Lycée Hoche
ECG1 A

2024-2025
Mathématiques

TP de Python numéro 4
Tracés avec `matplotlib`

Semaine du .

I. La bibliothèque graphique matplotlib

1. Première utilisation de `plt.plot`

Pour réaliser des figures avec Python, on utilisera la bibliothèque `matplotlib` de python, et plus particulièrement son *sous-module* `pyplot`.

Pour l'utiliser, on l'importera avec la commande suivante :

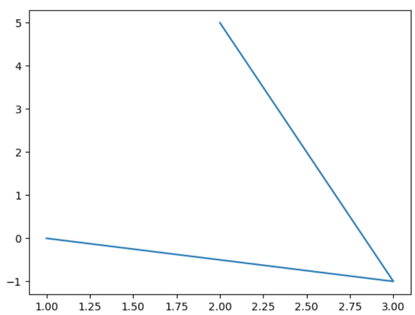
```
import matplotlib.pyplot as plt
```

Une fois cette commande effectuée, le module `matplotlib.pyplot` est donc importé avec, comme raccourci choisi `plt`.

Ce module permet de générer des graphiques, personnalisables, à l'aide d'une liste de points à placer.

Regardons un premier exemple :

```
import matplotlib.pyplot as plt
listeX=[1,3,2]
listeY=[0,-1,5]
plt.plot(listeX,listeY)
plt.show()
```



Dans cet exemple :

- La commande `plt.plot` génère un graphique à partir de deux listes données (`listeX` et `listeY` ici).
- La première liste est interprétée comme une liste d'abscisse, la seconde comme une liste d'ordonnées, et ces deux listes doivent avoir la même longueur.
- Ce graphique place la liste des points ainsi décrits, et les relie par une ligne brisée selon l'ordre spécifié dans la liste des abscisse. Ici, il s'agit donc des points de coordonnées $(1,0)$, $(3,-1)$ puis $(2,5)$.
- La commande `plt.show()` demande à Python d'afficher le graphique généré.

Ceci résume à première vue le comportement des commandes `plt.plot()` et `plt.show()`.

Exercice 1. Tracer avec Python une lignée brisée passant, dans cet ordre, pas les points de coordonnées $(-1,\frac{1}{2})$, $(0,1)$, $(1,2)$, $(2,4)$, $(3,8)$ et $(4,16)$. Que remarquez-vous ?

Remarque. La commande `plt.show()` ne prendra pas d'arguments, mais il ne faut pas oublier de mettre les parenthèses.

Remarque. À partir de maintenant et pour tout le TP, on supposera les imports suivants effectués :

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

Remarque. À la place de donner une liste d'abscisse en tant que `list`, on peut utiliser la fonction `range` pour décrire des abscisses.

Exemple 2. La code python suivant affiche un graphique des 48 premiers termes de la suite u définie par $u_n = n^2 + (-1)^n n$.

```
listeX=range(48)
listeY=[n**2+n*(-1)**n for n in listeX]
plt.plot(listeX,listeY)
plt.show()
```

Exercice 3. Tracer, avec Python, une ligne brisée passant par les points d'abscisses -5,-3,0,2,3,5,7,10 et 20 du graphe de la fonction $t \mapsto t^2$.

2. Personnalisation des tracés

Remarque. Ces commandes ne sont pas exigibles pour le concours, mais sont assez indispensables dans certains exercices de ce TP, et dans les contexte où vous devez tracer plusieurs courbes sur un même graphique.

a) Marqueurs de points, lignes et couleur

On dispose d'un bon nombre d'options de personnalisation des tracés effectués avec `plt.plot`. On utilisera pour cela les **code** présentés en partie 1 de l'annexe de ce TP.

