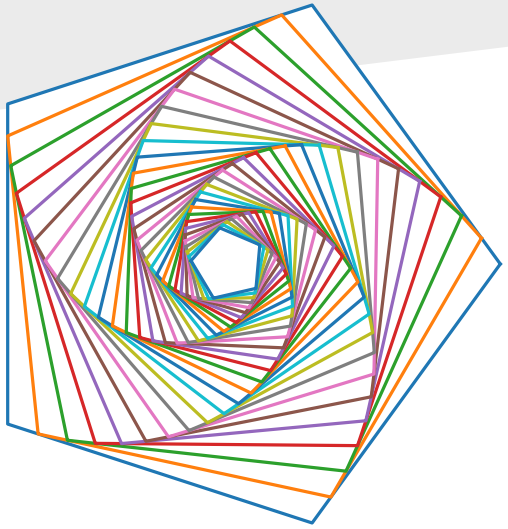
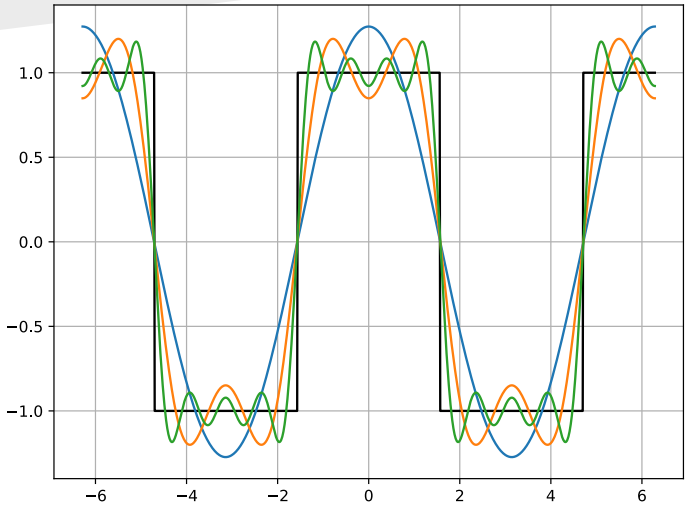
# Python pour le supérieur

## 3) Suites et séries





# Suites et séries

- Suites
  - Définition explicite
  - Récurrence d'ordre 1 et 2
  - Récurrence linéaire d'ordre $p$
- Séries
  - Calcul de sommes partielles
  - Développements limités
- Représentations graphiques
  - Suites et séries
  - Suites récurrentes et courbe de fonctions
- Accélération de convergence

# Suites : définition explicite

**Exemple 1 :** $u_n = 2^n \sin\left(\frac{\pi}{2^n}\right)$

```
1 from math import *
2
3 def u(n):
4     return 2**n * sin(pi/2**n)
5
6 print(u(10))
```

***Exécution du code***

```
3.1415877252771596
```

# Suites : définition explicite

**Exemple 1 :** $u_n = 2^n \sin\left(\frac{\pi}{2^n}\right)$

```
1 from math import *
2
3 def u(n):
4     return 2**n * sin(pi/2**n)
5
6 print([round(u(k), 4) for k in range(11)])
```

***Exécution du code***

```
[0.0, 2.0, 2.8284, 3.0615, 3.1214, 3.1365, 3.1403, 3.1413,
3.1415, 3.1416, 3.1416]
```

# Suites : définition par récurrence (ordre 1)

**Exemple 2 :** $u_0 = a \in \mathbb{R}^+, \quad u_{n+1} = \frac{1}{2}\left(u_n + \frac{a}{u_n}\right)$ (Héron)

**Propriété :** $u_n \longrightarrow \sqrt{a}$ pour $n \to +\infty$

```
1  a = 155
2  u = a
3
4  for _ in range(10):
5    u = (u + a/u) / 2
6
7  print(u)
```

*Exécution du code*

```
12.449899597988733
```

# Suites : définition par récurrence (ordre 1)

**Exemple 3 :** $C_0 = 1, \quad C_{n+1} = \dfrac{4n+2}{n+2} C_n$ (Nombres de Catalan)

```python
def C(n):
    u = 1
    for k in range(n):
        u = u*(4*k+2) // (k+2)
    return u

print(C(9))
```

***Exécution du code***

```
4862
```

# Suites : définition par récurrence (ordre 2)

**Exemple 4 :** $u_0 = 1, \quad u_1 = 5, \quad u_{n+2} = 3\,u_{n+1} - 2\,u_n$

```
1 u, v = 1, 5
2
3 for _ in range(6):
4     print(u)
5     u, v = v, 3*v - 2*u
```

*Exécution du code*

```
1
5
13
29
61
125
```

# Suites : définition par récurrence (ordre 2)

**Exemple 4 :** $u_0 = 1, \quad u_1 = 5, \quad u_{n+2} = 3\,u_{n+1} - 2\,u_n$

```
1 u = 1
2 v = 5
3
4 for _ in range(6):
5     print(u)
6     u = v
7     v = 3*v - 2*u
```



*Exécution du code*

```
1
5
5
5
5
5
```

# Suites récurrentes linéaires d'ordre $p$

**Définition :** On appelle **suite récurrente linéaire d'ordre $p$** toute suite à valeurs dans $\mathbb{K}$ ($= \mathbb{R}$ ou $\mathbb{C}$) définie pour tout $n \geq n_0$ par une relation de récurrence linéaire de la forme :

$$\forall n \geq n_0, \quad u_{n+p} = a_0 u_n + a_1 u_{n+1} + \cdots + a_{p-1} u_{n+p-1}$$

où $a_0, a_1, \ldots a_{p-1}$ sont $p$ scalaires fixés de $\mathbb{K}$, avec $a_0 \neq 0$.

**Étude :** Poser $U_n = \begin{pmatrix} u_n \\ u_{n+1} \\ \cdots \\ u_{n+p-1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \\ a_0 & a_1 & \cdots & \cdots & a_{p-1} \end{pmatrix}$

On a : $U_{n+1} = A\,U_n$ avec $A$, matrice compagnon du polynôme caractéristique de la suite $P(X) = X^p - \sum_{k=0}^{p-1} a_k\, X^k$

# Suites récurrentes linéaires d'ordre $p$

**Exemple 5 :** $u_0 = -3, \quad u_1 = 2, \quad u_2 = 5$

$$u_{n+3} = 3\,u_n - u_{n+1} + 4\,u_{n+2}$$

**Étude :** On pose $U_0 = \begin{pmatrix} -3 \\ 2 \\ 5 \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 3 & -1 & 4 \end{pmatrix}$

On a : $U_{n+1} = A\,U_n$

# Suites récurrentes linéaires d'ordre $p$

**Étude :** $U_0 = \begin{pmatrix} -3 \\ 2 \\ 5 \end{pmatrix}$, $A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 3 & -1 & 4 \end{pmatrix}$ et $U_{n+1} = A\,U_n$

```python
import numpy as np

U = np.mat('[-3 ; 2 ; 5]')
A = np.mat('[0 1 0] ; [0 0 1] ; [3 -1 4]')
L = []    # liste des termes

for _ in range(10):
    L.append(U[0, 0])
    U = A@U

print(L)
```

*Exécution du code*

```
[-3, 2, 5, 9, 37, 154, 606, 2381, 9380, 36957]
```

# Exercices

1) Définir $u_n = \left(1 + \frac{1}{n}\right)^n$ et donner les valeurs de $u_{100}$ et $u_{1000}$
2) Conjecturer la valeur du point fixe de la suite homographique $v_0 = 2,\ v_{n+1} = \frac{4v_n+14}{2v_n+1}$
3) Conjecturer la limite de $w_0 = \sqrt{2},\ w_{n+1} = \sqrt{2 + w_n}$
4) Créer une fonction `F(n)` qui renvoie le n-ième terme de la suite de Fibonacci. Combien de chiffres décimaux sont communs à `F(15)/F(14)` et $\varphi = \frac{\sqrt{5}+1}{2}$ ?
5) a) Afficher les 25 premiers termes de la suite de Padovan, définie par $p_0 = p_1 = p_2 = 1$ et $p_{n+3} = p_{n+1} + p_n$
   b) Vérifier que $p_{24} = 616$ et donner une valeur approchée de $\psi$, le nombre plastique, limite de deux termes consécutifs.

# Séries : définition explicite

**Exemple 6 :** $u_k = \ln k \, \dfrac{e^{\sqrt{k}}}{k!}$. Calculer $\displaystyle\sum_{k=1}^{100} u_k$

Après avoir saisi `from math import *` :

```
1  print(sum([log(k)* e**sqrt(k) / factorial(k) for k in
   range(1, 101)]))
```

**Exemple 7 :** $v_k = \dfrac{(-1)^k}{4k + (-1)^k}$. Calculer $\displaystyle\sum_{k=0}^{1000} v_k$

```
1  print(sum([(-1)**k/(4*k+(-1)**k) for k in range(1001)]))
```

# Séries : indices pairs et impairs

**Exemple 7 :** $v_k = \dfrac{(-1)^k}{4k+(-1)^k}$. Calculer $\displaystyle\sum_{k=0}^{1000} v_k$

$$\sum_{k=0}^{1000} v_k = \sum_{\substack{k=0 \\ k \text{ pair}}}^{1000} \frac{1}{4k+1} - \sum_{\substack{k=1 \\ k \text{ impair}}}^{1000} \frac{1}{4k-1}$$

```python
Sp = sum([1/(4*k+1) for k in range(0, 1001, 2)])
Si = sum([1/(4*k-1) for k in range(1, 1001, 2)])

print(Sp-Si)
```

