos

- Refírese ó cambio de tipo de datos dun array. Formas de facelo:
  - Volver a xerar o array con **`array(..., dtype="tipoNumpy")`**.
  - Usando unha función de NUMPY: **`tipoNumpy(array)`**.
  - Utilizando un método dos arrays:
    `nome_array.astype("tipoNumpy")`.
- Exemplos:
  - **`>>> from numpy import *`**
  - **`>>> a=array(['1','2','3'])`** # crea un array a partir dunha lista de caracteres.
  - **`>>> a`** # devolve array(['1', '2', '3'], dtype='|S1')

NumPy: `array`
Casting do tipo de datos

- Exemplos:
  - `>>> b=array(a, dtype="int")` # convirte **a** ao tipo enteiro
  - `>>> b` # devolve array([1, 2, 3])
  - `>>> c=array(a, dtype=numpy.float32)` # convirte datos a reais
  - `>>> c` # devolve array([1., 2., 3.], dtype=float32)
  - `>>> b.dtype` # devolve dtype('int64')
  - `>>> d=float32(a)` # convirte datos a reais utilizando función
  - `>>> d` # devolve array([1., 2., 3.], dtype=float32)
  - `>>> f=a.astype('int8')` # convirte datos a enteiro con 8 bits utilizando método de obxecto
  - `>>> f` # devolve array([1, 2, 3], dtype=int8)

NumPy: Entrada/saída

http://docs.scipy.org/doc/numpy/reference/routines.io.html

- **`savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ')`**: garda o array X nun arquivo de texto (só son obrigatorios os dous primeiros argumentos).
  - **`fname`** : nome do arquivo
  - **`fmt`**: cadea de caracteres co formato do tipo **`%ancho_columna[.num_decimais]tipo`** (tipo é i, f, g, e,...).
  - **`delimiter`**: caracter separando columnas.
- **`loadtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)`**: lee datos desde arquivo de texto (só é obrigatorio o primeiro argumento).
  - **`fname`** : nome do arquivo
  - **`dtype`**: tipo de datos.
  - **`delimiter`**: caracter separando columnas.
  - Devolve un array cos datos do arquivo.

NumPy: `array`
Ordenación de arrays

- NUMPY proporciona métodos para a ordenación dos datos dos arrays, independentemente do tipo de datos ou dimensión. Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.sort.html
  - **`sort(a,axis=-1)`**: devolve un array do mesmo tipo e forma ordeado de menor a maior segundo o eixo "axis".
  - **`argsort(a,axis=-1)`**: devolve un array das posicións que ordenaría o array de menor a maior segundo o eixo "axis". En ambos casos, se axis=None transforma nun array 1D.
- Exemplos:
  - **`>>> from numpy import *`**
  - **`>>> a=array([2,-3,4,1,0],dtype='int')`** # crea array
  - `>>> a` # devolve array([2, -3, 4, 1, 0])
  - **`>>> b=sort(a)`** # devolve a ordeado de menor a maior
  - **`>>> b`** # devolve array([-3, 0, 1, 2, 4])
  - **`>>> indices=argsort(a)`** # devolve posicións
  - **`>>> indices`** # devolve array([1, 4, 3, 0, 2])
  - **`>>> a[indices]`** # devolve array([-3, 0, 1, 2, 4])
  - **`>>> b[::-1]`** # devolve array([4, 2, 1, 0, -3]) que é o ordeamento inverso de a