Personnalisation des points, des lignes et de la couleur

Si, conformément à l'annexe :
- `p` désigne un code pour les marqueurs de points,
- `m` désigne un code pour les lignes,
- `c` désigne un code pour la couleur,
- `listeX` et `listeY` désignent respectivement, comme précédemment, des listes d'abscisses et d'ordonnées,

alors la personnalisation voulue du graphique se précise en argument supplémentaire de `plt.plot` de la manière suivante :

`plt.plot(listeX,listeY,"plc")`

On peut également ne préciser que certaines options de personnalisation. Par exemple :
- `plt.plot(listeX,listeY,"X:r")` engendre un graphique dont les points sont marqués par des grosses croix, reliés par des pointillés, en rouge.
- `plt.plot(listeX,listeY,"o--")` engendre un graphique avec des gros points reliés par des tirets,
- `plt.plot(listeX,listeY,"Dg")` engendre un graphique dont les points sont marqués par des diamants verts.

Exercice 4. Tracer les 10 premiers termes de la suite u définie par $u_n = (-1)^n\sqrt{n}$ en marquant les points par des grosses croix rouges.

b) Titre, légendes et quadrillage

On peut spécifier ces éléments selon la syntaxe suivante. Dans ce code, on suppose que `listeX` et `listeY` sont, à nouveau, des listes d'abscisses et d'ordonnées définies précédemment.

```
# Ajout d'une légende désignant la courbe
plt.plot(listeX,listeY,label="Légende de la courbe")

# Légendes supplémentaires : entre plt.plot() et plt.show()
# L'ordre entre les commandes ci-dessous est au choix
# On peut ne demander qu'une partie de ces personnalisations
plt.title("Titre du graphique")
plt.grid() # Ajoute un quadrillage
plt.xlabel("Légende en abscisse")
plt.ylabel("Légende en ordonne")

plt.legend()  # Demande l'affichage des légendes (obligatoire)

# Demande d'affichage
plt.show()
```

c) Personnalisation des axes

Pour préciser à Python les valeurs d'abscisse et d'ordonnée entre lesquelles afficher les axes, on utilise les commandes `plt.xlim` et `plt.ylim` entre `plt.plot` et `plt.show`.

On suppose toujours que `listex` et `listeY` sont des listes d'abscisses et d'ordonnée valide.

```
plt.plot(listex,listeY)

# Personnalisation des axes
# l'ordre est au choix
plt.xlim(xmin,xmax)  # xmin, xmax sont des nombres à spécifier
plt.ylim(ymin,ymax)  # idem

# Demande d'affichage
plt.show()
```

Par exemple, pour afficher un graphique où l'axe des abscisse va de 0 à 10, et l'axe des ordonnées de 0 à 20, on écrit :

```
plt.plot(listex,listeY)

plt.xlim(0,10)
plt.ylim(0,20)

plt.show()
```

Alternativement, on peut utiliser la commande `plt.axis('equal')` pour rendre le repère orthonormé.

3. Tracés multiples sur un même graphique

Pour effectuer plusieurs tracés sur un même graphique, on dispose de deux possibilités.

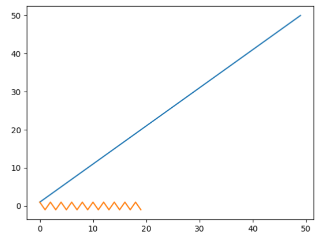
Première possibilité

On peut simplement appeler plusieurs fois la commande `plt.plot` avant la demande d'affichage `plt.show()`, comme dans l'exemple ci-dessous.

Exemple 5. Ce code trace sur un même graphique les 50 premiers termes de la suite u donnée par $u_n = n+1$ et les 20 premiers termes de la suite v donnée par $v_n = (-1)^n$.

```
import matplotlib.pyplot as plt
# Définition des listes
listeX1=range(50)
listeY1=[n+1 for n in listeX1]
listeX2=range(20)
listeY2=[(-1)**n for n in listeX2]

# Tracés
plt.plot(listeX1,listeY1)
plt.plot(listeX2,listeY2)

# Affichage
plt.show()
```

Et le résultat :



Remarque. Cette première syntaxe se généralise à un nombre quelconque de tracés.