***Exécution du code***

```
0.7044991080383882
```

# Séries : par récurrence

**Exemple 8 :** $u_0 = 0, \quad u_{k+1} = u_k + 3k^2 + 3k + 1$

Que vaut $S(n) = \sum_{k=0}^{n} u_k$ ?

```python
def S(n):
    s = 0
    u = 0
    for k in range(n+1):
        s += u
        u = u + 3*k**2 + 3*k + 1
    return s

print(S(10))
```

***Exécution du code***

```
3025
```

# Développements limités

**Exemple 9 :** $e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \mathrm{o}(x^n)$

```
1  from math import *
2
3  def exp(x, N=15):
4      return sum([x**k/factorial(k) for k in range(N+1)])
5
6  print(exp(1))
7  print(e)
```

*Exécution du code*

```
2.71828182845 89945
2.71828182845 9045
```

# Développements limités

**Exemple 10 :**

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{k=0}^{\left\lfloor \frac{n-1}{2} \right\rfloor} (-1)^k \frac{x^{2k+1}}{(2k+1)!} + \mathrm{o}(x^n)$$

```
1 from math import *
2
3 def sinus(x, N=15):
4     return sum([(-1)**k*x**(2*k+1)/factorial(2*k+1) for k
  in range(floor((N-1)/2))])
5
6 print(sinus(pi/4))
7 print(sqrt(2)/2)
```

***Exécution du code***

```
0.7071067811865 679
0.7071067811865 476
```

# Exercices

1) Soit $u_k = \dfrac{\sqrt{\ln k} + k^2}{3^k}$. Calculer $\displaystyle\sum_{k=1}^{200} u_k$

2) Soit $v_k = \dfrac{3k^2 + (-2)^k}{2k^3 + (-3)^k}$. Calculer $\displaystyle\sum_{k=0}^{100} v_k$

3) Soit $w_0 = 4,\ w_{n+1} = 2\,w_n - 3$. Calculer $\displaystyle\sum_{k=0}^{15} w_k$

4) Comparer les valeurs de $f(x) = \dfrac{1}{1+x}$ avec celles de son DL en 0 à l'ordre 10 pour $x = -0{,}2\ ;\ 0{,}1\ ;\ 0{,}5$ et $1{,}5$.

5) En combinatoire, un « **dérangement partiel d'ordre $k$** » est une permutation de $\{1, 2, \ldots, n\}$ ayant $n - k$ points fixes.

   Le nombre de dérangements partiels est $d_{n,k} = \dfrac{n!}{k!} \displaystyle\sum_{i=0}^{n-k} \frac{(-1)^i}{i!}$.

   Créer une fonction `d(n, k)` et vérifier que $d_{7,3} = 315$.

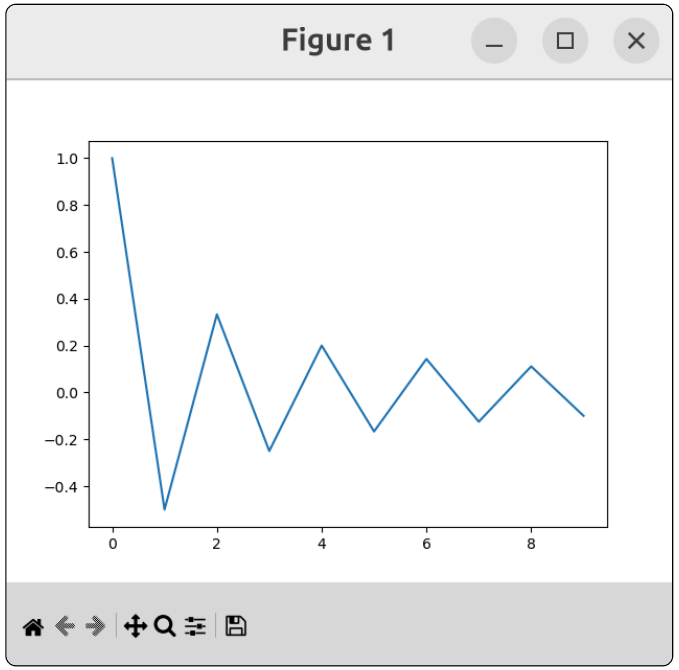
# Représentations graphiques

**Exemple 11 :** $u_n = \dfrac{(-1)^n}{n+1}$ pour $n \in [\![0\,;10]\!]$

```
1  import matplotlib.pyplot as plt
2
3  Y = [(-1)**n/(n+1) for n in range(11)]
4
5  plt.plot(Y)
6  plt.show()
```

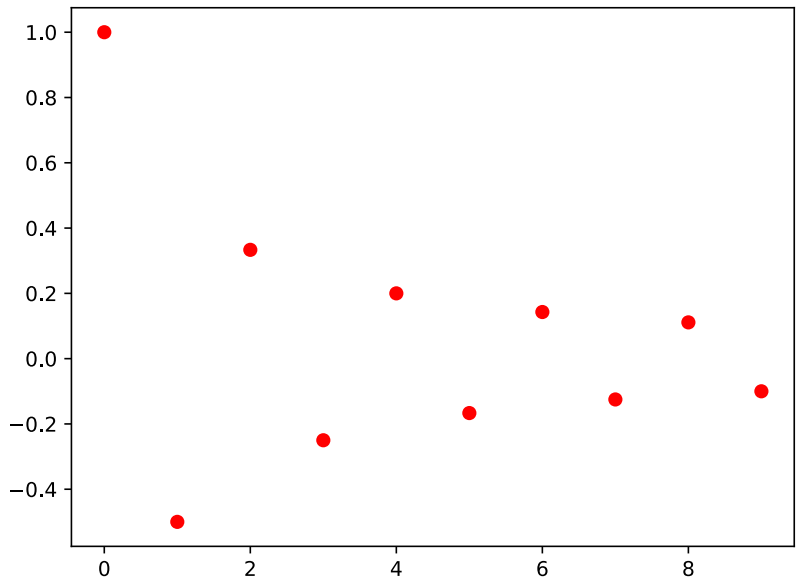
# Représentations graphiques

```
1  import matplotlib.pyplot as plt
2
3  Y = [(-1)**n/(n+1) for n in range(11)]
4
5  plt.plot(Y)
6  plt.show()
```

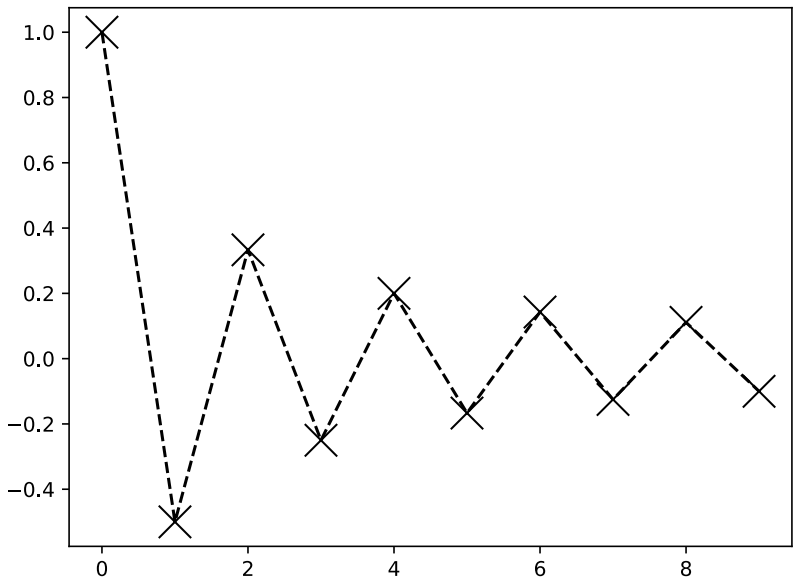


# Représentations graphiques

```
plt.plot(Y, linestyle='',
color='red', marker='o')
```

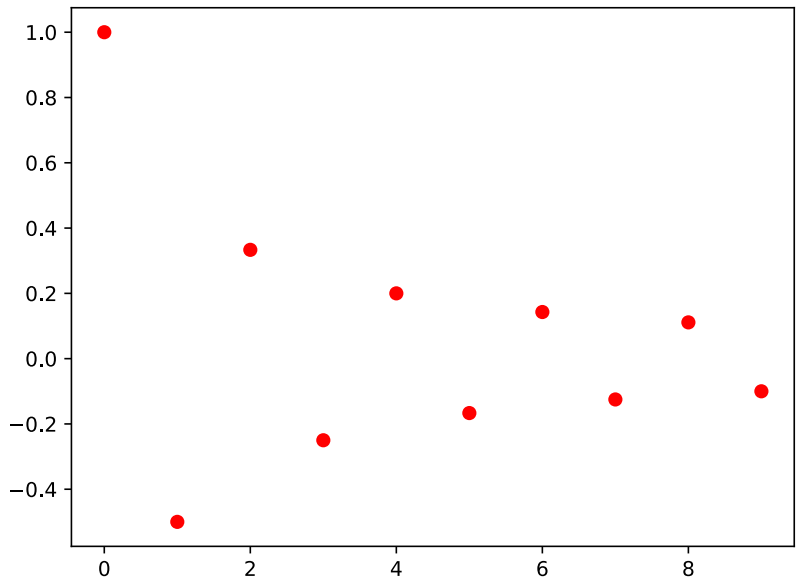



```
plt.plot(Y, linestyle='--', color='black',
marker='x', markersize='16')
```

