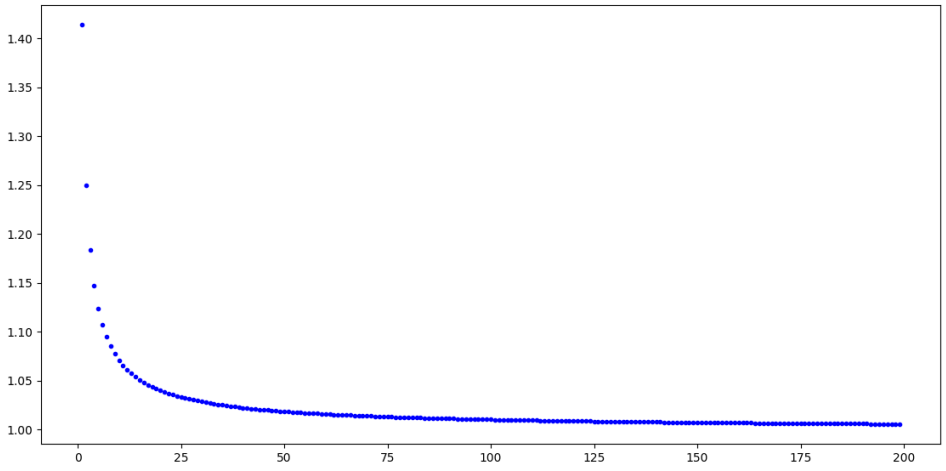
- Exercice 1 -

On considère une suite $(u_n)_{n\in\mathbb{N}}$ définie par son premier terme $u_0 = 1$ et pour tout entier n, $u_{n+1} = u_n + \dfrac{1}{u_n}$.

1. Écrire un programme en Python qui calcule et affiche la valeur de u_n lorsque l'utilisateur entre la valeur de n au clavier.
2. (a) Écrire une fonction en Python d'entête `"def suite_u(n) :"` prenant comme paramètre un entier n et renvoyant la valeur de u_n.
   (b) Quelle commande faut-il écrire pour calculer u_{100} ?
3. Écrire (de deux manières différentes) une fonction en Python prenant comme paramètre un entier n et renvoyant toutes les valeurs de u_0 à u_n rangées dans une liste.
   (a) Une fonction d'entête `"def Liste_U(n)"` utilisant la méthode `"append"`.
   (b) Une fonction d'entête `"def Tableau_U(n)"` utilisant la fonction `"zeros"` du module `"numpy"`.
4. Écrire un programme en Python qui détermine et affiche le plus petit entier naturel n pour lequel $u_n \geqslant 100$. *On obtient* $n = 4998$.
5. Écrire un programme en Python qui calcule et affiche la valeur de la somme $\displaystyle\sum_{n=0}^{100} u_n$.
   (a) En modifiant le programme de la question 1.
   (b) En utilisant la fonction `"Tableau_U"` de la question 3b et la fonction `"sum"` du module `"numpy"`.
6. On considère le programme Python et le graphique associé ci-dessous.
   A l'aide du graphique, conjecturer un équivalent de la suite $(u_n)_{n\in\mathbb{N}}$ en $+\infty$.

```
import matplotlib.pyplot as plt
import numpy as np
from math import sqrt

u = 1
eq = np.zeros(200)
for i in range(1,200) :
    u = u + 1/u
    eq[i] = u/sqrt(2*i)
abs = [k for k in range(1,200)]
plt.plot(abs, eq[1:], 'b.')    # tracé du graphe
plt.show()          # affichage du graphe
```



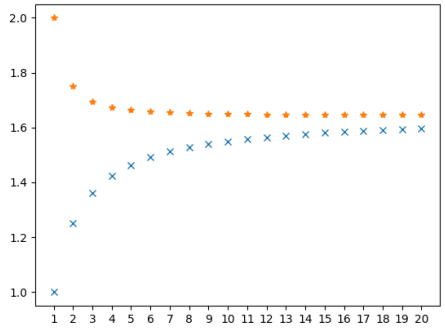
Python

Si T est un tableau `"numpy"`, alors la commande `"np.cumsum(T)"` du module `numpy` renvoie un autre tableau `"numpy"` contenant les sommes cumulées du tableau T.
Par exemple : `"np.cumsum([1,4,6])"` renvoie `"[1,5,11])"`.

