Numpy

Este pacote implementa o conceito de array n-dimensional. Também o conceito de funções universais além de centenas de funções e constantes a usar em computação matemática e científica. É pré-requisito em muitos outros pacotes (ex: scipy, sympy, matplotlib, tensorflow, pygame, astropy, opencv, pulp, pandas entre outros).

O pacote é muito extenso e completo, seu manual tem 1542 páginas e pode ser encontrado via google, buscando `numpy-ref-1.12.0.pdf` ou uma versão mais recente.

Instalação Tendo acesso à Internet, entre no diretório de instalação do Python, no subdiretório `Scripts` dentro da árvore de instalação do Python. Abrindo uma janela CMD nesse diretório execute

```
pip install numpy
```

Se não souber fazer isto ou não tiver direitos na máquina em questão, mude de estratégia e use o Winpython. Para seu governo este pacote bem como muitos outros, já está pré-instalado no `winpython`.

Importação Há duas maneiras de importar. A mais simples é `>>> from numpy import *` Aqui todos os objetos do pacote são incluidos no espaço de nomes atual. A segunda maneira, mais higiênica é atribuir um nome ao pacote e referenciar seus objetos qualificando-os por este nome. O comando agora é `>>> import numpy as np`. O nome do pacote agora é `np`. Pode-se usar qualquer outro nome, mas este material (e 99% da comunidade Python) o conhece como np.

Observar ainda que há diversos subpacotes dentro de numpy. Isso surgirá mais adiante em alguns exemplos. Os mais usados são `linalg` que inclue funções da álgebra linear, `fft` da transformada disreta de Fourier, `random` que inclue diversas funções de geração de aleatórios e de distribuições estatísticas.

arranjos Um arranjo em numpy é um array n-dimensional. Quando n=1, o array é unidimensional logo é um vetor e neste caso pode-se usar a lista convencional. A coisa começa a melhorar quando n=2 (matriz) ou mesmo quando n é maior do que 2. Em qualquer caso o a classe do pacote é ndarray e alguns de seus métodos são:

array Cria um array a partir de dados já existentes

```
>>> import numpy as np
>>> a=np.array(elementos, tipo)
>>> b=np.array([1,2,3,4])
>>> b
array([1, 2, 3, 4])
>>> c=np.array([1,2,3],float)
>>> c
array([1., 2., 3.])
>>> d=np.array([[1,2,3],[0,0,7],
                [3,3,9]])
>>> d
array([[1, 2, 3],
       [0, 0, 7],
       [3, 3, 9]])
>>> d1=np.array(d,float)
>>> d1
array([[ 1., 2., 3.],
       [ 0., 0., 7.],
       [ 3., 3., 9.]])
```

Interessante que se você mandar imprimir a matriz ela será exibida sem as vírgulas

```
>>> print(d)
[[1 2 3]
 [0 0 7]
 [3 3 9]]
```

Na criação as listas podem ser fornecidas como tuplas (usando-se parênteses ao invés de colchetes), mas esta estratégia não será desenvolvida aqui.

Indexação Elementos individuais podem ser recuperados dos arrays. Podem também ser alterados. Cada dimensão do array exigirá uma especificação, separadas por vírgulas. Acompanhe

```
>>> d[1,2]
7
>>> c[1]
2.0
>>> d[1,1]=99
>>> d
array([[ 1,  2,  3],
       [ 0, 99,  7],
       [ 3,  3,  9]])
```

Fatiamento Usando a mesma estratégia do python, sub-arrays podem ser obtidos usando fatiamento

```
>>> e=np.array([[4,7,8,10],
   [11,12,16,19],[20,21,23,25],
   [40,41,50,51]])
>>> e
array([[ 4,  7,  8, 10],
       [11, 12, 16, 19],
       [20, 21, 23, 25],
       [40, 41, 50, 51]])
>>> e[1:3,1:3]
array([[12, 16],
       [21, 23]])
>>> e[0:4,1:4]
array([[ 7,  8, 10],
       [12, 16, 19],
       [21, 23, 25],
       [41, 50, 51]])
>>> e[1,1:4]
array([12, 16, 19])
```

Zeros Outra maneira de criar o array contendo zeros. O formato é `zeros((forma),tipo)`. Acompanhe:

```
>>> f=np.zeros((3,2),complex)
>>> f
array([[0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j]])
>>> g=np.zeros((4,5),int)
>>> print(g)
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

Outros criadores O `eye` funciona como a matriz identidade, mas valendo para matrizes não quadradas. A identidade exige apenas a ordem da matriz (já que ela é quadrada).

