Computação II (MAB 225)

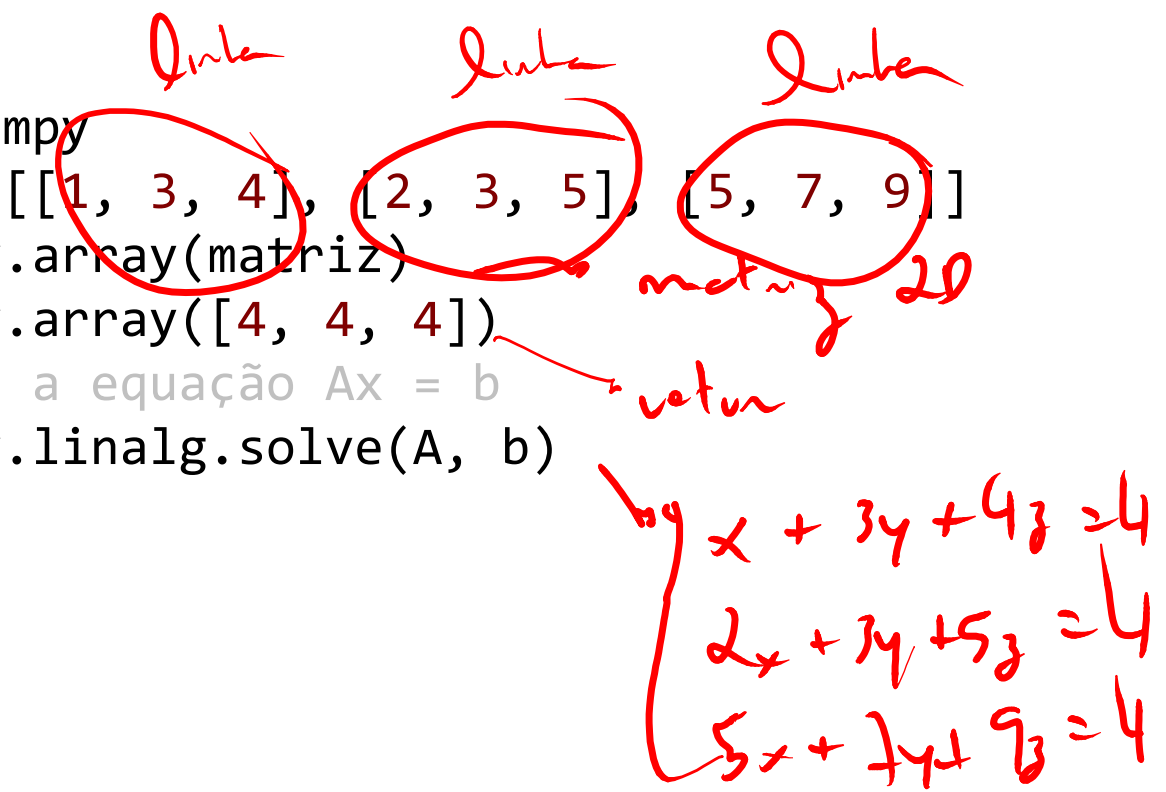
Fabio Mascarenhas - 2015.1

http://www.dcc.ufrj.br/~fabiom/pythonoo

numpy

- NumPy é uma biblioteca para trabalhar com matrizes de qualquer dimensão, contendo várias funções matemáticas em matrizes

```
import numpy
matriz = [[1, 3, 4], [2, 3, 5], [5, 7, 9]]
A = numpy.array(matriz)
b = numpy.array([4, 4, 4])
# Resolve a equação Ax = b
x = numpy.linalg.solve(A, b)
```



ndarray

- Matrizes em NumPy são instâncias do tipo `ndarray`
- Apesar do nome, um `ndarray` pode ser um vetor (dimensão 1), uma matriz tradicional (dimensão 2), um cubo de números (dimensão 3), etc.
- As operações em um `ndarray` são bem rápidas, mesmo para matrizes grandes
- Existem várias maneiras de criar uma matriz NumPy: a partir de listas, ou listas de listas, ou listas de listas de listas, a partir de intervalos, matrizes de zeros ou de uns, matrizes com valores aleatórios, matrizes lidas a partir de arquivos, etc.

Criando matrizes

- A maneira mais simples de criar uma matriz é passando uma lista (para um vetor) ou uma lista de listas (uma lista de linhas para uma matriz bidimensional):

```
A = numpy.array([[1, 3, 4], [2, 3, 5], [5, 7, 9]])
```

- Também podemos criar um vetor, e depois quebra-lo em linhas e colunas com o método `reshape`:

```
A = numpy.array([1, 3, 4, 2, 3, 5, 5, 7, 9])
A = A.reshape(3,3)
```

Matrizes de zeros, uns e identidade

- Podemos criar uma matriz de zeros com a função `zeros`, passando uma tupla com o tamanho de cada dimensão (linhas, colunas, etc.)

```
numpy.zeros((2, 3))
```

- Com a função `ones` criamos uma matriz de uns

```
numpy.ones((2, 3))
```

- Com a função `eye` criamos uma matriz identidade com a *ordem* que passarmos (uma matriz bidimensional com *ordem* linhas e *ordem* colunas)

```
numpy.eye(3)
```

Amostras lineares e logaritmícas

- A função `linspace(a, b, n)` retorna um vetor de n amostras igualmente espaçadas no intervalo fechado `[a,b]`

```
numpy.linspace(0, 100, 50)
```

- A função `logspace(a, b, n)` pega o vetor `linspace(a, b, n)` e faz todas as amostras serem expoentes na base 10:

```
numpy.logspace(0, 100, 50)
```

- Se quisermos uma matriz é só usar `reshape`!

