Sébastien Peronno

40 thèmes illustrés par le numérique pour l'agrégation interne de mathématiques

Développements, exercices et figures avec Python, Geogebra et LibreOffice

Oral
Agrégation
interne

ellipses

Références sciences

40 thèmes illustrés par le numérique pour l'agrégation interne de mathématiques

Développements, exercices et figures avec Python, Geogebra et LibreOffice

Sébastien Peronno

Oral
Agrégation
interne

ellipses

Collection Références sciences

dirigée par Paul de Laboulaye
paul.delaboulaye@editions-ellipses.fr

Retrouvez tous les livres de la collection et des extraits sur www.editions-ellipses.fr

ISBN : 9782340113725
dépôt légal : mars 2026

8/10 rue la Quintinie 75015 Paris

www.editions-ellipses.fr

Avant-propos

En 2025, j'ai obtenu l'agrégation interne de mathématiques après avoir suivi la formation dédiée à l'université de Paris Cité.

Durant ma préparation, j'ai souvent exploré des applications informatiques que ce soit pour créer des figures, programmer des algorithmes, tester des concepts théoriques, visualiser des courbes ou simuler des lois de probabilité.

Ce livre, destiné aux futurs agrégatifs, est avant tout le partage de ce travail.

Le rapport du jury 2024 note :

« L'enseignement des mathématiques nécessite l'utilisation d'outils informatiques, qu'il s'agisse de logiciels prêts à l'emploi ou d'algorithmes résolvant des problèmes de manière effective. Certaines tâches techniques (...) sont facilitées par des logiciels spécialisés et certains logiciels interviennent couramment comme outils pédagogiques.(...)

« L'usage pertinent des outils numériques pour illustrer ou clarifier un thème est à favoriser et est quasiment inévitable pour le traitement de certains sujets qui énoncent des aspects algorithmiques ou des calculs approchées d'intégrales (...) Dans les thèmes de probabilités, il serait agréable de voir des propositions de simulation informatique de variables aléatoires réelles, des calculs approchés de probabilités ou d'espérances. »

J'ai tenté de doser correctement la place accordée à la partie numérique selon le sujet abordé. Par exemple, pour les théorèmes d'approximation de Weierstrass ou de réarrangement de Riemann, d'intérêt théorique, on se contente d'un petit script qui illustre concrètement le résultat. Pour le chiffrement RSA, les séries de Fourier ou les statistiques, dans lesquels l'informatique est importante, une analyse détaillée est fournie.

J'espère que vous piocherez dans ces pages une matière utile et stimulante et vous souhaite pleine réussite dans votre belle aventure.

Je tiens à remercier chaleureusement toutes les personnes qui ont relu des parties du livre, me permettant d'éliminer des coquilles, reformuler des imprécisions et corriger des erreurs de calculs.

M'ont ainsi aidé mes anciens professeurs de Paris Cité, Thierry et Gentiana, ma camarade de promo Cindy, Gilles de la chaine Maths Adultes, Alex & Eve de la chaine Thomaths, Laurent en bon connaisseur de LaTeX, mon collègue Jérémie, Louis-Clément du lycée Hoche, Damien et Annie de Marolles-les-Buis.

Une dédicace aux anciens du campus de Versailles : Karim alias Turtle (le maitre du go spécialisé dans la coinche), Louis (l'explorateur des dimensions de Hausdorff), Guilhem (l'humanitaire agrégé) et Vincent (gloire à lui).

Une salutation collective à la promo 2024-2025 de préparation à l'agrégation interne de Paris Cité, portée par Catherine, qui m'a permis de garder constante ma motivation à préparer le concours, notamment grâce au travail avec l'équipe Triple A (Alexia-Amandine-Aurélie). Le titre du livre a été choisi par la promo.

Enfin, une pensée à mes parents Pascal et Marie-Jo, mon frère Dimitri et ma grand-mère Rachel qui m'accompagnent dans tous mes projets de vie, divers et variés.