```
>>> h=np.eye(3,2)
>>> h
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
>>> j=np.identity(3)
>>> j
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Atributos Um array tem os atributos `ndim` que é um escalar indicando a quantidade de dimensões e `shape` que é uma tupla indicando a quantidade de elementos em cada dimensão.

```
>>> j.ndim
2
>>> j.shape
(3, 3)
>>> g.shape
(4, 5)
```

Reorganizar Para reorganizar um array já existente, usa-se o método `reshape`.

```
>>> a1=np.arange(0,15,2)
>>> a1
array([0,2,4,6,8,10,12,14])
>>> a2=a1.reshape((2,4))
>>> a2
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14]])
```

Para reorganizar in-place usa-se o método `resize`.

```
>>> b1=np.array([1,7,8,9,10,11])
>>> b1.resize([3,2])
>>> print(b1)
[[ 1  7]
 [ 8  9]
 [10 11]]
```

Funções a seguir algumas (tem inúmeras mais, veja a documentação):

```
>>> a=np.array([[1,4,6],[7,8,1]])
>>> print(a)
[[1 4 6]
 [7 8 1]]
```

a.mean()	média
a.var()	variancia
a.sum()	soma
a.prod()	produto
a.max()	maior valor
a.min()	menor valor
a.argmax()	índice do maior
a.argmin()	índice do menor
np.unique(a)	valores únicos

```
>>> print(a.mean())
4.5
>>> print(a.max())
8
>>> print(a.argmax())
4
>>> print(np.unique(a))
[1 4 6 7 8]
```

Funções Universais As funções universais (ufuncs) se aplicam elemento a elemento dentro dos arrays. Acompanhe o exemplo abaixo para identificar o funcionamento das ufuncs.

```
>>> c1=[1,2,3]
>>> c2=[3,7,9]
>>> c1+c2
[1, 2, 3, 3, 7, 9]
>>> d1=np.array([1,2,3])
>>> d2=np.array([3,7,9])
>>> d1+d2
array([ 4,  9, 12])
```

Broadcasting Para broadcasting (aplicar um escalar a um array)

```
>>> a é array([1,2,3,4])
>>> a+3
array([4,5,6,7])
>>> b*5
array([0.5, 3.5, 40., 45.])
```

multiplicação matricial

```
>>> a1=np.array([[1,2,-4],
     [3,-1,5]])
>>> a2=np.array([[6,-3],
     [1,-2],[2,4]])
>>> np.dot(a1,a2)
array([[  0, -23],
       [ 27,  13]])
```

Produto vetorial

```
>>> x = (1, 2, 0)
>>> y = (4, 5, 6)
>>> print np.cross(x, y)
[12 -6 -3]
```

Determinante

```
>>> m=np.array([[2,3],[-1,-2]])
>>> m
[[ 2 3]
[-1 -2]]
>>> np.linalg.det(m)
-1.0
```

Matriz inversa

```
>>> good = np.array([[2.1,3.2],
       [4.3,5.4]])
>>> np.linalg.inv(good)
[[-2.23140496 1.32231405]
[ 1.7768595 -0.8677686 ]]
```

Sistema de equações lineares A solução de um sistema de equações lineares por exemplo

```
2x + y = 19
x - 2y = 2
cuja solução é (x=8 and y=3):
>>> coef=np.array([[2,1],[1,-2]])
>>> cont=np.array([19, 2])
>>> np.linalg.solve(coef, cont)
[ 8. 3.]
```

Se $m > n$, onde m são equações e n são incógnitas, o comando é `c=np.linalg.lstsq(a,b)` que devolve a solução que minimiza o erro cometido. (`solve`, acima, exige $m = n$).