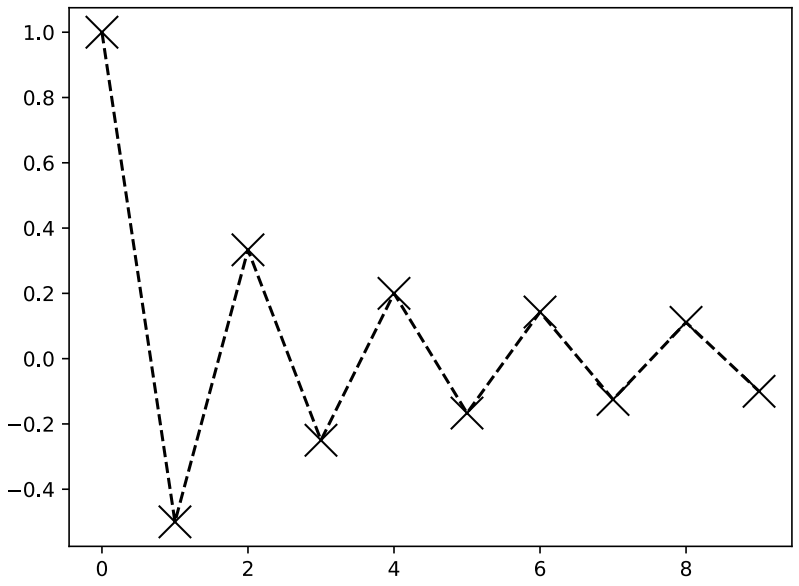


# Représentations graphiques

```
plt.plot(Y, 'ro')
```

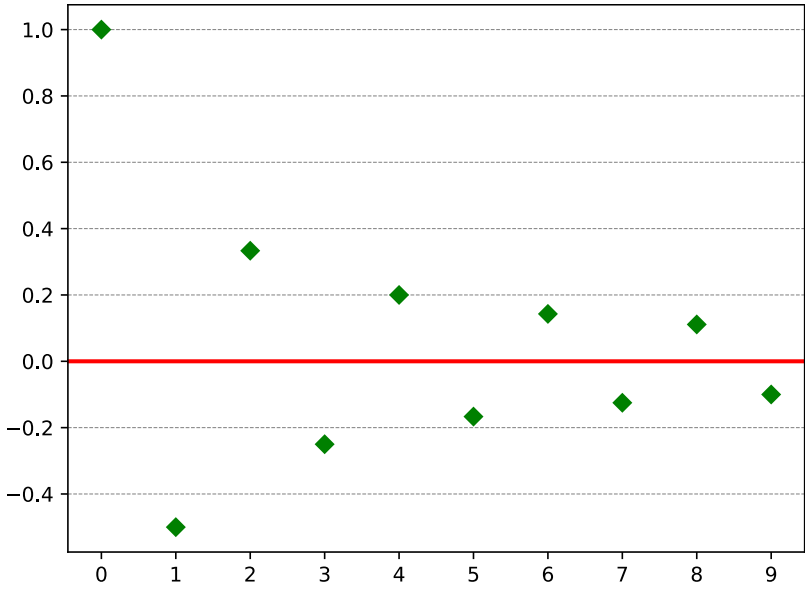



```
plt.plot(Y, '--xk', ms='16')
```



# Représentations graphiques

```
plt.plot(Y, 'gD')
plt.grid(axis='y', ls='--', lw=0.5,
c='gray')
plt.axhline(0, c='r', lw=2)
plt.xticks(range(10))
```




## Pour enregistrer :

1) Cliquer sur 


(formats JPG, PNG, SVG, PDF)

2)
```
plt.savefig("suite.svg")
```

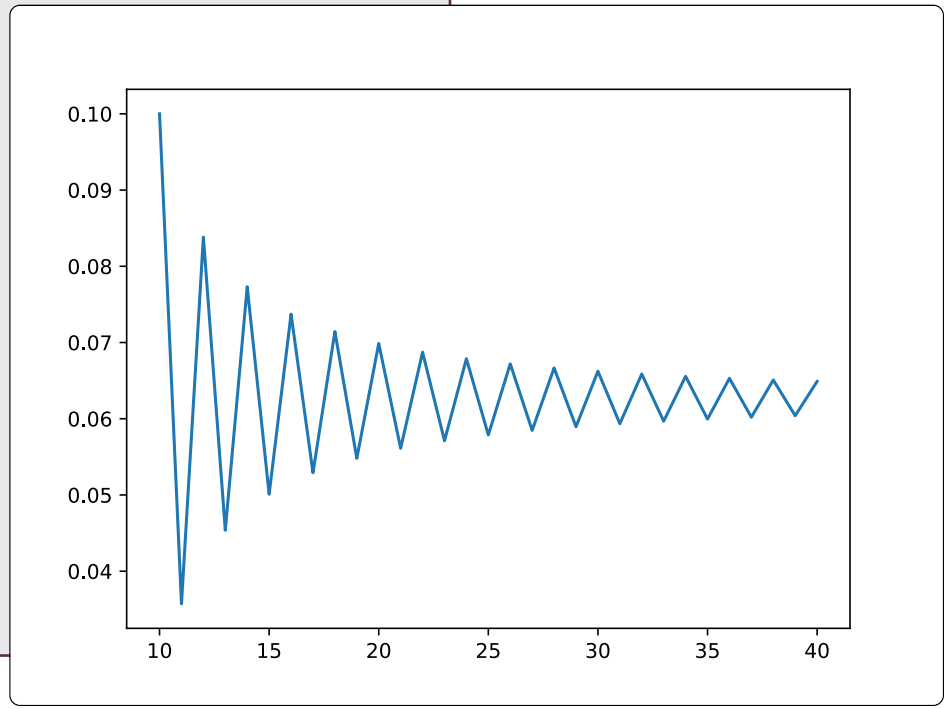
## Pour retirer les marges :

```
plt.figure(figsize=(5.5, 4.0))
...
plt.tight_layout(pad=0)
```

# Représentations graphiques

**Exemple 12 :** $v_k = \dfrac{(-1)^k}{k\sqrt{k-9}}$, $k \geq 10$. Valeurs de $S_n = \displaystyle\sum_{k=10}^{n} v_k$ ?

```
1  from math import *
2  import matplotlib.pyplot as plt
3
4  def v(k):
5      return (-1)**k/(k*sqrt(k-9))
6
7  X = [k for k in range(10, 41)]
8  Y = []
9  s = 0
10
11 for k in X:
12     s += v(k)
13     Y.append(s)
14
15 plt.plot(X, Y)
16 plt.show()
```

