Python

Uma breve introdução a operações com `NumPy` Arrays e visualizações de dados com `matplotlib`

𝔾. Marques

Python 3

Importante: usar uma versão do Python $\geq 3.2.0$

Documentação:

- Ver `http://www.python.org/`

Instalação:

- Ver `https://www.python.org/downloads/`

Para instalar de módulos de Python usar `pip` (depois de instalar Python)

- Usar a linha de commando para instalar módulos:
  ver versão: `pip --version`
  instalar módulo: `pip install nome-do-módulo`
- Exemplo: instalar vários módulos
  `pip install numpy scipy matplotlib scikit-learn`

NumPy

NumPy é um módulo de Python para cálculo científico, que incluí várias rotinas de álgebra linear, análise de Fourier, métodos estatísticos, geração de números aleatórios, e muitas outras. De particular importância é o objeto `ndarray` - uma forma de encapsular dados multi-dimensionais (vetores ou matrizes) em arrays, para os quais existem uma série de operações pré-compiladas de alto desempenho computacional.

Documentação:

- http://www.numpy.org/
- NymPy User Guide.
- **Leitura obrigatória**- NumPy
  Introduction to NumPy por M. Scott Shell

Convenção - para usar NumPy, fazer: `import numpy as np`

NumPy : Arrays

Criação de arrays:

Arrays são semelhantes a listas, exceto que todos os elementos dum array têm que ser do mesmo tipo:

```
>>> a=np.array([0,2,5,8],float)
```

Arrays podem ser multi-dimensionais:

```
>>> a=np.array([[0,1,3],[4,5,6]],float)
```

O array `a` têm vários atributos e métodos associados:

Atributos `a.shape`, `a.size`, `a.dtype`

Métodos `a.sort()` `a.transpose()` (ou `a.T`), `a.fill()`, `a.copy()`, `a.flatten()`, `a.clip()`, `a.min()`, `a.max()`, , `a.mean()`, `a.var()`, `a.std()`, ...

NumPy : Arrays

Criação de arrays:

Funções úteis para criação de arrays:

```
>>> a=np.arange(0,11,2,float)
>>> a=np.zeros((4,7),int); b=np.ones((4,7),float)
>>> a=np.identity(5,float); b=np.eye(5,k=0)
```

Funções para concatenar arrays:

```
>>> a=np.array([[1,2],[3,4]])
>>> b=np.array([5,6])      # b.shape=(2,)
>>> c=np.array([[5],[6]])# c.shape=(2,1)
>>> np.vstack((a,b))       # empilhar na vertical - ok
 array([[1   , 2],
        [3   , 4],
        [5   , 6]])
>>> np.hstack((a,b))      # empilhar na horizontal -not ok
ValueError:  arrays must have same number of dimensions
>>> np.hstack((a,c))       # já dá
 array([[1   , 2   , 5],
        [3   , 4   , 6]])
```

- Nota: ver igualmente a função `np.concatenate()`

NumPy : Arrays

Criação de arrays (L0AAex001.py):

```
1  # -*- coding: latin-1- -*-
2  #1ª linha para poder usar acentos
3  import numpy as np
4  a=np.array([1,3,5,2,4,6],np.uint8)
5  print(a.dtype)          #cuidado com o tipo dos dados
6  print('a=',a)
7  print('a-5=',a-5)
8  a=np.array([1,3,5,2,4,6],np.float)
9  print(a.dtype)
10 print(a.shape)
11 a=a.reshape((3,2))      #modificar dimensões
12 print(a.shape)
13 print(a[0,:])           #1ª linha
14 print(a[:,0])           #1ª coluna
15 b=a
16 a[0,0:2]=0              #2 primeiros elementos da 1ª linha = 0
17 print('b=',b)            # modificar um, modifica o outro
18 b=a.copy()              # para copiar usar .copy()
19 a[1,0:2]=-1
20 print('a=',a)
21 print('b=',b)
```

NumPy : Arrays

Operações com arrays:

- Operações matemáticas standard (adição, subtração, etc) em arrays são aplicadas elemento a elemento, o que significa que os arrays têm que ter as mesmas dimensões.
- Multiplicação e divisão são feitas elemento a elemento e por isso **não correspondem a multiplicação ou divisão matricial**.
- Quando dois arrays são de dimensões diferente, o array menor é "difundido" (*broadcasted* - copiado várias vezes) de maneira aos dois arrays terem as mesmas dimensões.

```
>>> a=np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> b=np.array([0,1,2])
>>> c=np.array([0,1])
>>> a+b # ok
>>> a+c # not ok
ValueError: operands could not be broadcast together with
shapes (3,3) (2)
```

- Algumas funções matemáticas: `abs`, `sign`, `sqrt`, `log`, `sin`, `cos`, `tan`, `arctan`, `arctan2`, `tanh`, `arctanh`, `sinc`, `exp`