NumPy: `array`
Procura de elementos de arrays

- NUMPY proporciona métodos para a procura de elementos nun array en base a un ou máis criterios. As condicións exprésanse como arrays de booleanos.
  - **`where(condición, a1, a2)`**: retorna cos elementos de a1 onde a condición é certa e cos elementos de a2 se a condición é falsa. Os tres arrays (condición, a1 e a2) son da mesma forma. where(condición) retorna as posicións devolve un array coas posicións dos elementos que cumpren a condición: Ex: where(x<5,x,-1): pon a -1 os elementos do array x maiores que 5
  - **`extract(condición, a)`**: devolve os elementos do array onde a condición é verdadeira (True). Sempre devolve un vector (dimensión 1D) independentemente da dimensión de a.
  - **`compress(condición, a,axis=None)`**: é a xeralización de extract. Devolve un array n-dimensional. Pódese especificar a dimensión na que se aplica a condición con “axis”.
  - **`argmax(a[, axis=None])`**: devolve as posicións dos valores de **a** iguais ó valor máximo de a segundo o eixo “axis”. Se axis=None busca en todo o array.
  - **`nanargmax(a[,axis=None])`**: igual que argmax ignorando os valores NaN (Not a Number).
  - **`argmin(a[, axis=None])`**: devolve as posicións dos valores de a iguais ó mínimo de a segundo o eixo “axis”. Se axis=None busca en todo o array.

NumPy: `array`
Procura de elementos de arrays

- Métodos para a para a procura de elementos continuación:
  - **`nanargmin(a[, axis=None])`**: igual que argmin ignorando os valores NaN.
  - **`nonzero(a)`**: devolve unha tupla de arrays, unha por cada dimensión de a, coas posicións dos elementos que son distintos de cero en cada dimensión.
  - **`flatnonzero(a)`**: igual que a anterior pero devolve un vector 1D.
  - **`nonzero(a)`**: devolve as posicións dos elementos distintos de cero.
  - **`isnan(a)`**: devolve un array de booleanos con True nas posicións onde hai un NaN e False no resto.
  - **`isinf(a)`**: devolve un array de booleanos con True nas posicións onde hai valores infinito e False no resto.
- Exemplos:
  - `>>> from numpy import *`
  - `>>> a=numpy.arange(12).reshape([3,4])` # crea unha matriz 3x4
  - `>>> a` # devolve array([[0, 1, 2, 3], [4, 5, 6, 7],[8, 9, 10, 11]])
  - `>>> condicion=mod(a,3)==0` # devolve true para elementos divisibles por 3

NumPy: `array`

Procura de elementos de arrays

- Exemplos:
  - `>>> condicion` # devolve array([[True, False, False, True], [False, False, True, False],[False, True, False, False]], dtype=bool)
  - `>>> extract(condicion,a)` # devolve elementos de a divisibles por 3, e dicir, array([0, 3, 6, 9])
  - `>>> b=array([[2,3],[1,4],[3,7]])` # crea unha matriz 3x2
  - `>>> b` # devolve array([[2, 3],[1, 4],[3, 7]])
  - `>>> compress([0,1],b,axis=0)` # devolve array([[1, 4]]) (a segunda fila)
  - `>>> compress([False,True],b,axis=0)` # devolve array([[1, 4]])
  - `>>> compress([1,0,1],b,axis=0)` # devolve array([[2, 3],[3, 7]]) (a primeira e terceira filas)
  - `>>> compress([0,1],b,axis=1)` # devolve array([[3],[4],[7]]) (devolve a segunda columna porque é a segunda dimensión)

NumPy: `array`

Procura de elementos de arrays

- Exemplos:
  - `>>> c=arange(12).reshape([2,6])` # crea matriz 2x6
  - `>>> c` # devolve array([[0, 1, 2, 3, 4, 5],[6, 7, 8, 9, 10, 11]])
  - `>>> argmax(c,axis=0)` # devolve array([1, 1, 1, 1, 1, 1]) (posición do máximo segundo o eixo das filas)
  - `>>> argmax(c,axis=1)` # devolve array([5, 5]) (posición do máximo segundo o eixo das columnas)
  - `>>> d=arange(5)` # crea un vector de 5 elementos
  - `>>> d` # devolve array([0, 1, 2, 3, 4])
  - `>>> e=d/0.0` # divide o vector d por 0.0
  - `>>> e` # devolve array([nan, inf, inf, inf, inf])
  - `>>> isnan(e)` # devolve array([True, False, False, False, False], dtype=bool) (Verdadeiro cando no vector hai un nan)
  - `>>> isinf(e)` # devolve array([False, True, True, True, True], dtype=bool)
- Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.sort.html