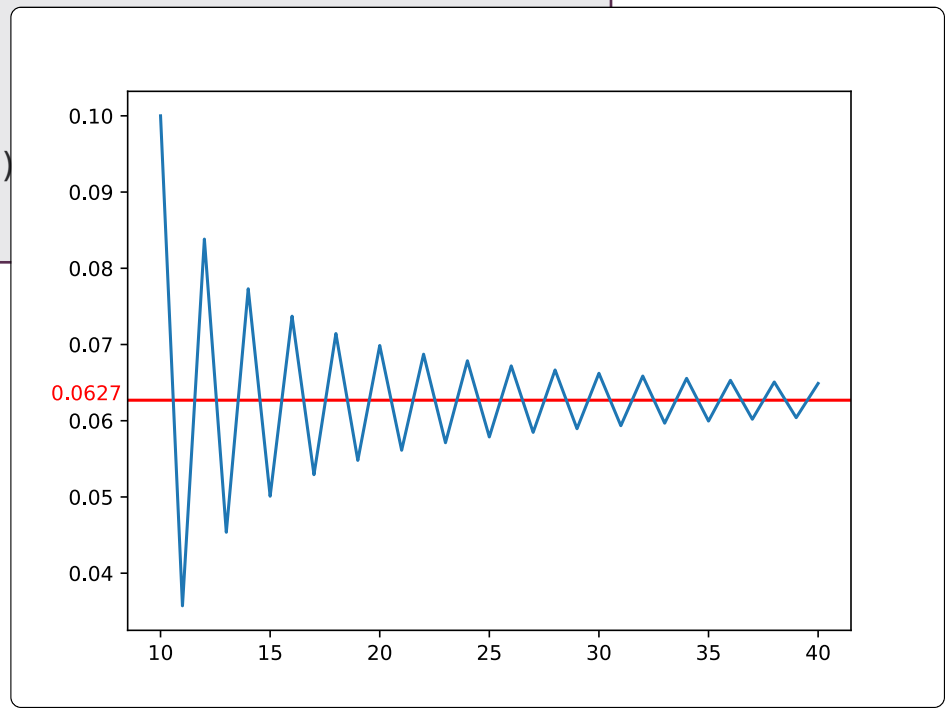


# Représentations graphiques

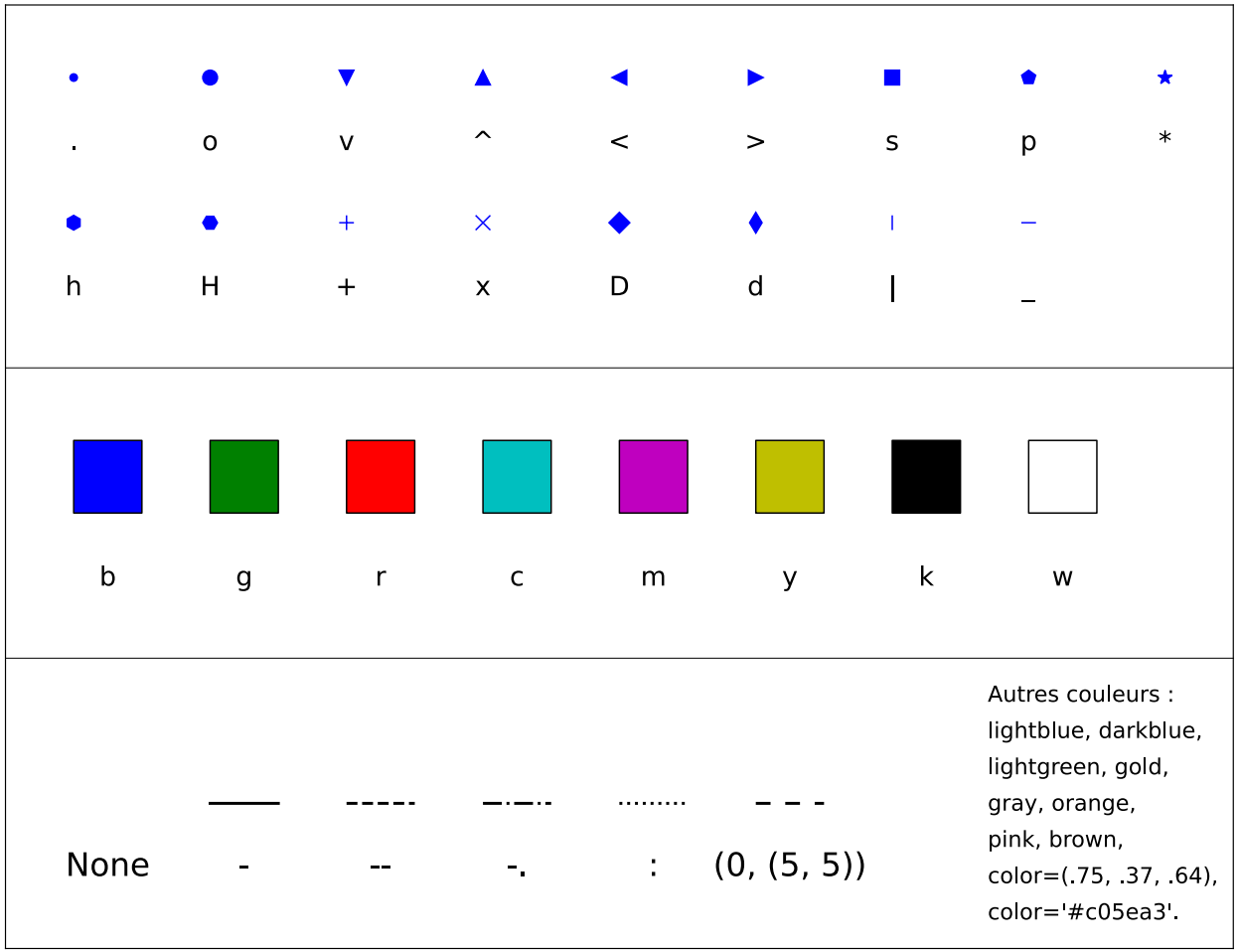
**Exemple 12 :** $v_n = \dfrac{(-1)^n}{n\sqrt{n-9}}$ pour $n \geq 10$

```
...
for k in range(41, 5001):
    s += v(k)

plt.axhline(s, c='r')
plt.text(5, s, round(s, 4), c='r')
...
```



# Options de matplotlib



# Exercices

1) Représenter la suite $u_n = \dfrac{1}{\sqrt{n+1}} \sin\left(\dfrac{n\pi}{4}\right)$ pour $n \in [\![0\,;10]\!]$.

   Pas de ligne. Marqueurs : carré de taille 16, couleur jaune.

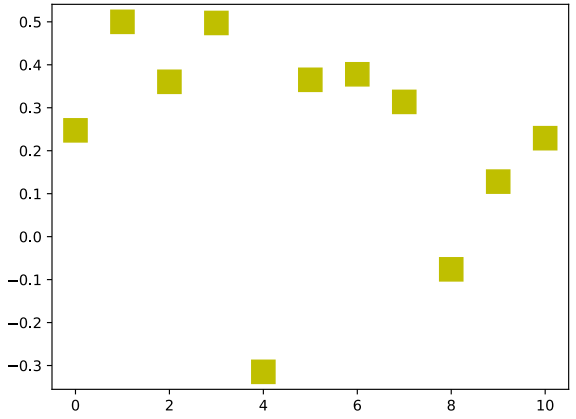
2) Représenter l'évolution de la somme partielle $S_n = \sum_{k=1}^{n} u_k$ avec $u_k = \dfrac{1}{k}\cos\left(\dfrac{k\pi}{6}\right)$ et $n \in [\![1\,;60]\!]$.

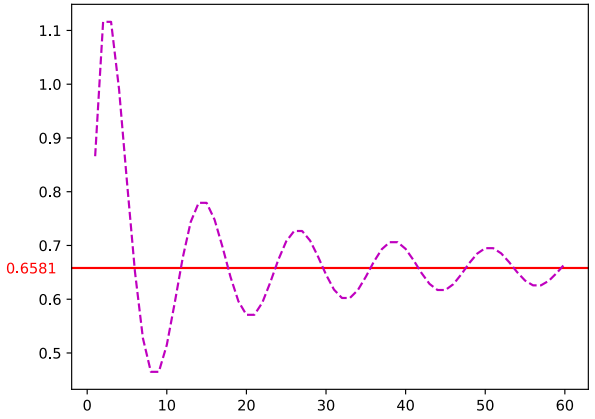
   Style de ligne : pointillés, couleur magenta. Pas de marqueurs.

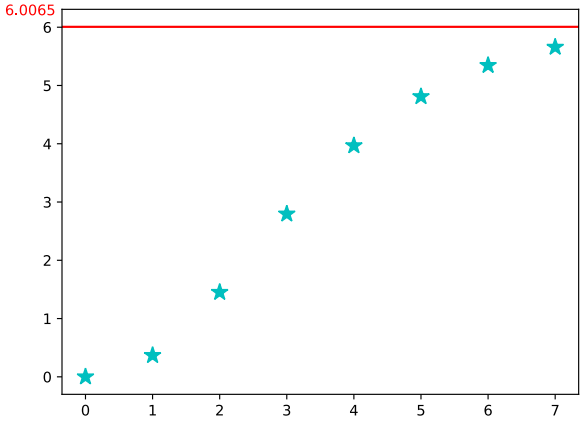
3) Représenter l'évolution de la somme partielle $T_n = \sum_{k=0}^{n} v_k$ avec $v_k = e^{-k}k^3$ et $n \in [\![0\,;10]\!]$.

   Pas de ligne. Marqueurs : étoiles de taille 12, couleur cyan.

# Exercices


0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8
10


1.1
1.0
0.9
0.8
0.7
0.6581
0.6
0.5
0
10
20
30
40
50
60


6.0065
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7

# Exercices

1) Représenter la suite $u_n = \dfrac{1}{\sqrt{n+1}} \sin\left(\dfrac{n\pi}{4}\right)$ pour $n \in [\![0\,;10]\!]$.

   Pas de ligne. Marqueurs : carré de taille 16, couleur jaune.

2) Représenter l'évolution de la somme partielle $S_n = \sum_{k=1}^{n} u_k$ avec $u_k = \dfrac{1}{k}\cos\left(\dfrac{k\pi}{6}\right)$ et $n \in [\![1\,;60]\!]$.

   Style de ligne : pointillés, couleur magenta. Pas de marqueurs.

3) Représenter l'évolution de la somme partielle $T_n = \sum_{k=0}^{n} v_k$ avec $v_k = e^{-k}k^3$ et $n \in [\![0\,;10]\!]$.

   Pas de ligne. Marqueurs : étoiles de taille 12, couleur cyan.

# Suites récurrentes définies par une fonction : rappels

**Cadre.** On considère une fonction $f : I \subset \mathbb{R} \to I$ et une suite $(u_n)$ définie par $u_0 \in I$ et $\forall n \in \mathbb{N}, u_{n+1} = f(u_n)$.

**Propriétés.**

1) **Si $f$ est croissante, alors $(u_n)$ est monotone.**
   De plus, $(u_n)$ est croissante (resp. décroissante) si la courbe représentative de $f$ est au-dessus (resp. en-dessous) de la droite d'équation $y = x$.

2) Si $f$ est décroissante, les suites extraites $(u_{2n})$ et $(u_{2n+1})$ sont monotones et de monotonies contraires.

# Suites récurrentes définies par une fonction : rappels

**Définition.**

Un élément $\ell$ de $I$ est appelé « point fixe » s'il vérifie $f(\ell) = \ell$.

Un point fixe $\ell$ est dit **attractif** s'il existe un voisinage $U$ de $\ell$ tel que $\forall u_0 \in U$, la suite $(u_n)$ converge vers $\ell$.

Un point fixe $\ell$ est dit **répulsif** s'il n'existe aucun voisinage $U$ tel qu'il existe $u_0 \in U \setminus \{\ell\}$ et tel que $(u_n)$ converge vers $\ell$.

**Propriété.** Si $f$ est dérivable en $\ell$, alors :

- si $|f'(\ell)| < 1$, $\ell$ est attractif
- si $|f'(\ell)| > 1$, $\ell$ est répulsif