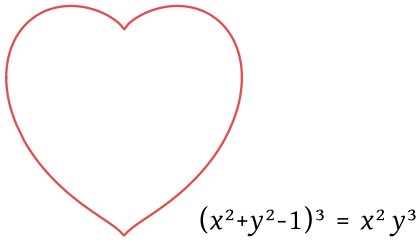


Table des matières

1

Introduction

1.1 Présentation générale

On trouve dans cet ouvrage une quarantaine de thèmes, couvrant une large partie du programme de l'agrégation interne. Leur dénominateur commun est que l'on en présente un aspect numérique.

Le pictogramme ❒ indique une illustration. Selon le cas, celle-ci peut être un simple visuel pour animer une leçon orale ou bien représenter un important théorème mais documenté ailleurs.

Les exercices sont indiqués par ✍. Ils fournissent des exemples d'applications pour l'oral 1 et de la matière pour l'oral 2. Ils sont à présenter uniquement durant la première phase de chacun des examens.

Enfin, les développements sont repérés par ❋. Pour ces derniers, des explications complètes sont données, accompagnées de ressources extérieures.

Les bases du tableur, une familiarité avec Geogebra et les notions fondamentales d'algorithmique (boucles, conditions...) sont présupposées connues.

Concernant spécifiquement Python, il est possible de prendre en main cet ouvrage en étant (faux) débutant. Cependant, le but n'est pas d'enseigner le langage mais de fournir des programmes lisibles, efficaces et utilisables pour accompagner les apprentissages puis être présentés aux épreuves orales.

Pour aller au plus simple, les scripts proposés sont prêts à l'emploi. Nous n'abordons ni la création d'interface graphique ni la structure orientée objet. Tous les scripts sont téléchargeables sur `joliesmaths.fr` et sur le site d'Ellipses.

1.2 Préparer l'agrégation interne

L'agrégation interne est un concours exigeant, qui se prépare souvent sur plusieurs années. Elle nous remet dans la position d'apprenant, celle de l'élève titubant entre savoirs mal consolidés et exploration de champs inconnus.

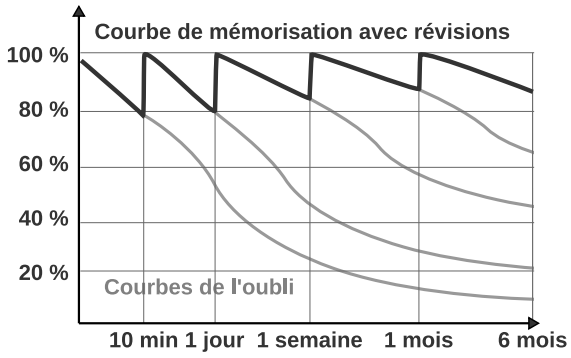
En tant qu'enseignant, c'est une belle opportunité pour prendre du recul sur sa propre pratique. Je vous encouragerais à avoir une réflexion méthodologique dès le lancement de votre projet. Un bon point d'entrée est la vidéo de David Louapre (Science étonnante) intitulée *« Mieux apprendre et étudier : les (vraies) techniques scientifiques*[1] *»*.

Une idée importante est celle de la **courbe de l'oubli**. Lorsque l'on a travaillé et même, semble-t-il, bien compris une nouvelle notion, on l'oublie si on ne la ré-emploie pas. C'est le fonctionnement normal du cerveau.

L'assimilation au long terme passe par plusieurs phases de répétitions espacées. Il existe des méthodes pour casser la courbe de l'oubli. Pour ma part, j'utilisais une *boite de Leitner* que m'avait offert un gentil collègue.

Il s'agit d'une boite avec plusieurs compartiments selon le degré de maitrise d'un sujet. Au fur et à mesure des apprentissages, on se crée un jeu de fiches, avec d'un côté une question du type « Démontrer l'inégalité de Cauchy-Schwarz » et de l'autre la réponse (ou l'idée principale). On se la donne ensuite à refaire le lendemain.