NumPy: `array`
Estatística básica

- NUMPY ofrece varias funcións de estatística básica que operan sobre un array x (se axis=None as operacións realizanse o array transformado a vector 1D):
  - **`amin(a, axis=None)`**: devolve un vector ou escalar co valor mínimo o longo do eixo axis.
  - **`amax(a, axis=None)`**: igual para máximo
  - **`nanmin(a, axis=None)`**: igual que amin ignorando valores NaN.
  - **`nanmax(a, axis=None)`**: igual que amax ignorando valores NaN.
  - **`ptp(a, axis=None)`**: peak to peak (devolve o rango de valores máximo-mínimo no eixo dado).
  - **`average(a,axis=None,weights=None)`**: devolve un escalar ou array coa media ponderada no eixo dado.
  - **`mean(a, axis=None,dtype=None)`**: realiza media aritmética no eixo dado.
  - **`median(a, axis=None)`**: realiza mediana no eixo dado.
  - **`std(a, axis=None,dtype=None, ..., ddof=0)`**: desviación típica
  - **`var(a, axis=None,dtype=None, ..., ddof=0)`**: varianza.
- Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.statistics.html

NumPy
Subpaquetes de NumPy

- **`linalg`** : álxebra lineal (resolución de sistemas de ecuacións, operacións con vectores e matrices: determinante, inversa,...). Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.linalg.html
- **`random:`** estatística aleatoria (datos aleatorios, permutacións e distribucións). Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.random.html
- **`fft`** : transformada de Fourier. Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.fft.html.
- **`Polynomial`**: permite crear, manipular e axustar a polinomios. Máis información en: http://docs.scipy.org/doc/numpy/reference/routines.polynomials.html

Manipulación de polinomios

Máis información en:http://docs.scipy.org/doc/numpy/reference/routines.polynomials.poly1d.html

- Numpy manipula polinomios (considerando un polinomio como o vector dos seus coeficientes onde o coeficiente do monomio máis significativa está na posición 0 e nos monomios que non existen poñemos un 0) para realizar as seguintes operacións:
  - **`poly1d(c_or_r[, r, variable])`**: clase para un polinomio en 1D.
  - **`polyval(p, x)`**: avalía un polinomio p nun punto ou vector x.
  - **`poly(sequencia_de_ceros)`**: devolve os coeficientes do polinomio que representa a secuencia de ceros.
  - **`roots(p)`**: devolve as raices do polinomio p.
  - **`polyfit(x, y, deg[, rcond, full, w, cov])`**: axusta os puntos dos vectores x e y a un polinomio utilizando o método de mínimos cadrados. Devolve os coeficientes de polinomio
  - **`polyder(p[, m])`**: devovle a derivada do polinomio p de grao m.
  - **`polyint(p[, m, k])`**: devolve a integral indefinida do polinomio p de orde m.
  - **`polyadd(a1, a2)`**: suma polinomios a1 e a2.
  - **`polydiv(u, v)`**: devolve cociente e resto da division de u entre v.
  - **`polymul(a1, a2)`**: devolve o producto de polinomios a1 e a2.
  - **`polysub(a1, a2)`**: diferencia de polinomios (a1 – a2).

