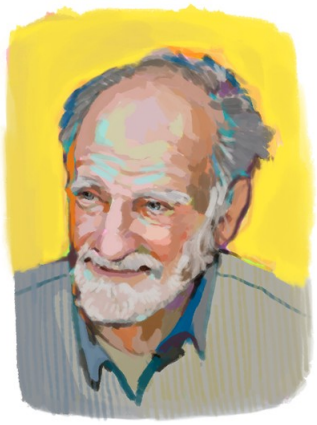
TP N°1 :

Problème 1 : Algorithme de Mariages Stables de Gale et Shapley

Supposons la situation suivante : on dispose d'un groupe d'individus, w femmes et h hommes (hétérosexuels) qu'on souhaite marier. Pour cela, on dispose pour chaque femme de son classement des hommes préférés et pour chaque homme, de son classement des femmes préférées. On souhaiterait que les mariages soient stables, c'est-à-dire que dans deux couples formés différents (h_1, w_1) et (h_2, w_2), on ne se retrouve pas dans la situation où h_1 préfèrerait la femme w_2 à sa femme w_1 et que w_2 préfèrerait l'homme h_1 à son marie h_2 (sinon il y a risque d'adultère).

David Gale (1921-2008)
PROFESSOR, UC BERKELEY



Lloyd Shapley
PROFESSOR EMERITUS, UCLA

L'algorithme de *David Gale* et *Llyod Shapley* permet de répondre à cette question lorsque le nombre de femmes w et le nombre d'hommes h sont identiques. De plus, ils ont démontré que leur algorithme donnait la meilleure partenaire possible pour les hommes permettant d'obtenir des mariages stables. Le but de ce problème est de comprendre et de programmer cet algorithme sur Python.

Exemple simple

Supposons qu'on a un groupe de 3 filles : Lucie, Camille et Marwa et un groupe de trois garçons : Alexis, Rémi et Nicolas. De plus on possède le classement (du préféré au moins aimé) de chacun d'entre eux :

Pour les garçons :

- **Alexis** : Camille, Lucie, Marwa
- **Rémi** : Camille, Marwa, Lucie
- **Nicolas** : Marwa, Camille, Lucie

Pour les filles :

- **Lucie** : Nicolas, Alexis, Rémi
- **Camille** : Alexis, Rémi, Nicolas
- **Marwa** : Alexis, Rémi, Nicolas

- Au premier tour : tous les garçons proposent à leur premier choix, les filles acceptent leurs préférés parmi les propositions qu'elles ont reçues. Ici, Lucie ne reçoit pas de proposition, elle reste seule à la fin du premier tour. Camille a deux propositions : Rémi et Alexis, comme elle préfère Alexis, elle l'accepte et refuse Rémi. Marwa accepte sa seule proposition : Nicolas. Les couples formés sont alors : (Alexis, Camille) et (Nicolas, Marwa). Cependant ces couples sont provisoires : aux tours suivants, les filles peuvent accepter la proposition d'un homme qu'elle préfère à celui qui est en couple avec elle.
- Au second tour : tous les garçons "non casés" proposent à leur choix suivant dans leur classement respectif. Ici seul Rémi n'est pas en couple chez les garçons : il propose à Marwa. Marwa préfère Rémi a Nicolas, elle le jette et récupère Rémi. A la fin du tour, les couples sont : (Alexis, Camille) et (Rémi, Marwa).

- Au troisième tour : les garçons "non casés" proposent à leur choix suivant dans la liste (qui n'est pas nécessairement leur 3e choix s'ils n'ont pas participé à tous les tours précédents). Ici, Nicolas propose à Camille, mais Camille préfère Alexis (avec qui elle est en couple) que Nicolas, elle reste avec lui. Nicolas reste seul et les couples n'ont pas changé.
- Au quatrième tour : les garçons "non casés" proposent à leur choix suivant dans la liste. Ici, Nicolas propose à Lucie qui l'accepte.
- L'algorithme s'arrête lorsque tous les garçons ont trouvé une partenaire.

1 Expliquer brièvement pourquoi l'algorithme se termine en renvoyant n couples (Garçon,Fille) lorsque le nombre de garçons et de filles sont égaux et vaut n. (On pourra raisonner par l'absurde)

2 (a) Pourquoi avec un tel algorithme, un couple (Garçon,Fille) dont les deux sont en tête du classement de l'autre termine marié (ce qui est le cas de (Alexis,Camille) dans l'exemple) ?

(b) Est-ce toujours le cas si, au lieu de s'arrêter, les garçons continuent de proposer aux filles lorsqu'ils sont en couple ?