Ma boite de Leitner



Courbe d'apprentissage
(inspirée par Ebbinghaus)

1. `https://www.youtube.com/watch?v=RVB3PBPxMWg` – septembre 2023

Si le sujet est assimilé, on se le redonne trois jours plus tard et s'il ne l'est pas, de nouveau le lendemain. Plus le sujet est maitrisé, plus on espace sa révision : une semaine, un mois, trois mois.

Pour ne pas multiplier les fiches, cette technique est utilisée plutôt pour assimiler les fondamentaux : définitions de bases, théorèmes les plus utiles, exercices types et contre-exemples usuels.

Il est important de noter **qu'il ne s'agit pas de fiches synthèses**. Celles-ci, hélas largement utilisées, consistent à résumer le cours et à le relire inlassablement : il a été scientifiquement montré qu'elles sont inefficaces [2]. Le cerveau retient s'il est activement mobilisé ; la lecture simple est trop passive.

Une camarade de promo utilisait le logiciel libre **Anki**. Il permet de créer des cartes-mémoires et d'organiser son apprentissage par répétitions espacées. Le principe est le même que la boite de Leitner, avec les sophistications du numérique.

Quelle que soit la méthode utilisée, l'important est de connaitre au mieux son cerveau et de ne pas se mentir à soi-même. La recopie simple d'une démonstration ne vaut rien si l'on n'est pas capable de la refaire une semaine plus tard sans regarder.

Aussi, lorsque l'on a trop travaillé et que le cerveau est surchargé, il est inutile de le charger encore davantage. On fait une pause : promenade, sport, méditation, jeu vidéo, peu importe, de quoi laisser le cerveau assimiler en arrière-plan. On n'apprend pas nécessairement mieux en y passant plus de temps.

Enfin, le passage de l'agrégation est une belle aventure intellectuelle. Une condition nécessaire de réussite, bien que non suffisante, est d'aimer faire des mathématiques.

1.3 Faire l'impasse sur l'informatique ?

Préparer l'agrégation interne nécessite, dans la pratique, de faire des choix de sujets qui seront plus ou moins travaillés.

Apprendre à programmer en Python lorsque l'on ne connait pas le langage peut apparaitre comme « non rentable » en terme de temps investi. Certaines

2. Voir les travaux modernes de sciences cognitives sur l'apprentissage et l'héritage de Hermann Ebbinghaus.

formations universitaires font elles-mêmes l'impasse sur l'informatique.

C'est un choix personnel de fixer ses priorités. En tant qu'auteur d'un livre dédié, je souhaiterai cependant apporter des arguments en faveur de son usage.

- Les rapports de jury notent l'importance voire la nécessité des outils numériques pour certains sujets.
- Les algorithmes *décrits* (comme RSA, LU, Bézout...) sont mieux assimilés s'ils sont par ailleurs *programmés*.
- Faire vivre les mathématiques au-delà du marqueur noir sur tableau blanc est la preuve d'un attachement à la matière et d'une maitrise pédagogique qui ne peut qu'être valorisée.
- De nombreuses applications (notamment avec Geogebra) sont très simples et très rapides. En parfois quelques minutes, il est possible d'avoir un rendu utile et de qualité.

Je me permettrai ici de donner mon témoignage personnel. En oral 1, je tombe sur deux leçons peu préparées et je choisis « 128 – Groupe orthogonal d'un espace vectoriel euclidien de dimension 2, de dimension 3 ». J'ai en effet déjà travaillé un développement « recasable » sur les isométries du cube, pris dans [KARM].

