Des outils pour étudier les suites : documentation

Dans le cadre de cet atelier, un certain nombre d'outils vous sont déjà donnés. Voici la documentation associée. Ces outils dépendent de deux modules, `numpy` et `matplotlib.pyplot`[1]. Les programmes Python qui vous sont donnés commencent donc tous par la commande

```
import numpy, matplotlib.pyplot
```

La fonction f_μ est implémentée ainsi :

```
def fonction (x) :
   return 3,56995*x*(1-x)
```

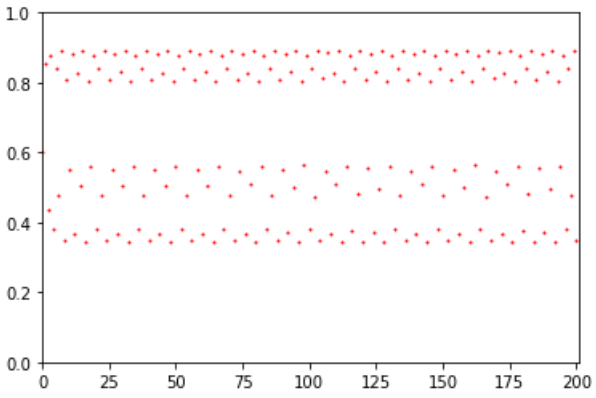
qui prend en entrée un réel (`float`) et ressort un réel (`float`). Dans cet exemple, le paramètre μ vaut $3,56995$; dans le cadre de cet atelier, il faudra modifier ce paramètre directement dans la définition de `fonction`.

1 Graphe de la suite

La fonction `suite_graphe(ListeValeurs, Y_Min, Y_Max)` prend en entrée :

- Une liste de réels `ListeValeurs` ;
- Deux réels `Y_Min` et `Y_Max` ;

et affiche l'ensemble des points (`n`, `ListeValeurs[n]`). Les paramètres `Y_Min` et `Y_Max` sont les bornes inférieures et supérieures des ordonnées de la fenêtre d'affichage.



Sortie de
`suite_graphe(ListeValeurs, 0, 1)`
où `ListeValeurs` consiste en 200 itérations de $f_{3,56995}$.

```
def suite_graphe (ListeValeurs, Y_Min, Y_Max) :
   Range = [x for x in range(len(ListeValeurs))]
   matplotlib.pyplot.clf()
   matplotlib.pyplot.plot(Range, ListeValeurs,
       'r.', markersize=2)
   matplotlib.pyplot.axis([0, len(ListeValeurs),
       Y_Min, Y_Max])
   matplotlib.pyplot.show()
```

Le code de la fonction `suite_graphe`.

2 Diagramme en toile d'araignée

Pour afficher les diagrammes en toile d'araignée, nous avons prévu deux fonctions.

La fonction `graphe_fonction(Fonction, X_Min, X_Max)` prend en entrée :

- Une fonction `Fonction` ;
- Deux réels `X_Min` et `X_Max` ;

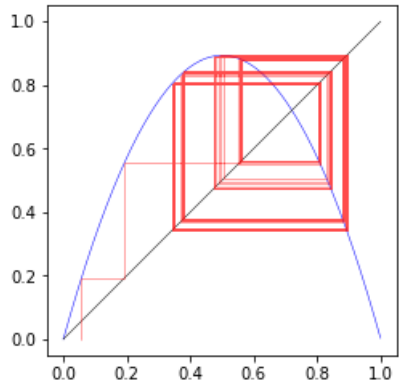
et crée le graphe de la fonction `Fonction` entre les abscisses `X_Min` et `X_Max`. Le graphe est construit en bleu avec 1000 valeurs équiréparties entre `X_Min` et `X_Max`. À cela on ajoute en noir la droite d'équation $y = x$. **Cette fonction n'affiche pas le graphe ainsi construit.**

La fonction `suite_araignee(CourbePolygonale)` prend en entrée :

[1] Dans le cadre de cet atelier, nous n'utiliserons pas les abréviations habituelles `np` pour `numpy` ni `plt` pour `matplotlib.pyplot`. Si vous le faites, il faudra modifier les fonctions présentées en conséquence.

- Une liste de paires de points $[(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})]$

et ajoute en rouge au graphe la ligne brisée reliant les sommets $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$. Elle affiche le graphe ainsi obtenu. Pour obtenir le graphe en toile d'araignée, il faut lancer la fonction `graphe_fonction` puis la fonction `suite_araignee`.



