



$$\begin{cases} 5x + 2y + 3z = 0 \\ x + 3y + 2z = 0 \\ 3x - 5y + z = -1 \\ 2x - y + z = 0 \end{cases}$$

# NumPy

+

# SciPy

**Facundo Batista**

[facundo@taniquetil.com.ar](mailto:facundo@taniquetil.com.ar)

<http://www.taniquetil.com.ar/>

Py  Ar

# NumPy



# ¿Qué es NumPy?

- ▶ Es una biblioteca de Python para trabajar con arreglos multidimensionales.
- ▶ El principal tipo de dato es el arreglo o *array*
- ▶ También nos permite trabajar con la semántica de matrices
- ▶ Nos ofrece muchas funciones útiles para el procesamiento de números

# Disclaimer

- ▶ Pueden ver toda la info de esta presentación en [http://www.scipy.org/Tentative\\_NumPy\\_Tutorial](http://www.scipy.org/Tentative_NumPy_Tutorial)
- ▶ Los ejemplos suponen que primero se hizo en el intérprete:  

```
>>> from numpy import *
```

# El Array

- ▶ Es una tabla de elementos
  - ▶ normalmente números
  - ▶ todos del mismo tipo
  - ▶ indexados por enteros
- ▶ Ejemplo de arreglos multidimensionales
  - ▶ Vectores
  - ▶ Matrices
  - ▶ Imágenes
  - ▶ Planillas
- ▶ ¿Multidimensionales?
  - ▶ Que tiene muchas dimensiones o ejes
  - ▶ Un poco ambiguo, mejor usar *ejes*
  - ▶ Rango: cantidad de ejes

# Propiedades del Array

- ▶ Como tipo de dato se llama `ndarray`
- ▶ `ndarray.ndim`: cantidad de ejes
- ▶ `ndarray.shape`: una tupla indicando el tamaño del array en cada eje
- ▶ `ndarray.size`: la cantidad de elementos en el array
- ▶ `ndarray.dtype`: el tipo de elemento que el array contiene
- ▶ `ndarray.itemsize`: el tamaño de cada elemento en el array

## Propiedades del Array

```
>>> a = arange(10).reshape(2,5)
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a.shape
(2, 5)
>>> a.ndim
2
>>> a.size
10
>>> a.dtype
dtype('int32')
>>> a.itemsize
4
```

# Creando Arrays

- ▶ Tomando un iterable como origen

```
>>> array( [2,3,4] )  
array([2, 3, 4])  
>>> array( [ (1.5,2,3), (4,5,6) ] )  
array([[ 1.5,  2. ,  3. ],  
       [ 4. ,  5. ,  6. ]])
```



## Creando Arrays

- ▶ Con funciones específicas en función del contenido

```
>>> arange(5)
array([0, 1, 2, 3, 4])
>>> zeros((2, 3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> ones((3, 2), dtype=int)
array([[1, 1],
       [1, 1],
       [1, 1]])
>>> empty((2, 2))
array([[ 9.43647120e-268,  7.41399396e-269],
       [ 1.08290285e-312,          NaN]])
>>> linspace(-pi, pi, 5)
array([-3.141592, -1.570796,  0.      ,  1.570796,  3.141592])
```

## Manejando los ejes

```
>>> a = arange(6)
>>> a
array([0, 1, 2, 3, 4, 5])
>>> a.shape = (2, 3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> a.shape = (3, 2)
>>> a
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> a.size
6
```

## Operaciones básicas

- ▶ Los operadores aritméticos se aplican por elemento

```
>>> a = arange(20, 60, 10)
>>> a
array([20, 30, 40, 50])
>>> a + 1
array([21, 31, 41, 51])
>>> a * 2
array([ 40,  60,  80, 100])
```

- ▶ Si es *inplace*, no se crea otro array

```
>>> a
array([20, 30, 40, 50])
>>> a /= 2
>>> a
array([10, 15, 20, 25])
```

# Operaciones básicas

- ▶ Podemos realizar comparaciones

```
>>> a = arange(5)
>>> a >= 3
array([False, False, False,  True,  True], dtype=bool)
>>> a % 2 == 0
array([ True, False,  True, False,  True], dtype=bool)
```

- ▶ También con otros arrays

```
>>> b = arange(4)
>>> b
array([0, 1, 2, 3])
>>> a - b
array([20, 29, 38, 47])
>>> a * b
array([ 0, 30, 80, 150])
```