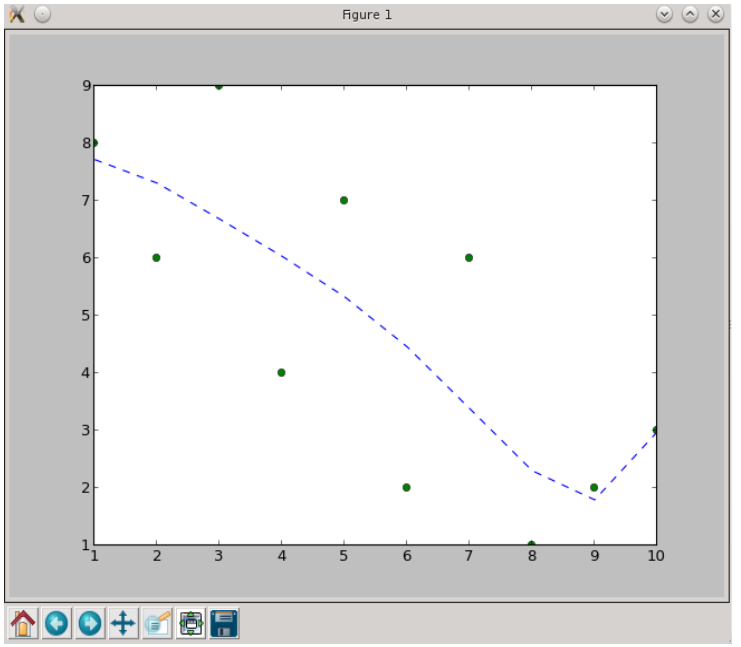
Axuste de función a polinomios

Máis información en: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

- Problema: dado un conxunto de puntos $\{(x_i,y_i), i=1...m\}$, atopa-lo polinomio $p(x)$ de grao n que mellor se axusta aos puntos (minimiza a suma dos erros cadráticos entre os puntos (x_i,y_i) e os valores $(x_i,p(x_i))$ do polinomio).

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

- **`polyfit(x, y, deg [, rcond, full, w, cov])`**: axusta os puntos dos vectores x e y a un polinomio de grao deg utilizando o método de mínimos cadrados. Devolve os coeficientes de polinomio que minimizan o erro.
- Exemplo: axustar os puntos de y a un polinomio de orde 5
  - from numpy import *
  - x=arange(1,11)
  - y=array([8,6,9,4,7,2,6,1,2,3], dtype='float')
  - p=polyfit(x,y,5)
  - from matplotlib.pyplot import *
  - plot(x,y, 'go', x, polyval(p,x), 'b—')
  - show()



Axuste de función potencial

Máis información en: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

- **Potencial**: $y = ax^b$: Tomando logaritmos: $log(y) = log(a) + b \cdot log(x)$: $log(y)$ é un polinomio de orde 1 con variábel $log(x)$:

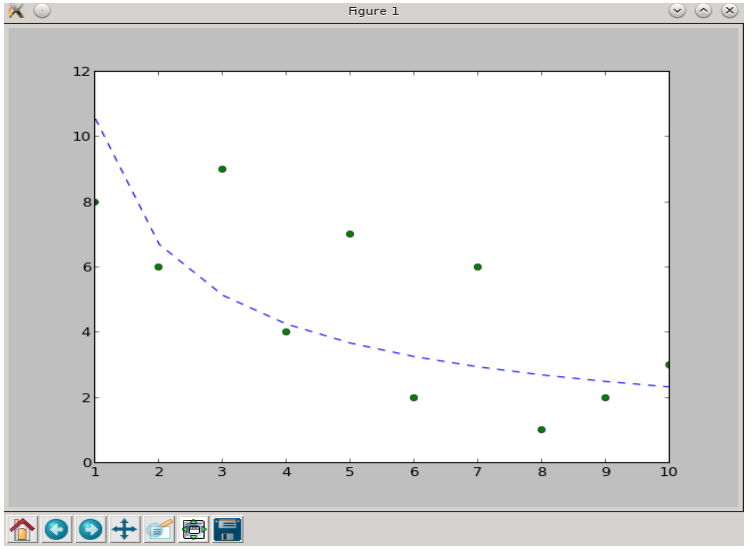
$$p = polyfit(log(x),log(y),1);$$

Como axusto a un polinomio de grado 1, $length(p)=2$, e entón

$$\left.\begin{array}{l} log(y)=b \cdot log(x)+log(a) \\ log(y)=p(0)*log(x)+p(1) \end{array}\right\} \quad b=p(0),\ p(1)=log(a) \rightarrow a=e^{p(1)}$$

E a función potencial é: $y=e^{p(1)}x^{p(0)}$

- Exemplo cos datos anteriores:
  - x=arange(1,11);
  - y=array([8,6,9,4,7,2,6,1,2,3], dtype='float')
  - p=polyfit(log(x),log(y),1)
  - plot(x,y,'go', x, exp(p[1])*x**p[0], 'b--')
  - show()



