TP-3 : Modules et fichiers

1 La bibliothèque Matplotlib

1.1 Tracé de courbe $y = f(x)$

Il y a plusieurs modules qui permettent de faire cela. Ici, on utilise le module `pyplot` de la bibliothèque `matplotlib`. D'autres préfèrent le module `pylab` de la même bibliothèque.

Pour représenter graphiquement une fonction f sur un segment $[a, b]$ on peut construire une subdivision $(a_i)_{0\leqslant i\leqslant n}$ de $[a, b]$ avec $a_i = a + i\,\frac{b-a}{n}$, créer la liste `X=[a+i*(b-a)/n for i in range(n+1)]` (on veut $n+1$ points), puis la liste `Y=[f(x) for x in X]` et on demande `plot(X,Y, options)` qui représente la ligne brisée, réunion des segments reliant les points de coordonnées $\big(a_i, f(a_i)\big)$ pour i variant de 0 à n.

```
import matplotlib.pyplot as plt
def dessin(f, a, b, n):
    """retourne la représentation graphique de f sur le segment [a,b]
    avec n+1 points """
    X = [a+i*(b-a)/n for i in range(n+1)]
    Y = [f(x) for x in X]
    plt.figure() # crée, ouvre une fenêtre pour le dessin
    plt.plot(X, Y, 'r') # 'r' pour red

f1 = lambda x : x**3-3*x**2+x-5
dessin(f1,-3, 5, 100)
```

Cela crée une image que l'on peut ensuite enregistrer sous divers format, par exemple 'pdf' ou 'png'. Pour fermer la fenêtre graphique à l'intérieur du script, on utilise `plt.close()`.

On peut rajouter des axes, un titre, fixer le cadre (abscisses et coordonnées minimales et maximales) et beaucoup d'autres choses encore.

On charge le module `math` pour avoir plus de fonctions usuelles.

Dans le module `numpy` il y a la fonction `arange(a, b, h)` qui crée la subdivision de points limites a et b et de pas $h = \frac{b-a}{n}$ et la fonction `linspace(a,b,n)` qui crée une subdivision de n points, donc de pas $\frac{b-a}{n-1}$.

Pour représenter la fonction f définie sur $\mathbb{R}$ par $f(0) = 1$ et $f(x) = \frac{\sin x}{x}$ pour $x \neq 0$, on peut donc entrer

```
import numpy as np
def f(x):
    if x == 0:
        return 1
    return np.sin(x)/x # le else est inutile
plt.figure()
X = np.linspace(-15, 15, 600) # 30 points par unité
Y = [f(x) for x in X]
plt.plot(X, Y, 'b')
plt.axhline(linewidth=2,color='k') # rajoute un axe des abscisses en noir ('k')
plt.axvline(linewidth=2,color='k') # rajoute un axe des ordonnées
plt.title('la fonction f(x)=sin(x)/x')
plt.axis([-15,15,-0.5,1.3])
plt.show()
```

Dans `numpy` toutes les fonctions usuelles sont vectorialisées, c'est-à-dire que par exemple pour `np.sin`, si X est une liste, alors `np.sin(X)` donne directement la liste `[np.sin(x) for x in X]`. On peut aussi vectorialiser toutes les fonctions que l'on crée avec la syntaxe `fvec = np.vectorize(f)` et ainsi remplacer la ligne `Y=[f(x) for x in X]` par `Y=fvec(X)`.

À vous de représenter quelques fonctions bien choisies sur des intervalles pertinents, après interaction visuelle avec les retours du logiciel, en utilisant de manière pertinente et efficace les fonctions et les méthodes du module `pyplot`.

On se contente ici de cette introduction et du cas le plus simple. On peut aussi, représenter, pour ceux qui savent ce que c'est, des arcs paramétrés, des courbes planes définies par une équation cartésienne, des courbes et des surfaces, en perspective, dans l'espace à trois dimensions, des lignes de niveaux, des figures géométriques de toutes sortes. Tout ceci est désormais en dehors du programme de mathématiques.

1.2 Représentation d'histogrammes

Il y a aussi dans `pyplot` beaucoup de fonctions, qui s'utilisent pour nous comme des boites noires, qui permettent de donner des représentations graphiques de données statistiques. On donne ici la plus célèbre et si on en veut plus, si on a besoin de fonctions, de représentations plus sophistiquées, il suffit d'aller voir la documentation de la bibliothèque et de ses modules. Il y a aussi beaucoup de tutoriels accessibles avec internet.

On se donne une liste `data` de valeurs numériques (ou autres, ici on prendra des entiers) et on veut afficher un histogramme. On utilise ici le module `random` pour créer une série statistique « au hasard » et on utilise la fonction `hist` du module `pyplot`.

```
from random import randint
data = [randint(0,20) for i in range(51)]
print(data)
plt.figure()
plt.hist(data, range = (0, 20), bins = 20, color = 'red', edgecolor = 'black')
plt.title('notes du DS 1')
plt.xlabel('notes')
plt.ylabel('effectifs')
plt.show()
```

On peut aussi faire des regroupements par intervallles.

```
plt.figure()
plt.hist(data, range = (0, 20), bins = 5)
plt.title('histogramme des notes du DS 1')
plt.xlabel('notes')
plt.ylabel('effectifs')
plt.show()
```

Les intervalles ne sont pas nécessairements de même longueur.

```
plt.figure()
plt.hist(data, range = (0, 20), bins = [0,8,12,20])
plt.title('histogramme des notes du DS 1')
plt.xlabel('notes')
plt.ylabel('effectifs')
plt.show()
```

Avec `help(plt.hist)` dans la console on a la spécification de la fonction avec toute la liste des paramètres possibles, dont la plupart sont dotés d'une valeur par défaut. Plus simplement, on peut aussi aller sur une des nombreuses pages qui expliquent, en français si on veut, comment on peut utiliser tout cela.

