N. Laillet
nlaillet.math@gmail.com

Préparation à l'oral de Maths-Informatique à Centrale

Table des matières

1 Remarques générales **2**

2 Algèbre **4**
2.1 Algèbre linéaire 4
2.2 Arithmétique et polynômes 5

3 Analyse **7**
3.1 Suites et séries numériques 7
3.2 Suites et séries de fonctions 8
3.3 Intégrales généralisées et intégrales à paramètres 8
3.4 Équations différentielles 9
3.5 Calcul différentiel 9

4 Probabilités **10**

5 Corrigé des exercices **11**

 N. Laillet
nlaillet.math@gmail.com

1 Remarques générales

Le principe À l'oral de Centrale, vous aurez à utiliser python à l'oral de Maths-Informatique (à chaque fois), à l'oral de Physique-Informatique (parfois)

Ces oraux ont **30 minutes de préparation** et **30 minutes de passage**.

L'idée n'est pas d'utiliser python pour faire de l'algorithmique complexe mais, au contraire, pour faire des simulations : il est très peu probable que l'on vous demande de programmer la méthode d'Euler, des trapèzes, que l'on vous demande de faire un tri, etc.

Bien au contraire, il faut utiliser tous les outils disponibles fournis par python. Un aide-mémoire vous sera donné pendant l'oral, avec différentes commandes python : appropriez-vous cet aide-mémoire dès maintenant et ne le découvrez pas le jour de l'oral !

À l'oral de maths, Python doit **servir à conjecturer un résultat**, puis l'oral en tant que tel servira à démontrer cette conjecture. Dans certaines planches, python servira simplement à illustrer ce que vous aurez mathématiquement démontré.

En mathématiques toujours, essayez de ne pas passer plus de **10 minutes** sur le python (quitte à y revenir ensuite) : c'est pourquoi il faut savoir retrouver très rapidement les commandes de l'aide-mémoire. Lors de l'oral, en revanche, si vous avez complètement négligé le python, on vous demandera de le faire en direct, quitte à ce que vous y passiez 15-20 minutes.

Nous allons présenter, sur des points particuliers, avec des exemples d'énoncés, de quelle manière utiliser l'outil informatique.

Conseils généraux et très pratiques Une demi heure pour faire préparation du python + des maths, c'est **très peu**. Voici quelques conseils.

- il est indispensable de **s'approprier** l'aide-mémoire : ne le découvrez pas le jour de l'oral. Il est disponible sur `https://www.concours-centrale-supelec.fr/CentraleSupelec/SujetsOral/MP`
- sachez rapidement exécuter un code python ! Pas besoin de cliquer, mais les raccourcis :
  - `F5` pour tout exécuter
  - `Ctrl+Entrée` pour exécuter le morceau de code délimité par des `##`

  Par exemple si j'ai

```
1  ##Partie 1
2
3  a = 2
4  b = 3
5  print(b)
6
7  ##Partie 2
8
9  a = 230
10 b = 5
11 print(a)
```

  Si je clique sur la ligne 2,3,4,5 ou 6 et que je tape `Ctrl+Entrée`, alors j'aurai la valeur de `b` (3) qui s'affichera. Si je clique sur la ligne 8,9,10 ou 11, alors j'aurai la valeur de `a` (230) qui s'affichera.

- apprenez à **lire vos erreurs en python**. J'ai vu beaucoup de candidats paniquer en disant « ça marche pas », alors que l'erreur python était assez claire (erreur de syntaxe, de out of range, etc.)
- **repérez** lorsque python vous indique une erreur. J'ai vu énormément de candidats dire « il doit y avoir un problème avec python car cela ne me renvoie pas le bon résultat ». Puis, quand je demande d'exécuter le programme, je vois la console de remplir de messages en rouge : il faut s'inquiéter lorsqu'il y a ce genre de message !
- sachez trouver **la fenêtre des graphes**. Lorsque vous tapez `plt.show()` (avec deux parenthèses, s'il vous plaît !) et que vous exécutez, le graphe s'affiche parfois en arrière-plan. Il ne faut pas paniquer quand aucune figure ne s'affiche : il y en a une, il suffit de cliquer sur l'icône correspondant dans la barre des tâches.
- sachez **reconnaître** les graphes des différentes fonctions que vous tracez. Les légendes peuvent être utiles à ce sujet.
- apprenez à **lire avec des arrondis**. Si le résultat qui vous est renvoyé par python est

```
12 [ 1.37605094e-01+1.234429123e-17.j ,
13 5.65477599e-01+0.j ,
14 -3.29392851e-16+0.j ,
15 -5.42212563e-01+0.j]
```

alors sachez remarquer que toutes les coordonnées de ce vecteur sont **réelles** et que la troisième coordonnée est **nulle**. En effet,

— un nombre complexe s'écrit comme `a + bj` en python. Par exemple, `1.5+0.5j` représente $\frac{3}{2} + \frac{1}{2}i$

— `1.234429123e-17` et `-3.29392851e-16` sont des valeurs à qu'il faut considérer comme nulles.

Derniers conseils. Le fichier python doit aussi être en ligne sur cahier-de-prépa : il faudra commenter/décommenter les lignes correspondantes pour faire tourner les programmes. Utilisez Ctrl+Entrée pour ne faire tourner que les lignes de code entre deux double-dièses.
Il se peut qu'il y ait des erreurs ou incohérences. N'hésitez pas à m'écrire à ce sujet !

N. Laillet
nlaillet.math@gmail.com

2 Algèbre

2.1 Algèbre linéaire

Il faut utiliser l'aide-mémoire « Calcul matriciel ».

Type de questions et conseils.

- il faut pouvoir remplir une matrice de taille n quelconque. Il arrivera régulièrement que l'on vous donne une forme générale de matrice et que vous deviez la remplir. Sachez donc initialiser une matrice à 0 avec `np.zeros((a,b))` (attention aux doubles parenthèses, erreur classique !), puis utiliser deux boucles `for` imbriquées.
- attention à la numérotation qui commence à 0 en python (j'ai déjà vu des matrices « décalées » à cause de ce problème)
- deux questions classiques en algèbre linéaire :
  - calculer un déterminant en essayant de trouver une expression en fonction de n,
  - étudier les éléments propres d'une matrice. Faites alors attention qu'il existe deux fonctions, `alg.eig` et `alg.eigvals` : la première donne un tableau constitué des **valeurs propres** et la matrice constituée des **vecteurs propres** associés. Parfois, on vous demandera de commenter ce que vous obtenez. Quelques commentaires classiques à faire :
    - « je trouve n valeurs propres distinctes pour une matrice $n \times n$, donc cette matrice semble diagonalisable »
    - « toutes les valeurs propres de la matrice semblent réelles »
    - « la plus petite valeur propre semble être (...) »
- vous aurez parfois des suites de matrices à calculer par récurrence. Pensez surtout au fait que le produit matriciel n'est pas `*` : il est décrit dans l'aide-mémoire.
- la chose la plus difficile que l'on peut vous demander en algèbre linéaire est de faire l'algorithme de Gram-Schmidt/de calculer une distance à un sous-espace vectoriel.