Seconde possibilité

Pour tracer sur un même graphique les points donnés par des listes `listeX1, listeY1` d'un côté, et `listeX2, listeY2` de l'autre, on peut utiliser la syntaxe suivante :
`plt.plot(listeX1,listeY1,...,listeX2,listeY2,...)`
où les points de suspensions représentent les options de styles (légende et codes pour les marqueurs et la couleur) spécifiées pour chaque graphique.

Exemple 6. Ce code trace sur un même graphique les 40 premiers termes des suites u et v données par $u_n = n^2+1$ et $v_n = n^n$, les termes de u étant données par des petites croix et les termes de v par des diamants.

```
# Définition des listes
listeX=range(40)
listeY1=[n**2+1 for n in listeX]
listeY2=[n**n for n in listeX]

# Tracé
plt.plot(listeX,listeY1,"x",listeX,listeY2,"D")

# Affichage
plt.show()
```

Remarque. Cette seconde syntaxe est moins pratique et je vous conseille la première, mais vous devez être capable de la lire.

4. Bilan des deux derniers parties

Voici un code Python permettant de tracer, sur un même graphique, les 20 premiers termes des suites u, v et w définies sur $\mathbb{N}$ par :

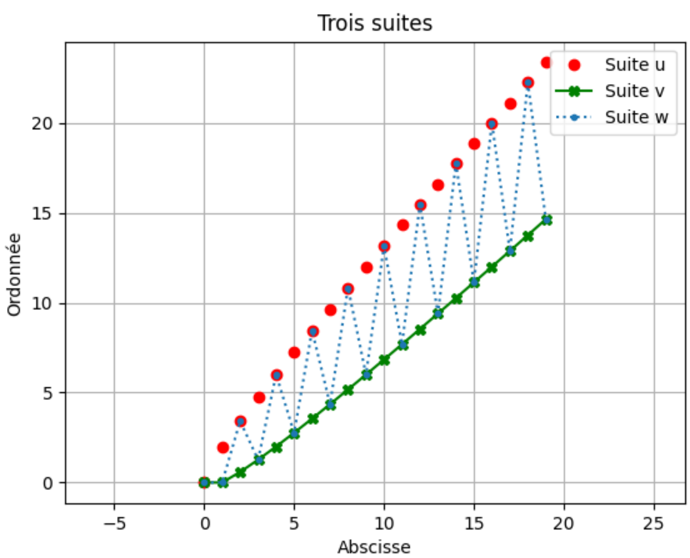
$$\begin{cases} u_n = n + \sqrt{n} \\ v_n = n - \sqrt{n} \\ w_n = n + (-1)^n \sqrt{n} \end{cases}$$

avec des options de personnalisation permettant de rendre le graphique lisible.

Remarque. Toutes les options de personnalisation sont à mettre entre le dernier appel de `plt.plot` et `plt.show()`.

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

# Listes des abscisses et ordonnées
listeX = range(20) # entiers de 0 à 19

listeU=[n + np.sqrt(n) for n in listeX] # ordonnées pour u
listeV=[n - np.sqrt(n) for n in listeX] # ordonnées pour v
listeW = [n + np.sqrt(n)*(-1)**n for n in listeX] # ordonnées pour w

# Tracés avec légende
plt.plot(listeX,listeU,"or",label="Suite u") # u avec gros points rouges
plt.plot(listeX,listeV,"X-g",label="Suite v") # v avec grosses croix vertes reliées par un trait
plt.plot(listeX,listeW,".:",label="Suite w") # w avec des petits points reliés par pointillés

# Axes et légendes

plt.title("Trois suites")
plt.xlabel("Abscisse")
plt.ylabel("Ordonnée")
plt.grid() # ajout d'un quadrillage
plt.axis('equal') # axes orthonormés