Pendant les dernières minutes des trois heures de préparation, je décide de tracer sur Geogebra 3D un simple cube avec quelques segments, peu convaincu que cela ait un intérêt. Au final, celui-ci se révèlera un point d'appui, utile à la fois pour mon développement et pour une question du jury (« comment caractériser les isométries autrement que par les diagonales ? »). L'écran tactile permettant de tourner dans l'espace, le rendu en sera d'autant plus élégant.

En oral 2 pour « 203 – Séries à termes réels ou complexes : convergence absolue, semi-convergence », je prépare un algorithme en Python qui applique le théorème de réarrangement de Riemann à la série harmonique alternée (voir graphique page 96). Ayant oublié mon chronomètre, je gère mal mon temps pour terminer correctement la démonstration. Le graphique m'aura au moins permis d'en présenter le dernier argument à l'oral, tout en réalisant une *preuve de concept*.

Ayant été admis « en queue de peloton », il est possible que ces deux applications numériques aient fait la petite différence nécessaire.

2

Outils numériques

2.1 Python : prise en main express

2.1.1 Variables

Les variables en Python sont notées avec des lettres non accentuées, des chiffres et le symbole _. Les mots réservés aux fonctions (`for`, `while`, etc.) ne doivent pas être utilisés. Majuscules et minuscules ne sont pas équivalentes (« sensibilité à la casse ») : attention à ne pas confondre `u` et `U`.

Les variables sont de différents types : décimal (*float*), entier (*integer*), tableau (*array*), booléen (*boolean*), chaine de caractères (*string*), t-uplet (*tuple*), etc. Selon le type, une même opération n'a pas la même fonction.

`d = 4.5` est un décimal et `d*2` renvoie `9`. En revanche, `s = "4.5"` avec les guillemets est une chaine de caractères et `s*2` renvoie `'4.54.5'`

2.1.2 Opérations mathématiques

Les opérations suivantes sont utilisables directement :

Division décimale	Division euclidienne	Reste modulaire	Puissance
`32/5`	`32//5`	`32%5`	`3**4`
`6.4`	`6`	`2`	`81`

Racine n-ième	Arrondi à l'unité	Arrondi au millième	Valeur abs.
`27**(1/3)`	`round(5.2148)`	`round(5.2148,3)`	`abs(-7)`
`3.0`	`5`	`5`	`7`

Ces fonctions sont utilisables **après** avoir saisi `from math import *`

Racine	ln	log base 10	Factorielle
`sqrt(5)`	`log(e)`	`log(10000, 10)`	`factorial(5)`
`2.236...`	`1.0`	4	120

Cosinus	Sinus	Tangente	Arccos
`cos(pi/4)`	`sin(pi/2)`	`tan(0)`	`acos(1/2)`
`0.707...`	`1.0`	`0.0`	`1.047...`

En raison de limites liées à la représentation des décimaux, certaines opérations ne donnent pas les valeurs exactes attendues :

- `9 + 0.1 + 0.1 + 0.1 - 0.3` renvoie `8.999999999999998`
- `log(1000, 10)` renvoie `2.9999999999999996`

2.1.3 Opérations sur les tableaux

Un tableau se déclare avec des crochets : `V = [5, 8, 10, 13]`

Les indices commencent à 0, ce qui prête parfois à confusion. `V[1]` renvoie donc `8`. De plus, `V[-1]` renvoie `13` et `len(V)` renvoie 4 (taille du tableau).

Pour ajouter le nombre 18 à la fin, on écrit `V += [18]` ou `V.append(18)`

De base, les tableaux en Python ne sont pas considérés comme des matrices, ce qui fait que les opérations `+` et `*` ne sont pas celles mathématiques.