NumPy : Arrays

Operações com arrays (L0AAex002.py):

```
1  import numpy as np
2  a=(np.arange(-6,6,2,float)+1).reshape((2,3))
3  print('a=',a ,'\nsoma=' , a.sum(),'\nproduto=',a.prod(),\
4          '\nmedia=',a.mean(),'\nDesvio Padrao=',a.std())
5  #ou em alternativa
6  print('a=',a ,'\nsoma=' , np.sum(a),'\nproduto=',np.prod(a),\
7          '\nmedia=',np.mean(a),'\nDesvio Padrao=',np.std(a))
8  b=np.array([[1,2,3],[4,3,2]],dtype=float)
9  print('a*b=',a*b,'\na**b=',a**b)
10 c=b[:,0]
11 print('\nc=',c)
12 #print('a+c',a+c ,'a*c',a*c)
13 #ERRO: a.shape=(2,3) e c.shape=(2,)
14 print('a+c=',a+c[:,np.newaxis],'\n a*c=',a*c[:,np.newaxis])
15 #funciona c[:,np.newaxis].shape=(2,1)
16 #c -> adicionada a cada coluna de a
```

Selecção e Manipulação de Elementos de Arrays:

A selecção e indexação de elementos de Arrays pode ser feita de várias maneiras (ver página Array Indexing)

1. *Slicing*
2. Indexação com arrays booleanos
3. Indexação com arrays de inteiros

NumPy : Arrays

Selecção e Manipulação de Elementos de Arrays:

A selecção e indexação de elementos de Arrays pode ser feita de várias maneiras (ver página Array Indexing)

1. *Slicing* - sintaxe : `start:end:step`

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

valores negativos para `start` e `end` são interpretados como `n+start/end` onde `n` é o numero total de elementos do array.

```
>>> a[-3:2:-1]
array([7, 6, 5, 4, 3])
>>> a[-3:]# o mesmo que a[-3:10]
array([7, 8, 9])
```

2. Indexação com arrays booleanos
3. Indexação com arrays de inteiros

NumPy : Arrays

Selecção e Manipulação de Elementos de Arrays:

A selecção e indexação de elementos de Arrays pode ser feita de várias maneiras (ver página Array Indexing)

1. *Slicing*
2. Indexação com arrays booleanos

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

3. Indexação com arrays de inteiros

NumPy : Arrays

Selecção e Manipulação de Elementos de Arrays:

A selecção e indexação de elementos de Arrays pode ser feita de várias maneiras (ver página Array Indexing)

1. *Slicing*
2. Indexação com arrays booleanos
3. Indexação com arrays de inteiros

```
>>> a=np.array([3,0,2,4,1])
>>> b=np.array([3,3,2,1,0,0,4])
>>> a[b]
array([4, 4, 2, 0, 3, 3, 1])
>>> c=np.array([5,3,2,1,0,0,4])
>>> a[c]
IndexError:  index 5 out of bounds 0<=index<5
```

NumPy : Arrays

Cálculo Vectorial com Arrays:

O NumPy fornece várias funções para a realização de operações vectorias matriciais e de álgebra linear.

- Produto interno: função `np.dot()`

```
>>> a=np.array([3,0,2,4,1])
>>> b=np.array([2,1,5,4,7])
>>> np.dot(a,b)
39
```

A função `np.dot()` é também aplicável a multiplicação matricial

```
>>> a=np.array([[3,0,2],[2,4,1],[1,5,3]])
>>> b=np.array([2,1,5])
>>> c=np.array([[1,2,3],[6,5,4],[2,1,0]])
>>> np.dot(a,b)
array([16, 13, 22])
>>> np.dot(b,a)
array([13, 29, 20])
>>> np.dot(a,c)
array([[ 7, 8, 9],[28, 25, 22],[37, 30, 23]])
>>> np.dot(c,a)
array([[10, 23, 13],[32, 40, 29],[ 8, 4, 5]])
```

NumPy : Arrays

Cálculo Vectorial com Arrays:

O NumPy fornece várias funções para a realização de operações vectorias matriciais e de álgebra linear.

- Como alternativa, pode-se declarar variávies da classe `matrix`.

```
>>> A=np.matrix([[3,0,2],[2,4,1],[1,5,3]])
>>> b=np.array([2,1,5]);B=np.matrix(b)
>>> B.shape
(1,3)
>>> B*A
matrix([[13, 29, 20]])
>>> np.dot(b,A)
matrix([[13, 29, 20]])
>>> b*A
matrix([[13, 29, 20]])
>>> A*b
ValueError:  objects are not aligned
```

NumPy : Arrays

Cálculo Vectorial com Arrays:

O NumPy fornece várias funções para a realização de operações vectorias matriciais e de álgebra linear.