# affichages
plt.legend()
plt.show()
```

Et le résultat :



II. Représentation graphique des suites réelles

Pour représenter une suite réelle $(u_n)_{n\in\mathbb{N}}$ (disons ici, ses 101 premiers termes), on peut simplement :

- Construire une liste `listeX` formée des entiers de 0 à 100 dans l'ordre croissant,
- construire la liste `listeY` qui est $[u_0, u_1, ..., u_{100}]$, puis
- utiliser `plt.plot(listeX,listeY)`.

On trace ainsi une ligne brisée passant par les points de coordonnées $(0, u_0)$, $(1, u_1)$, $(2, u_2)$, ..., $(100, u_{100})$ ce qui correspond bien à nos attentes.

1. Suites définies explicitement

Une suite définie explicitement est une suite u dont le terme u_n est donné par une formule explicite en fonction de n.

On peut alors assez simplement tracer les premiers termes de cette suite.

Par exemple, ce code trace les 75 premiers termes de la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_n = \dfrac{3n}{n^2+1}$:

```
import matplotlib.pyplot as plt
listeX=range(75)
listeY=[3*n/(n**2+1) for n in listeX]
plt.plot(listeX,listeY)
plt.show()
```

Il faut être attentif aux premiers termes...

Exercice 7. Compléter le code ci-dessous pour qu'il affiche les 75 premiers termes de la suite u définie sur $\mathbb{N}^*$ par $u_n = \dfrac{3}{n}$.

```
import matplotlib.pyplot as plt
listeX=range(...,...)

listeY=[   ...    for n in    ...  ]
plt.plot(listeX,listeY)
plt.show()
```

On peut également (et c'est conseillé) construire en amont une fonction donnant le terme u_n en fonction de n.

Par exemple, ce code trace les 50 premiers termes de la suite u définie sur $\mathbb{N}$ par $u_n = \exp(n+1) - \dfrac{\ln(n)}{\sqrt{n+1}}$.

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

def U(n): # entrée : entier n, sortie : n ième terme de u
    return( np.exp(n+1)-np.log(n)/np.sqrt(n+1) )

listeX=range(50)
listeY=[U(n) for n in listeX]

plt.plot(listeX,listeY)
plt.show()
```

Exercice 8. Soit $(u_n)_{n\in\mathbb{N}}$ la suite réelle définie par : $\forall n \in \mathbb{N}, u_n = \dfrac{1+(-1)^n}{2}$. Représenter graphiquement les valeurs de u_n pour $n \in [\![0, 40]\!]$. Que remarquez-vous? En déduire une autre expression de u_n en fonction de n.

Exercice 9. Soient u et v les suites réelles définies sur $\mathbb{N}$ par :

$$\forall n \in \mathbb{N}, \begin{cases} u_n = n \\ v_n = 2\lfloor \frac{n}{2} \rfloor \end{cases}.$$

Représenter sur un même graphique les valeurs $(u_n)_{n\in[\![0,20]\!]}$ et $(v_n)_{n\in[\![0,20]\!]}$. Quelle propriété mathématique est illustrée par cette figure ?

2. Suites définies par une relation de récurrence

Pour tracer les termes d'une suite $(u_n)_{n\in\mathbb{N}}$ définie par récurrence, on pourrait aussi écrire une fonction d'entête `def U(n):` prenant en entrée un entier `n` et renvoyant en sortie $u_{\mathtt{n}}$. On pourrait alors utiliser cette fonction pour former, par exemple, la liste $[u_0, u_1, ..., u_{50}]$ si on désire tracer les valeurs de u_n pour $n \in [\![0, 50]\!]$.

Cette méthode est trop inefficace pour être conservée.

On préférera définir une fonction `termesU` renvoyant directement la liste des termes dont on a besoin pour construire le graphique, selon les méthodes du TP précédent.