A partir de funções

- A função `fromfunction(func, tamanho)` cria um vetor com o tamanho de cada dimensão dado pela tupla `tamanho`, e onde cada elemento é dado pela função `func` (que recebe o índice do elemento em cada dimensão)

```
def triang(i, j):
    if i >= j:
        return 0.0
    else:
        return float(i+j)

numpy.fromfunction(triang, (3, 3))
```

Matrizes aleatórias

- A função `random.sample(tamanho)` recebe uma tupla com os tamanhos de cada dimensão e retorna uma matriz com números aleatórios entre 0 e 1, em uma distribuição uniforme

- A função `random.normal(mediana, desvio, tamanho)` retorna uma matriz com números aleatórios em volta da mediana dada, em uma distribuição normal com o desvio padrão dado

- A biblioteca `numpy.random` possui funções para retornar matrizes aleatórias segundo diversas outras distribuições

Gravando e lendo uma matriz

- É normal trabalhar com grandes matrizes com o numpy, por isso ele oferece mecanismos para escrever e ler matrizes de arquivos
- O padrão é uma linha por linha do arquivo, com os elementos separados por espaços
- A função `savetxt(arquivo, matriz)` grava uma matriz no arquivo dado
- A função `loadtxt(arquivo)` lê uma matriz do arquivo dado

Atributos das matrizes

- Matrizes são objetos, e têm alguns atributos úteis

```
mat = numpy.random.sample((2, 5))
mat.ndim  # número de dimensões
mat.shape # tamanho (tupla)
mat.size  # número de elementos
mat.T     # matriz transposta
```

- Elas também têm vários métodos; já vimos o `reshape`, outros são `min()`, que retorna o elemento mínimo, `max()`, que retorna o máximo, `mean()`, que retorna a média, `sum()`, que retorna a soma de todos os elementos...

Operações aritméticas

- Uma operação aritmética entre uma matriz e um escalar faz a operação em todos os elementos da matriz, dando uma outra matriz

```
m1 = numpy.random.sample((5,5))
m2 = mat * numpy.pi / 2
```

- Uma operação aritmética entre duas matrizes de mesmo tamanho faz a operação elemento operação entre elementos correspondentes (`A * B` não é a multiplicação de matrizes!)

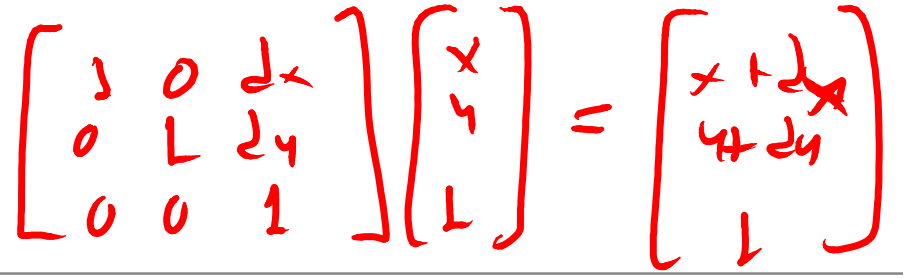
```
m1 = numpy.random.sample((3,3))
m2 = numpy.random.sample((3,3))
m3 = m1 * m2
```

- Pode-se também fazer operações entre uma matriz linha e uma coluna (ou vice-versa)

Outras operações em vetores e matrizes

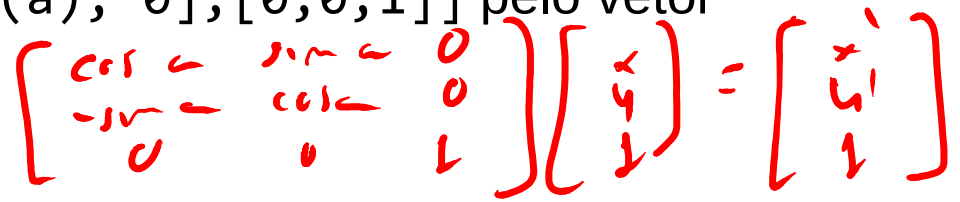
- Produto interno entre vetores: `numpy.dot(v1, v2)`
- Produto cruzado: `numpy.cross(v1, v2)`
- Multiplicação de matrizes também usa `numpy.dot(A, B)`
  - Vetores são tratados como matrizes coluna
- Inversa é `numpy.linalg.inv(A)`
- `numpy.linalg.solve(A, b)` resolve o sistema Ax = b, onde A é uma matriz quadrada e b um vetor