3 Démontrer brièvement que l'algorithme se termine en au plus n^2 tours lorsque le nombre de garçons est égal au nombre de filles et vaut n.

Programmation de l'algorithme sur Python

On se propose de programmer l'algorithme avec l'aide de python, pour cela, on dispose des variables suivantes :

- `NomG` qui est la liste des Noms des Garçons (en chaine de caractères) ;
- `NomF` qui est la liste des Noms des Filles (en chaine de caractères) ;
- `choixG` qui est une liste contenant des listes : pour `k` indice de la liste `choixG`, `choixG[k]` est une liste contenant le classement du garçon `NomG[k]`. Le premier choix de `NomG[k]` est donc `choixG[k][0]`, puis `choixG[k][1]`, etc...
- `choixF` est la même liste que `choixG` mais pour les filles.

Voici ce que cela donne pour notre exemple :

```
NomG=["Alexis","Rémi","Nicolas"]
NomF=["Lucie","Camille","Marwa"]
choixG=[["Camille","Lucie","Marwa"],["Camille","Marwa","Lucie"],["Marwa","Camille","Lucie"]]
choixF=[["Nicolas","Alexis","Rémi"],["Alexis","Rémi","Nicolas"],["Alexis","Rémi","Nicolas"]]
```

Attention : A partir d'ici, le nombre de garçons n'est pas nécessairement égal au nombre de filles. De plus, pour plus de simplicité, on supposera qu'il n'y a pas d'homonymes (chaque fille et garçon ont des prénoms différents, comme dans l'exemple).

4 (a) Que renvoie l'instruction `choixF[1][2]` si le script précédent a été exécuté ?

(b) Même question pour `choixG[2]` ?

5 On veut créer une liste `ind` tel pour `k` indice de `ind`, `ind[k]` sera l'indice du prochain choix proposé par le garçon `NomG[k]` au cours de l'algorithme dans son classement `choixG[k]`. Par exemple, si Rémi a déjà proposé Camille mais pas Marwa, alors `ind[1]=1`. Au départ, cette liste contient que des zéros et a la même taille que le nombre de garçon. Donner une instruction python permettant de créer cette liste.

6 On veut créer une liste `selection` qui, pour `k` indice de la liste, `selection[k]` contient le nom du garçon (chaine de caractères) en couple avec `NomF[k]`. Au départ, la variable est initialisée à une liste contenant que des chaines de caractères vide `' '` et ayant autant d'éléments que le nombre de filles. Donner une instruction permettant d'initialiser cette variable.

Dans notre exemple, on obtient ces deux variables :

```
indice=[0,0,0]
selection=['', '', '']
```

7 Dans l'exemple cité dans la section précédente : "Exemple simple", quel est la valeur des variables `ind` et de `selection` à la fin de l'algorithme ?

8 Écrire une fonction python `verification(L,M)` qui prend en arguments deux listes `L` et `M` et qui renvoie `True` si tous les éléments de la liste `L` sont des éléments également de la liste `M`, sinon `False`. Pour cela, on peut utiliser l'instruction `k in P` qui vaut `True` si la valeur de la variable `k` est dans la liste `P`, `False` sinon.

9 Écrire une fonction python `verification2(LL,M)` qui prend en arguments une liste de listes `LL` et une liste `M` et qui renvoie `True` si toutes les listes contenu dans la liste `LL` ne contiennent que des éléments de la liste `M`, sinon `False`. Pour rappel, toute fonction crée dans les questions précédentes peuvent être utilisée.

10 Donner deux instructions permettant de vérifier que le classement fait par chaque garçon (et respectivement chaque fille) ne possède que des noms de filles de la liste `NomF` (respectivement des noms de garçons de la liste `NomG`).

11 Écrire une fonction python `ordre(a,b,L)` qui prend en arguments `a` et `b`, deux éléments de la liste `L` et la liste `L` et qui renvoie `True` si l'indice de `a` est plus petit que celui de `b` (`a` est placé avant `b` dans la liste) sinon `False`. On pourra s'aider d'une boucle `for` sur la liste `L` et s'aider du fait que la liste est parcourue dans l'ordre des indices croissants dans une boucle for.

12 Comme le nombre de filles et de garçons n'est pas nécessairement le même, on modifie le critère d'arrêt : l'algorithme s'arrête si pour tous les garçons considérés, soit ils ont proposé toutes les filles de leur classement soit ils sont en couple avec une fille. Créer une fonction `arrêt(ind,nb,NomG,selection)` qui prend en argument la liste `ind` qui aura le même rôle que la liste homonyme sus-citée, l'entier `nb` représentant le nombre de fille, la liste des noms des garçons `NomG` et la liste des choix faits par les filles `selection` et qui renvoie `True` si l'algorithme ne doit pas s'arrêter au vu des valeurs des arguments, `False` sinon.