Exemples corrigés.

1. Proposer deux fonctions python : une fonction `matmin(L)` qui prend en argument une liste de réels triée et qui renvoie la matrice de terme général $(\texttt{min(L[i],L[j]})_{1\leqslant \texttt{i},\texttt{j}\leqslant n}$, et une fonction `matpgcd(L)` qui prend en argument une liste d'entiers naturels non nuls et qui renvoie la matrice de terme général $(\texttt{L[i]} \wedge \texttt{L[j]})_{1\leqslant \texttt{i},\texttt{j}\leqslant n}$. Vérifier, pour quelques valeurs de n, que ces matrices sont toujours dans $\mathcal{S}_n^+(\mathbb{R})$.

2. Soient $n \geqslant 2$ et $A = (a_{i,j})_{1\leqslant i,j\leqslant n}$ la matrice de $\mathcal{M}_n(\mathbb{R})$ telle que $a_{i,j} = 1$ si $|i-j| = 1$, les autres coefficients étant nuls.

$$\text{Pour } j \in \{1, \dots, n\}, \text{ on pose } X_j = \left(\sin\left(\frac{j\pi}{n+1}\right)\sin\left(\frac{2j\pi}{n+1}\right)\dots\sin\left(\frac{nj\pi}{n+1}\right)\right)^T.$$

On note $P \in \mathcal{M}_n(\mathbb{R})$ la matrice dont les colonnes sont $X_1, \dots, X_n$.

(a) Écrire une fonction $A(n)$ (resp. $P(n)$) qui renvoie la matrice A (resp. P).

(b) Écrire une fonction $B(n)$ qui renvoie la matrice $P^{-1}AP$. Calculer $B(n)$ pour différentes valeurs de n. Émettre une conjecture sur le spectre de A et sur la famille $(X_1, \dots, X_n)$. On admet la validité de ces conjectures.

2.2 Arithmétique et polynômes

L'arithmétique est un terrain de prédilection des planches de Maths II en MP. C'est simple de faire du python dessus ! Les polynômes sont un peu moins présents.

Type de questions et conseils.

- en arithmétique, l'avantage est que les commandes python sont assez simples : `%` désigne le reste et `//` le quotient dans la division euclidienne.
- python est pratique pour faire du calcul dans $\mathbb{Z}/n\mathbb{Z}$ par exemple : il suffit juste de travailler modulo n.
- pour l'objet « polynômes », les choses se corsent un peu : on vous demandera d'utiliser l'aide-mémoire du module `Polynomial`. Faites **très attention** au fait que dans ce module, le premier coefficient correspond au coefficient constant (c'est décrit sur l'exemple de l'aide-mémoire mais, comme toujours, dans la précipitation, on peut paniquer).
- **CONSEIL ABSOLU, présent dans le rapport de jury** : si jamais vous avez des polynômes à manipuler, il peut être très malin de poser `X = Polynomial([0,1])`, puis de manipuler les polynômes comme d'habitude.
- **non affichée dans l'aide-mémoire**, la manière dont les polynômes s'affichent ne doivent pas vous faire peur.

```
16 >>> X = Polynomial([0,1])
17
18 >>> P = X**2 - 3*X + 4
19
20 >>> P
21 Polynomial([ 4., -3.,  1.], domain=[-1.,  1.], window=[-1.,  1.])
```

  Les parties `domain` et `window` ne sont pas utiles du tout, à aucun des planches de Centrale. Elles sont utiles lorsque l'on fait de l'interpolation polynomiale.

- dernier point : on vous demandera parfois de regarder le **polynôme caractéristique** d'une matrice. C'est terrible, les coefficients sont affichés **dans l'ordre inverse** de celui donné par `Polynomial`. Il faut donc y faire attention.

Exemples corrigés.

3. Soit μ la fonction définie sur $\mathbb{N}^*$ de la manière suivante :

- $\mu(1) = 1$.
- $\mu(n) = 0$ si n possède au moins un facteur carré.
- $\mu(n) = (-1)^k$ si $n = p_1 \dots p_k$ est la décomposition en facteur premiers de n (sans facteur carré, donc).

(a) Écrire une fonction Python `liste_P` qui prend en paramètre un entier n et renvoie la liste des diviseurs premiers de n.

(b) Écrire en Python une fonction mu qui renvoie $\mu(n)$.

(c) Vérifier, sur certains exemples, que pour $n \wedge m$, $\mu(mn) = \mu(m)\mu(n)$.

4. Soit p un nombre premier. On rappelle que $\mathbb{Z}/p\mathbb{Z}$ est un corps, on note $(\mathbb{Z}/p\mathbb{Z})^* = \mathbb{Z}/p\mathbb{Z}\setminus\{0\}$. On dit que $x \in (\mathbb{Z}/p\mathbb{Z})^*$ est une racine primitive modulo p si $\{x^n, n \in \mathbb{Z}\} = (\mathbb{Z}/p\mathbb{Z})^*$. À l'aide de python, déterminer les racines primitives modulo 107.

N. Laillet
nlaillet.math@gmail.com

5. Soit E l'ensemble des polynômes **unitaires** (i.e. de coefficient dominant égal à 1) à coefficients dans $\mathbb{Z}$. Si $P(X) = (X - \lambda_1) \dots (X - \lambda_n) \in E$, avec $(\lambda_1, \dots, \lambda_n)$ les racines complexes de P, on définit

$$r_q(P) = (X - \lambda_1^q) \dots (X - \lambda_n^q).$$

(d) Écrire une fonction python `r(q,P)` qui renvoie $r_q(P)$ sous forme développée.

(e) Tester cette fonction pour quelques valeurs de $q \in \mathbb{N}^*$ et $P \in E$. Que peut-on conjecturer ?

3 Analyse

3.1 Suites et séries numériques

La difficulté, lorsque vous avez des suites ou des séries, c'est la manière dont elles sont définies.

Type de questions et conseils.

- définition **par récurrence** : une fonction itérative ou récursive est possible. Faites attention aux complexités trop élevées lorsqu'il y a une suite récurrente d'ordre 2 ou plus. Par exemple, on évitera, pour la suite de fibonacci $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$, d'écrire

```
22 def fibo(n):
23     if n==0 or n==1:
24         return n
25     else:
26         return fibo(n-1) + fibo(n-2)
```

  On préfèrera éviter de faire trop d'appels récursifs. On peut poser

```
27 def fibo(n):
28     L = [0,1]
29     for k in range(2,n+1):
30         L.append(0)
31         L[k] = L[k-1] + L[k-2]
32     return L[n]
```

- définition **implicite** : beaucoup d'exercices d'oral vous font étudier des suites $(x_n)_{n\in\mathbb{N}}$ où x_n est l'unique solution d'une équation dépendant de n. Lisez **attentivement** l'aide-mémoire sur la résolution d'équation : notamment, il est dit que la fonction `fsolve(f,x0)` prend en arguments une **fonction** f et un **point de départ de l'algorithme** x_0.
- pour les séries, souvent, calculer une somme tronquée à 100 termes suffit pour avoir une bonne approximation de la valeur : on calculer $\sum_{k=0}^{100} u_k$ au lieu de $\sum_{k=0}^{+\infty} u_k$.