- O NumPy tem uma série de funções de álgebra linear que se encontram disponíveis no sub-módulo `linalg`.
- Funções úteis (funcionam com classes `array` e `matrix`):

```
>>> A=np.array([[3,0,1],[0,4,1],[1,0,2]])
>>> np.linalg.det(A)                    ⟶ Determinante de A
    20.000000000000007
>>> np.linalg.inv(A)                    ⟶ Inverso de A
     array([[ 0.   , 0.5          , -0.5 ],
            [ 1.   , -0.80901699  , -0.30901699],
            [ 0.   , 0.30901699   , 0.80901699]])
>>> val,Vec=np.linalg.eig(A) ⟶ Valores e vectores próprios de A
>>> val
    array([ 4.   , 3.61803399, 1.38196601])
>>> Vec
     array([[ 0.   , 0.5          , -0.5 ],
            [ 1.   ,-0.80901699  ,-0.30901699],
            [ 0.   , 0.30901699  , 0.80901699]])
```

matplotlib

`matplotlib` é um módulo de Python para visualizações de dados (inspirado nos comandos de MatLab). `matplotlib` permite fazer gráficos (2D e 3D) de funções, dados temporais espaciais (ex: ficheiros de áudio e imagem), histogramas, gráficos de contornos, barras, pontos individuais, entre muitas outras funcionalidades.

Documentação:

- `http://matplotlib.org/`
- matplotlib User Guide.
- Anatomy of Matplotlib por Benjamin Root do GitHub.
- **Leitura obrigatória** para novos utilizadores de `matplotlib`: Capítulo 2.1 do matplotlib User Guide (release 3.2.2), sobre como usar `pyplot`.

Convenção fazer: `import matplotlib.pyplot as plt`
ou: `from matplotlib import pyplot as plt`

matplotlib: Gráficos 2D

Visualização de funções

Para funções de variáveis contínuas é necessário criar a ilusão de continuidade

- Selecionar um intervalo de visualização da função (ex: para $f(x)$ escolher x entre $[x_{\min}, x_{\max}]$)
- Criar um array com valores de x no intervalo escolhido, com incrementos idênticos e suficientemente pequenos para obter uma boa definição.

Comando: `plot()`

comandos relacionados: `figure()`, `subplot()`, `axis()`, `grid()`, ...

matplotlib: Gráficos 2D

L0AAex003a.py: Fazer plot da função $f(x) = \frac{\sin(\pi x)}{\pi x}$

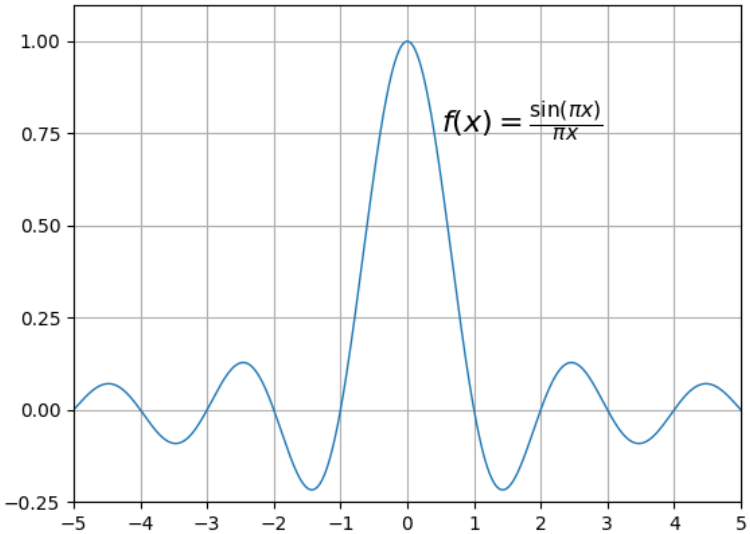
```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  #desenhar a função sinc(x)
5  x=np.linspace(-5,5,1000) #1000pts equi-espaçados entre [-5,5]
6  fx=np.sin(np.pi*x)/(np.pi*x) #fx é a função "sinc(x)"
7  plt.plot(x,fx,lw=1)
8  plt.axis([-5,5,-0.25,1.1])    #eixos
9  plt.grid()                    #grelha
10 plt.xticks(np.arange(-5,6))   #intervalos no x = 1
11 plt.yticks(np.arange(-.25,1.25,.25))  #intervalos no y = 1/4
12 plt.text(.5,.75,r'$f(x)=\frac{\sin(\pi x)}{\pi x}$',fontsize=16)
13 plt.savefig('../figs/L0AAex003a.png') #guardar  em ficheiro ".png"
14 plt.show()                           #(na directoria "../figs/")
```