- Exercice 2 -

On considère les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ suivantes : $u_n = \displaystyle\sum_{k=1}^{n} \frac{1}{k^2}$ et $v_n = u_n + \dfrac{1}{n}$.

1. On rappelle que la commande `"np.arange(a,b)"` renvoie un tableau `numpy` contenant les entiers de l'intervalle $[\![a; b-1]\!]$ et que les opérations algébriques sur les tableaux `"numpy"` sont appliquées à chaque coefficient.
   Écrire une fonction Python d'entête `"def U(n) :"` qui renvoie un tableau `"numpy"` contenant les n premiers termes de la suite $(u_n)_{n\in\mathbb{N}^*}$.
2. Écrire une fonction Python d'entête `"def V(n) :"` qui renvoie un tableau `"numpy"` contenant les n premiers termes de la suite $(v_n)_{n\in\mathbb{N}^*}$.
3. Écrire un programme Python qui affiche les 20 premiers termes des suites u et v sur un même graphique.
4. Le programme de la question précédente renvoie le graphique suivant.



   (a) Que peut-on conjecturer sur les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$?
   (b) En admettant que la conjecture est vraie, écrire un programme Python qui, pour une valeur donnée de $\epsilon > 0$, renvoie une approximation de $\displaystyle\sum_{k=1}^{+\infty} \frac{1}{k^2}$ à ϵ près.

- Exercice 3 -

1. On considère la suite à récurrence linéaire d'ordre 2 (u_n) définie, pour tout $n \in \mathbb{N}$, par :

$$u_0 = 0,\ u_1 = 2,\ \text{et}\ u_{n+2} = 7u_{n+1} - 3u_n.$$

   Écrire une fonction Python d'entête `"def Suite_Rec2_U(n) :"` afin qu'elle renvoie le terme général u_n en fonction de n.
2. On considère les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ suivantes :

$$a_0 = 1,\ b_0 = 2,\quad \text{et}\ \forall n \in \mathbb{N},\quad a_{n+1} = \frac{a_n^2}{a_n + b_n}\quad \text{et}\quad b_{n+1} = \frac{b_n^2}{a_n + b_n}$$

   Écrire une fonction Python d'entête `"def Suites_ab(n) :"` afin qu'elle renvoie les valeurs de a_n et b_n lorsque n est donné en paramètre.

- Exercice 4 -

On considère une suite $(v_n)_{n\in\mathbb{N}^*}$ définie par : $v_1 = 1$ et pour tout entier n non nul, $v_{n+1} = v_n + \dfrac{1}{n^2 v_n}$.

1. Écrire une fonction en Python d'entête `"def Suite_V(n) :"` prenant comme paramètre un entier n et renvoyant la valeur de v_n.
2. On admet que la suite (v_n) converge vers une limite ℓ vérifiant : pour tout $p \geqslant 2$, $0 \leqslant \ell - v_p \leqslant \dfrac{1}{p-1}$.
   Écrire un programme Python qui renvoie une valeur approchée de ℓ à 10^{-4} près.

- EXERCICE 5 (DICHOTOMIE) -

- **Problème et objectif :**
  On considère une fonction f continue et strictement monotone sur un intervalle $[a,\ b]$.
  On suppose que $f(a)f(b) < 0$ de telle sorte que l'équation $f(x) = 0$ admet une unique solution α d'après le théorème de la bijection monotone.
  La méthode de recherche itérative par dichotomie permet d'obtenir une valeur approchée à ϵ près de la solution α de l'équation $f(x) = 0$ sur l'intervalle $[a,\ b]$.
  Les données du problème sont donc : la fonction f, l'intervalle $[a,\ b]$ et la précision ϵ souhaitée.
- **Principe :**
  Le principe de la méthode de recherche par dichotomie est une méthode itérative.
  A chaque itération, on partage en deux l'intervalle de recherche (en son milieu) et on garde pour l'itération suivante l'intervalle (deux fois plus petit) qui contient la solution (l'autre ne la contenant pas par unicité de la solution).
  La largeur de l'intervalle est divisée par 2 à chaque itération.
  On poursuit les itérations jusqu'à ce que cet intervalle soit de largeur inférieur ϵ (degré de précision souhaité pour l'approximation.
  L'une ou l'autre extrémité de l'intervalle final donne une approximation de la solution α à ϵ près.
- **Mise en œuvre :**
  - ⋆ On initialise l'intervalle $I = [a,\ b]$.
  - ⋆ A chaque itération que l'on poursuit tant que la longueur $b-a$ de l'intervalle I est supérieure à la précision ϵ voulue :
    - On calcule le milieu $c = (a+b)/2$ de l'intervalle puis la valeur $f(c)$ de la fonction f au point c.
    - On remplace soit le point a soit le point b par le point c :
      - soit à l'aide du signe de $f(c)$ et du sens de variation de f si celui-ci est connu.
      - soit à l'aide du signe du signe du produit $f(a) \times f(c)$ si le sens de variation de f n'est pas connu.
    - On obtient alors un nouvel intervalle $I = [a,\ b]$ pour la prochaine itération (la valeur de a ou b ayant été modifiée et la longueur de l'intervalle divisée par 2).
  - ⋆ On renvoie la dernière valeur de c calculée qui donnera une approximation de la solution α à ϵ près.
- **Illustration :**