Exemples corrigés.

6. On définit $u_0 = 1$ et $u_{n+1} = \sqrt{1 + u_n}$. Représenter les 100 premiers termes de $(u_n)_{n\in\mathbb{N}}$.

7. On définit $u_0 = 1$ et, pour tout n dans $\mathbb{N}$, $u_{n+1} = \sum_{k=0}^{n} u_k u_{n-k}$. Représenter les 20 premiers termes de la suite.

8. (a) Pour $n \in \mathrm{N}$, montrer que l'équation $\mathrm{e}^{-x}\left(1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}\right) = \frac{1}{2}$ possède une unique solution $a_n \in \mathbb{R}$. *Non corrigé car ce n'est pas du python !*

(b) Tracer sur une même figure a_n, n et $n+1$ pour $0 \leqslant n \leqslant 50$. Que peut-on conjecturer ?

(c) Tracer $3\,(a_n - n)$ pour $0 \leqslant n \leqslant 50$. Que peut-on conjecturer ?

3.2 Suites et séries de fonctions

Type de questions et conseils.

- Là, on vous demandera souvent de tracer des suites ou séries de fonctions et de conjecturer des comportements asymptotiques de ces suites/séries de fonctions.
- La principale difficulté sera le mélange des variables (n et x). Faites-y bien attention ! Sinon, les conjectures doivent être assez faciles à faire.

Exemples corrigés.

9. Pour $(x, n) \in \mathbb{R} \times \mathbb{N}$, on pose $u_n(x) = \dfrac{x^n}{1 + nx}$ et $S_n(x) = \displaystyle\sum_{k=0}^{n} u_k(x)$. On pose $N(x, \varepsilon) = \left\lfloor \dfrac{\ln(\varepsilon(1-x))}{\ln(x)} \right\rfloor$. On peut identifier $S(x)$ à $S_{N(x,\varepsilon)}(x)$ en prenant $\varepsilon = 10^{-3}$. Tracer la courbe de S et énoncer une conjecture.

3.3 Intégrales généralisées et intégrales à paramètres

Type de questions et conseils.

- **Attention à la gestion des variables** lorsque vous avez une intégrale à paramètres à tracer.
- Vous aurez très souvent besoin de définir une fonction **à l'intérieur d'une fonction** (un peu comme pour la recherche de zéros). C'est normal, il faut s'habituer à ça.
- Il faut savoir que l'on peut mettre `np.inf` comme argument de `np.quad`, ce qui permet de calculer des intégrales allant jusqu'à $+\infty$.
- C'est déjà dit dans l'aide-mémoire de Centrale, mais attention : `np.quad(f,a,b)` renvoie **deux réels**, le premier étant la valeur approchée de l'intégrale et le deuxième étant une estimation de l'erreur.
- Python calcule mal les valeurs approchées d'intégrales semi-convergentes.

Exemples corrigés.

10. On étudie la suite a définie par $a_n = 1 - \displaystyle\int_0^1 \left(3x^2 - 2x^3\right)^{\frac{1}{n}} dx$. Écrire un programme $a(n)$ renvoyant la valeur de a_n. Puis déterminer, à l'aide d'un programme, la convergence éventuelle de la suite (na_n).

11. On pose :

$$F(x) = \int_0^{+\infty} \frac{\sin(xt)}{t + t^2} dt,\ H(x) = \int_0^{+\infty} \frac{t\cos(xt)}{t + t^2} dt \text{ et } G(x) = \int_0^{+\infty} \frac{\sin(u)}{x^2 + u^2} du.$$

Afficher sur Python $x \mapsto (F(x+h) - F(x))/h$ et $x \mapsto H(x)$ avec $h = 0.1$ et établir une conjecture.

3.4 Équations différentielles

Type de questions et conseils.

- Attention, **toute** équation différentielle, en python, s'écrit sous la forme $y'(t) = F(y(t), t)$. Il faudra **toujours** définir proprement une fonction F.
- Notamment, si vous avez une équation du second ordre, il faudra vraiment bien faire attention à l'écrire sous la forme d'une équation du premier ordre.

Exemples corrigés.

12. Soient a un réel, $g \in C^0(\mathbf{R}, \mathbf{R})$ et (L) l'équation différentielle $f' + af = g$.

(a) Écrire la fonction $f(g, a, b, x)$ donnant la valeur en x de la solution f de (L) valant b en zéro.

(b) Soit $g : x \longmapsto \sin(x)$. On choisit $a = 1$ et un b arbitraire. Tracer le graphe de f. Tracer ensuite le graphe de f pour $b = \left(e^{2\pi a} - 1\right)^{-1} \int_0^{2\pi} g(t)e^{at}dt$. Qu'en déduit-on ?

3.5 Calcul différentiel

Type de questions et conseils.

- Il y aura assez peu de calcul différentiel à Centrale.
- Cependant, on peut vous demander de tracer les lignes de niveau d'une fonction de deux variables, ou des surfaces. Pour ce faire, **copiez intégralement** le code donné dans l'aide : il ne faut pas que vous preniez d'initiative dans ce genre d'exercice, ce sont toujours les mêmes lignes de code à taper.

Exemples corrigés.

13. Soit $n \in \mathbb{N}^*$. On munit $\mathbb{R}^n$ de son produit scalaire canonique et de la norme euclidienne associée. Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction $\mathscr{C}^1$. Soit $\rho > 0$ et $x_0 \in \mathbb{R}^n$. On définit la suite $(x_k)_{k\in\mathbb{N}}$ par, pour tout k dans $\mathbb{N}$,

$$x_{k+1} = x_k - \rho\nabla f(x_k).$$

Dans cette question, on pose $n = 2$, $A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$, $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$, $f : x \mapsto x^T Ax + b^T x$, $x_0 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$ et $\rho = 0.1$.

(a) **[Py]** Tracer le graphe de f (comme $f : \mathbb{R}^2 \to \mathbb{R}$, il s'agit d'une surface).

(b) **[Py]** Tracer les lignes de niveau de f, et repérer ainsi le minimum de la fonction f.

(c) **[Py]** Justifier que pour tout entier k, $x_{k+1} = x_k - \rho(2Ax_k + b)$, puis conjecturer numériquement la convergence et la limite éventuelle de la suite $(x_k)_{k\in\mathbb{N}}$.

14. Pour tout $(x, y) \in \mathbf{R}^2$, on pose $f(x, y) = x^3 - 3x\left(1 + y^2\right)$.

(a) Tracer les lignes de niveau $f(x, y) = h$ pour $(x, y) \in [-2, 2]^2$ et $h \in [\![-3, 3]\!]$. Énoncer une conjecture sur les points critiques de f.