Transposta

```
>>> a=np.array(range(6),
     float).reshape((2,3))
>>> a é array([[ 0., 1., 2.],
              [ 3., 4., 5.]])
>>> a.transpose()
array([[0.,3.],[1.,4.],[2.,5.]])
```

Concatenação

```
>>> a=np.array([1,2],float)
>>> b=np.array([3,4,5],float)
>>> c=np.array([7,8,9],float)
>>> np.concatenate((a,b,c))
array([1.,2.,3.,4.,5.,7.,8.,9.])
>>> a=np.array([[1,2],[3,4]])
>>> b=np.array([[5,6],[7,8]])
>>> np.concatenate((a,b))
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=1)
array([[ 1., 2., 5., 6.],
[ 3., 4., 7., 8.]])
```

Funções matemáticas abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, and arctanh. floor (inteiro inferior), ceil (inteiro superior), e rint (inteiro mais proóximo = arredondado).

Geração de aleatórios Gera números reais entre 0 e 1.

```
>>> np.random.rand(2,2)
array([[0.07724462, 0.93519669],
       [0.44908731, 0.12275935]])
```

Inteiros aleatórios. O formato é início (inclusive), final (exclusive) e quantidade. Os dois últimos são opcionais.

```
>>> np.random.randint(1,50,3)
array([40,  4, 49])
>>> np.random.randint(6) # dado
4
```

Ordenação

```
>>> a=np.array([[1,4,7],[8,3,9],
      [22,11,13]])
>>> a
array([[ 1,  4,  7],
       [ 8,  3,  9],
       [22, 11, 13]])
>>> np.sort(a)
array([[ 1,  4,  7],
       [ 3,  8,  9],
       [11, 13, 22]])
>>> np.sort(a,axis=None)
array([1,3,4,7,8,9,11,13,22])
>>> np.sort(a,axis=0)
array([[ 1,  3,  7],
       [ 8,  4,  9],
       [22, 11, 13]])
```

Distribuição normal Esta função retorna simulações de uma distribuição normal. Para obter $N(\mu, \sigma^2)$, basta fazer $sigma * np.random.randn(...) + mu$. Onde μ = média da distribuição e σ^2 é a variância. Veja-se o exemplo

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49, 4.00, -1.81, 7.29],
       [ 0.39, 4.68,  4.99, 4.84]])
```

Cria um array $2, 4$ contendo simulações de $3, 6.25$. Lembre que $2.5^2 = 6.25$.

Distribuição binomial Cria simulações desta distribuição. Usa 2 parâmetros: `n`, que significa o número de tentativas e `p` que vem a ser a probabilidade de sucesso da distribuição ($0 \leq p \leq 1$). A função retorna para cada simulação o número de sucessos. Lembrando que a probabilidade é

$$P(N) = \left(\begin{array}{c} n \\ N \end{array} \right) p^N (1-p)^{n-N}$$

Veja-se o exemplo:

```
>>> n=10
>>> p=0.5
>>> s=np.random.binomial(n,p,1000)
```

O exemplo acima faz 1000 simulações, cada uma jogando 10 moedas e informando a quantidade de caras em cada simulação.

Desvio padrão em arrays

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1., 1.])
>>> np.std(a, axis=1)
array([ 0.5, 0.5])
```

Embaralhando uma lista Supondo uma lista L já existente e querendo embaralhá-la, no próprio local

```
>>> L=[1,4,7,22,11,5,4,77]
>>> L
[1, 4, 7, 22, 11, 5, 4, 77]
>>> np.random.shuffle(L)
>>> L
[4, 5, 7, 1, 4, 22, 77, 11]
```

Ler e Gravar dados Suponha uma variável x como em

```
>>> x=np.random.rand(3,2)
>>> x
array([[0.65615773, 0.64436913],
       [0.54172899, 0.9726057 ],
       [0.27936652, 0.09892232]])
```

Para guardá-la em disco (note que este formato é ilegível por humanos) para posterior recuperação pode-se fazer

```
>>> x.dump('c:/apl2/lixo.dad')
```

E depois basta fazer

```
>>> y=np.load('c:/apl2/lixo.dad')
```

Para colocar uma constante Imagine um array de 6 linhas por 10 colunas que deve ser preechido com 22

```
>>> z=np.empty((10,6))
>>> z.fill(22)
```

Iteração com elementos

```
>>> a=np.arange(9).reshape(3,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> for x in np.nditer(a):
    print(x)
...
0 1 2 3 4 5 6 7 8
```

Para medir alguma execução

```
import time
ini=time.time()
... o que quer medir
fim=time.time()
print("tempo em segs",fim-ini)
```