Exemple 10. Considérons la suite u définie par la relation de récurrence :

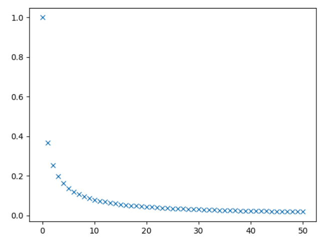
$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = u_n e^{-u_n} \end{cases}.$$

Alors, la fonction `TermesU` définie par le code suivant prends en entrée un entier naturel `n` et renvoie en sortie la liste $[u_0, u_1, ..., u_{\mathtt{n}}]$.

```
import numpy as np # Pour l'exponentielle

def TermesU(n):
    L = [1] # L contiendra la liste voulue, initialisée avec u(0)
    for k in range(n): # faire n fois
        L.append( L[-1]*np.exp(-L[-1]) ) # ajout du terme suivant
    return(L)
```

À la suite de ce code, on peut alors tracer les termes $(u_n)_{n\in[\![0,50]\!]}$ avec le code suivant :

```
import matplotlib.pyplot as plt #import

listeX=range(51) # Entiers de 0 à 50
listeY=TermesU(50) # Termes u(n) pour n entre 0 et 50
plt.plot(listeX,listeY,"x") # Marquage des points avec des croix, optionnel
plt.show() # Demande d'affichage
```



Exercice 11. Considérons la suite u donnée par $\begin{cases} u_0 = 0 \\ \forall n \in \mathbb{N}, u_{n+1} = \dfrac{1}{1+u_n} \end{cases}$. Représenter graphiquement les termes u_n pour $n \in [\![0, 20]\!]$. Que conjecturer ?

Il faut éventuellement faire attention aux premiers termes, et si l'entier n est présent dans la relation de récurrence.

Exercice 12. Représenter les 30 premiers termes de la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par $\begin{cases} u_1 = 1 \\ \forall n \in \mathbb{N}^*, u_{n+1} = \dfrac{u_n}{\sqrt{n}} \end{cases}$.

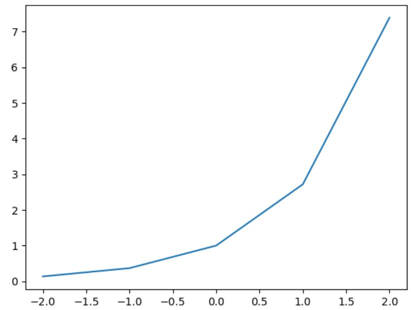
On peut également tracer ainsi des suites définies par une relation de récurrence sur deux rangs.

Exercice 13. Représenter les termes $(u_n)_{n\in[\![0,20]\!]}$ de la suite u définie par $\begin{cases} u_0 = 1, u_1 = 2 \\ \forall n \in \mathbb{N}, u_{n+2} = -u_{n+1} - u_n \end{cases}$.

III. Représentation graphique des fonctions

Voici une première tentative de représentation de la fonction $x \mapsto \mathrm{e}^x$ sur l'intervalle $[-2, 2]$.

```python
import matplotlib.pyplot as plt
import numpy as np
listeX=[-2, -1, 0, 1, 2]
listeY=[np.exp(x) for x in listeX]
plt.plot(listeX,listeY)
plt.show()
```



Ceci approche bien le graphe de l'exponentielle, les 5 points placés sont bien des points de son graphe, mais l'approximation *en ligne brisée* est visuellement bien trop grossière.

Pour adapter cette méthode et avoir un graphe convenable, il suffit de placer (beaucoup) plus de points, *jusqu'à ce que la ligne brisée ne soit plus vraiment visible.* On dispose pour cela de fonctions du module `numpy`, qui se chargent de générer une grande liste d'abscisses pour effectuer facilement nos tracés.

