Informatique en CPGE (2014-2015)
Exercices 9
Les bibliothèques Numpy, Matplotlib, Scipy

1 Installation de bibliothèques

Attention, suivant les versions, les bibliothèques ne sont pas toutes compatibles.

Un choix qui fonctionne est Idle version 3.3.2 : https://www.python.org/download/releases/3.3.2/ avec les versions de bibliothèques citées ci-dessous.

Choisir le fichier d'installation correspondant à votre ordinateur : Windows X86 MSI installer ou Windows X86-64 MSI Installer pour les systèmes Windows 32 bits ou 64 bits et Mac Os X 64-bits ou Mac Os X 32-bits pour les systèmes Mac 32 ou 64 bits. En général Python est déjà livré avec les différentes distributions Linux.

Installer les bibliothèques complémentaires, par exemple pour Windows 32 bits :

- matplotlib : choisir la version matplotlib-1.2.0.win32-py3.3.exe (http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.2.0/)
- numpy : choisir la version numpy-1.8.2-win32-superstack-python3.3.exe (http://sourceforge.net/projects/numpy/files/NumPy/1.8.2/)
- scipy : choisir la version scipy-0.14.0-win32-superstack-python3.3.exe (http://sourceforge.net/projects/scipy/files/scipy/0.14.0/)

Afin de vérifier si les bibliothèques fonctionnent correctement, recopier dans un fichier le code suivant et l'exécuter.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sc
```

Si le code ne fonctionne pas il faut tester d'autres versions ou alors installer la dernière version de Pyzo (http://http://www.pyzo.org/downloads.html). Deux avantages à Pyzo : fonctionne avec une version 3 de Python et les principales bibliothèques complémentaires sont déjà installées. Au lycée le raccourci pour Pyzo se trouve dans le répertoire "Logiciels" sur le bureau.

Une autre possibilité qui foncionne partout est d'utiliser une version portable de Python où tout est déjà installé : http://portablepython.com/

2 Calculs avec des tableaux

Si nous souhaitons établir un tableau de valeurs pour une fonction f, nous pouvons utiliser des objets de type **list**. Par exemple :

```
def f(x):
    return x**2-3
n=11   # nombre de points en abscisse
dx=2/(n-1)  # espace entre les points sur [0;2]
xliste=[i*dx for i in range(n)]
yliste=[f(x) for x in xliste]
```

Mais dans le cas, comme ici, où les éléments de chaque liste sont du même type et où la longueur de chaque liste est connue, il existe dans la bibliothèque **Numpy** un type plus approprié et avec lequel les calculs seront simplifiés :

```
import numpy as np
a=np.array(xliste)  # a est du type numpy.ndarray
```

La fonction **np.array** transforme une liste en tableau.
Note : pour la suite nous utiliserons le mot "liste" pour un objet de type "list" et le mot "tableau" pour un objet de type "ndarray".

Pour créer un tableau de longueur n rempli de zéros, nous écrivons :

```
a=np.zeros(n)
```

Les éléments de a sont du type **float** ; nous pouvons spécifier le type, par exemple **int** :

```
a=np.zeros(n,int)
```

Pour établir un tableau de valeurs d'une fonction, nous avons souvent besoin de générer un tableau de n nombres uniformément répartis sur un intervalle [p ; q] :

```
x=np.linspace(p,q,n)
```

Les éléments d'un tableau peuvent être extraits commme pour les listes, mais attention, si un élément de l'extrait est modifié, l'élément de l'original l'est aussi :

```
x=np.linspace(0,2,11)
#  x=[ 0.   0.2  0.4  0.6  0.8  1.   1.2  1.4  1.6  1.8  2. ]
z=x[2:9:2]
#  z=[ 0.4  0.8  1.2  1.6]
z[1]=5
#  z=[ 0.4  5.   1.2  1.6]
#  x=[ 0.   0.2  0.4  0.6  5.   1.   1.2  1.4  1.6  1.8  2. ]
```

Tout l'intérêt pour établir un tableau de valeur de la fonction f vue au début se voit ici :

```
x=np.linspace(0,2,11)
y=f(x)
```

Il n'y a plus besoin d'utiliser des boucles for, la fonction f s'applique directement.

Attention : les fonctions du module **math** ne peuvent pas s'appliquer sur les tableaux. Nous devons utiliser les versions de ces fonctions existant dans Numpy.