Axuste de función exponencial

Máis información en: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

- **Exponencial**: $y = ae^{bx}$ (ou ben $y = a10^{bx}$): tomando logaritmos: $log(y) = log(a) + b \cdot x$: $log(y)$ é un polinomio de orde 1 con variábel x:

$$p = polyfit(x,log(y),1);$$

Como axusto a un polinomio de grado 1, $length(p)=2$, e entón:

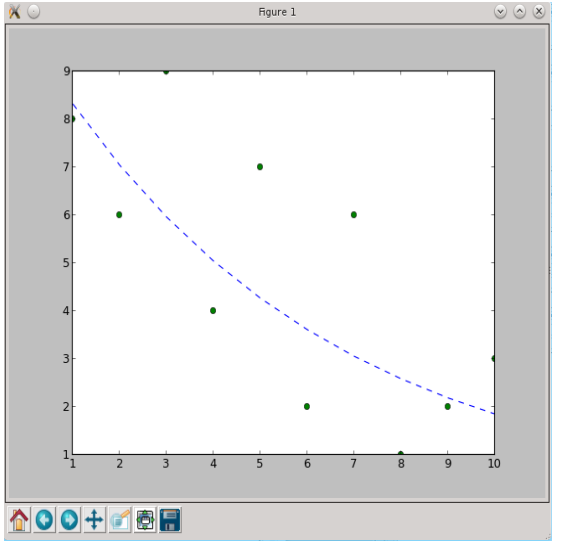
$log(y)=b \cdot x+log(a)$

$log(y)=p(0)*x+p(1)$

$b=p(0)$, $p(1)=log(a) \rightarrow a=e^{p(1)}$

E a función exponencial é: $y=e^{p(1)}e^{p(0)x} \rightarrow y=e^{p(1)+p(0)x}$

- Exemplo cos datos anteriores:
  - x=arange(1,11);
  - y=array([8,6,9,4,7,2,6,1,2,3], dtype='float')
  - p=polyfit(x,log(y),1)
  - plot(x,y,'go', x, exp(p[1]+p[0]*x), 'b--')
  - show()



Axuste de función logaritmica

Máis información en: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

- **Logarítmica**: $y = a \cdot log(x)+b$ (ou ben $y = a \cdot log_{10}x+b$): y é un polinomio de orde 1 en $log(x)$:

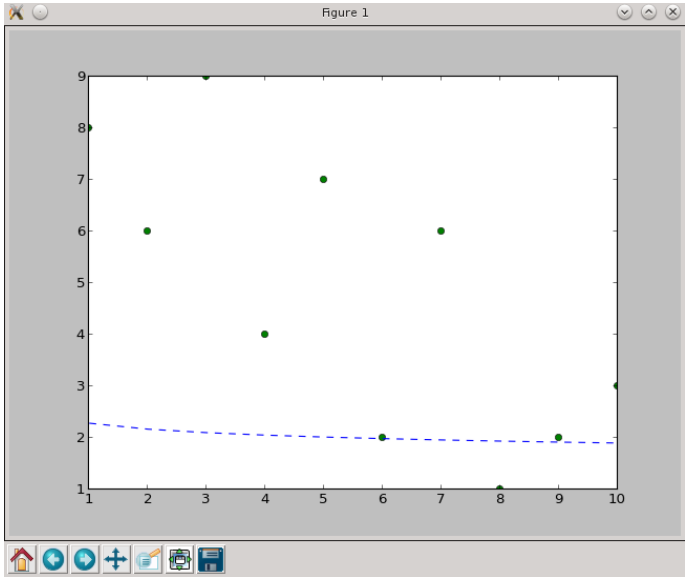
$$p = polyfit(log(x),y,1)$$

Temos que:

$$\left.\begin{array}{l} y=a \cdot log(x)+b \\ y=p(0) \cdot log(x)+p(1) \end{array}\right\} \quad a=p(0), b=p(1)$$

E a función logarítmica é: $y=p(0) \cdot log(x)+p(1)$

- Exemplo cos datos anteriores:
  - x=arange(1,11);
  - y=array([8,6,9,4,7,2,6,1,2,3], dtype='float')
  - polyfit(log(x), y, 1)
  - plot(x,y,'go', x, p[0]*log(x)+p[1], 'b--')
  - show()