matplotlib: Gráficos 2D

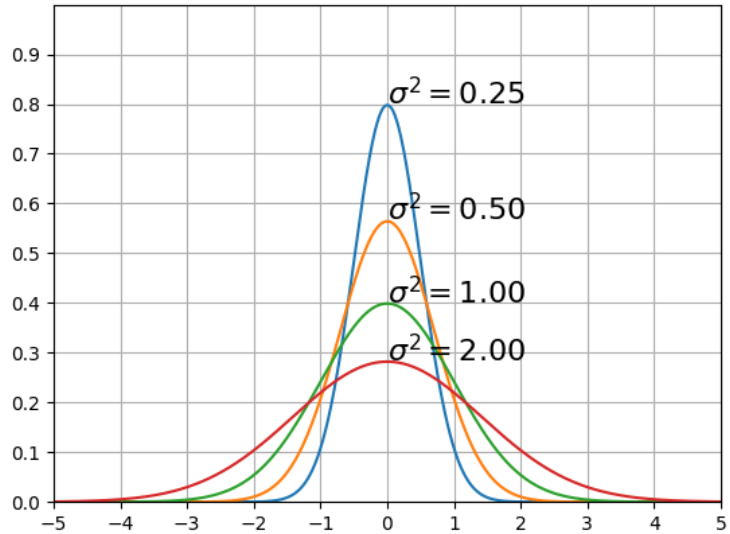
L0AAex003b.py: Fazer plot da função $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  #gráfico de Gaussiana de média 0 (para dif.  variancias)
5  x=np.linspace(-5,5,1000) #1000pts equi-espaçados entre [-5,5]
6  sigma2=np.array([1./4,1./2,1.,2.]) #valores da variancia
7  plt.figure()
8  for s in sigma2:
9      fx=1/np.sqrt(2*np.pi*s)*np.exp(-1./(2*s)*x**2)
10     plt.plot(x,fx)
11     strTmp=r'$\sigma^2=$%0.2f'%s
12     plt.text(0,fx[500],strTmp,fontsize=16)
13 plt.axis([-5,5,0,1])
14 plt.xticks(np.arange(-5,6))     #intervalos no x =  1
15 plt.yticks(np.arange(0,1,.1))  #intervalos no y =  0.1
16 plt.grid()
17 #plt.savefig('../figs/L0AAex003b.png') #guardar em ficheiro ".png"
18 plt.show()
```

matplotlib: Gráficos 2D



L0AAex003a.png



L0AAex003b.png

matplotlib: Gráficos 2D

Sinais Discretos (comando `stem()`)

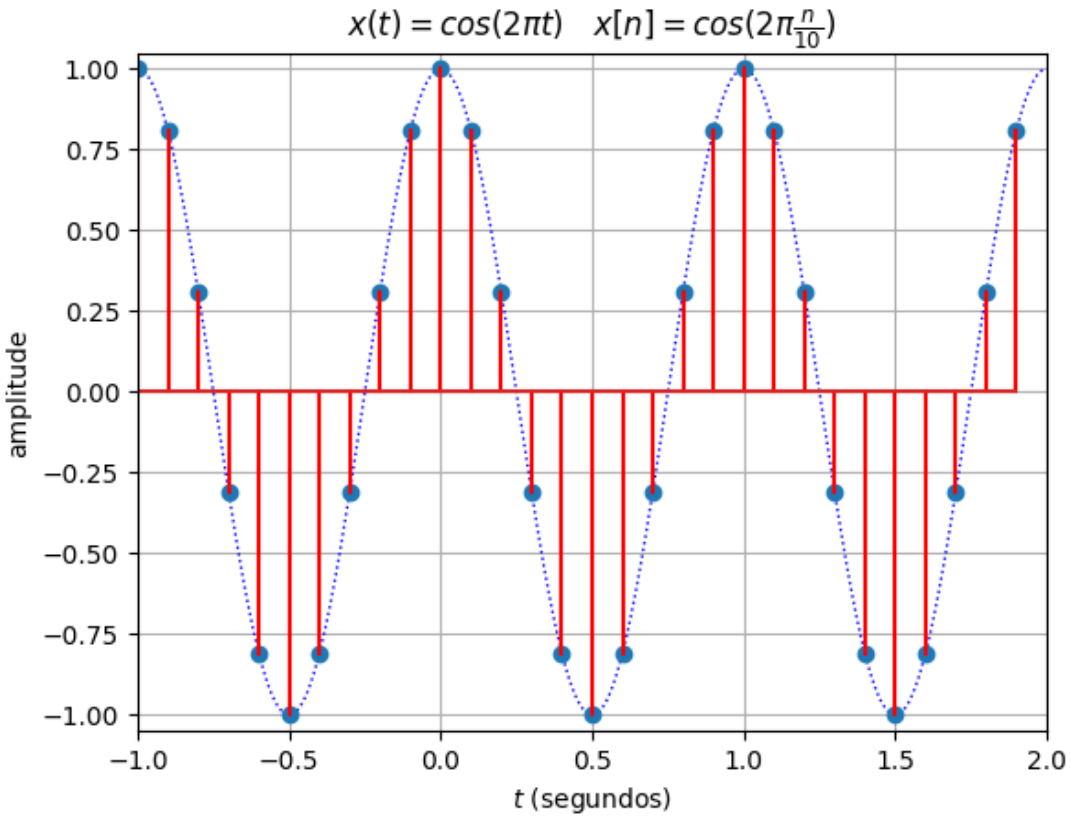
`L0AAex003c.py`: Visualizar $x(t) = \cos(2\pi t)$ amostrado a $F_s = 10$Hz

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  #representar versão amostrada de sinosóide
5  t=np.linspace(-1,2,1000) #1000pts equi-espaçados entre [-1,2]
6  fc=1 #frequência da sinosóide
7  x_t=np.cos(2*np.pi*fc*t) #fc Hz
8  N=10 #n° amostras segundo (N=fs, freq. de amostragem)
9  #tempo discreto (T=1/fs-> amostras de 0.1 em 0.1 segundos)
10 n=np.arange(-10,20) #[-1xN,2xN]
11 x_n=np.cos(2*np.pi*fc*n/N)
12 plt.plot(t,x_t,':b',lw=1)
13 plt.stem(n*1./N,x_n,'r')
14 plt.axis([-1,2,-1.05,1.05])  #eixos
15 plt.grid(True)               #grelha
16 plt.xlabel(r'$t$␣(segundos)'),plt.ylabel('amplitude')
17 plt.title(r'$x(t)=cos(2\pi␣t)\quadx[n]=cos(2\pi\frac{n}{10})$')
18 plt.xticks(np.arange(-1,2.1,.5))  #intervalos no x = 1/2
19 plt.yticks(np.arange(-1,1.1,.25))  #intervalos no y = 1/4
20 plt.savefig('../figs/L0AAex003c.png') #guardar  em ficheiro ".png"
21 plt.show()                          #(na directoria "../figs/")
```