1. Les fonction `linspace` et `arange` de `numpy`

On suppose dans cette partie que numpy est importé avec : `import numpy as np.`

La fonction `np.linspace`

Soient `a` et `b` deux réels tels que `a < b`, et `N` un entier naturel supérieur à 2. Alors, la commande

`np.linspace(a, b, N)`

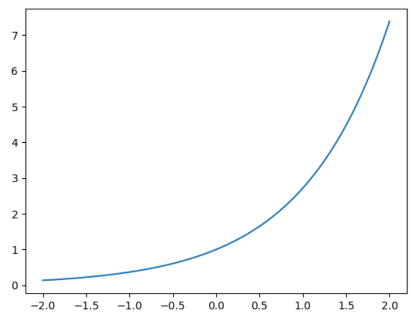
génère la liste des `N+1` réels de `a` à `b`, régulièrement répartis "tous les $\dfrac{\mathtt{b}-\mathtt{a}}{\mathtt{N}}$".

Exemple 14. `np.linspace(0, 1, 4)` génère la liste `[0., 0.25, 0.5, 0.75, 1.]`. On a bien 5 réels de $[0, 1]$, à partir de 0, régulièrement répartis.

Remarque. Ces réels sont les termes $u_0, \ldots, u_{\mathtt{N}}$ de l'unique suite arithmétique u telle que $u_0 = \mathtt{a}$ et $U_{\mathtt{N}} = \mathtt{b}$. La raison de cette suite est $\dfrac{\mathtt{b}-\mathtt{a}}{\mathtt{N}}$.

On utilise très souvent cette commande pour avoir une liste d'abscisse convenable pour le tracé de graphe de fonctions. Par exemple, avec 100 points de -2 à 2 pour le graphe de l'exponentielle, on ne voit déjà plus que notre tracé n'est qu'une ligne brisée.

```python
import matplotlib.pyplot as plt
import numpy as np
listeX=np.linspace(-2, 2, 100)
listeY=[np.exp(x) for x in listeX]
plt.plot(listeX,listeY)
plt.show()
```



La fonction np.arange

Soient `a` et `b` deux réels tels que `a < b`, et `p` un réel. Alors, la commande

`np.arange(a,b,p)`

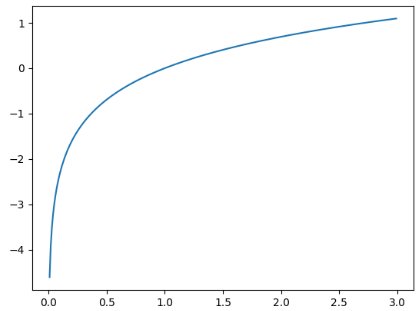
génère la liste `[a, a+p, a+2*p, a+3*p,...]` des points obtenus à partir de `a` en ajoutant progressivement `p` tout en restant strictement inférieur à `b`.

Exemple 15. `np.arange(0, 2, 0.1)` génère la liste `[0, 0.1, 0.2, 0.3,..., 1.8, 1.9]`.

On peut donc également utiliser `np.arange` pour avoir une liste donnant beaucoup d'abscisses pour les tracés de graphes de fonctions : au lieu de spécifier le nombre de points qu'on veut (comme avec `np.linspace`), on spécifie l'espacement entre ceux-ci.

Par exemple, pour tracer le graphe du logarithme sur $]0,3]$, on peut écrire le code suivant (on commence à 0.001, et on place un point tous les 0.01) :

```
import matplotlib.pyplot as plt
import numpy as np
listeX=np.arange(0.001, 3, 0.01)
listeY=[np.log(x) for x in listeX]
plt.plot(listeX,listeY)
plt.show()
```



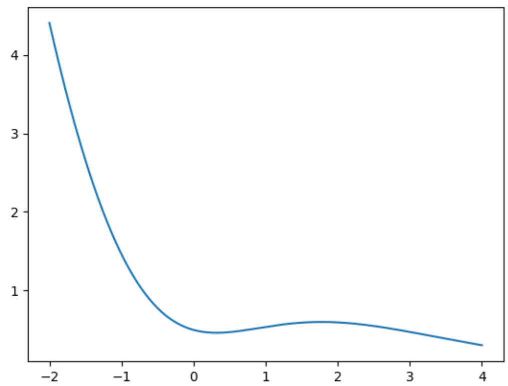
2. Représenter une fonction

Une fois le point précédent clair, la représentation de fonctions f devient assez simple.