# Operaciones básicas

- ▶ Tenemos algunos métodos con cálculos típicos

```
>>> c
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> c.min(), c.max()
(0, 9)
>>> c.mean()
4.5
>>> c.sum()
45
>>> c.cumsum()
array([ 0,  1,  3,  6, 10, 15, 21, 28, 36, 45])
```

- ▶ Hay **muchas** funciones que nos dan info del array
  - ▶ all, alltrue, any, argmax, argmin, argsort, average, bincount, ceil, clip, conj, conjugate, corrcoef, cov, cross, cumprod, cumsum, diff, dot, floor, inner, inv, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sometrue, sort, std, sum, trace, transpose, var, vdot, vectorize, where

# Trabajando con los elementos

- ▶ La misma sintaxis de Python

```
>>> a = arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2]
2
>>> a[2:5]
array([2, 3, 4])
>>>
>>> a[1] = 88
>>> a[-5:] = 100
>>> a
array([ 0, 88, 2, 3, 4, 100, 100, 100, 100, 100])
```

## Trabajando con los elementos

- ▶ Pero también podemos trabajar por eje

```
>>> a = arange(8).reshape((2,4))
>>> a
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
>>> a[:,1]
array([1, 5])
>>> a[0,-2:]
array([2, 3])
```

## Cambiando la forma del array

- Podemos usar `.shape` directamente

```
>>> a = arange(8)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> a.shape = (2,4)
>>> a
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

- Usando `.shape` con comodín

```
>>> a.shape = (4,-1)
>>> a
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
>>> a.shape
(4, 2)
```



## Cambiando la forma del array

- ▶ Transponer y aplanar

```
>>> a
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
>>> a.transpose()
array([[0, 4],
       [1, 5],
       [2, 6],
       [3, 7]])
>>> a.ravel()
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> a
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

## Juntando y separando arrays

- ▶ Tenemos vstack y hstack

```
>>> a = ones((2,5)); b = arange(5)
>>> a
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
>>> b
array([0, 1, 2, 3, 4])
>>> juntos = vstack((a,b))
>>> juntos
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 0.,  1.,  2.,  3.,  4.]])
```

## Juntando y separando arrays

- ▶ También `hsplit` y `vsplit`

```
>>> hsplit(juntos, (1,3))
[array([[ 1.],
        [ 1.],
        [ 0.]]) ,
 array([[ 1.,  1.],
        [ 1.,  1.],
        [ 1.,  2.]]) ,
 array([[ 1.,  1.],
        [ 1.,  1.],
        [ 3.,  4.]])]
```

## Indexado avanzado

- Podemos indizar con otros arrays

```
>>> a = arange(10) ** 2
>>> i = array([ (2,3), (6,7) ])
>>> a
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
>>> a[i]
array([[ 4,  9],
       [36, 49]])
```

- O elegir elementos

```
>>> a = arange(5)
>>> b = a % 2 == 0
>>> a
array([0, 1, 2, 3, 4])
>>> b
array([ True, False,  True, False,  True], dtype=bool)
>>> a[b]
array([0, 2, 4])
```

# Matrices

- ▶ Es un caso específico del array

```
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> A = matrix(a)
>>> A
matrix([[0, 1, 2],
        [3, 4, 5]])
>>> A.T
matrix([[0, 3],
        [1, 4],
        [2, 5]])
>>> A.I
matrix([[-0.77777778,  0.27777778],
        [-0.11111111,  0.11111111],
        [ 0.55555556, -0.05555556]])
```

# Matrices

- ▶ Se comportan, obviamente, como matrices

```
>>> A
matrix([[0, 1, 2],
        [3, 4, 5]])
```

```
>>> M
matrix([[2, 3],
        [4, 5],
        [6, 7]])
```

```
>>> A * M
matrix([[16, 19],
        [52, 64]])
```

# Polinomios

```
>>> p = poly1d([2, 3, 4])
>>> print p
      2
2 x + 3 x + 4
>>> print p*p
      4      3      2
4 x + 12 x + 25 x + 24 x + 16
>>> print p.deriv()
4 x + 3
>>> print p.integ(k=2)
      3      2
0.6667 x + 1.5 x + 4 x + 2
>>> p(range(5))
array([ 4,  9, 18, 31, 48])
```

# SciPy





# Intro

- ▶ Colección de algoritmos matemáticos y funciones
  - ▶ Construido sobre NumPy
- ▶ Poder al intérprete interactivo
  - ▶ Procesamiento de datos y prototipado de sistemas
  - ▶ Compite con Matlab, IDL, Octave, R-Lab, y SciLab

# Disclaimer

- ▶ Pueden ver toda la info de esta presentación en <http://docs.scipy.org/doc/>
- ▶ ¿Les conté que me recibí de ingeniero hace más de 9 años?