matplotlib: Gráficos 2D

Sinais Discretos (comando `stem()`)

`L0AAex003c.py`: Visualizar $x(t) = \cos(2\pi t)$ amostrado a $F_s = 10$Hz



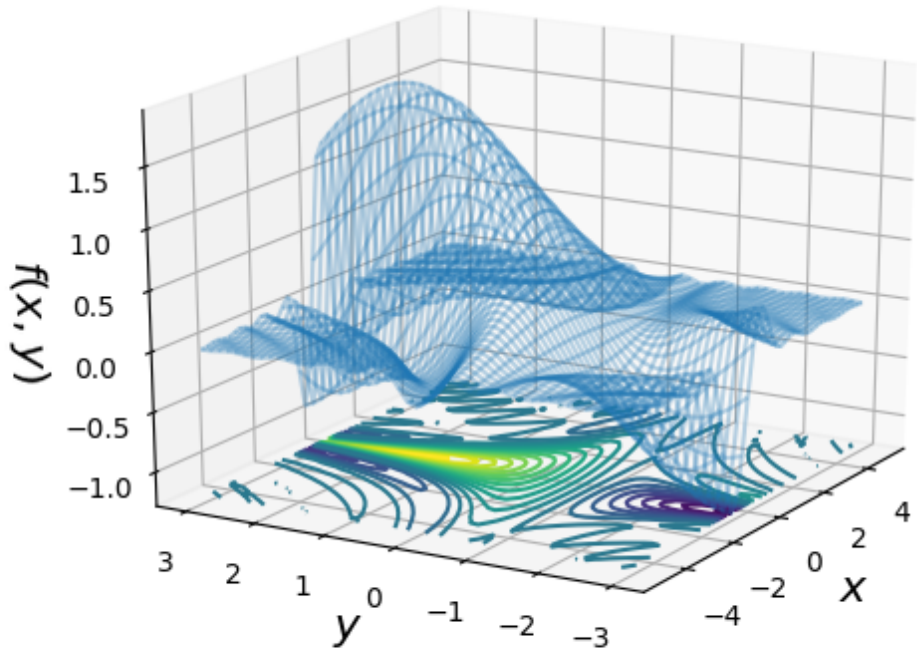
matplotlib: Gráficos 3D

L0AAex004a.py: Fazer plot da função $f(x,y) = \frac{\sin(x+xy)\sin(y+xy)}{(x+xy)x}$