13 Créer une fonction `Recherche_indice(L,a)` qui prend en arguments une liste `L` et un élément de la liste `a` et qui renvoie l'indice de `a` dans `L`. (On suppose que `a` apparait une et une seule fois).

14 Écrire un script permettant de simuler l'algorithme de Gale et Shapley, pour cela on pourra s'aider du raisonnement suivant :

```
Tant que mon critère d'arrêt est respecté :
-- pour chaque garçon de ma liste des garçon :
-- -- si le garçon n'est pas en couple et n'a pas parcouru tout son classement :
-- -- -- je recherche l'indice de la fille qu'il a choisi
-- -- -- je regarde ce qu'elle a pour le moment selectionner :
-- -- -- si elle n'a toujours rien sélectionné :
```

```
-- -- -- -- la fille sélectionne le garçon
-- -- -- sinon :
-- -- -- -- on regarde si dans le classement de la fille, le nouveau garçon
-- -- -- -- est mieux placé:
-- -- -- -- si c'est le cas :
-- -- -- -- --  on remplace le garçon en couple avec la fille par
-- -- -- -- --  celui qu'on est en train de regarder
On affiche à la fin la liste des garçons choisis et la liste de leurs partenaires.
```

Problème 2 : Suite logistique

On appelle suite logistique la suite définie par récurrence comme suit :

$$\begin{cases} x_0 \in]0,1[\\ \forall n \in \mathbb{N},\ \ x_{n+1} = \mu x_n(1-x_n) \end{cases}$$

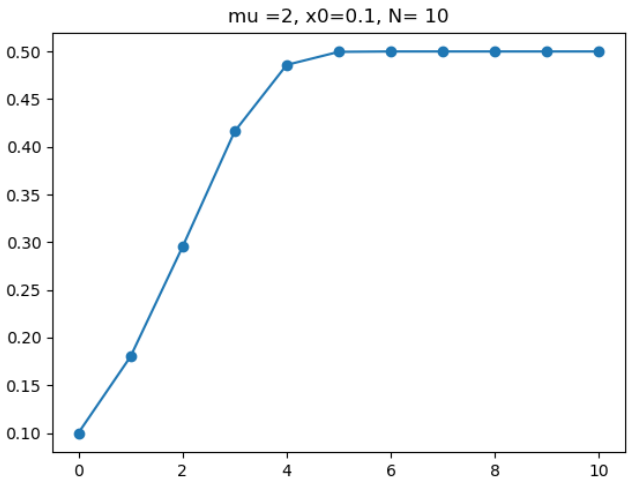
Nous allons tout d'abord étudier la convergence de certaines de ces suites pour des μ bien choisis.

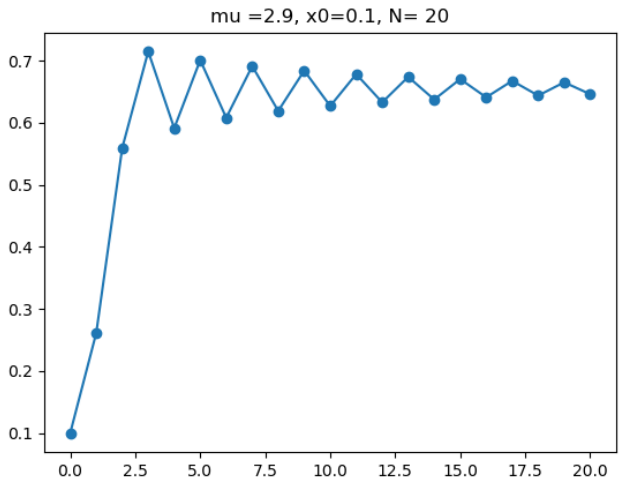
Convergence de cas particuliers

1 Écrire une fonction python `pas(mu,x)` qui prend deux arguments `mu` et `x` et qui renvoie la valeur `mu*x*(1-x)`.