2 Création, enrichissement et consultation d'une liste des nombres premiers

On veut créer, en vue de consultation, un fichier qui contient les premiers nombres premiers dans l'ordre croissant. On en mettra un par ligne. On commence par la création du fichier dans lequel on va rentrer soi-même les tout premiers. Le fichier est créé dans le dossier courant, que l'on peut connaitre avec la commande `os.getcwd`. S'il ne convient pas on peut en changer, voir le cours.

```
#Création du fichier et premier remplissage
fich = open('listepremiers.txt','w') # création du fichier pour écriture
fich.write('2\n') # le \n marque le passage à la ligne
fich.write('3\n')
fich.write('5\n')
fich.write('7\n')
fich.close()
```

On a ainsi les quatre premiers nombres premiers et si on veut voir le contenu de ce fichier on entre

```
fich = open('listepremiers.txt','r')
print(fich.read())
fich.close()
```

Pour rajouter des nombres premiers on utilise l'algorithme du crible d'Ératosthène : un nombre est premier si et seulement si il n'admet aucun diviseur premier inférieur ou égal à sa racine carrée.

On rappelle que 1 n'est pas premier ; le premier nombre premier est 2 et c'est le seul nombre premier pair. Ainsi pour rajouter p nombres premiers on ouvre `liste_premiers`, on extrait la liste des nombres premiers déjà connus, on prend successivement les impairs strictement supérieurs au dernier nombre premier connu et on teste leur primalité avec la propriété donnée ci-dessus et en cas de succès on rajoute le nouveau nombre premier à la liste en l'écrivant dans une nouvelle ligne à la fin du fichier. On met tout cela dans une fonction `ajout` de paramètre `p`.

Au choix, on peut essayer d'écrire tout cela tout seul, ou recopier le script suivant, en prenant le temps de le comprendre, voire en le modifiant si c'est pertinent. On rajoute la mesure du temps de l'ajout à l'aide de la fonction `time` du module `time`.

```
from time import time
def ajout(p):
    t0 = time()
    fich = open('listepremiers.txt','r')
    L = fich.read()
    fich.close()
    L1 = L.split('\n')
    L2 = L1[0:-1]
    l = len(L2)
    P = [int(L2[i]) for i in range(l)]
    fich = open('listepremiers.txt','a')
    n = P[-1]
    while len(P) < l + p:
        n += 2
        prem = True
        k = 0
```

```
        while P[k]**2 <= n and prem == True:
            if n % P[k] == 0:
                prem = False
            k +=1
        if prem == True:
            P.append(n)
            ch = str(n)+'\n'
            fich.write(ch)
    fich.close()
    return time()-t0

def afficheliste():
```

Si on veut voir la nouvelle liste, on peut créer une fonction qui l'affiche.

```
    L=fich.readlines()
    fich.close()
    P=[int(x.rstrip('\n')) for x in L]
    print (len(P),P)
```

consultation et utilisation

Avec `afficheliste()` (ne pas oublier les parenthèses) on a la liste des premiers nombres premiers connus.

Il est temps de s'amuser et d'enrichir notre fichier, en essayant d'aller raisonnablement loin. On peut en rajouter 500, 1000, etc... et on pourra constater que suivant le nombre N de nombres premiers déjà calculés le temps d'éxécution de `ajout(p)` augmente avec N, ce qui peut interroger le mathématicien, la mathématicienne, et aussi l'informaticien, l'informaticienne mais pas forcément pour les mêmes raisons.

On peut maintenant interroger notre fichier pour savoir si un nombre est premier ou non (s'il est dans la liste), pour connaître, dans le cas où il serait premier ,son rang dans la liste des premiers et inversement pour un rang p donné, retourner le p-ième nombre premier. L'intérêt étant que les calculs sont faits une fois pour toutes et donc l'accès à l'information est plus rapide.

Écrire une fonction `niemepremier(n)` qui retourne le n-ième nombre premier. Il faudra prévoir le cas où n est trop grand pour la liste calculée soit, au choix, en répondant que le nombre est trop grand pour l'état actuel des données, soit en rajoutant les nombres premiers nécessaires (les deux versions sont intéressantes).

Écrire une fonction `rangpremier(n)` qui pour un entier n précise s'il est premier ou non (il faudra là encore gérer le dépassement comme ci-dessus) et, dans le cas où il est premier, donne son rang dans la liste des premiers déjà calculés.

Ceci étant fait, on s'intéresse en mathématiques à la fréquence des nombres premiers, c'est-à-dire si on note $\pi(n)$ le nombre de nombres premiers inférieurs ou égaux à n, on s'intéresse au quotient $\frac{\pi(n)}{n}$. En utilisant la liste obtenue, on peut définir une fonction Python `pipremier` et calculer `pipremier(n)` et considérer numériquement ou graphiquement la suite $\left(\frac{\pi(n)\ \ln(n)}{n}\right)_{n\geqslant 2}$ et on voit des choses que l'on interprète. On sait qu'il y a de plus en plus de nombres premiers (l'ensemble des nombres premiers est infini), mais on voit qu'ils sont de plus en plus rares et on peut quantifier cette rareté progressive, c'est le célèbre théorème des nombres premiers (voir par exemple Wikipedia) que l'on découvre numériquement, visuellement, ici.