`V*2` donne `[5, 8, 10, 13, 5, 8, 10, 13]`

On charge généralement un module dédié en début de programme :

`import numpy as np` (voir page 57 pour plus de détails)

On déclare `A = np.array([[1,4],[5,7]])` et `A*2` renvoie bien $2 \cdot A$

2.1.4 Instructions

Observez ces exemples : `range(n)` renvoie tous les nombres de 0 à $n-1$

Exécution du code

```
>>> [k**2 for k in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
>>> [2*k+1 for k in range(5,12)]
[11, 13, 15, 17, 19, 21, 23]
```

L'instruction `a = a + 10` ajoute 10 à la variable `a` et s'écrit aussi `a += 10`

La commande `for` se termine par « : » et toutes les instructions répétées doivent être décalées avec la touche `TAB` du clavier (elles sont dites ***indentées***).

Le script suivant affiche **uniquement** la valeur finale de $s = \sum_{k=1}^{10} \frac{1}{k}$ car le `print` ne fait pas partie de la boucle. Mais si on aligne la ligne 4 comme la ligne 3, on affiche les 10 sommes partielles. Testez.

Somme des 10 premiers inverses

```
1 s = 0
2 for k in range(1, 11):
3     s += 1/k
4 print(s)
```

Lorsqu'on utilise une condition de type `if(a==2):` ou `while(n<200):`, les instructions doivent être bien indentées en-dessous. Attention aussi à la confusion courante entre « == » (test d'égalité) et « = » (affectation de valeur).

Les comparateurs `<=`, `>=` et `!=` correspondent respectivement à $\leqslant$, $\geqslant$ et $\neq$.

2.1.5 Fonctions

Une fonction est créée avec `def nom_fonction(arguments):` et se termine par `return`. Toutes les lignes de sa définition doivent être indentées pareillement.

Voici les couples $(x, f(x))$ pour $f(x) = x^2 - 3x + 2$ et $x \in [\![-5,5]\!]$:

fonction.py

```
1 def f(x):
2     return x**2 - 3*x + 2
3
4 for x in range(-5, 6):
5     print((x, f(x)))
```

Découvrons maintenant un premier exemple complet, utilisable aux oraux.

2.2 Premier exemple : la suite de Syracuse ✍

2.2.1 Exercice

On définit par récurrence la « suite de Syracuse de premier terme u_0 » par

$$u_0 \in \mathbb{N}^* \text{ et } u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

On appelle, sous réserve d'existence :

- **temps de vol :** le plus petit indice N tel que $u_N = 1$.
- **altitude maximale :** le plus grand entier atteint par la suite, *i.e.* si $u_0 \notin \{1\ ;2\}$, $\max\{u_n, n \in [\![0, N]\!]\}$.

1. Que peut-on dire de la suite pour $u_0 = 1$?
2. Montrer que pour tout u_0 de $\mathbb{N}^*$, la suite est divergente.
3. On utilise un programme Python.
   a. Montrer que pour tout $u_0 \in [\![1{,}100]\!]$, le temps de vol est fini et construire le graphique des temps de vol sur cet intervalle.
   b. Donner u_0 pour lequel la suite a le plus long temps de vol en précisant la valeur de celui-ci.
4. Dans un nouveau fichier, programmer la suite de Syracuse pour $u_0 = 27$. Le script doit afficher son temps de vol, son altitude maximale et un graphique de ses valeurs jusqu'à $u_N = 1$.