Sortie de
`graphe_fonction(fonction,0,1)`
`suite_araignee(Toile)`
où `Toile` consiste en 50 itérations de $f_{3,56995}$.

```
def graphe_fonction (Fonction, X_Min, X_Max) :
    Resolution = 1000
    Step = (X_Max-X_Min) / Resolution
    Range = numpy.arange(X_Min, X_Max, Step)
    Image = fonction(Range)
    matplotlib.pyplot.clf()
    matplotlib.pyplot.gca().set_aspect('equal',
        adjustable='box')
    matplotlib.pyplot.plot(Range, Image, 'b-',
        linewidth=0.5)
    matplotlib.pyplot.plot(Range, Range, 'k-',
        linewidth=0.5)

def suite_araignee (CourbePolygonale) :
    matplotlib.pyplot.plot([x[0] for x in
        CourbePolygonale], [x[1] for x in
        CourbePolygonale], 'r-', alpha=0.7,
        linewidth=0.5)
    matplotlib.pyplot.show()
```

Le code des fonctions `graphe_fonction` et `suite_graphe`.

3 Affichage des valeurs de la suite

3.1 La fonction `affichage_valeurs`

La fonction `affichage_valeurs(ListeValeurs, Nom)` prend en entrée :

- Une liste de réels (de précision arbitraire) `ListeValeurs` ;
- Une chaîne de caractères `Nom` ;

et imprime les réels de la liste `ListeValeurs` les uns au-dessus des autres dans un fichier `Nom.txt` situé dans le même répertoire que le programme.
Il est souvent nécessaire de spécifier l'adresse complète du fichier texte, par exemple : `Nom = 'C:/UserFiles/MaListe'`.

```
0.057136145995794307417270374571671709418296813964843750
0.192318942754712630992578885718652295744769542715164324813928862114474382891921300983130
0.554528783615996615483009961854446573056743245633754050968695836325384356163417636157620
0.881872652643186675388233061843541282784942999464570889261903700516165772040042589788360
0.371893390808966083487709971216750686331678578359657218116639001254889323879738106422910
0.833899967718391176036610310175198946969026434403614107821905737052156252957095813638150
0.494476671720258781159509702425370519552290973891366427027723456027429443881791864841500
0.892378590980987486397249994884888594977773828903152141837235479323827122308943880443725
0.342854575630929491473341053236226023628306221720742345508982601668692262217742562056650
0.804328711425737121536262224250700499645309594002530689026401765940928870364804426900930
0.561853137183192894967473707410468476124999842372298733914779185041268949915952476932970
0.878829547522066079683085257184701167472820093237849418796187144474857864000966266888260
0.380157456500807629696313576111876802107684122096722870515542636133416712368649211262440
0.841215038332251346399665836557282030247548235618521527931570015924795792542048447595240
0.476846423873954470402834330674759417773737729434821510243605308478648785254568138751860
0.890573692332298590132240484054092055385497767276682299126431247438184780942729105299234
0.347899448753213512586220741714769227372609307081599766048450947816628475616959884831260
0.809898214377096997690615165241828542208989369276698984463608219311986305738392857387985
0.549640557156579688081457324389245065053826733096519090261685160326968925260933730020870
0.883690483063353842107123068113227551025192671651208419849624067614369243587509528136660
0.366925220066938002067041806077236865326573815832744373966225473012719966150490019059160
0.829267622961241319236593659066598566901828734437620032454880356612717342782060579301750
0.505443632774326688451638683626914291452917684056115786761849855708884847578094702386550
0.892381711179776124717659615498506029409745987188775781128243889869045048910678560300270
0.342845834222285199440922467761964435071673417568459618928259148747024398965552728364630
0.804318903269587661856072028806052534371893289981609870954423056182728303545111842133440
0.561874448752423745380179000599728127949569587279850011406015555281200977803064346060730
0.878820134174324174945167374198186419511161958985472517627390038359644834399494588483760
0.380182917455307682795329155113904360716752164983930707998951820358702706392159187183760
0.841236822034471309715984869278693226099537131813377181362821476930155468634752596367870
0.476793351825933567130686426519909592976635617781720214831133677843856881471120946359630
0.890564908712980566709876451415127730975163108340006518685707528360440984904666729119320
0.347923942980434511352739132352352667040962777982456405688039549281361090213133586707240
0.809924812542889183643323497194427442673739578287607371670806254981392155687341056622660
0.549581702405281828258044512052884651424741005661723069563352810966557910878423581136160
0.883711330505401490061085215298573726367709433743150999067556108018738047421722113231940
0.366868106704393028448547362691453119255630344298159072854371141038682056512905142463280
0.829213455905224301890827475337163563434072333880118032839126906847589607573267011490100
```

Le fichier `03_AffichageValeurs.txt` créé par la commande `affichage_valeurs (ListeValeurs, '03_AffichageValeurs')`, où `ListeValeurs` consiste en 200 itérations de $f_{3,56995}$ avec une précision de 100 décimales.

```
def affichage\_valeurs (ListeValeurs, Nom) :
    TextFile = open(str(Nom)+".txt", "w")
    for x in ListeValeurs:
        TextFile.write(str(x)+'\n')
    TextFile.close()
```

Le code de la fonction `affichage_valeurs`.