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5  #visualizar função f(x,y)=sin(x+x*y)*sin(y+x*y)/(x+x*y)/x
6  #criar grelha com função meshgrid
7  X,Y=np.meshgrid(np.linspace(-5,5.,50),np.linspace(-3,3.,50))
8  #X e Y arrays bi-dimensionais de 50x50
9  Z=np.sin(X+X*Y)*np.sin(Y+X*Y)/(Y+X*Y)/X
10 f1=plt.figure() #criar figura
11 ax=f1.add_subplot(111,projection='3d') #3D
12 ax.plot_wireframe(X,Y,Z,alpha=.3)
13 ax.contour(X,Y,Z,25,offset=Z.min()) #25=nº contornos
14 ax.elev=20   #posição da câmera:elevação
15 ax.azim=-150 #posição da câmera:azimute (em graus)
16 ax.set_xlabel('$x$',fontsize=16)
17 ax.set_ylabel('$y$',fontsize=16)
18 ax.set_zlabel('$f(x,y)$',fontsize=16)
19 #plt.savefig('../figs/L0AAex004a.png') #guardar em ficheiro ".png"
20 plt.show()
```

matplotlib: Gráficos 3D

`L0AAex004a.py`: Fazer plot da função $f(x,y) = \dfrac{\sin(x+xy)\sin(y+xy)}{(x+xy)x}$



matplotlib: Gráficos 3D

L0AAex004b.py:

Fazer plot da curva dada por:

$$x = \sin(t/2)sin(t^2/10)$$
$$y = \cos(t)$$
$$z = (t/20)^2$$

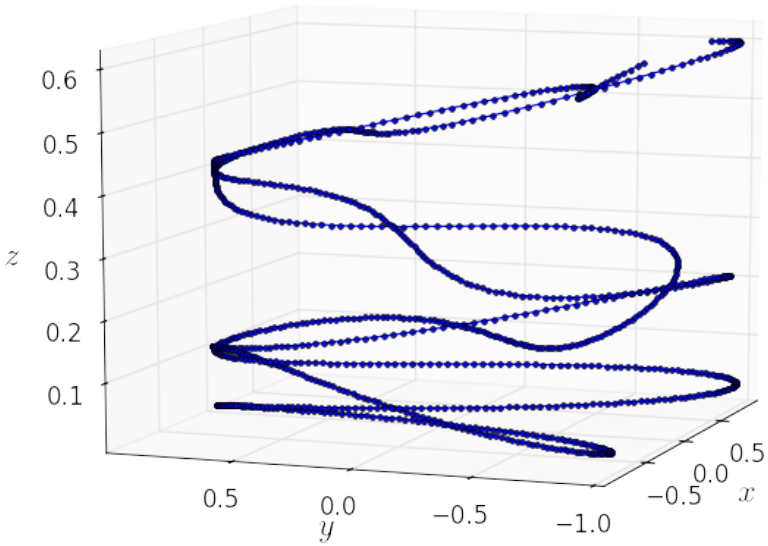
```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5  #visualizar  curva dada por: t=[-5pi,5pi]
6  # x=sin(t/2)*sin(t**2/10)  y=cos(t)  z=(t/20)**2
7  t=np.linspace(-np.pi,np.pi,1000)*5
8  x=np.sin(t/2)*np.sin(t**2/10)
9  y=np.cos(t);z=(t/20)**2
10 f1=plt.figure() #criar figura
11 ax=f1.add_subplot(111,projection='3d') #3D
12 ax.plot(x,y,z,'.-b')
13 ax.elev=40   #posição da câmera:elevação
14 ax.azim=-92 #posição da câmera:azimute (em graus)