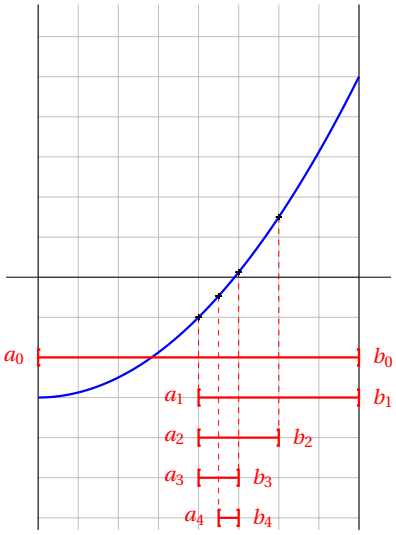
Voici un exemple faisant apparaître les quatre premières itérations dans le cas d'une fonction croissante.

⋆ $c_0 = \dfrac{a_0+b_0}{2}$ et $f(c_0) < 0$ donc on choisit l'intervalle de droite : $a_1 \leftrightarrow c_0$ et $b_1 \leftrightarrow b_0$.

⋆ $c_1 = \dfrac{a_1+b_1}{2}$ et $f(c_1) > 0$ donc on choisit l'intervalle de gauche : $a_2 \leftrightarrow a_1$ et $b_2 \leftrightarrow c_1$.

⋆ $c_2 = \dfrac{a_2+b_2}{2}$ et $f(c_2) > 0$ donc on choisit l'intervalle de gauche : $a_3 \leftrightarrow a_2$ et $b_3 \leftrightarrow c_2$.

⋆ ...



On suppose qu'une fonction `f` définie sur $[0,\ 1]$ est déjà programmée dans Python.

```
def f(x) :
  return (formule de la fonction f)
```

On peut donc accéder à la valeur de `f` en un point `a` par la commande `f(a)`.
Remarque : si f n'est pas programmée dans Python, il faut recopier la formule de f au moment de faire le calcul.

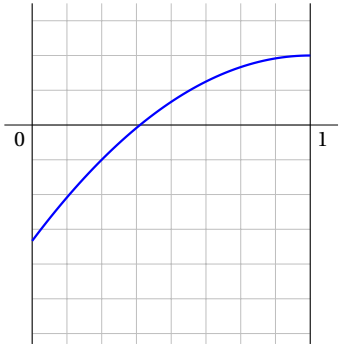
Compléter la fonction suivante pour qu'elle renvoie une valeur approchée à `eps` près de la racine de l'équation $f(x) = 0$ sur l'intervalle $[0,\ 1]$ par la méthode de dichotomie.

1. Dans le cas général en utilisant le signe du produit $f(a) \times f(c)$.

```
def Dichotomie(f,eps) :
    # initialisation
    a = ............
    b = ............
    # tant que l'intervalle
    # est trop grand
    while ..............  :
        c = ..............
        if f(a)*f(c)>0 :
            ............
        else :
            ............
    return(............)
```

2. Dans le cas d'une fonction f strictement croissante sur l'intervalle $[0,\ 1]$.

```
def Dichotomie_Croiss(f,eps) :
    a = ............
    b = ............
    while ..............  :
        c = ..............
        if f(c)>0 :
            ............
        else :
            ............
    return(............)
```



3. Dans le cas d'une fonction f strictement décroissante sur l'intervalle $[0,\ 1]$.

```
def Dichotomie_Decroiss(f,eps) :
    a = ............
    b = ............
    while ..............  :
        c = ..............
        if f(c)>0 :
            ............
        else :
            ............
    return(............)
```