(b) Tracer le graphe de $g : x \mapsto f(1+x, x)$. Quelle conjecture peut-on faire sur un des points critiques de f ?

4 Probabilités

C'est **LA** partie du programme où les simulations python ont le plus de sens : vous aurez de grandes chances de tomber sur un oral de probabilités à Centrale.

Type de questions et conseils.

- **SIMULER** une variable aléatoire : vous aurez beaucoup de questions du type « écrire une fonction renvoyant une simulation de l'expérience » ou « simuler en python la variable X », ou « écrire une fonction python renvoyant une réalisation de X ».
  Dans tous ces cas, il faut créer une fonction qui renvoie donc un **résultat**, et pas une **probabilité**.
- Vous aurez à votre disposition toutes les lois usuelles.
  Attention ! Dans l'aide-mémoire python, il est juste précisé que `rd.binomial(n,p,k)` renvoie un tableau avec `k` simulations d'une loi binomiale de paramètres `(n,p)`. Il n'est pas écrit, plus simplement, que `rd.binomial(n,p)` renvoie **une** simulation d'une loi binomiale de paramètres `(n,p)`.
- Deuxième type de question : **estimer** une espérance. Là, la méthode sera **toujours la même.**
  - vous faites un grand nombre de simulations de la variable aléatoire (1000 par exemple), et vous faites la moyenne de ces simulations.
  - l'interrogatrice ou l'interrogateur vous demandera presque toujours pourquoi cette méthode fonctionne : vous répondrez que c'est à cause de la **loi des grands nombres**, qui dit que la moyenne d'un grand nombre de réalisations indépendantes d'une variable aléatoire se rapproche de l'espérance.

Exemples corrigés.

15. On choisit un entier X uniformément au hasard entre 1 et n, puis un entier Y uniformément au hasard entre 1 et X. Écrire une fonction python `Y(n)` simulant la variable aléatoire `Y`. Estimer l'espérance de Y en fonction de n.

16. Soient X et Y deux variables aléatoires uniformes sur $\mathbb{Z}/n\mathbb{Z}$. Estimer la loi de $X+Y$.

17. Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables aléatoires i.i.d. telle que $\mathbb{P}(X_0=1)=\mathbb{P}(X_0=-1)=\dfrac{1}{2}$.
On pose $S_n = x_0+\cdots+x_n$. Soit $(a,b)\in\mathbb{Z}^2$ avec $a<b$. On pose $T=\min\{n\in\mathbb{N}, S_n\notin[a,b]\}$, si cet ensemble est non vide, $T=+\infty$ sinon. On dit que T est le temps de sortie de $[a,b]$.

(a) Écrire, en Python, une fonction `temps(a,b)` qui renvoie le temps de sortie T. Tester avec $a=-5$ et $b=7$.

(b) Écrire, en Python , une fonction `moyenne(a,b)` qui renvoie la moyenne de T sur un nombre de 10000 expériences. Tester avec $a=-5$ et $b=7$ et d'autres valeurs de a et b. Que peut-on conjecturer ?

 N. Laillet
nlaillet.math@gmail.com

5 Corrigé des exercices

1. On propose les programmes suivants (difficulté : calculer un pgcd).

```
33 import numpy as np
34 import matplotlib.pyplot as plt
35 import numpy.linalg as alg
36 import numpy.random as rd
37
38
39 def matmin(L):
40     n = len(L)
41     A = np.zeros((n,n))
42     for i in range(n):
43         for j in range(n):
44             A[i,j] = min(L[i],L[j])
45     return A
46
47
48 def pgcd(x,y):
49     a,b = max(x,y),min(x,y)
50     while b!=0:
51         a,b = b,a%b
52     return a
53
54 def matpgcd(L):
55     n = len(L)
56     A = np.zeros((n,n))
57     for i in range(n):
58         for j in range(n):
59             A[i,j] = pgcd(L[i],L[j])
60     return A
61
62 def estpos(L):
63     for x in L:
64         if x<-10**(-10):
65             return False
66     return True
67
68 compt = 0
69 for _ in range(1000):
70     L = np.sort(100*rd.random(10))
71     A = alg.eigvals(matmin(L))
72     compt+=estpos(A)
73
74 for _ in range(1000):
75     L = rd.randint(1,1000,10)
76     A = alg.eigvals(matpgcd(L))
77     compt+=estpos(A)
```

(j'ai fait attention aux problèmes de comparaison des flottants à 0) On trouve bien `compt` = 2000 après tests.

2. On propose les fonctions suivantes. On a bien fait attention à `np.zeros((n,n))`, on a fait

attention aux i et j qui deviennent $i+1$ et $j+1$ dans la fonction `P(n)`.

```
def A(n):
    M = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if abs(i-j)==1:
                M[i,j] = 1
    return M

def P(n):
    M = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            M[i,j] = np.sin((i+1)*(j+1)*np.pi/(n+1))
    return M

def B(n):
    Q = P(n)
    return alg.inv(Q).dot(A(n).dot(Q))
```

Pour l'affichage des résultats, `P(3)` peut faire peur :

```
>>> B(3)
array([[ 1.41421356e+00,  8.65956056e-17, -5.55111512e-17],
       [ 0.00000000e+00, -1.35963107e-32,  2.77555756e-16],
       [ 5.55111512e-17,  8.65956056e-17, -1.41421356e+00]])
```

Deux possibilités pour parer à ce problème. Ou bien on remonte ses manches et on repère ce qui a l'air nul ou pas, ou bien on utilise `.round(d)` qui arrondit à `d` décimales.

```
>>> B(3).round(2)
array([[ 1.41,  0.  , -0.  ],
       [ 0.  , -0.  ,  0.  ],
       [ 0.  ,  0.  , -1.41]])

>>> B(4).round(2)
array([[ 1.62,  0.  , -0.  ,  0.  ],
       [-0.  ,  0.62,  0.  ,  0.  ],
       [ 0.  ,  0.  , -0.62, -0.  ],
       [-0.  , -0.  ,  0.  , -1.62]])

>>> B(5).round(2)
array([[ 1.73,  0.  , -0.  ,  0.  ,  0.  ],
       [-0.  ,  1.  ,  0.  ,  0.  , -0.  ],
       [-0.  , -0.  , -0.  ,  0.  ,  0.  ],
       [-0.  , -0.  ,  0.  , -1.  , -0.  ],
       [ 0.  ,  0.  , -0.  ,  0.  , -1.73]])
```

On conjecture alors que A est diagonalisable et que ses vecteurs propres sont $X_1, \dots, X_n$. Son spectre a l'air réel et, si λ est valeur propre de A, alors $-\lambda$ l'est aussi. Enfin, si n est impair, $0 \in^(A)$.

3. On propose d'abord un test de primalité, puis on liste tous les diviseurs premiers, puis enfin on calcule la fonction de Möbius.