# Suites récurrentes définies par une fonction

**Exemple 13 :** $u_0 \in \mathbb{R}^+, \quad u_{n+1} = f_1(u_n)$

avec $f_1(x) = 6 - \dfrac{9}{x+1{,}5}$ sur $\mathbb{R}^+$

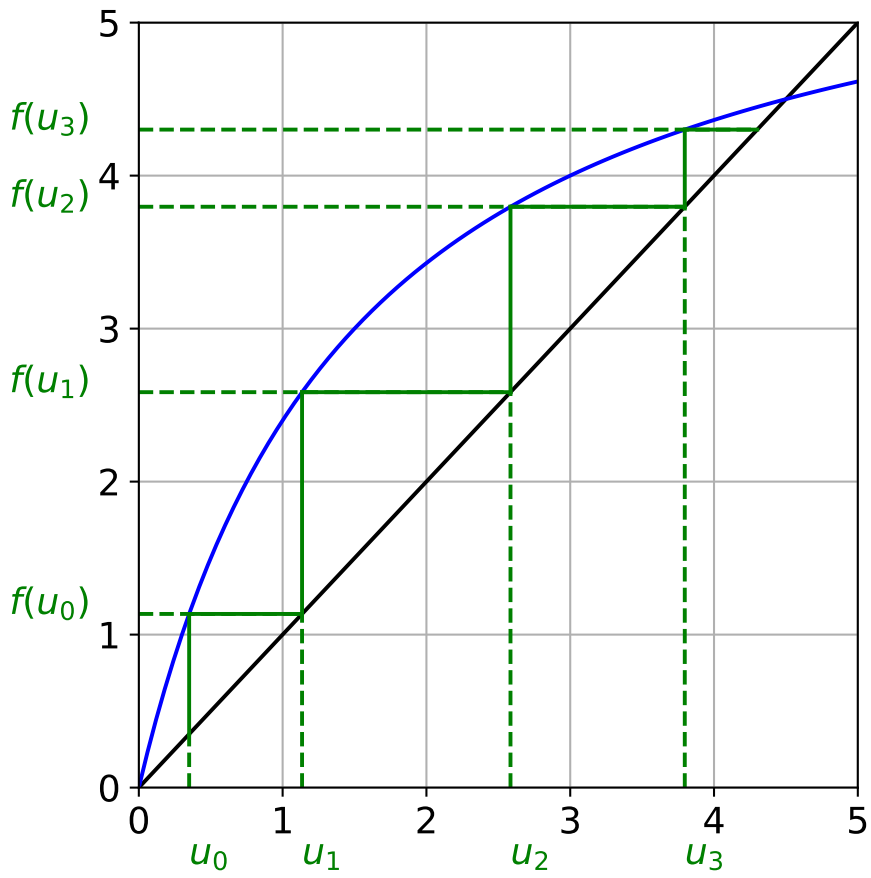
$f_1'(x) = \dfrac{9}{(x+1{,}5)^2} > 0$

$f_1(\ell) = \ell \quad \Leftrightarrow \quad \ell = 0 \quad \text{ou} \quad \ell = 4{,}5$

$f_1'(0) = 4 > 1$ donc 0 est répulsif.

$f_1'(4{,}5) = \dfrac{1}{4} < 1$ donc 4,5 est attractif.

Exemple avec $u_0 = 0{,}35 \rightarrow$



# Suites récurrentes définies par une fonction

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return 6 - 9 / (x + 1.5)

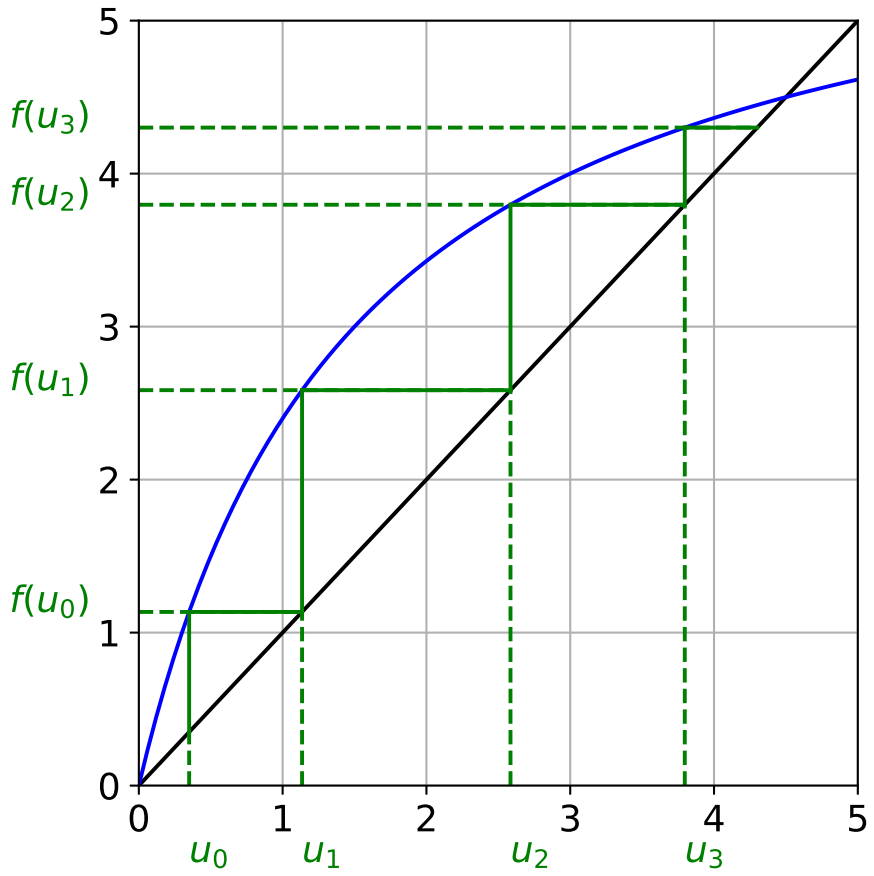
X = np.linspace(0, 5, 101)

plt.figure(figsize=(5, 5))
plt.rcParams['font.size'] = 14

plt.plot(X, X, 'k-')
plt.plot(X, f(X), 'b-')

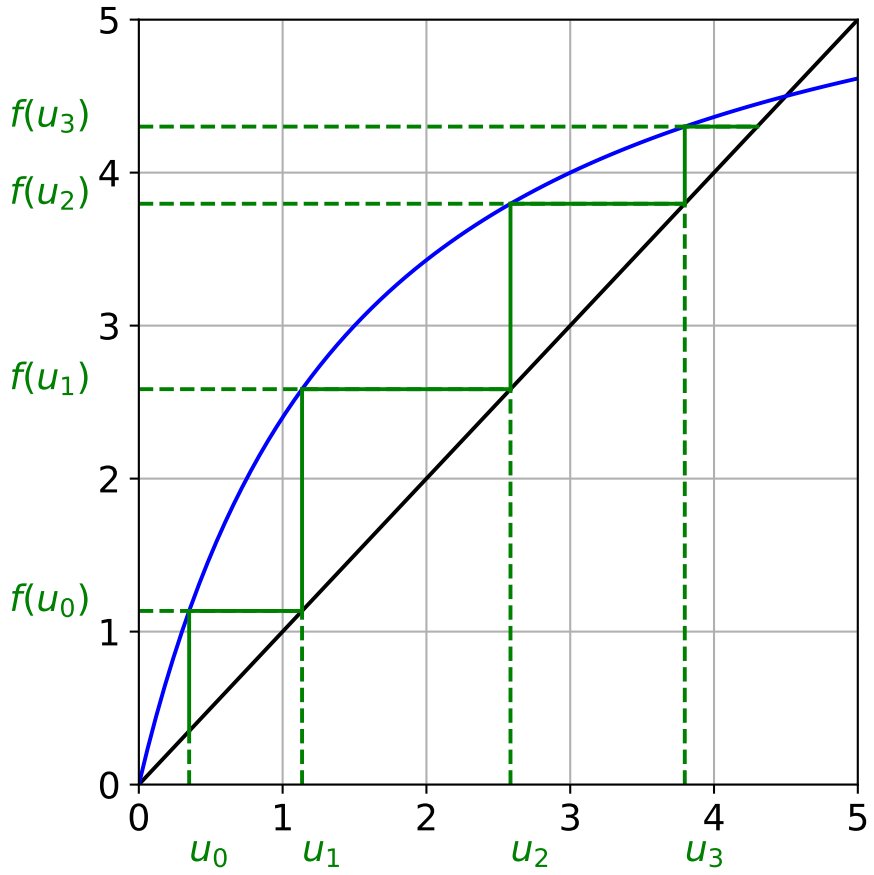
u = 0.35

for n in range(4):
    # Escalier
    plt.plot([u, u], [u, f(u)], 'g-')
    plt.plot([u, f(u)], [f(u), f(u)], 'g-')
```



# Suites récurrentes définies par une fonction