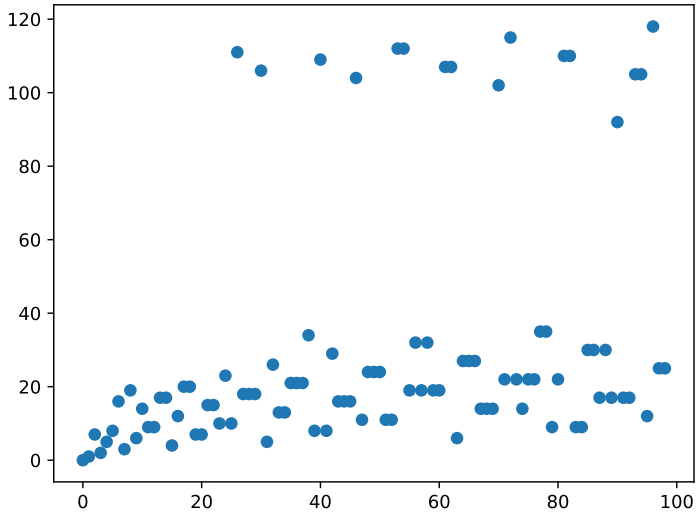
Corrigé

1. Les six premiers termes sont 1, 4, 2, 1, 4 et 2. La suite est divergente. Elle boucle sur un cycle de trois entiers qui constituent des points d'adhérence.
2. Soit $u_0 \in \mathbb{N}^*$. Supposons que u_n converge. Par une récurrence immédiate, on montre que les termes de u_n sont tous des entiers naturels non nuls. Donc la suite $(u_n)_{n\in\mathbb{N}}$ est nécessairement stationnaire.

   Notons n_0 tel que $\forall n \geqslant n_0, u_n = l$. Cet entier l ne pourrait pas être impair car $u_{n_0+1} = l$ serait pair. Donc l doit vérifier $l = \dfrac{l}{2}$, *i.e.* $l = 0$. Or, on a montré que les termes de $(u_n)_{n\in\mathbb{N}}$ sont non nuls. Donc $(u_n)_{n\in\mathbb{N}}$ est divergente.
3. a. On vérifie la propriété par recherche exhaustive.

syracuse-vols.py

```
1  import matplotlib.pyplot as plt
2
3  V = []  # valeurs de la suite
4
5  for k in range(1, 101):
6      t = 0   # temps de vol
7      u = k   # u_0
8      while(u!=1):    # tant que u_n est différent de 1
9          t += 1
10         if(u%2==0): # si u est pair
11             u = u//2
12         else:       # si u est impair
13             u = 3*u + 1
14     V.append(t)
15
16 print(V)
17 M = max(V)
18
19 print("\nLe plus long temps de vol est", M, "pour u_0 =",
   ↪  V.index(M)+1)
20 plt.plot(V, 'o', ls="None")
21 plt.show()
```



Exécution du code

```
[0, 1, 7, 2, 5, 8, 16, 3, 19, 6, 14, 9, 9, 17, 17,
(...)
92, 17, 17, 105, 105, 12, 118, 25, 25, 25]

Le plus long temps de vol est 118 pour u_0 = 97
```

C'est parti pour coder !

Idéalement, installez Python 3 et Pyzo sur votre ordinateur. Copiez le code et exécutez-le. Si l'erreur `ModuleNotFoundError: No module named 'matplotlib'` s'affiche, c'est que l'« extension » `matplotlib` n'est pas installée. Si ce côté informatique vous rebute, il est possible de coder sur un site web en cherchant *"python en ligne avec matplotlib"* dans un moteur de recherche. Passons aux explications...

`import matplotlib.pyplot as plt` charge le module pour tracer le graphique. La syntaxe parait compliquée au début mais à force de l'écrire, vous retiendrez l'instruction.

`V = []` déclare un tableau vide. Le dièse `#` permet d'ajouter un commentaire (non interprété par le langage).

`for k in range(1,101):` démarre une boucle qui va tester tous les entiers de 1 à **100**. Les deux lignes dessous, indentées pareillement, correspondent aux deux variables, réinitialisées à chaque début de boucle.

On compte le temps de vol `t` avec l'instruction conditionnelle `while(u!=1):` qui signifie « tant que u_n est différent de 1 ». On ajoute 1 à `t` à chaque répétition de la boucle.

`if(u%2==0):` signifie « si u_n congru à 0 modulo 2 », autrement dit s'il est pair. Deux lignes plus loin, `else:` (« sinon ») donne l'instruction pour u_n impair. Attention aux alignements et aux deux points en fin de ligne.

`u = u//2` traduit $u_{n+1} = \frac{u_n}{2}$. Il serait possible de noter `u = u/2` mais, en mettant une seule barre oblique, on passe en division décimale et cela affiche un zéro non significatif pour tous les nombres.