# Funciones y funciones!

- ▶ De todo tipo!
  - ▶ airy
  - ▶ elliptic
  - ▶ bessel
  - ▶ gamma
  - ▶ beta
  - ▶ hypergeometric
  - ▶ parabolic cylinder
  - ▶ mathieu
  - ▶ spheroidal wave
  - ▶ struve
  - ▶ kelvin

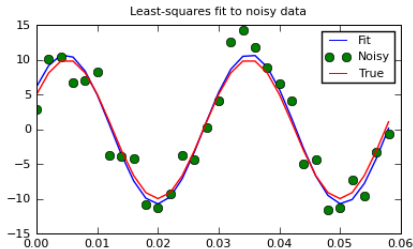
# Integration

- ▶ General integration
- ▶ Gaussian quadrature
- ▶ Integrating using samples
- ▶ Ordinary differential equations

# Optimization

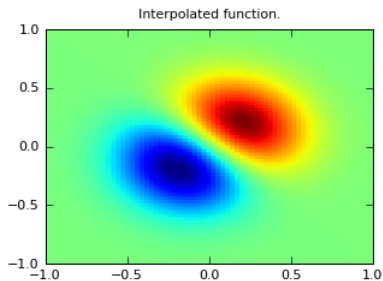
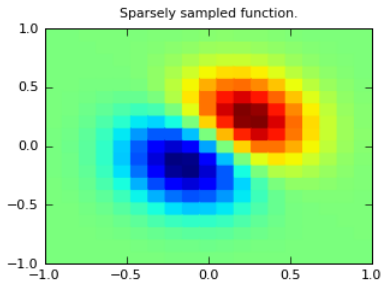
- ▶ Nelder-Mead Simplex algorithm
- ▶ Broyden-Fletcher-Goldfarb-Shanno algorithm
- ▶ Newton-Conjugate-Gradient
- ▶ Least-square fitting
- ▶ Scalar function minimizers
- ▶ Root finding

```
>>> f = poly1d([1, 4, 8])
>>> print f
2
1 x + 4 x + 8
>>> roots(f)
array([-2.+2.j, -2.-2.j])
>>> f(-2.+2.j)
0j
```



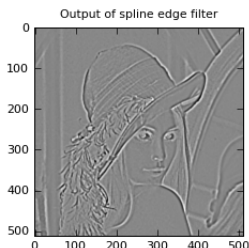
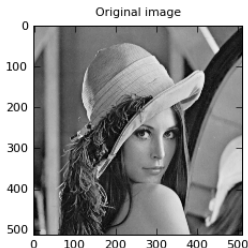
# Interpolation

- ▶ Linear 1-d interpolation
- ▶ Spline interpolation in 1-d
- ▶ Two-dimensional spline representation



# Signal Processing

- ▶ B-splines
  - ▶ second- and third-order cubic spline coefficients
  - ▶ from equally spaced samples in one- and two-dimensions
- ▶ Filtering
  - ▶ Convolution/Correlation
  - ▶ Difference-equation filtering
  - ▶ Other filters: Median, Order, Wiener, Hilbert, ...



# Algebra lineal

- ▶ Matrices
  - ▶ Inversas
  - ▶ Determinantes
  - ▶ Resolución de sistemas lineales
- ▶ Descomposiciones
  - ▶ Eigenvalues and eigenvectors
  - ▶ Singular value, LU, Cholesky, QR, Schur
- ▶ Funciones de matrices
  - ▶ Exponentes y logaritmos
  - ▶ Trigonometría (común e hiperbólica)



# Estadísticas

- ▶ Masked statistics functions
  - ▶ 64!
- ▶ Continuous distributions
  - ▶ 81!
- ▶ Discrete distributions
  - ▶ 12!
- ▶ Statistical functions
  - ▶ 72!

# ¡Muchas gracias!

¿Preguntas?

¿Sugerencias?

Facundo Batista

[facundo@taniquetil.com.ar](mailto:facundo@taniquetil.com.ar)

<http://www.taniquetil.com.ar>



Licencia: Creative Commons

Atribución-NoComercial-CompartirDerivadasIgual 2.5 Argentina

[http://creativecommons.org/licenses/by-nc-sa/2.5/deed.es\\_AR](http://creativecommons.org/licenses/by-nc-sa/2.5/deed.es_AR)