```
21      # Abscisses
22      plt.plot([u, u], [0, u],
    'g--')
23      plt.text(u, -0.5, f"$u_{n}$",
    c='g')
24
25      # Ordonnées
26      plt.plot([0, f(u)], [f(u),
    f(u)], 'g--')
27      plt.text(-0.9, f(u),
    f"$f(u_{n})$", c='g')
28
29      u = f(u)
30
31  plt.grid(True)
32  plt.xlim(0, 5)
33  plt.ylim(0, 5)
34  plt.tight_layout(pad=1)
35  plt.show()
```



# Suites récurrentes définies par une fonction

**Exemple 14 :**

$u_0 \in [0;4{,}5], u_{n+1} = f_2(u_n)$

avec $f_2(x) = -1{,}5 - \dfrac{9}{x-6}$ sur $[0;4{,}5]$

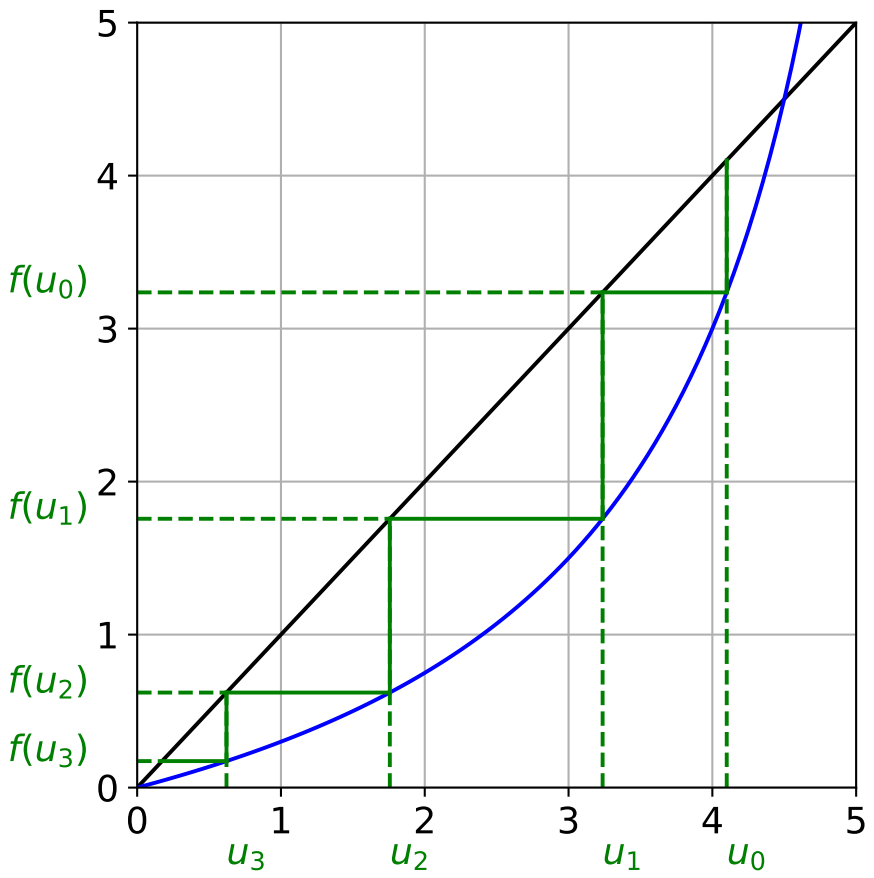
$f_2'(x) = \dfrac{9}{(x-6)^2} > 0$

$f_2(\ell) = \ell \quad \Leftrightarrow \quad \ell = 0$ ou $\ell = 4{,}5$

$f_2'(0) = \dfrac{1}{4} < 1$ donc 0 est attractif.

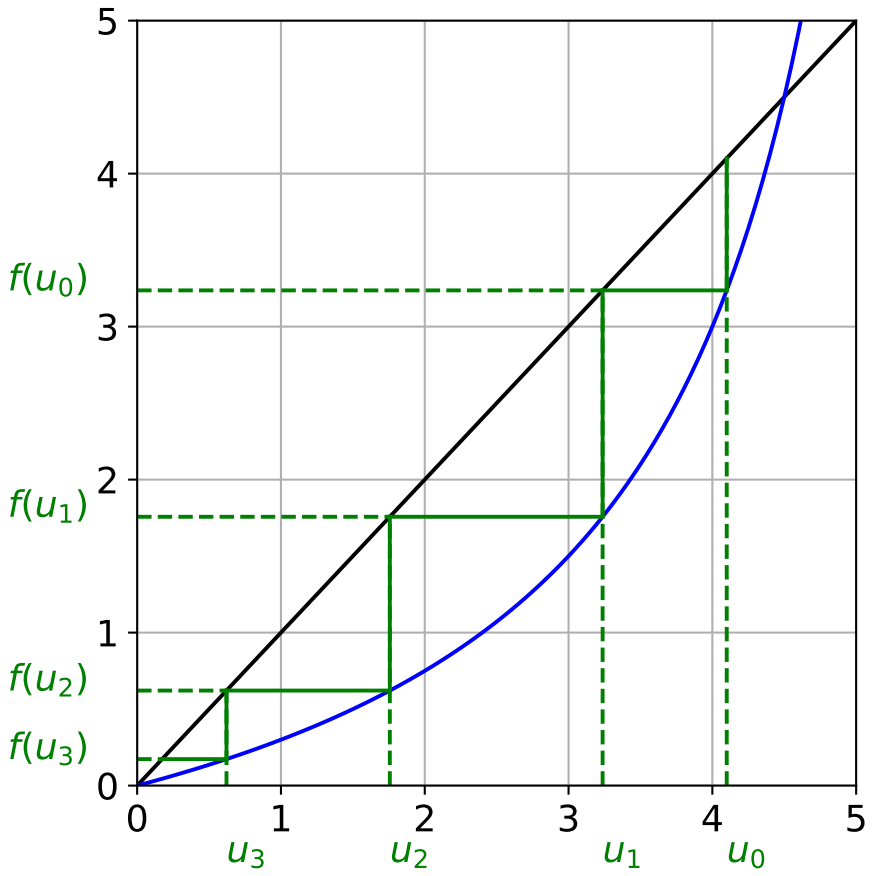
$f_2'(4{,}5) = 4 > 1$ donc 4,5 est répulsif.

Exemple avec $u_0 = 4{,}1 \rightarrow$



# Suites récurrentes définies par une fonction

```
def f(x):
    return -1.5 - 9 / (x - 6)
...
u = 4.1
```



# Suites récurrentes définies par une fonction

**Exemple 15 :** $u_0 \in \mathbb{R}^+, u_{n+1} = f_3(u_n)$

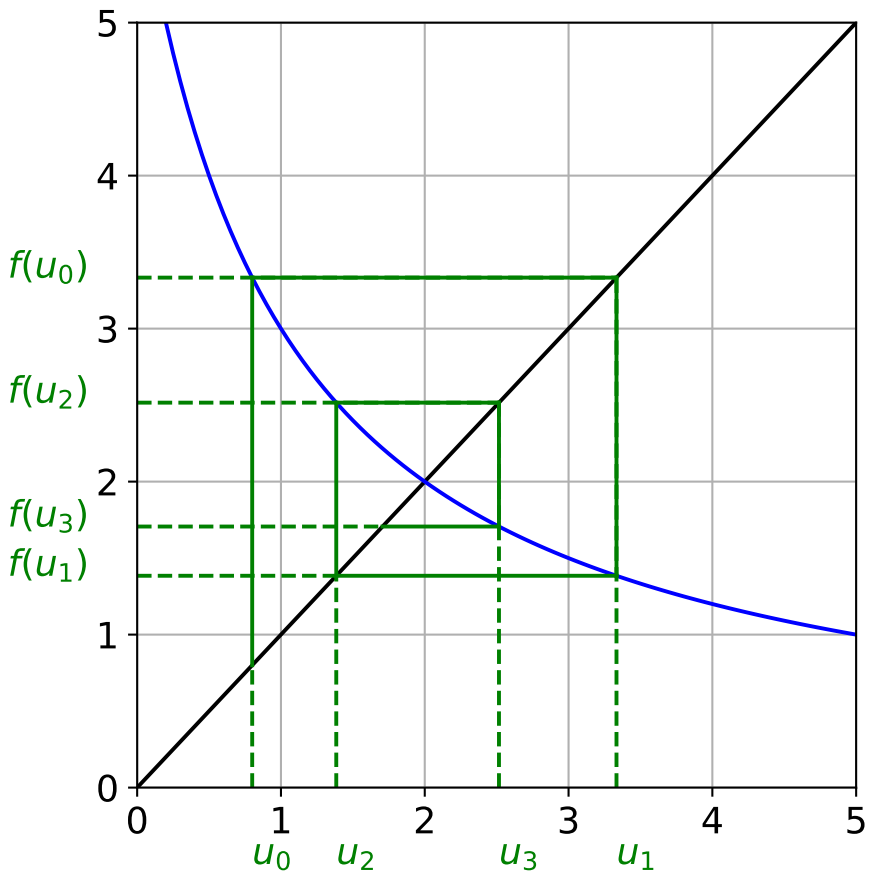
avec $f_3(x) = \dfrac{6}{x+1}$ sur $\mathbb{R}^+$

$f_3'(x) = \dfrac{-6}{(x+1)^2} < 0$

$f_3(\ell) = \ell \quad \Leftrightarrow \quad \ell = -3 \quad \text{ou} \quad \ell = 2$

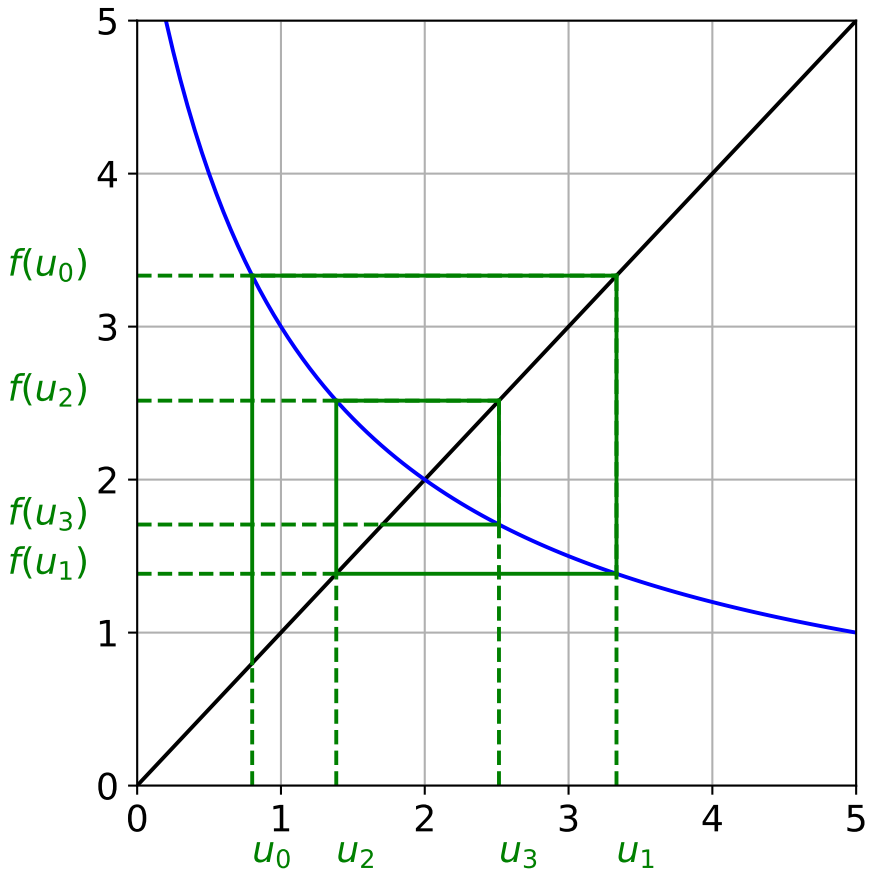
$|f_3'(2)| = \left|-\dfrac{2}{3}\right| < 1$ donc 2 est attractif.

Exemple avec $u_0 = 0{,}8 \rightarrow$



# Suites récurrentes définies par une fonction

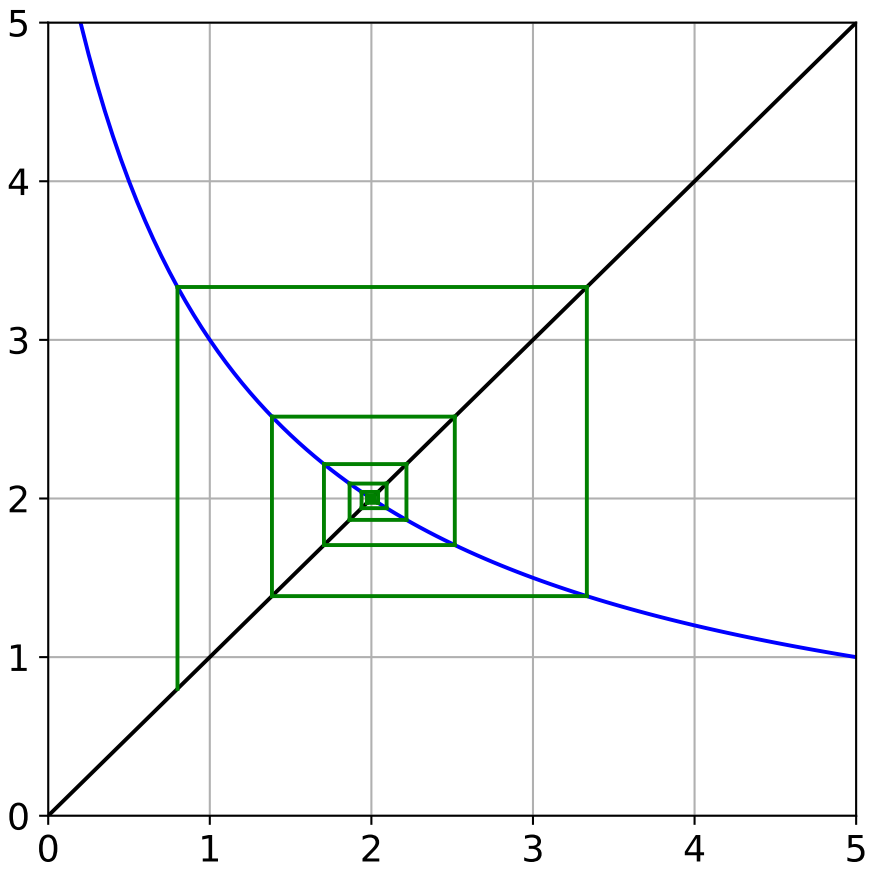
```
def f(x):
    return 6 / (x + 1)
...
u = 0.8
```



# Suites récurrentes définies par une fonction