N. Laillet
nlaillet.math@gmail.com

```
118 def est_premier(n):
119     if n in [0,1]:
120         return False
121     else:
122         for k in range(2,n):
123             if n%k==0:
124                 return False
125         return True
126 
127 def liste_P(n):
128     return [k for k in range(n+1) if est_premier(k) and n%k==0]
129 
130 def mu(n):
131     if n==1:
132         return 1
133     else:
134         L = liste_P(n)
135         for k in L:
136             if n%(k**2)==0:
137                 return 0
138         return (-1)**(len(L))
```

4. On va ruser : l'idée, pour tester si x est une racine primitive modulo n, est de compter le plus petit k tel que $x^k = 1$. Si $k = n$, c'est que x était une racine primitive modulo n !

```
139 def est_inv(x,n):
140     k=1
141     y = x
142     while y !=1:
143         y = (y*x)%n
144         k = k+1
145     return k==n-1
146 
147 for k in range(1,107):
148     if est_inv(k,107):
149         print(k)
```

5. On propose le programme

```
150 from numpy.polynomial import Polynomial
151 
152 def r(q,P):
153     L = P.roots()
154     res = Polynomial([1])
155     for a in L:
156         res = res*Polynomial([-a**q,1])
157     return res
```

N. Laillet
nlaillet.math@gmail.com

On trouve, par exemple

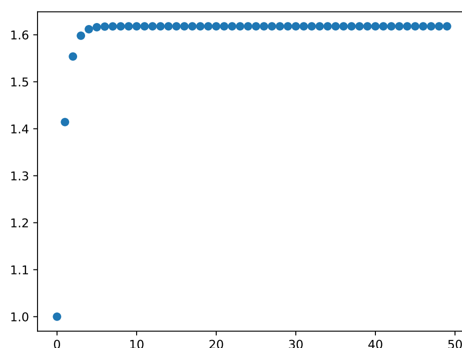
```
158 >>> P = Polynomial([2,3,1,2,1])
159
160 >>> r(4,P)
161 Polynomial([ 16.+0.j, -81.+0.j, 37.+0.j, -18.+0.j, 1.+0.j],
162 domain=[-1., 1.], window=[-1., 1.])
163
164 >>> r(3,P)
165 Polynomial([8.-4.4408921e-16j,33.+0.0000000e+00j,
166 28.+0.0000000e+00j, 11.+0.0000000e+00j, 1.+0.0000000e+00j],
167 domain=[-1., 1.], window=[-1., 1.])
```

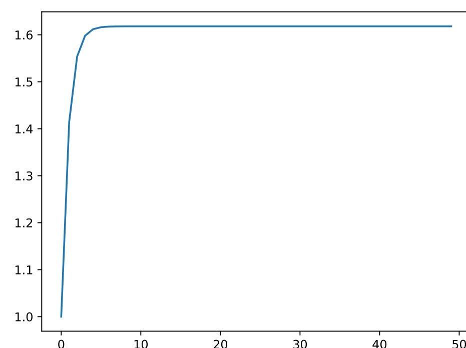
En pensant que tout ce qui est du type `-4.4408921e-16j` ou `+0.0000000e+00j` est nul, on conjecture que ce polynôme est toujours à coefficients entiers ! Le but de l'exercice est ensuite de démontrer ce résultat grâce aux polynômes caractéristiques d'une matrice.

6. Là, on peut faire un programme récursif tout simple :

```
168 def a(n):
169     if n == 0:
170         return 1
171     else:
172         return np.sqrt(1+a(n-1))
173
174 L = [k for k in range(50)]
175 A = [a(n) for n in L]
176 plt.plot(L,A,'o')
177 plt.show()
```

On remarque que dans le `plt.plot`, on met `'o'`, afin de représenter les termes de la suite par des points (dessin de gauche). Si on avait juste écrit `plt.plot(L,A)`, on aurait eu le dessin de droite.





 nlaillet.math@gmail.com

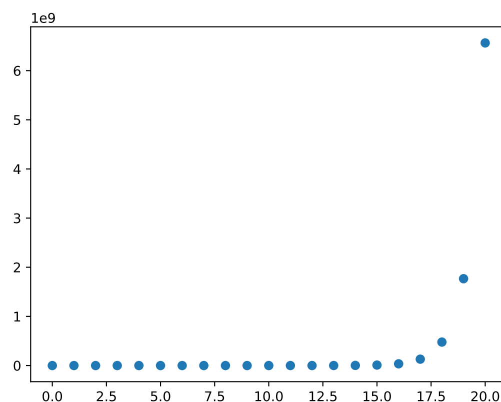
7. La grosse erreur de complexité serait d'écrire

```
178 def proglourd(n):
179     if n == 0:
180         return 1
181     else:
182         res = 0
183         for k in range(n):
184             res += proglourd(k)*proglourd(n-1-k)
185         return res
```

On a fait attention au fait que k allait bien de 0 à $n-1$, donc il n'y a aucun problème d'indices. Cependant, ce programme fait beaucoup trop d'appels à lui-même ! $n = 10$ se calcule sans problème. À partir de $n = 13$, on voit le temps augmenter. Pour $n = 20$, le temps est beaucoup trop gros !
On va donc **stocker les valeurs** de u_n dans une liste ! On propose alors

```
186 def u(n):
187     L = [1]
188     for k in range(1,n+1):
189         L.append(0)
190         for i in range(k):
191             L[k] = L[k] + L[i]*L[k-1-i]
192     return L
193
194 N = 20
195 Ln = [k for k in range(N+1)]
196 Un = u(N)
197
198 plt.plot(Ln,Un,'o')
199 plt.savefig('unrecdure.pdf')
200 plt.show()
```

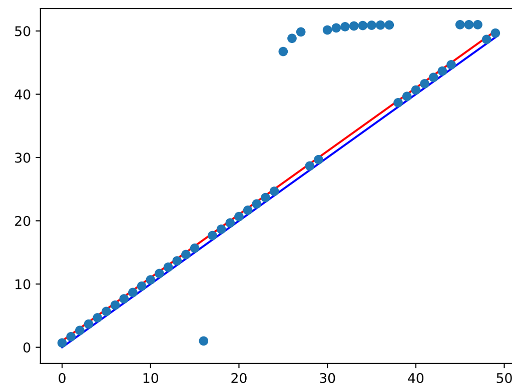
On obtient alors



8. (a) Là, comme je l'ai indiqué en conseil, il faut vraiment faire attention à la fonction que l'on va utiliser : la fonction `fsolve` : j'ai fait plusieurs tests, et prendre la fonction indiquée dans l'énoncé donne des choses bizarres, en fonction du point de départ de la méthode.

```
201 def a(n):
202     def f(x):
203         px = 1 #x**k/k!
204         res = 1
205         for k in range(1,n+1):
206             px *= x/k
207             res += px
208         return np.exp(-x)*res -1/2
209     return opt.fsolve(f,1)
210
211 Ln = [n for n in range(50)]
212 Lnp = [n+1 for n in Ln]
213 An = [a(n) for n in Ln]
214
215 plt.plot(Ln,Ln,'b')
216 plt.plot(Ln,Lnp,'r')
217 plt.plot(Ln,An,'o')
218 plt.show()
```

On obtient...