```
x=np.linspace(0,2,11)
y=np.cos(x)*np.exp(-x**2/2)
```

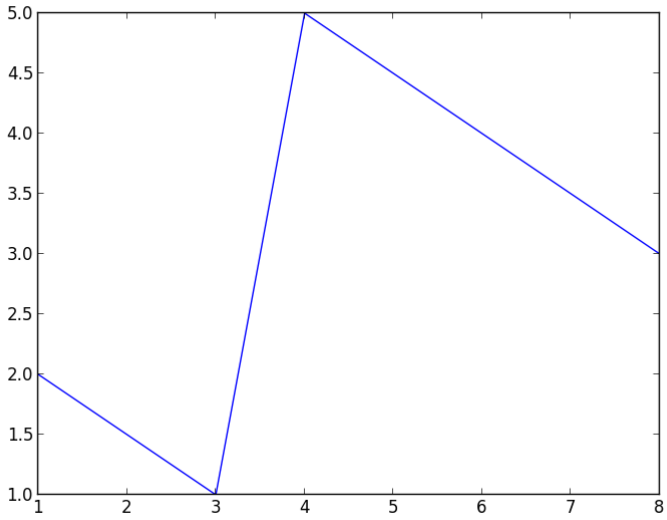
3 Tracé de courbes

Nous allons utiliser la bibliothèque **Numpy** et le module **Pyplot** de la bibliothèque **Matplotlib**.
Le code usuel pour commencer est :

```
import numpy as np
import matplotlib.pyplot as plt
```

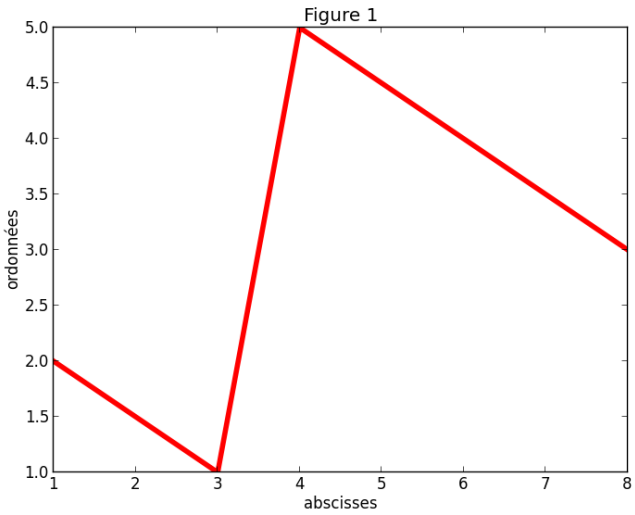
Le principe : on crée une liste d'abscisses et une liste d'ordonnées ; les points sont alors placés et reliés par des segments. Le code est le suivant :

```
x=[1,3,4,8]
y=[2,1,5,3]
plt.plot(x,y)
plt.savefig('figure') #pour sauvegarder la figure
plt.show()
```



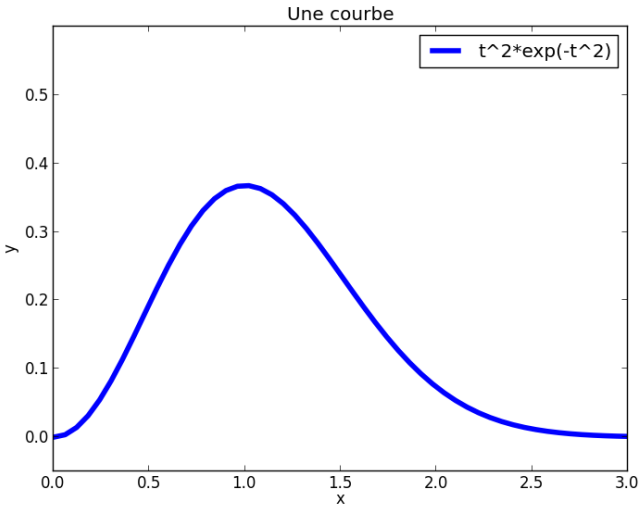
Dans le code suivant, on change la couleur et l'épaisseur du trait, on ajoute un titre et des étiquettes :