```
def f(x):
    return 6 / (x + 1)
...
u = 0.8
```

- Désactivations des lignes 22, 23, 26, 27 (après `# Abscisses` et `# Ordonnées`).
- Augmentation du nombre de termes



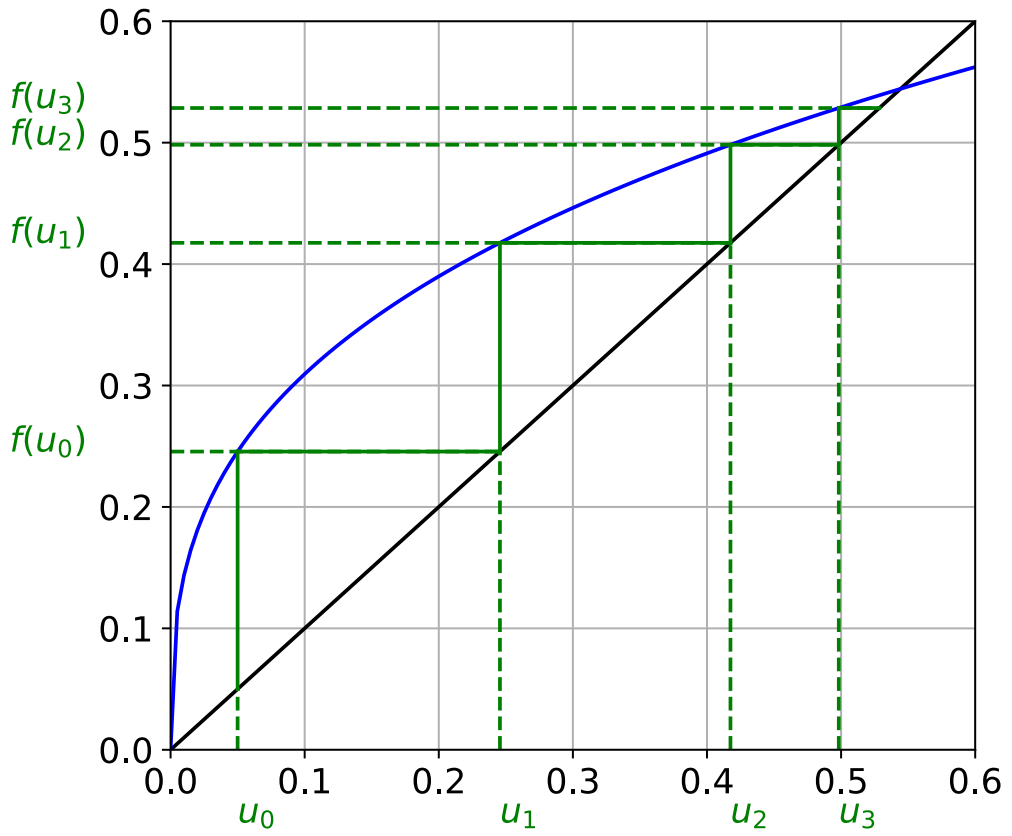
# Exercice

1) Tracer le graphique suivant avec :

- $f(x) = \frac{2}{3}\sqrt[3]{x}$
- $u_0 = 0{,}05$ et $u_{n+1} = f(u_n)$
- une fenêtre de $[0\,;0{,}6] \times [0\,;0{,}6]$

Remarques :

- utiliser `x**(1/3)` et non `sqrt(x, 3)`
- augmenter le nombre de points de `np.linspace` à 1001
- agrandir la largeur à 6 : `figsize=(6, 5)`
- ajuster abscisses et ordonnées des textes.

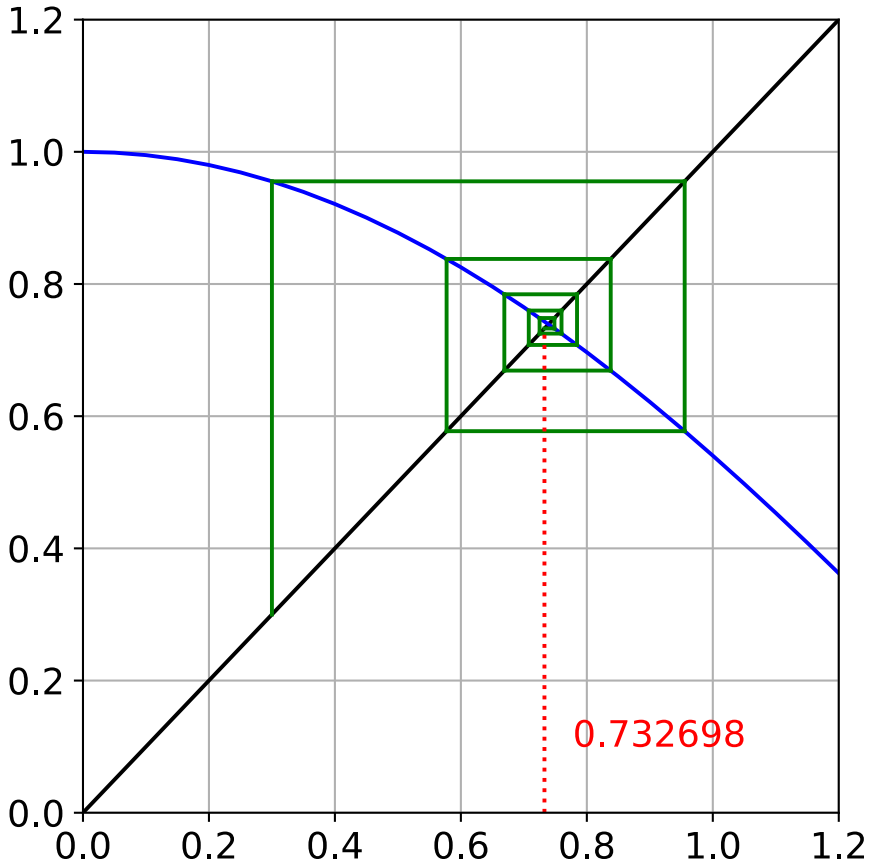


# Exercice

2) Tracer le graphique suivant avec :

- $f(x) = \cos x$
- $u_0 = 0{,}3$ et $u_{n+1} = f(u_n)$
- un colimaçon de 10 termes.
- un segment vertical donnant une valeur approchée de $\lim\limits_{n\to+\infty} u_n$ (« nombre de Dottie »)
- une fenêtre de $[0\,;1{,}2] \times [0\,;1{,}2]$

Remarque : utiliser `np.cos(x)`



# Problème de Bâle

**Exemple 16 :** Trouver une (bonne) valeur approchée de

$$\zeta(2) = \sum_{k=1}^{+\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