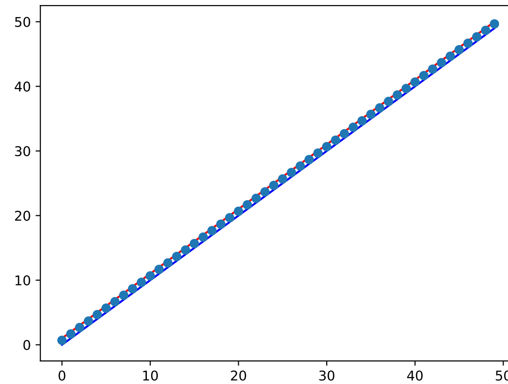


On a l'impression que $n \leqslant a_n \leqslant n+1$, sauf erreurs... une manière que j'ai trouvée pour éviter ce genre d'erreur est de poser

```
219 def a(n):
220     def f(x):
221         px = 1 #x**k/k!
222         res = 1
223         for k in range(1,n+1):
224             px *= x/k
225             res += px
226         return res -1/2*np.exp(x)
227     return opt.fsolve(f,100)
```

On obtient

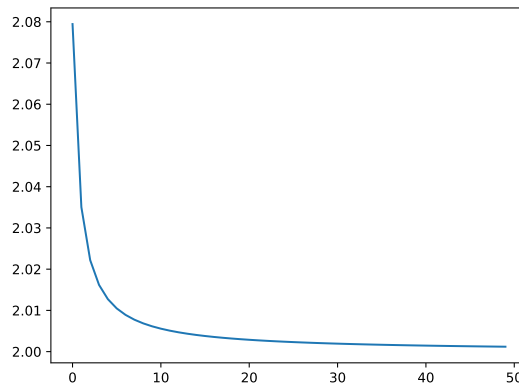
N. Laillet
nlaillet.math@gmail.com



(b) On écrit

```
228 Rn = [3*(a(n)-n) for n in Ln]
229 plt.plot(Ln,Rn)
230 plt.show()
```

et on obtient



9. On rappelle que ln est `np.log` en python. Pour rendre les programmes plus propres, on écrit

```
231 def N(x,eps):
232     return np.floor(np.log(eps*(1-x))/np.log(x))
```

Puis on définit $u_n(x)$:

```
233 def u(n,x):
234     return x**n/(1+n*x)
```

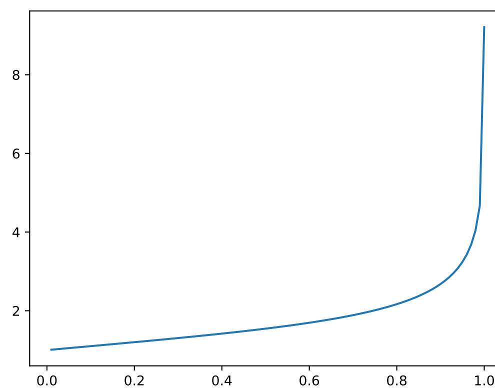
Enfin, on définit $S(x)$

```
eps = 10**(-3)
def S(x):
    n = N(x,eps)
    res = 0
    for k in range(n+1):
        res = res + u(k,x)
    return res
```

On peut enfin tracer la courbe de S :

```
LX = np.linspace(0.01,0.9999,100)
LY = [S(x) for x in LX]
plt.plot(LX,LY)
plt.show()
```

On obtient le graphe suivant, qui nous fait conjecturer que $S(x) \underset{x\to 1}{\longrightarrow} +\infty$.

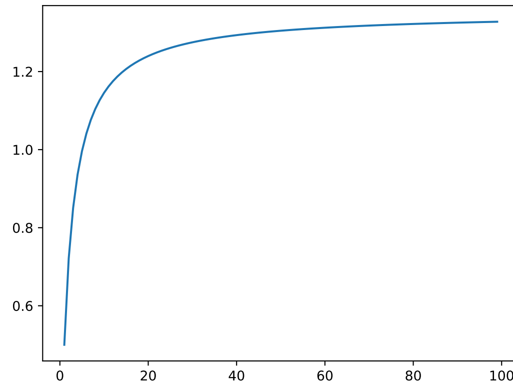


10. En faisant attention à bien définir une fonction dans une fonction, on propose

```
def a(n):
    def f(x):
        return (3*x**2 - 2*x**3)**(1/n)
    return 1 - integr.quad(f,0,1)[0]

LN = [n for n in range(1,100)]
LAN = [n*a(n) for n in LN]
plt.plot(LN,LAN)
plt.show()
```

On obtient

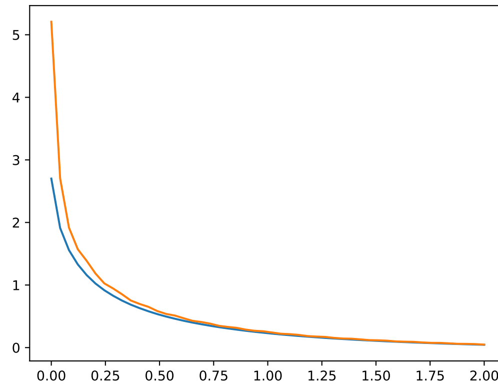


ce qui nous fait conjecturer la convergence de (na_n) vers une valeur finie non nulle.

11. Alors, là, on vous demandera une initiative, car l'intégrale définissant H ne converge pas vraiment... On propose alors

```
255 def F(x):
256     def f(t):
257         return np.sin(x*t)/(t+t**2)
258     return integr.quad(f,0.1,200)[0]
259
260 def H(x):
261     def f(t):
262         return t*np.cos(x*t)/(t+t**2)
263     return integr.quad(f,0.1,200)[0]
264
265 h = 0.1
266 def Fp(x):
267     return (F(x+h) - F(x))/h
268
269 LX = np.linspace(0,2)
270 LFp = [Fp(x) for x in LX]
271 LH = [H(x) for x in LX]
272
273 plt.plot(LX,LFp)
274 plt.plot(LX,LH)
275 plt.show()
```

On conjecture que $F' = H$! (cf. graphe suivant)



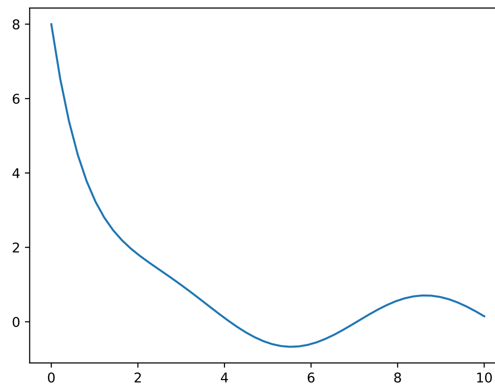
12. On propose

```
276 def f(g,a,b,x):
277     def F(y,t):
278         return -a*f + g(t)
279     T = np.linspace(0,x)
280     Y = integr.odeint(F,b,T)
281     return T[-1]
```