Axuste de función reciproca

Máis información en: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

- **Recíproca**: $y = \frac{1}{ax+b}$

Invertindo temos: *1/y=ax+b*, e *1/y* é un polinomio de orde 1 en *x*:

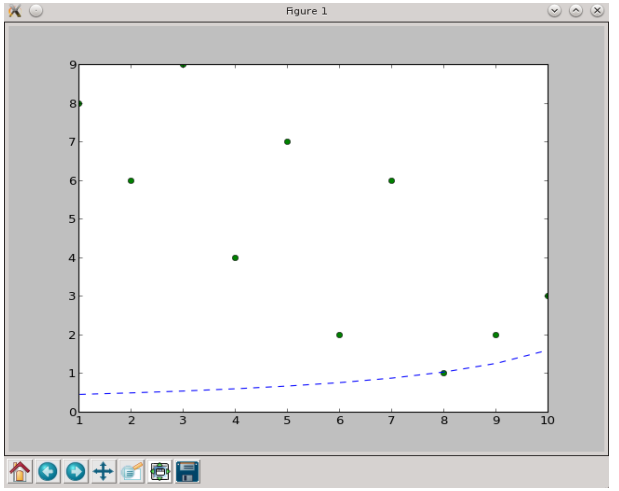
p = polyfit(x,1./y,1)

Temos que:

$$\left.\begin{array}{l} 1/y = a \cdot x + b \\ 1/y = p(0) \cdot x + p(1) \end{array}\right\} \quad a = p(0), b = p(1)$$

E a función recíproca é: $y = \frac{1}{p(0) \cdot x + p(1)}$

- Exemplo cos datos anteriores:
  - x=arange(1,11);
  - y=array([8,6,9,4,7,2,6,1,2,3], dtype='float')
  - polyfit(x, 1.0/y, 1)
  - plot(x,y,'go', x, 1.0/(p[0]*x+p[1]), 'b--')
  - show()



Alxebra lineal

Máis información en: https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

- Algunhas operacións da alxebra linea (hai que cargar o submodulo **`linalg`** de Numpy):
  - **`det(a)`**: determinante do array a.
  - **`trace(a)`**: suma dos elementos da diagonal principal de a.
  - **`diag(a)`**: devolve a diagonal de a.
  - **`triu(a) e tril(a)`**: devolve, respectivamente, o triangulo superior e inferior de a.
  - **`diagflat(v)`**: crea unha matriz con todos ceros e o array v na diagonal.
  - **`inv(a)`**: devolve a matriz inversa.
  - **`dot(a,b)`**: producto da matrices a e b.
  - **`matrix_power(a, n)`**: potencia dunha matriz cadrada a a un enteiro n ($\mathbf{a}^n$).
  - **`solve(a,b)`**: resolve un sistema de ecuacións lineais ax=b (a matriz de coeficientes, b vector co termo independente e x a solución).
  - **`eig(a)`**: devolve os autovalores e autovectores dunha matriz a.
  - **`matrix_rank(a)`**: devolve o rango da matriz a.
  - **`exception numpy.linalg.LinAlgError`**: excepción que lanzan os obxetos de linalg cando ocorre un erro.

Números aleatorios

Máis información en: https://numpy.org/doc/2.0/reference/random/legacy.html

- Hai que cargar o submódulo random de numpy. Exemplo:
  - **from numpy.random import ***
- Algunhas funcións para xerar números aleatorios son:
  - **`rand(d0,d1,...,dn)`**: devolve un array de números aleatorios no intervalo [0,1) nas dimensións dadas.
  - **`random(shape)`**: igual comportamento que **`ones()`**, pero xerando números aleatorios en [0,1).
  - **`seed(seed=None)`**: xera unha semente.
  - **`shuffle(x)`**: baralla a secuencia x, e deixa a secuencia barallada na secuencia x.
  - **`permutation(x)`**: realiza permutacións aleatorias de x. Se x é un enteiro, realiza as permutacións sobre **`arange(x)`**. Se x é un vector devolve un vector barallado, e se x e unha secuencia, realiza as permutacións só na primeira dimensión.

