Jour 6

TOILES D'ARAIGNÉES

solutions

Question 1

Alors on ne le répétera jamais assez, mais de 0 à $n_{\max}$, il y a $n_{\max}+1$ nombres.

Question 2

On veut donc $n_{\max}+1$ cases dans cette liste. Il y en a une pour 0, une pour 1, ce qui laisse $n_{\max}-1$ cases initialisées à la valeur `None`, c'est bien ce que fait le programme.

Question 3

Commençons par la ligne facile : à l'intérieur de la boucle, on traduit juste la formule de récurrence. Ensuite, réfléchissons à la borne du `range`. Dans un `range`, la différence des deux bornes est égale au nombre d'itérations faites par la boucle. Il y a, on l'a vu, $n_{\max}-1$ cases à remplir (on ne va pas laisser les `None`, on veut les valeurs de la suite !). Si la borne de gauche est zéro, la borne de droite doit donc être égale à ce nombre d'itérations.

```
def SuiteFibonacci(n_max) :
    F = [0, 1] + [None] * (n_max - 1)
    for n in range(0, n_max - 1) :
        F[n + 2] = F[n + 1] + F[n]
    return F
```

Question 4

Bon alors le programme de la version initiale de l'énoncé n'avait précisément pas les opérations dans le bon ordre (donc en particulier, les dessins n'étaient pas les bons). Petite colère noire, j'ai encore écrit n'importe quoi dans un énoncé ; on se lève, on fait le tour de la pièce pour se calmer, et on corrige tout. Et maintenant que l'énoncé est corrigé, on va pouvoir justifier le *bon* programme.

La liste `absc` est celle des abscisses, on doit la construire de sorte à avoir `[`u_0`, `u_0`, `u_1`, `u_1`, `u_2`, `u_2`, ...]`. À la première itération de la boucle, on commence avec la variable u qui vaut $f(u_0)=u_1$. Et donc, il faut bien ajouter deux fois de suite u à la liste `absc`, avant de modifier u en le terme suivant de la suite.

La liste `ords` est celle des ordonnées, on doit la construire de sorte à avoir `[0, `u_1`, `u_1`, `u_2`, `u_2`, `u_3`, ...]`. À la première itération de la boucle, on commence avec la variable u qui vaut u_1 (comme on l'a déjà dit), on l'ajoute donc une fois, puis l'instruction `u = f(u)` modifie u en le terme suivant de la suite, et on peut ajouter cette nouvelle valeur à la liste. De la même manière, on voit qu'au cours de l'itération i de la boucle, on ajoute u_i et u_{i+1} dans la liste `ords`, c'est ce qu'il faut.

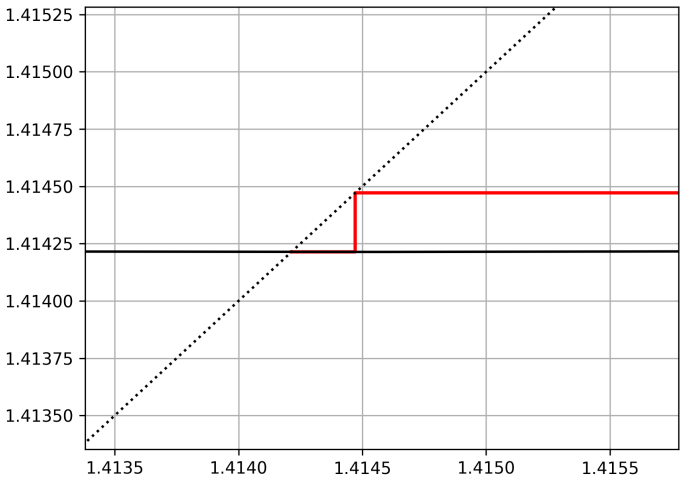
Question 5

Ici $f_1(x)=\dfrac{x}{2}+\dfrac{d}{2x}=\dfrac{x}{2}+\dfrac{2}{2x}=\dfrac{x}{2}+\dfrac{1}{x}$. Ce qui s'écrit ainsi en Python.

```
def f1(x) :
    return x / 2 + 1 / x
```

Question 6

Pour mieux voir, il faut zoomer. Et on peut zoomer fort, parce que ça converge très, très, très vite



Apparemment vers un truc environ égal à 1,414…, ce qui n'est pas loin de ressembler à $\sqrt{2}$. Et justement, on a pris $d = 2$, et l'algorithme de Héron, c'est pour calculer les racines carrées…

Question 7

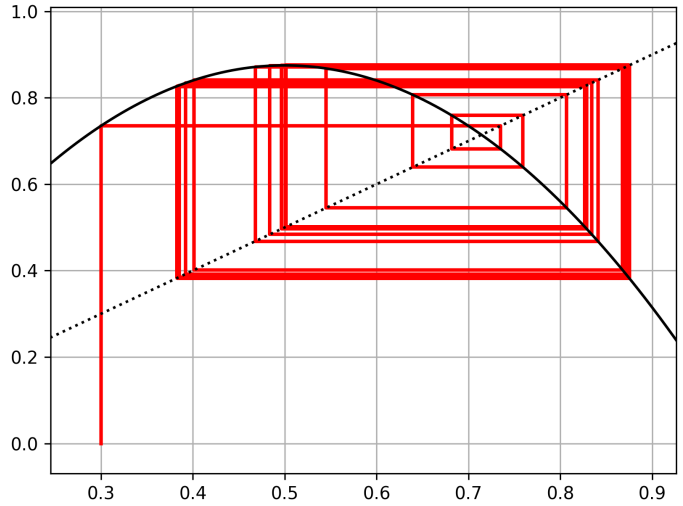
Ici $f_2(x) = \mu \times (x - x^2)$. Ce qui s'écrit ainsi en Python (il faut remplacer μ par la valeur qu'on veut).

```
def f2(x) :
    return 3.5 * (x - x ** 2)
```

Question 8

On peut zoomer un peu, mais là, ce n'est pas ce qui va nous aider. On va plutôt augmenter le nombre de termes : cinquante plutôt que dix.

```
>>> ToileDAraignée(f2, 0.3, 50)
```



Si l'on suit la toile d'araignée (vous avez compris pourquoi cela s'appelle ainsi ?) à partir du début, on fait quelques premiers rectangles hésitants, puis ensuite on continue à tourner (pas en rond, mais en rectangles) en alternant entre quatre abscisses différentes. La suite n'a donc pas de limite, elle oscille entre quatre valeurs, dont on peut démontrer qu'elle s'approche de plus en plus ; on n'appelle pas cela des limites (parce qu'il ne peut pas y avoir plusieurs limites) mais des *valeurs d'adhérence*.