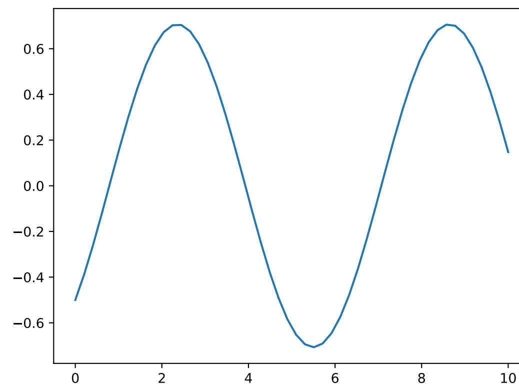
On remarque que l'on a défini la fonction F, et que dans `integr.odeint`, on y a mis F, la condition initiale et le tableau de temps. On aurait pu prendre un tableau `T` à deux éléments : `T = [0,b]`. Le nombre d'éléments du tableau `T` dans `odeint` n'a **absolument aucun impact** sur la précision de la solution.

```
282 def f(g,a,b,x):
283     def F(y,t):
284         return -a*y + g(t)
285     T = np.linspace(0,x)
286     Y = integr.odeint(F,b,T)
287     return Y[-1]
288
289 def g(x):
290     return np.sin(x)
291
292 def g2(x):
293     return g(x)*np.exp(a*x)
294
295 a = 1
296 b1 = 8
297 b2 = (np.exp(2*np.pi*a)-1)**(-1)*integr.quad(g2,0,2*np.pi)[0]
298 T = np.linspace(0,10)
299
300 LY1 = [f(g,a,b1,x) for x in T]
301 LY2 = [f(g,a,b2,x) for x in T]
302
303 plt.plot(T,LY1)
304 plt.show()
```

Avec un b arbitraire, on obtient



C'est une solution qui décroît vers 0.
Avec le b indiqué par l'énoncé, on obtient



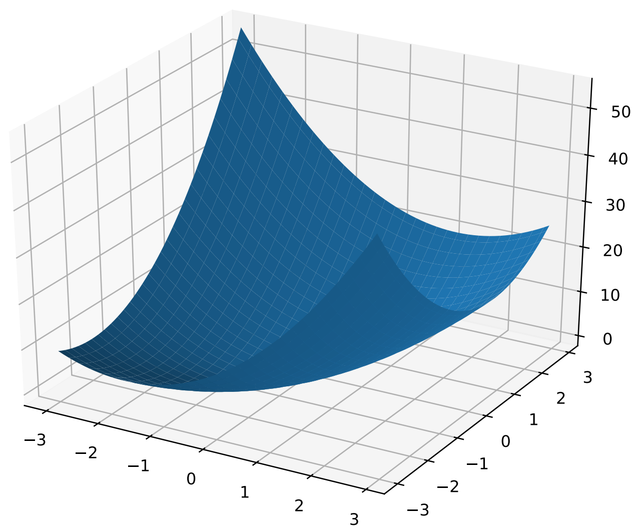
On retrouve une sinusoïde : c'est la seule solution non amortie du système, peut-être ?

13. (a) On propose

```
305 import numpy as np
306 import numpy.linalg as alg
307 import matplotlib.pyplot as plt
308 from mpl_toolkits.mplot3d import Axes3D
309
310 A = np.array([[2,-1],[-1,2]])
311 b = np.array([[1],[2]])
312
313 def F(x,y):
314     X = [x,y]
315     return np.dot(X,np.dot(A,X)) + np.dot(X,b)
316
```

```
317 ax = Axes3D(plt.figure())
318 F=np.vectorize(F)
319 X = np.arange(-3, 3, 0.1)
320 Y = np.arange(-3, 3, 0.1)
321 X, Y = np.meshgrid(X, Y)
322 Z = F(X, Y)
323 ax.plot_surface(X, Y, Z)
324
325 plt.show()
```

On obtient alors

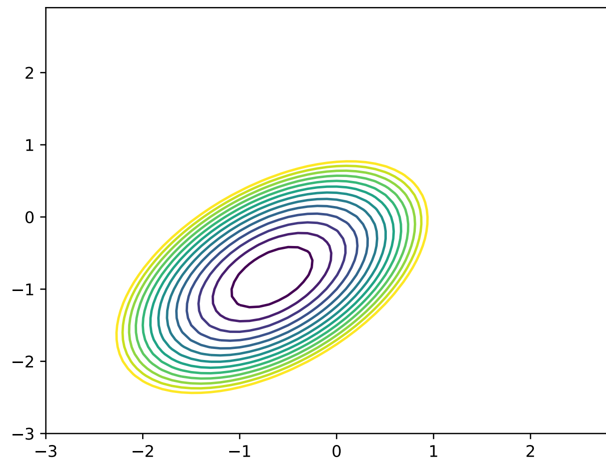


(b) On propose alors

```
326 plt.figure()
327 plt.contour(X,Y,F(X,Y), np.arange(-3,3,.3))
328 plt.show()
```

et on obtient

N. Laillet
nlaillet.math@gmail.com



On a bien l'impression d'un minimum vers $(-0.6, -0.8)$.

(c) Déjà, la justification de la relation de récurrence vient du fait que

$$\nabla f : x \mapsto 2Ax + b.$$

On propose enfin deux choses :

- convergence de la suite :

```
329 rho=.1
330 Xn=np.array([[3],[3]])
331 for n in range(100):
332     Xn=Xn-rho*(2*A.dot(Xn)+b)
333     print(Xn)
```

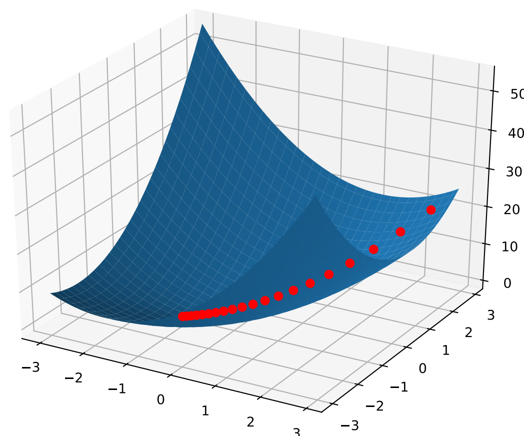
On a bien une convergence vers `[[-0.66666667] [-0.83333333]]`.