Números aleatorios

Máis información en: https://numpy.org/doc/2.0/reference/random/legacy.html

- Algunhas funcións para xerar números aleatorios son:
  - **`randint(a,b,shape)`**: devolve un array de números aleatorios enteiros no intervalo [a,b) das dimensións dadas.

```
In [29]: import numpy.random as rd

In [30]: rd.randint(3, 10, [2,3])
Out[30]:
array([[6, 3, 8],
       [9, 7, 9]])
```

Interpolación lineal de funcións

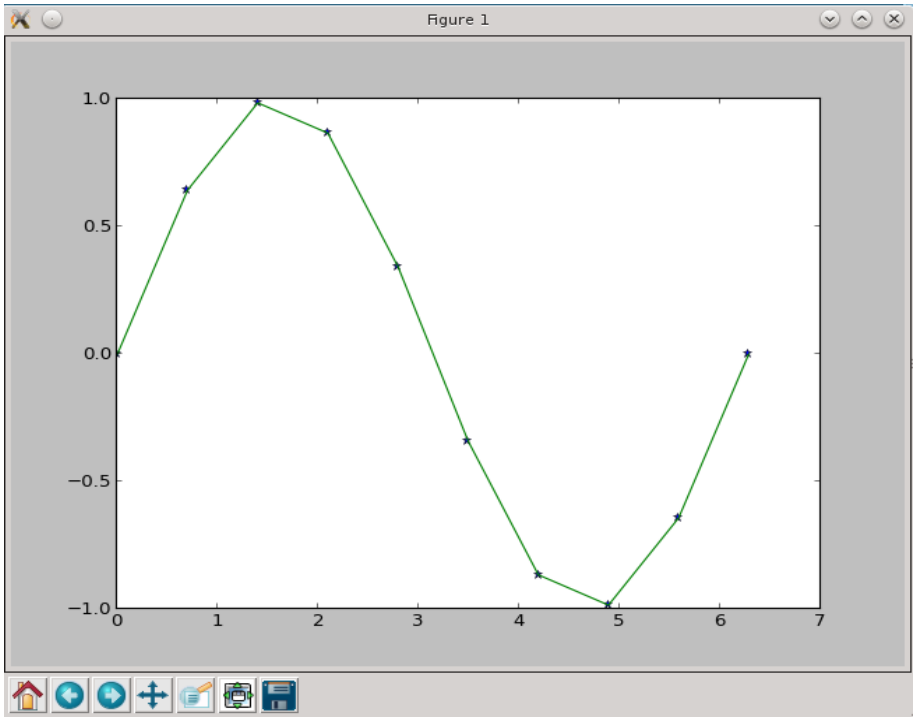
Máis información en:http://docs.scipy.org/doc/numpy/reference/generated/numpy.interp.html

- Dados **m** puntos (x_j, y_j), j=1...m dunha función f(x) descoñecida, quérese calcular y_i=f(x_i) con $x_i \neq x_j$, j=1...m. Os valores y_i chámaselles valores interpolados ou estimados. A función **`y=interp(x, xp, fp, left=None, right=None)`**: devolve os puntos interpolados utilizando unha interpolación lineal onde:
  - X : son as coordenadas x dos valores interpolados.
  - xp : son un vector unidimensional cos valores x_j, j=1...m
  - fp : son un vector unidimensional cos valores y_j, j=1...m
  - Left : (opcional) valor que devolve para valores x < xp[0] (por defecto fp[0]).
  - right : (optional) valor que devolve para valores x > xp[-1] (por defecto fp[-1]).
  - Cando ocorren erros lanza a excepción **`ValueError`**.

Interpolación lineal de funcións

Máis información en:http://docs.scipy.org/doc/numpy/reference/generated/numpy.interp.html

- Exemplo:
  - from numpy import *
  - x=linspace(0, 2*pi,10)
  - y=sin(x)
  - xx=linspace(0,2*pi, 100)
  - yy=interp(x, x, y)
  - from matplotlib.pyplot import *
  - plot(x, y, 'b*', xx, yy, 'g-')
  - show(False)



- Os puntos azuis son os puntos reais (vectores x e y) e a liña verde son os puntos interpolados (vectores xx e yy) utilizando interpolación lineal.