- On définit une fonction Python `f` prenant en entrée un réel x et renvoyant $f(x)$,
- on génère une liste `listeX` des abscisses avec `np.linspace` ou `np.arange`,
- on génère la liste `listeY` des ordonnées avec la commande :
  `listeY = [ f(x) for x in listeX ]`
- et on utilise les commandes `plt.plot(listeX,listeY)` et `plt.show()`.

Exemple 16. Le code suivant affiche une représentation graphique la fonction $x \mapsto \dfrac{x^2+1}{e^x+1}$ sur $[-2,4]$.

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

# Définition de la fonction f
def f(x):
    return( (x**2+1)/(np.exp(x)+1) )

# Génération des listes
listeX=np.linspace(-2,4,500) # 500 points
listeY=[ f(x) for x in listeX ]

# Tracé et affichage
plt.plot(listeX,listeY)
plt.show()
```



Remarque. On peut avoir à faire quelques ajustements pour tracer des fonctions sur des domaines qui ne sont pas des segments.

- Pour une représentation sur un intervalle avec une (ou des) borne ouverte comme $]0,1]$, on commence le tracé à un point suffisamment proche de la (ou des) borne concernée, comme $[0.01,1]$.
- Pour une représentation sur un intervalle non borné (comme $\mathbb{R}_+$), on choisit un segment *suffisamment grand* sur lequel effectuer le tracé (comme $[0,100]$ ou $[0,1000]$).
- Pour une représentation sur un domaine qui n'est qu'une réunion d'intervalles (comme $[-2,-1] \cup [1,2]$), on effectue un tracé par intervalle intervenant dans cette réunion, et on met ces tracés sur un même graphique.

Exercice 17. Représenter sur un même graphique les courbes des fonctions $x \mapsto 1+x$, $x \mapsto e^x$ et $x \mapsto 1+xe^x$, sur l'intervalle $[-2,2]$ et avec un quadrillage. On ajoutera une légende pour identifier ces courbes. Quelle propriété mathématique est représentée par cette figure ?

Exercice 18. Représenter avec Python la fonction signe : $x \mapsto \begin{cases} 1 \text{ si } x>0 \\ 0 \text{ si } x=0 \\ -1 \text{ si } x<0 \end{cases}$ sur l'intervalle $[-3,3]$.

Exercice 19. Considérons les fonctions définies sur $\mathbb{R}$ par $f : x \mapsto x^3-3x-1$ et $g : x \mapsto |f(x)|$. Représenter sur une même figure les courbes de f et g sur l'intervalle $[-3,3]$, avec une légende, un quadrillage, des pointillés pour la courbe de f et un trait plein pour la courbe de g.

Exercice 20. Soit f la fonction définie sur $\mathbb{R}$ par $f(x)=2x^3-5x^2+x-5$.

f est dérivable en tant que polynôme, le but de cet exercice est de représenter sur un même graphique les courbes de f et de f' et ce, sans calculer f' à la main.

On rappelle que par définition, pour tout réel x :

$$f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}.$$

Cette limite justifie *l'approximation* $f'(x) \simeq \dfrac{f(x+h)-f(x)}{h}$, pour une valeur très petite de h.

A l'aide de cette approximation pour $h = 10^{-6}$, tracer les courbes de f et f' sur un même graphique, sur l'intervalle $[-2,4]$, avec des légendes pour distinguer les courbes.

L'intérêt de cette méthode est qu'on peut représenter une dérivée sans savoir la calculer explicitement.

IV. Un exercice de synthèse sur le problème de Syracuse