- graphe :

```
334 ax = Axes3D(plt.figure())
335 F=np.vectorize(F)
336 X = np.arange(-3, 3, 0.1)
337 Y = np.arange(-3, 3, 0.1)
338 X, Y = np.meshgrid(X, Y)
339 Z = F(X, Y)
340 ax.plot_surface(X, Y, Z)
341
342 def GF(X):
343     return np.dot(A,X) + b
344
345 def gradient(gf,X0,rho,n):
346     res = X0
347     for _ in range(n):
348         res = res - rho*GF(res)
349     return res
```

```
350
351
352
353 def gradgraphe(gf,X0,rho,n):
354     res = X0
355     LX = []
356     LY = []
357     LZ = []
358     for _ in range(n):
359         res = res - rho*gf(res)
360         x,y = res[0,0],res[1,0]
361         LX.append(x)
362         LY.append(y)
363         LZ.append(F(x,y)[0])
364     return LX,LY,LZ
365
366 LX,LY,LZ = gradgraphe(GF,[3,3],0.1,20)
367
368 plt.plot(LY,LY,LZ,'or')
369
370 plt.show()
```

On obtient alors



14. (a) Là, je recopie bêtement l'aide-mémoire de Centrale en changeant la fonction de deux variables ! Le concours dit

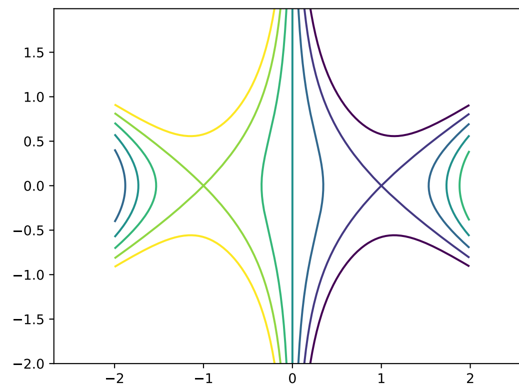
```
371 def f(x,y):
372     return x**2 + y**2 + x*y
373
374 f=np.vectorize(f)
375 X = np.arange(-1, 1, 0.01)
376 Y = np.arange(-1, 1, 0.01)
377 X, Y = np.meshgrid(X, Y)
378 Z = f(X, Y)
379 plt.axis('equal')
380 plt.contour(X, Y, Z, [0.1,0.4,0.5])
381 plt.show()
```

N. Laillet
nlaillet.math@gmail.com

Je change donc la valeur de f, le domaine de (x, y) et les valeurs prises par Z. Je tape donc

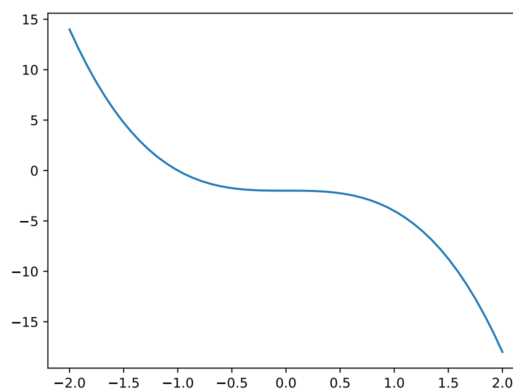
```
382 def f(x,y):
383     return x**3 -3*x*(1+y**2)
384
385 f=np.vectorize(f)
386 X = np.arange(-2, 2, 0.01)
387 Y = np.arange(-2, 2, 0.01)
388 X, Y = np.meshgrid(X, Y)
389 Z = f(X, Y)
390 plt.axis('equal')
391 plt.contour(X, Y, Z, [k for k in range(-3,4)])
392 plt.show()
```

On obtient



On conjecture que $(-1, 0)$ et $(1, 0)$ sont les deux deux points critiques de f.

(b) On trace

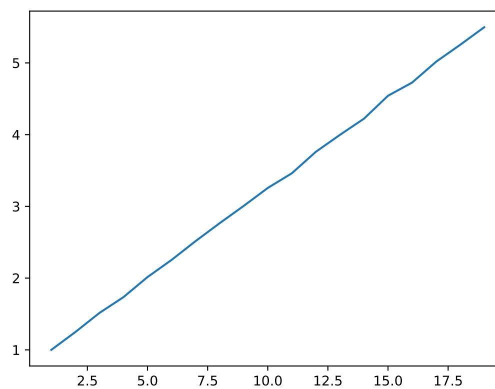


On remarque que la fonction n'a pas de minimum ou de maximum local en 0 : le point critique $(1, 0)$ n'est ni un maximum, ni un minimum local de f.

15. On va prendre une initiative ici ! Pour estimer l'espérance **en fonction** de n, on trace `Y(n)` en fonction de `n`. Ce n'est pas obligatoire : s'arrêter à la fonction `esp(n)` aurait pu suffire. Mais là, on obtiendra un beau graphique.

```
393 import numpy.random as rd
394 import matplotlib.pyplot as plt
395
396 def X(n):
397     return rd.randint(1,n+1)#attention à la borne de fin
398
399 def Y(n):
400     return rd.randint(1,X(n)+1)
401
402 def esp(n):
403     N = 10000
404     res = 0#on va faire la somme des réalisations de la variable
405     for k in range(N):#k est juste
406     #une variable d'itération
407         res = res + Y(n)
408     return res/N #on renvoie la moyenne
409
410 Ln = [k for k in range(1,20)]
411 Lesp = [esp(n) for n in Ln]
412 plt.plot(Ln,Lesp)
413 plt.show()
```

On obtient le graphique suivant, qui fait penser à une dépendance **affine** de $\mathbb{E}(Y_n)$ en fonction de n.



16. (a) Là, on cherche à chercher **le premier instant** auquel S_n sort de $[\![a, b]\!]$. Il faudra donc utiliser une boucle `while`.
Pour simuler X_n, il est simple de simuler une variable de Bernoulli de paramètre $\frac{1}{2}$, ou bien avec `rd.randint(0,2)`, ou bien avec `rd.binomial(1,1/2)`. Ensuite, si Y est une variable de Bernoulli, on remarque que $2Y - 1$ suit la loi de X_n ! On propose donc

```
414 def X():
415     return 2*rd.randint(0,2)-1
416
417 def T(a,b):
418     S = 0
419     temps = 0
420     while S>=a and S<=b:
421         S += X()
422         temps += 1
423     return temps
```

(b) Là, la simulation, c'est comme ce que l'on a fait précédemment :

```
424 def esp(a,b):
425     N = 10000
426     res = 0
427     for k in range(N):
428         res = res + T(a,b)
429     return res/N
```

Ce qui est **vraiment dur**, c'est la conjecture. Voici plein de valeurs

```
430 >>> esp(-3,3)
431 16.2254
432
433 >>> esp(-3,4)
434 20.0676
435
436 >>> esp(-3,5)
437 24.0218
438
439 >>> esp(-3,6)
440 28.4745
```

```
441 >>> esp(-1,6)
442 14.3992
443
444 >>> esp(-2,6)
445 20.7774
446
447 >>> esp(-2,8)
448 26.7196
449
450 >>> esp(-6,3)
451 27.763
```

Et là, en se disant que l'on doit remarquer quelque chose, on a l'impression que l'espérance est égale à $(-a+1)(b+1)$... mais, le trouver pendant un oral de 30 minutes, avec le stress, cela ne me paraît pas immédiat.