15 ax.set_xlabel('$x$',fontsize=16)
16 ax.set_ylabel('$y$',fontsize=16)
17 ax.set_zlabel('$z$',fontsize=16)
18 #plt.savefig('../figs/L0AAex004b.png') #guardar em ficheiro ".png"
19 plt.show()
```

matplotlib: Gráficos 3D

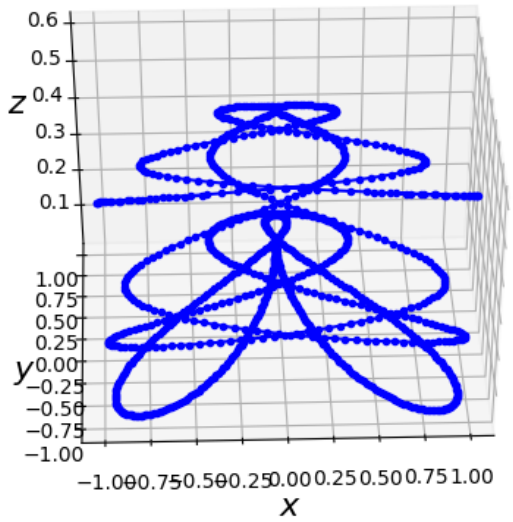
L0AAex004b.py:

Fazer plot da curva dada por:

$$x = \sin(t/2)sin(t^2/10)$$
$$y = \cos(t)$$
$$z = (t/20)^2$$



(ax.elev=10, ax.azim=-160)



(ax.elev=40, ax.azim=-92)

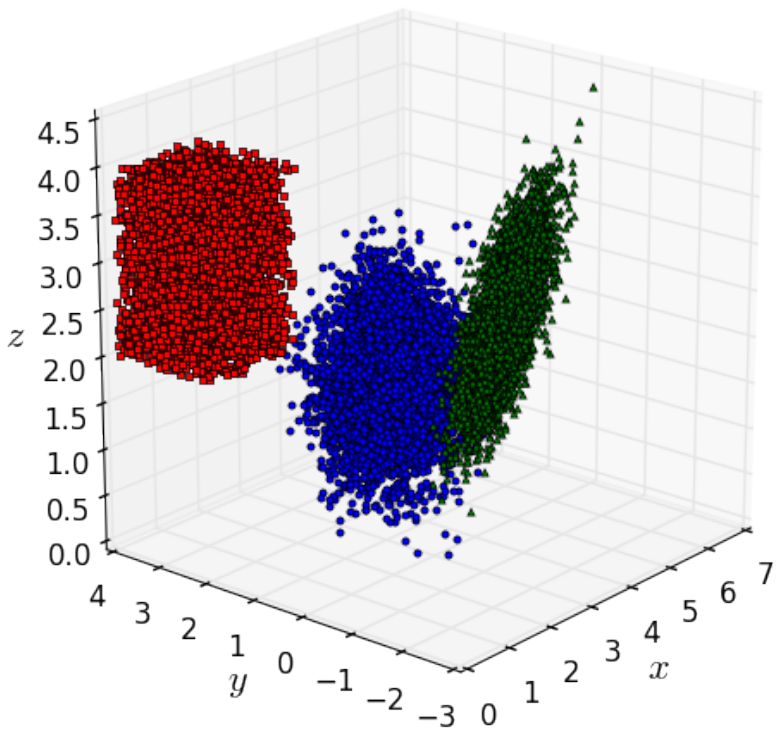
matplotlib: Gráficos 3D

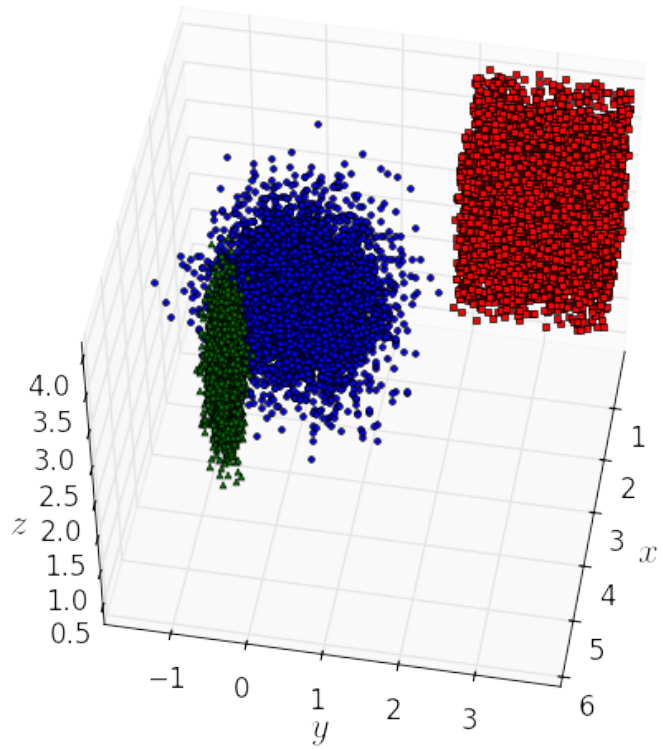
L0AAex004c.py: Gráfico de pontos gerados aleatoriamente:

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5  #Gerar pontos 3D (três grupos com 5000 pts cada)
6  x1=np.random.randn(3,5000)*1/2.+np.array([[2],[0],[2]])
7  x2=np.random.rand(3,5000)*2+np.array([[0],[2],[2]])
8  A3=np.array([[1,.5,0],[.1,.0,.1],[.1,1,.1]])
9  x3=A3.dot(x1)-np.array([[-2],[1],[-2]])
10 f1=plt.figure() #criar figura
11 ax=f1.add_subplot(111,projection='3d') #3D
12 ax.plot(x1[0,:],x1[1,:],x1[2,:],'ob',markersize=2,alpha=.25)
13 ax.plot(x2[0,:],x2[1,:],x2[2,:],'sr',markersize=2,alpha=.25)
14 ax.plot(x3[0,:],x3[1,:],x3[2,:],'^g',ms=2,alpha=.25)
15 ax.elev=20   #posição da câmera:elevação
16 ax.azim=-140 #posição da câmera:azimute (em graus)
17 ax.set_xlabel('$x$',fontsize=16)
18 ax.set_ylabel('$y$',fontsize=16)
19 ax.set_zlabel('$z$',fontsize=16)
20 #ax.set_aspect('equal','box')  #todos eixo à mesma escala
21 #plt.savefig('../figs/L0AAex004c.png',\
22 #bbox_inches='tight',transparent=True)
23 plt.show()
```