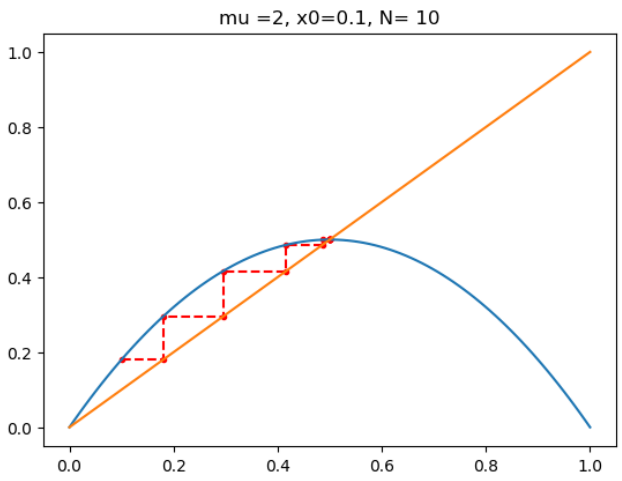
2 Écrire une fonction python `suitelogi(mu,x0,n)` qui prend en arguments un flottant `x0`, un flottant `mu` et un entier `n` et qui renvoie la liste $L := [x_0, x_1, ..., x_n]$ des valeurs de la suite définie précédemment avec (μ, x_0) =(`mu,x0`).

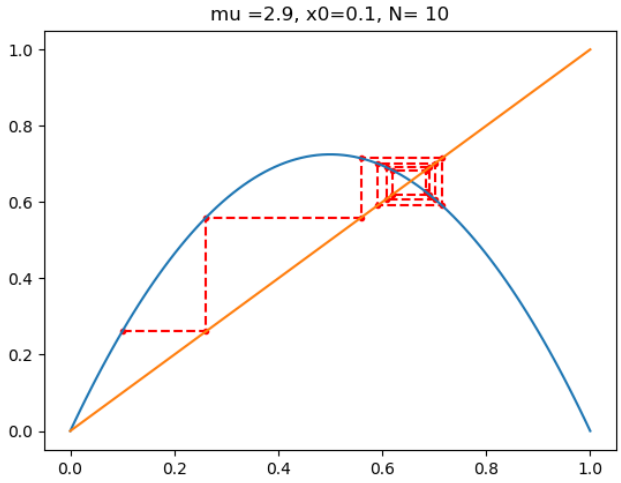
3 Écrire une fonction python `trace(N)` qui prend en argument un entier `N` et qui renvoie une graphe des points (j, x_j) pour $j \in [\![0, n]\!]$ comme sur les graphes ci-dessous :





On remarque des comportement différents selon les valeurs de μ : pour $\mu = 2.9$, les sous-suites d'indices pairs et impairs semblent être adjacentes, alors que pour $\mu = 2$, la suite semble être croissante et convergente vers 0.5. Par la suite, on aimerait utiliser une autre visualisation du comportement asymptotique de la suite. Cela consiste, à partir du graphe de la fonction $x \mapsto \mu x(1-x)$ et de la droite d'équation $y = x$ à tracer les lignes brisées reliant les points de coordonnées (x_0, x_1) puis (x_1, x_1), (x_1, x_2), $x_2, x_2)$, (x_2, x_3), (x_3, x_3),..., (x_{n-1}, x_{n-1}), (x_{n-1}, x_n). Voici ce que cela donne pour les deux paramètres μ précédents :





4 Écrire une fonction python `beg_gauche` qui prend en argument la liste `L` de valeurs $L := [l_0, l_1, ..., l_n]$ et qui renvoie la liste suivante $[l_0, l_1, l_1, l_2, l_2, ..., l_{n-1}, l_{n-1}]$.

5 Écrire une fonction python `beg_droite` qui prend en argument la liste `L` de valeurs $L := [l_0, l_1, ..., l_n]$ et qui renvoie la liste suivante $[l_1, l_1, l_2, l_2, ..., l_{n-1}, l_{n-1}, l_n]$.

6 Écrire une fonction `pointsequidi(a,b,N)` : python qui prend en argument deux flottants `a` et `b` et un entier `N` et qui renvoie N point équidistants dont les bornes sont a et b (on supposera $a < b$ pour plus de simplicité). Par exemple l'instruction `pointsequidi(0,1,11)` renvoie `[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]` (enfin dans un monde parfait sans encodage car python renvoie plutôt : `[0, 0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6, 0.7, 0`