Interpolación funcións (paquete scipy)

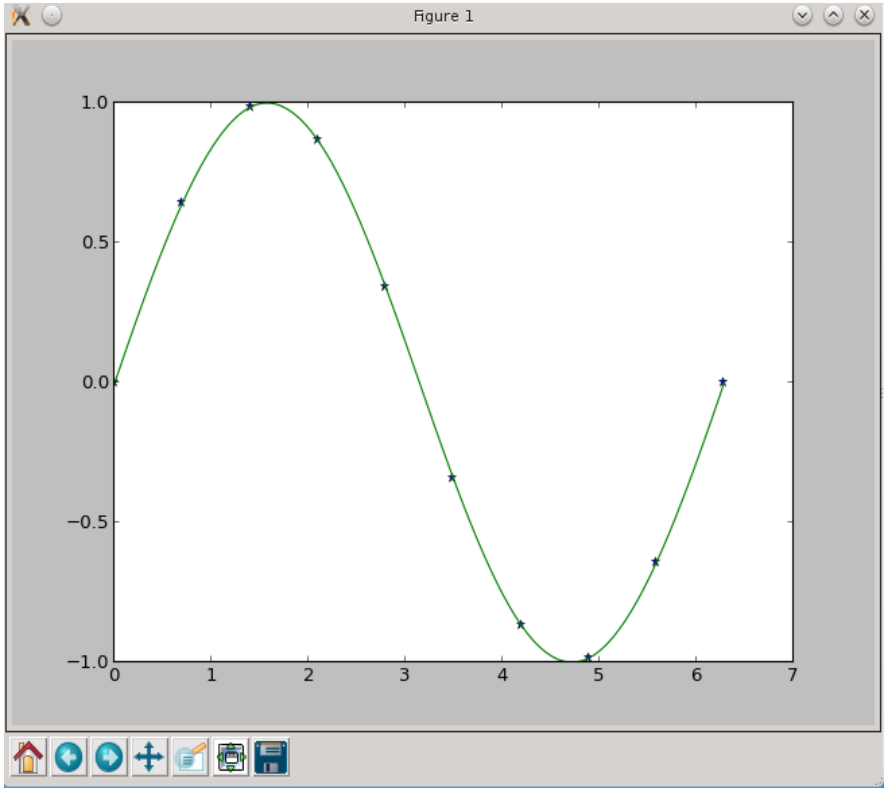
Máis información en:
http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.interpolate.interp1d.html

- O paquete **NUMPY** só nos permite interpolación lineal. Para outros tipos de interpolación hai que utilizar o paquete **SCIPY** (submodulo interpolate) e a función: **`interp1d(x, y, kind='linear', axis=-1, copy=True, bounds_error=True,fill_value=np.nan, assume_sorted=False)`** onde:
  - x : son un vector unidimensional cos valores x_j, j=1...m
  - y : son un vector unidimensional cos valores y_j, j=1...m
  - kind : especifica o tipo de interpolación e pode ser ‘linear’, ‘nearest’, ‘zero’, ‘slinear’, ‘quadratic, ‘cubic’ where ‘slinear’, ‘quadratic’ e ‘cubic’. Por defecto usa interpolación lineal.
  - O resto dos parámetros son opcionais e non teñen importancia para o propósito deste curso.
  - Devolve a función f que aproxima y=f(x)

Interpolación funcións (paquete scipy)

Máis información en:
http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.interpolate.interp1d.html

- Exemplo:
  - from scipy import *
  - x=linspace(0, 2*pi,10)
  - y=sin(x)
  - from scipy.interpolate import *
  - f=interp1d(x, y, 'cubic')
  - xx=linspace(0,2*pi, 100);
  - yy=f(xx)
  - from matplotlib.pyplot import *
  - plot(x, y, 'b*', xx, yy, 'g-')
  - show(False)



- Os puntos azuis son os utilizados para a interpolación (vectores x e y) e os verdes son os puntos xx interpolados (vector yy) utilizando unha interpolación cúbica.