Transfomações 2D



- Podemos fazer translação de um ponto `(x,y)` multiplicando a matriz `[[1,0,dx],[0,1,dy],[0,0,1]]` pelo vetor `[x,y,1]`

- Rotação de um ponto `(x,y)` em volta da origem é multiplicar a matriz `[[cos(a), sin(a), 0],[-sin(a), cos(a), 0],[0,0,1]]` pelo vetor `[x,y,1]`



- O resultado sempre é um vetor `[x',y',1]`, então podemos compor várias operações em uma única matriz

- Vamos aplicar essas transformações no programa da tartaruga

Fatiando matrizes e acessando elementos

- Podemos pegar um pedaço de uma matriz com a mesma sintaxe que fatiamos listas, usando um intervalo para cada dimensão:

```
A = numpy.arange(1,26).reshape(5,5)
print A[1:3,0:3]
```

- Podemos acessar um elemento da matriz separando os índices com vírgulas, mas isso raramente é necessário: `A[1,2]` acessa o terceiro elemento da segunda linha

- Podemos extrair uma linha com `A[lin,:]`, e uma coluna com `A[:,col]`

matplotlib

- matplotlib é uma biblioteca para plotar gráficos 2D: linhas, pontos, histogramas, etc.

- Suas funções estão na biblioteca `matplotlib.pyplot`

- A função principal é `matplotlib.pyplot.plot` para gráficos de linhas; na sua forma mais básica o primeiro argumento é um vetor com as abcissas e o segundo com as ordenadas

```
X = numpy.linspace(0,10,1000)
Y = numpy.power(X,2)
matplotlib.pyplot.plot(X, Y)
```

- Para um gráfico de pontos usa-se `matplotlib.pyplot.scatter`

Configurando e salvando

- A função `matplotlib.pyplot.xlim` recebe um par com o intervalo que o eixo das abcissas mostra
- A função `matplotlib.pyplot.ylim` faz o mesmo com o eixo das ordenadas
- As funções `matplotlib.pyplot.xlabel` e `matplotlib.pyplot.ylabel` recebem uma string com o título de cada eixo
- A função `matplotlib.pyplot.title` recebe o título do gráfico
- Finalmente, `matplotlib.pyplot.savefig` grava o gráfico no arquivo com o nome dado (extensão png)

Múltiplas linhas

- Podemos fazer um gráfico de várias linhas, passando diversos pares vetores de coordenadas

```
X = numpy.linspace(0,10,1000)
Y1 = numpy.power(X,2)
Y2 = numpy.power(X,3)
Y3 = numpy.power(X,4)
matplotlib.pyplot.plot(X, Y1, X, Y2, X, Y3)
```

- Podemos dar legendas com a função `matplotlib.pyplot.legend`, passando uma lista de legendas

Histograma

- Um histograma é um gráfico que conta quantos elementos do vetor dado estão em cada faixa:

```
matplotlib.pyplot.hist(dados, faixas, intervalo)
```

- Exemplo:

```
A = numpy.random.normal(100, 15, 100000)
matplotlib.pyplot.hist(A, 100, (50, 150))
```

Figuras

- Uma 3-matriz (cubo) de dimensões (M, N, 4) pode ser uma imagem com largura M e altura N se cada interseção linha/coluna é uma quádrupla de números entre 0.0 e 1.0

- O primeiro elemento do trio é o componente vermelho, o segundo é o verde, o terceiro é o azul e o último é transparência (0.0 transparente e 1.0 opaco)

- Podemos ler uma figura de um arquivo com `matplotlib.pyplot.imread`, passando o nome do arquivo

- Para mostrar a imagem usamos `matplotlib.pyplot.imshow` passando a 3-matriz

Convolução

- Uma operação muito comum em imagens:

```
def convolucao(M, K):
    MS = []
    n = len(K)
    a = (n-1)/2
    for i in range(a,M.shape[0]-a-1):
        for j in range(a,M.shape[1]-a-1):
            for k in range(3):
                MS.append((M[i-a:i+a+1,j-a:j+a+1,k] * K).sum())
            MS.append(1.0)
    return numpy.array(MS).reshape((M.shape[0]-n,M.shape[1]-n,4))
```

- M é uma imagem, K um *kernel*, aplicado em cada componente de cor

```
G = numpy.array([[0.00000067, 0.00002292, 0.00019117, 0.00038771, 0.00019117, 0.00002292, 0.00000067],
                 [0.00002292, 0.00078634, 0.00655965, 0.01330373, 0.00655965, 0.00078633, 0.00002292],
                 [0.00019117, 0.00655965, 0.05472157, 0.11098164, 0.05472157, 0.00655965, 0.00019117],
                 [0.00038771, 0.01330373, 0.11098164, 0.22508352, 0.11098164, 0.01330373, 0.00038771],
                 [0.00019117, 0.00655965, 0.05472157, 0.11098164, 0.05472157, 0.00655965, 0.00019117],
                 [0.00002292, 0.00078633, 0.00655965, 0.01330373, 0.00655965, 0.00078633, 0.00002292],
                 [0.00000067, 0.00002292, 0.00019117, 0.00038771, 0.00019117, 0.00002292, 0.00000067]])
```