Soit A un entier naturel. On rappelle que la *suite de Syracuse de paramètre* A est la suite u définie par la relation de récurrence :

$$\begin{cases} u_0 = A \\ \forall n \in \mathbb{N}, u_{n+1} = \begin{cases} \dfrac{u_n}{2} \text{ si } n \text{ est pair} \\ 3u_n + 1 \text{ sinon} \end{cases} \end{cases}.$$

On remarque que si, pour un certain n_0, on a $u_{n_0} = 1$, alors les termes suivants de la suite $u_{n_0+1}, u_{n_0+2}, ...$ sont indéfiniment $4, 2, 1, 4, 2, 1....$

La conjecture de Syracuse (1937), non résolue a ce jour (2024), prédit que quelque soit la valeur de A, la suite de Syracuse de paramètre A termine par ce cycle $4, 2, 1, ...$

Exercice 21. Toutes ces questions sont bien sûr à faire en Python.

1. Écrire une fonction d'entête `def Syracuse(A,n)` prenant en entrée deux entiers naturels `A` et `n` et renvoyant en sortie la liste $[u_0, u_1, .., u_{\mathtt{n}}]$ des $\mathtt{n}+1$ premiers termes de la suite de Syracuse de paramètre `A`.
2. Représenter graphiquement les n premiers termes de la suite de Syracuse de paramètre A dans les cas suivants :

   - $A = 15$ et $n = 25$,
   - $A = 9$ et $n = 25$,
   - $A = 27$ et $n = 100$.

   On pourra chercher à automatiser ce processus.

Soit A un entier naturel et $(u_n)_n$ est la suite de Syracuse de paramètre A.

- On appelle *durée de vol* du paramètre A l'entier $d_A = \min\{n \in \mathbb{N}, u_n = 1\}$.
- On appelle *altitude maximal de vol* du paramètre A la valeur $M_A = \max(u_0, u_1, ..., u_{d_A})$.

3. Déterminer la durée de vol et l'altitude maximal de vol des paramètres $A = 3$ et $A = 15$.
4. Écrire une fonction d'entête `def Vol(A):` prenant en entrée un entier naturel `A` et renvoyant en sortie le couple $(d_{\mathtt{A}}, M_{\mathtt{A}})$ formé par la durée de vol et l'altitude de vol du paramètre `A`. Tester cette fonction pour $A \in \{3, 15, 27\}$.
5. Déterminer un paramètre A_0 ayant une durée de vol maximale parmi les paramètres A tels que $A \leq 1000$, ainsi que son altitude de vol correspondante.
6. Représenter les termes $u_0, u_1, ..., u_{d_{A_0}}$ de la suite de Syracuse de paramètre A_0.
7. Déterminer un paramètre A_1 ayant une altitude de vol maximale parmi les paramètres A tels que $A \leq 1000$, ainsi que sa durée de vol correspondante.
8. Représenter les termes $u_0, u_1, ..., u_{d_{A_1}}$ de la suite de Syracuse de paramètre A_1.
9. Déterminer le nombre de suites de Syracuse dont la durée de vol est inférieur à 20, et lister les paramètres correspondants.
10. Déterminer le nombre de suites de Syracuse dont l'altitude maximale est inférieur à 100, et lister les paramètres correspondants.

V. Annexe

1. Personnalisation des lignes et des points

Options de personnalisation des marqueur de points :

Code	.	o	v ou ˆ ou > ou <	s	*	+
Points	petit point	gros point	triangle	carré	étoile	plus

Code	P	x	X	p	h	D
Points	gros plus	croix	grosse croix	pentagone	hexagone	diamants

Options de personnalisation des lignes :

Code	−	−−	-.	:
Ligne	trait plein	tirets	alternance tiret-point	pointillés

Options de personnalisation de la couleur :

Code	w	k	r	g	b	c	m	y
Couleur	blanc	noir	rouge	vert	bleu	cyan	magenta	jaune