matplotlib: Gráficos 3D

L0AAex004c.py: Gráfico de pontos gerados aleatoriamente:





`matplotlib`: Leitura e Visualização Imagens

Funções `imread()` e `imshow()`.

Há alguns pontos a ter em consideração quando se lida com imagens:

- Tipicamente o valor dos pixeis está representado por inteiros de 8 bit (unsigned). Atenção que com este tipo de variável é só possível representar números inteiros entre $[0, 255]$ ($255+1=0$, ou $0-1=255$).
- O próximo exemplo mostra uma imagem e a mesma imagem multiplicada por dois, quando estas estão guardadas em arrays do tipo `uint8`.
- Para processar a imagem convém converter os valores dos pixeis para `float`

matplotlib: Leitura e Visualização Imagens

L0AAex005.py: ler e visualizar imagens

```
1  # -*- coding: latin-1 -*-
2  #AA - script para ler e visualizar imagens
3  import matplotlib.pyplot as plt
4  fName="lena.tif"   #necessário estar na mesma dir. que código
5  I=plt.imread(fName)#ler imagens (I-> numpy array uint8)
6  plt.subplot(1,2,1) #1x2 figuras (3º valor = índice da figura 1 ou 2)
7  plt.imshow(I) #origem no canto inferior esquerdo
8  #tirar eixos - plt.axis('off') tb dá
9  plt.xticks([]),plt.yticks([]),plt.box(True)
10 #atenção que pixeis estão em  uint8
11 #só é possível representar valores entre 0-255
12 plt.subplot(1,2,2)
13 plt.imshow(I*2)
14 plt.xticks([]),plt.yticks([]),plt.box(True)
15 #guarda figura
16 #plt.savefig('../figs/L0AAex005.png', transparent=True, bbox_inches
       ='tight', pad_inches=0)
17 plt.show() #não é necessário em     #guardar  em ficheiro ".png"
18           #iPython usar show()     #(na directoria "../figs/")
19                                    #(dá erro se não existir)
```

matplotlib: Leitura e Visualização Imagens

L0AAex005.py: ler e visualizar imagens

Imagem **I**

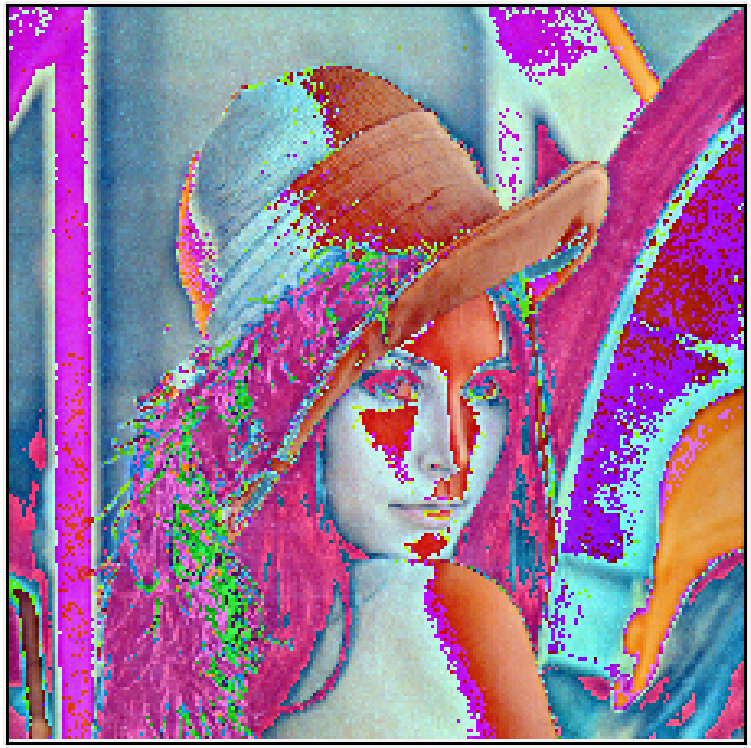


Imagem **I**×2

`matplotlib`: Visualizações

Função `imshow()` - pontos a ter em consideração:

- A função escala as imagens para um tamanho pré-definido. Isto pode introduzir artefactos tais como a sobreposição espectral (*aliasing*).
- A função faz uma filtragem passa-baixo - para desativar, chamar a função com o parâmetro `interpolation='none'`.
- A função usa, por omissão, o "colormap" `jet` (mesmo para imagens a tons de cinzento). Para visualizar imagens em tons de cinzento, chamar a função com o parâmetro `cmap='gray'`.

A função `imshow()`, além de servir para visualizar imagens, permite igualmente visualizar Numpy arrays.

Python `pickle`

Para guardar dados usar o módulo `pickle` (vem com o Python).

Guardar variáveis

- O módulo `pickle` permite guardar um objecto de Python num ficheiro:

```
>>> import pickle
>>> pickle.dump(Obj,open(filename,'wb'))
```

- Caso se queira guardar várias variáveis, pode-se usar um dicionário.
  Exemplo: para guardar variáves `X1, X2, DBinfo, trueClass'`

```
>>> Dados={'X1':X1,'X2':X2,'info':DBinfo,'Classes':trueClass}
>>> pickle.dump(Dados,open('dados.p','wb'))
```

Ler ficheiros `pickle`

```
>>> X=pickle.load(open('dados.p','rb'))
>>> X.keys()
dic_keys(['X1','X2','info','Classes'])
```