```python
from math import *

n = 3*10**6
S = sum([1/k**2 for k in range(1, n+1)])

print(S)
print(pi**2/6)
```

*Exécution du code*

```
1.64493 37335149488
1.64493 40668482264
```

**Pas terrible. Comment améliorer la vitesse de convergence ?**



# Problème de Bâle

**Exemple 16 :** Utilisation de la formule d'Euler-Maclaurin

$$\zeta(2) \approx \sum_{k=1}^{n} \frac{1}{k^2} + \frac{1}{n} - \frac{1}{2n^2} + \frac{1}{6n^3} - \frac{1}{30n^5} + \frac{1}{42n^7} - \frac{1}{30n^9} + R_4$$

avec $R_4 = \mathrm{O}\left(\frac{1}{n^{11}}\right)$

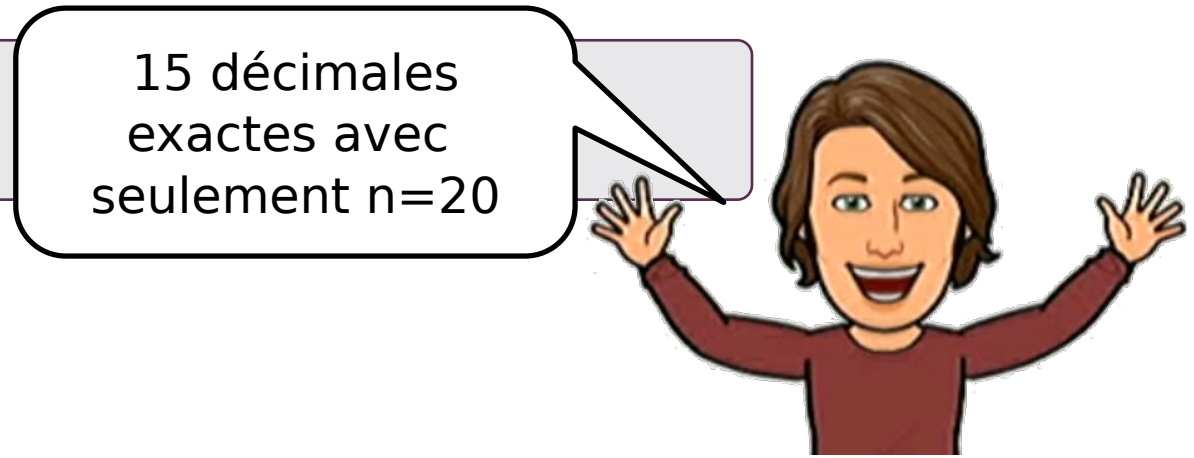
```
1  from math import *
2
3  n = 20
4  S = sum([1/k**2 for k in range(1, n+1)])
5
6  S = S + 1/n - 1/(2*n**2) + 1/(6*n**3) - 1/(30*n**5) + 1/
   (42*n**7) - 1/(30*n**9)
7
8  print(S)
9  print(pi**2/6)
```

# Problème de Bâle

```
1  from math import *
2
3  n = 20
4  S = sum([1/k**2 for k in range(1, n+1)])
5
6  S = S + 1/n - 1/(2*n**2) + 1/(6*n**3) - 1/(30*n**5) + 1/
   (42*n**7) - 1/(30*n**9)
7
8  print(S)
9  print(pi**2/6)
```

*Exécution du code*

```
1.644934066848226 2
1.644934066848226 4
```



# Rappel : vitesse de convergence

**Vitesse de convergence.**

Soit $(u_n)$ une suite (réelle) qui converge vers une limite réelle $L$.

On dit que cette suite converge **linéairement** vers $L$ s'il existe $\lambda \in ]0,1[$ tel que $\lambda = \lim\limits_{n\to+\infty} \dfrac{u_{n+1}-L}{u_n-L}$

**Remarque.**

On notera « $|\lambda| = 1^-$ » si $|\lambda| = 1$ et pour tout $n$, $\left|\dfrac{u_{n+1}-L}{u_n-L}\right| < 1$

# Procédé Delta-2 d'Aitken

**Principe.** Soit $(u_n)$ une suite (réelle) qui converge vers une limite finie dont on ne connait, *a priori*, pas la valeur.

On introduit **$\Delta$, l'« opérateur de différence finie avant »**, défini par $\Delta u_n := u_{n+1} - u_n$

On associe à cette suite la suite $(v_n)$ définie par :

$$v_n = u_n - \frac{(\Delta u_n)^2}{\Delta^2 u_n} = u_n - \frac{(u_{n+1} - u_n)^2}{u_{n+2} - 2u_{n+1} + u_n}$$

**Remarque :**

$$\begin{aligned}\Delta^2 u_n &= \Delta(\Delta u_n) = \Delta u_{n+1} - \Delta u_n \\ &= (u_{n+2} - u_{n+1}) - (u_{n+1} - u_n) = u_{n+2} - 2u_{n+1} + u_n\end{aligned}$$

# Procédé Delta-2 d'Aitken

**Théorème.** Soit $(u_n)$ une suite (réelle) qui converge linéairement vers $L \in \mathbb{R}$.

Alors la suite $(v_n)$ définie par

$$v_n = u_n - \frac{(\Delta u_n)^2}{\Delta^2 u_n} = u_n - \frac{(u_{n+1} - u_n)^2}{u_{n+2} - 2u_{n+1} + u_n}$$

**converge vers $L$ plus rapidement**, c'est-à-dire que

$$\lim_{n \to +\infty} \frac{v_n - L}{u_n - L} = 0$$

# Procédé Delta-2 d'Aitken

**Exemple 17 :** $\displaystyle\sum_{k=0}^{+\infty} \frac{(-1)^k}{k+1} = \ln 2$ (avec « $|\lambda| = 1^-$ »)

On considère la **suite** $u_n = \displaystyle\sum_{k=0}^{n} \frac{(-1)^k}{k+1}$. Comparons $u_{2000}$ et $v_{2000}$.

```python
from math import *

n = 2000
s, u = 0, []

for k in range(0, n+3):
    s += (-1)**k/(k+1)
    u.append(s)

vn = u[n] - (u[n+1]-u[n])**2 / (u[n+2]-2*u[n+1]+u[n])

print(f"u{n}  =", u[-3])
print(f"v{n}  =", vn)
print("ln 2   =", log(2))
```

# Procédé Delta-2 d'Aitken

```python
from math import *

n = 2000
s, u = 0, []

for k in range(0, n+3):
    s += (-1)**k/(k+1)
    u.append(s)

vn = u[n] - (u[n+1]-u[n])**2 / (u[n+2]-2*u[n+1]+u[n])

print(f"u{n}  =", u[-3])
print(f"v{n}  =", vn)
print("ln 2   =", log(2))
```

*Exécution du code*

```
u2000  = 0.693 3969931 848778
v2000  = 0.693 1471805 755146
ln 2   = 0.693 1471805 599453
```

# Exercice

1. On considère la **suite** $u_n = \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$

Sachant que $\lim_{n \to +\infty} u_n = \frac{\pi}{4}$, comparer les valeurs de :

   a. $u_{2000}$

   b. $v_{2000}$, avec $(v_n)$, suite associée du procédé Delta-2 d'Aitken

   c. $\frac{\pi}{4}$, valeur pré-enregistrée du logiciel.

2. Même exercice avec $u_n = \sum_{k=1}^{n} \frac{1}{k^2}$

Le procédé est-il aussi efficace que précédemment ?

# Conclusion

## Approfondissements possibles

- « Suite de polygones »
- Séries de Fourier
- Théorème de réarrangement de Riemann

## Quelques conseils

- Apprendre les syntaxes basiques de `matplotlib` par cœur
- Observer des exemples sur `https://matplotlib.org/` pour les adapter, jeter un coup d'œil à la documentation
- Utiliser l'IA de manière pertinente
- Avoir une bibliographie de référence

# Bibliographie


- Philippe Caldero, *Carnet de voyage en Analystan*, Calvage et Mounet, 2024
- Julien Guillod, *Programmation Python par la pratique* (2e édition), Dunod, 2024. `https://python.guillod.org/fr/`
- Thierry Meyre, *Séries et intégrales*, Calvage et Mounet, 2025
- Sébastien Peronno, *40 thèmes illustrés par le numérique pour l'agrégation interne de mathématiques*, Ellipses, 2026
- Georges Skandalis, *Agrégation interne. Analyse : résumés de cours et exercices corrigés*, Calvage et Mounet, 2024
- `joliesmaths.fr/sup` → téléchargement des codes et diapos

# Solutions

**Suites**

1) `u(100) = 2.704813...,  u(1000) = 2.7169239....`

2) Point fixe : (3,5 ; 3,5)

3) $\lim w_n = 2$

4) `F(15)/F(14) = 1.61803 2786885246`
   `(1+5**.5)/2 = 1.61803 3988749895`

5) a. `[1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37,`
   `49, 65, 86, 114, 151, 200, 265, 351, 465, 616]`

   b. $\psi \approx 1,3247179572447 46$

# Solutions

## Séries

1) `1.6538910093242722`

2) `8.057100033329066`

3) `65583`

4)

```
1 def f(x, n=10):
2     return sum([(-1)**k * x**k for k in range(n+1)])
```

$-0{,}2$ et $0{,}1 \rightarrow$ TB convergence ; $0{,}5 \rightarrow$ convergence OK

$1{,}5 \rightarrow$ divergence

5)

```
1 def d(n, k):
2     return int(factorial(n)/factorial(k)*sum([(-1)**i/
  factorial(i) for i in range(n-k+1)]))
```

# Solutions

## Accélération de convergence

1) Avec `s += (-1)**k/(2*k+1)` :

```
u2000= 0.785 523100 9208814
v2000= 0.785 398163 4052369
pi/4 = 0.785 398163 3974483
```

2) Avec `s += 1/(k+1)**2` :

```
u2000 = 1.644 4344415775836
v2000 = 1.644 6843790698442
pi^2/6= 1.644 9340668482264
```

# Fin