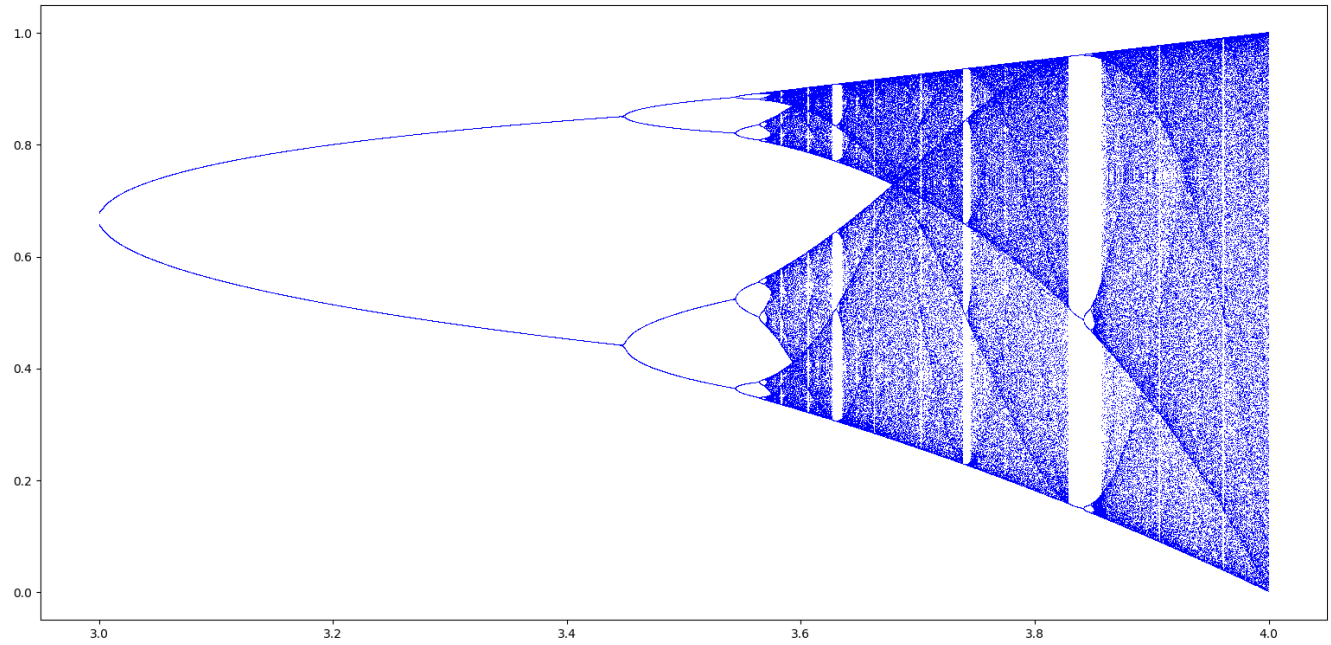
7 Écrire une fonction python `trace2(mu,x0,N)` qui renvoie le graphe ci-dessus pour les valeurs de x_0 jusqu'à x_N pour μ et x_0 donnés en argument. Pour cela on utilisera trois fois l'instruction `plt.plot` :

(a) Une première pour tracer la fonction identité $x \mapsto x$, on utilisera 101 points équidistants entre 0 et 1.

(b) Une seconde fois pour tracer la fonction $x \mapsto \mu x(1-x)$. On réutilisera la liste des abscisses du cas précédent.

(c) Pour tracer les points de coordonnées (x_0, x_1) puis $(x_1, x_1), (x_1, x_2), x_2, x_2), (x_2, x_3), (x_3, x_3)$,..., $(x_{n-1}, x_{n-1}), (x_{n-1}, x_n)$. On utilisera les fonctions des questions précédentes pour cela.

En étudiant cette suite, il est possible de démontrer que lorsque μ est compris entre 3 et $1+\sqrt{6} \approx 3.45$ la suite finit par osciller proche de deux valeurs. En fait, la suite des termes pairs converge vers une valeur différente de celle de la suite des termes impairs. Entre $1+\sqrt{6}$ et environ 3.54, la suite oscille entre 4 valeurs limites de sous-suite. Ce nombre augmente à 8 puis 16, enfin à partir d'environ 3.57, il est très difficile de d'écrire le comportement asymptotique de la suite. On appelle ***valeur d'adhérence*** *toute limite d'une sous suite convergente, le diagramme suivant montre l'ensemble de ces valeurs selon μ.*



Le script suivant permet d'obtenir la figure précédente pour $N = 5000$.

```
1 def diag_bifurcation(N):
2     for mu in pointsequidi(3,4,N):
3         Y=suitelogi(mu,0.5,500)[400:]
4         X=[mu for i in Y]
5         plt.plot(X,Y,',b')
6     plt.show()
```

8 A quoi sert la variable N ?

9 Que fait exactement l'instruction en ligne 3 ? Quelles sont les hypothèses qui ont été faites, selon vous, et qui permettent de dire que ce diagramme est le diagramme des valeurs d'adhérence de la suite logistique selon le paramètre μ.

10 En supposant que `pointsequidi(a,b,N)` et `suitelogi(mu,x0,N)` sont des algorithmes de complexité linéaire selon la variable `N`, donner la complexité de `diag_bifuraction`. Pourquoi utilise-t-on `N` comme instance ? Expliquer brièvement.