```
plt.plot(x,y,color='red',linewidth=6)
plt.title('Figure 1')
plt.xlabel('abscisses')
plt.ylabel('ordonnées')
```



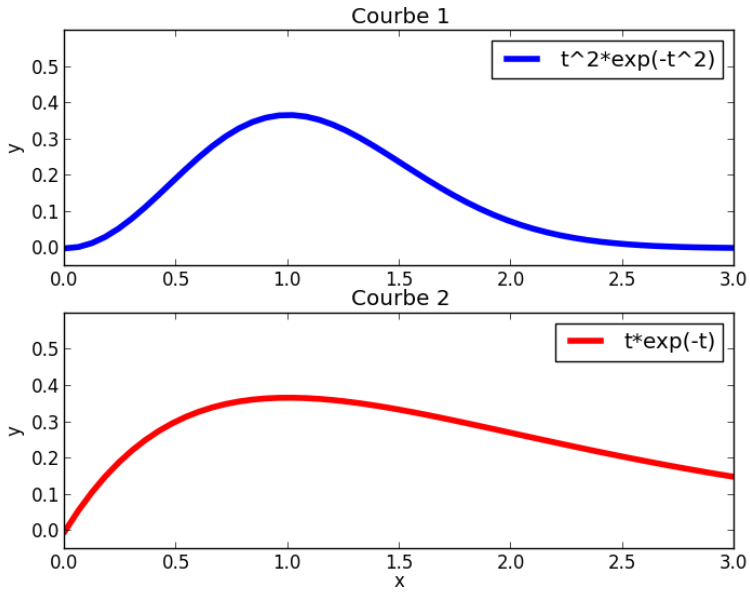
Pour tracer la courbe représentant une fonction :

```
def f(x):
    return x**2*np.exp(-x**2)
x=np.linspace(0,3,51)
y=f(x)
plt.plot(x,y,linewidth=4)
plt.title('Une courbe')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['t^2*exp(-t^2)'])
plt.axis([0,3,-0.05,0.6])
plt.show()
```



On obtient deux courbes sur la même figure avec la commande **subplot(l,c,n)** où l est le nombre de lignes, c le nombre de colonnes et n un compteur :

```
def f(x):
    return x**2*np.exp(-x**2)
def g(x):
    return x*np.exp(-x)
x=np.linspace(0,3,51)
plt.subplot(2,1,1)
y=f(x)
plt.plot(x,y,linewidth=4)
plt.title('Courbe 1')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['t^2*exp(-t^2)'])
plt.axis([0,3,-0.05,0.6])
plt.subplot(2,1,2)
y=g(x)
plt.plot(x,y,linewidth=4)
plt.title('Courbe 2')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['t*exp(-t)'])
plt.axis([0,3,-0.05,0.6])
plt.show()
```



4 Algèbre linéaire

Les bibliothèques Numpy et Scipy contiennent un module **linalg** utile en algèbre linéaire.
Exemples avec Numpy :

```
import numpy as np

N=[[3,1,1],[0,2,-1],[1,1,4]]
```

```
# determinant de N

print('\ndet de N',np.linalg.det(N))

# inverse de N
N1=np.linalg.inv(N)

print('\ninverse de N',N1)


P=np.array([[1,1,1],[-1,-1,1],[0,2,0]])
Pinv=np.array([[0.5,-0.5,-0.5],[0,0,0.5],[0.5,0.5,0]])
print("produit de matrices",np.dot(P,Pinv))

T=np.array([[2,0,0],[0,3,1],[0,0,3]])
print("\npuissance d'une matrice",np.linalg.matrix_power(T,3))
```

5 Calcul d'intégrales

```
import numpy as np
import scipy.integrate as integ

def f(x):
    return x**2
lx=[i/10 for i in range(11)]
ly=[f(x) for x in lx]
# méthode des trapèzes
s1=0
for i in range(10):
    s1+=(lx[i+1]-lx[i])*(ly[i]+ly[i+1])/2

# avec scipy.integrate
s2=integ.trapz(ly,lx) #attention à l'ordre (ly,lx)

#calcul de l'intégrale de f entre 0 et 1
s3=integ.quad(f,0,1)
```

6 Equation f(x)=0

Dichotomie et méthode de Newton avec scipy.optimize :

```
import scipy.optimize

def f(x):
    return x**2-2

a=1
b=2

x=scipy.optimize.bisect(f,a,b)
print("sol=",x)
```

```
x=scipy.optimize.newton(f,a)
print("sol=",x)
```

Les fonctions **root** et **fsolve** pour trouver des valeurs approchées des zéros d'une fonction :

```
import scipy.optimize
def f(x):
    return x**2-2

a=1

x=scipy.optimize.fsolve(f,a)
print("sol=",x)

x=scipy.optimize.root(f,a)
print("sol=",x)
```

Pour plus d'informations, consulter les documentations officielles :
http://matplotlib.org/contents.html
http://docs.scipy.org/doc/