Les sous-sections 3.2 et 3.3 résument les principales informations sur les réels en précision arbitraire (de type `decimal`) et l'écriture dans un fichier texte.

3.2 Variables de type `Decimal`

Afin de travailler en précision arbitraire, on utilisera le module `decimal`, qu'il faut importer au début de votre programme :

```
from decimal import *
```

Ensuite, fixez la précision, c'est-à-dire le nombre de décimales des variables de type `decimal`. Pour commencer, on utilisera une centaine de décimales :

```
getcontext().prec = 100
```

Quand vous voudrez utiliser des variables de type `decimal`, il faudra les déclarer explicitement dans le programme. Par exemple :

```
x = Decimal(3)
```

déclare x comme une variable de type `decimal` égale à 3 (autrement dit, $x = 3.0\ldots0$, avec 100 zéros). Il est important de **ne pas additioner ou multiplier entre elles des variables de précisions différentes (par exemple, une variable de type `decimal` avec une variable de type `float`).** Par conséquent, la fonction f_μ est implémentée différemment :

```
def fonction (x) :
   return Decimal(3,56995)*x*(1-x)
```

et prend en entrée un réel (`decimal`) et ressort un réel (`decimal`).

3.3 Écrire dans un fichier texte

Pour créer un fichier texte et écrire dedans, il faut, en Python :

- ouvrir ce fichier, avec les droits d'écriture ;
- écrire dans ce fichier ;
- fermer ce fichier.

Par exemple, pour écrire "Hello world!" dans `NomDeFichier.txt`, on entre le code suivant :

```
TextFile = open("NomDeFichier.txt", "w")
TextFile.write('Hello world!')
TextFile.close()
```

La commande `TextFile.write()` prend en argument une chaîne de caractères. Pour écrire dans le fichier une variable numérique x (de type `float` ou `decimal`), il faut donc écrire :

```
TextFile.write(str(x))
```

Le saut de ligne correspond à la chaîne de caractères `\n`.

4 Algorithme de convergence

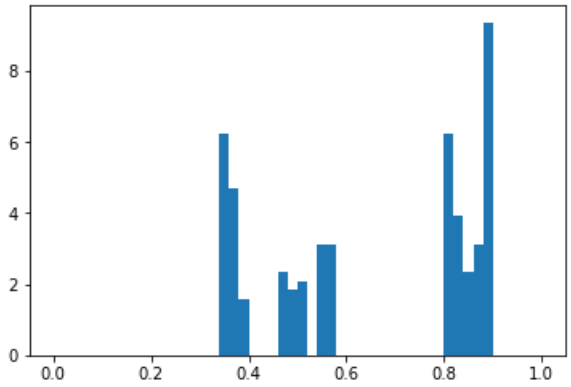
À vous de l'écrire !

5 Histogramme des valeurs de la suite

La fonction `suite_histogramme(ListeValeurs, X_Min, X_Max, Classes)` prend en entrée :

- Une liste de réels `ListeValeurs` ;
- Deux réels `X_Min` et `X_Max` ;
- Un entier strictement positif `Classes` ;

et affiche l'histogramme de la liste `ListeValeurs[n]`. Les paramètres `X_Min` et `Y_Max` sont les bornes inférieures et supérieures des abscisses de la fenêtre d'affichage. Le paramètre `Classes` est le nombre de classes de l'histogramme. L'ordonnée de l'histogramme est la densité, c'est-à-dire le rapport du nombre d'éléments dans la classe par la largeur de la classe.



Sortie de
`suite_histogramme(ListeValeurs, 0, 1, 50)`
où `ListeValeurs` consiste en 10000 itérations de $f_{3,56995}$.

```
def suite_histogramme (ListeValeurs, X_Min,
    X_Max, Classes):
   matplotlib.pyplot.clf()
   matplotlib.pyplot.hist(ListeValeurs,
       bins=Classes, range=[X_Min, X_Max],
       normed=True)
   matplotlib.pyplot.show()
```

Le code de la fonction `suite_histogramme`.