`print("\n...")` affiche un message à l'écran, sachant que `\n` désigne un retour à la ligne.

`plt.plot(V, 'o', ls="None")` crée un graphique avec des points (argument `'o'`) non reliés entre eux (argument `ls="None"` pour *linestyle*).

`plt.show()` montre ce graphique à l'écran. Il est ensuite possible de l'enregistrer en cliquant sur l'icône 🖫. Différents formats d'images sont possibles. Privilégiez `SVG` (format vectoriel, sans pixel), sinon `PNG` et `JPG` sont possibles. Sauvegardez l'image : c'est un bon réflexe pour les oraux.

Si vous débutez en Python, tout cela peut sembler un peu dense au premier abord. C'est à l'image du travail à mener à l'agrégation : le démarrage d'un tout nouveau sujet donne le vertige quand on mesure son degré d'ignorance. La persévérance nous fait assimiler progressivement.

Il est à peu près certain que vous devrez corriger plusieurs fois le code avant qu'il ne marche. C'est normal, *no worries*. La « trousse de secours du débogage » est disponible page 19.

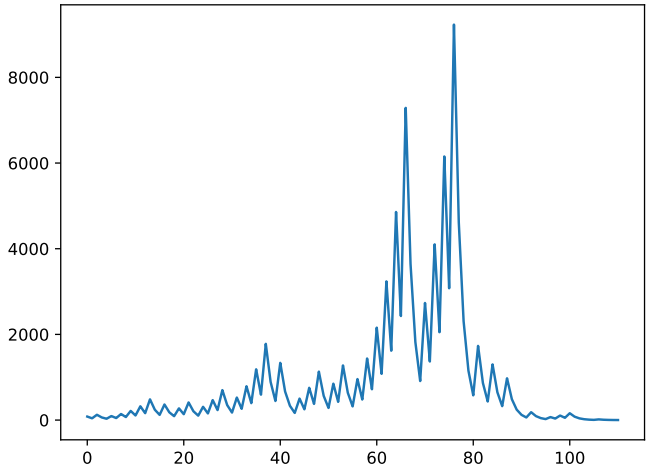
Passons au deuxième code, celui de la question 4.

syracuse27.py

```
import matplotlib.pyplot as plt

u = 27
print("u_0 =", u)

V = [] # valeurs de u

while(u!=1):
    if(u%2==0):
        u = u//2
    else:
        u = 3*u + 1
    V.append(u)

print("Temps de vol =", len(V))
print("Altitude maximale =", max(V))

plt.plot(V)
plt.show()
```

Exécution du code

```
u_0 = 27
Temps de vol = 111
Altitude maximale = 9232
```



2.2.2 Conclusion

Le deuxième code vous a paru sûrement plus compréhensible. Vous voilà entré dans le monde de Python.

Concernant maintenant la suite mathématique étudiée, une question vient : est-il vrai que pour tout entier u_0, la suite a un temps de vol fini ?

L'assertion a été vérifiée pour tout $u_0 < 2{,}36 \times 10^{21}$ mais elle n'a pas été actuellement démontrée sur $\mathbb{N}^*$. Ce problème a été présenté à l'université de Syracuse dans l'État de New York dans les années 1950. Le fameux mathématicien Paul Erdős en a dit : *« les mathématiques ne sont pas encore prêtes pour de tels problèmes »*.

En oral 1, on peut la présenter en quelques mots, illustrée de l'un des deux scripts, comme une ouverture sur l'actualité de la recherche. En oral 2, on peut l'ajouter aux côtés d'exercices plus consistants. Cela montre un intérêt et une culture mathématique. On justifie la présence d'un exercice où le contenu mathématique est plus limité par le caractère intéressant de la démarche : tester une conjecture à l'aide de l'informatique fait désormais partie intégrante de notre discipline.