

Plaquelette 4 - Algèbre linéaire numérique : pivot, itérations, applications

Python et calcul scientifique — Licence 1 (L1)

Corrections détaillées

Boîte à outils

Pivot de Gauss : $L_i \leftarrow L_i - \frac{a_{ij}}{a_{jj}} L_j$ sous le pivot, puis remontée ; coût $\approx \frac{n^3}{3}$ multiplications.

— **Déterminant** : produit des pivots, changé de signe à chaque échange de lignes.

— **Inverse** (Gauss-Jordan) : on échelonne $(A \mid I_n)$ jusqu'à $(I_n \mid A^{-1})$.

— **Méthode de la puissance** : $x_{k+1} = \frac{Ax_k}{\|Ax_k\|}$ converge vers un vecteur propre de la valeur propre dominante, à la vitesse $\left| \frac{\lambda_2}{\lambda_1} \right|^k$.

— **Moindres carrés** (droite $y = ax + b$) :

$$a = \frac{\text{cov}(x, y)}{V(x)}, \quad b = \bar{y} - a\bar{x}$$

— **Conditionnement** : une matrice presque singulière amplifie les erreurs d'arrondi (matrices de Hilbert).

Correction de l'exercice 1 - Programmer le pivot de Gauss

a)

```
from fractions import Fraction
```

```
def gauss(A, b):
    n = len(A)
    M = [[Fraction(x) for x in A[i]] + [Fraction(b[i])] for i in range(n)]
    for j in range(n):
        p = j
        while M[p][j] == 0:
            p = p + 1
        M[j], M[p] = M[p], M[j]
        for i in range(j + 1, n):
            c = M[i][j] / M[j][j]
            M[i] = [M[i][k] - c * M[j][k] for k in range(n + 1)]
    x = [Fraction(0)] * n
    for i in range(n - 1, -1, -1):
        s = M[i][n] - sum(M[i][k] * x[k] for k in range(i + 1, n))
        x[i] = s / M[i][i]
    return x
```

b) $2 + 2 - 3 = 1 \checkmark$; $1 - 2 + 6 = 5 \checkmark$; $3 + 4 + 3 = 10 \checkmark$

c) À l'étape j , on modifie $n - j - 1$ lignes de $n - j + 1$ coefficients environ :

$$\sum_{j=0}^{n-1} (n-j)^2 \approx \frac{n^3}{3}$$

d) En fractions, le résultat est **exact** : une erreur de programmation se voit immédiatement, sans être confondue avec une erreur d'arrondi.

Réponse. Solution (1, 2, 3) ; coût $\approx \frac{n^3}{3}$ multiplications.

Correction de l'exercice 2 - Déterminant par élimination

a)

```
def det(A):
    n = len(A)
    M = [[Fraction(x) for x in ligne] for ligne in A]
    d = Fraction(1)
    for j in range(n):
        p = j
        while p < n and M[p][j] == 0:
            p = p + 1
        if p == n:
            return Fraction(0)
        if p != j:
            M[j], M[p] = M[p], M[j]
            d = -d
        d = d * M[j][j]
        for i in range(j + 1, n):
            c = M[i][j] / M[j][j]
            M[i] = [M[i][k] - c * M[j][k] for k in range(n)]
    return d
```

b) Sarrus : $0 + 2 + 2 - 0 - 0 - 0 = 4$ ✓

c) $a_{11} = 0$: on ne peut pas diviser par ce pivot ; on échange L_1 et L_2 , ce qui change le signe du déterminant.

d) $n = 20$:

$$\frac{20^3}{3} \approx 2700 \quad \text{contre} \quad 20! \approx 2,4 \times 10^{18}$$

— soit, à 10^9 opérations par seconde, plus d'un millénaire de calcul pour la formule développée, contre quelques microsecondes par élimination.

Réponse. $\det = 4, 1, 0$; coût ≈ 2700 opérations contre $20!$ pour $n = 20$.

Correction de l'exercice 3 - Inverse par Gauss-Jordan

a)

```
def inverse(A):
    n = len(A)
    M = [[Fraction(x) for x in A[i]] + [Fraction(int(i == j)) for j in range(n)]
          for i in range(n)]
    for j in range(n):
        p = j
        while M[p][j] == 0:
            p = p + 1
        M[j], M[p] = M[p], M[j]
        piv = M[j][j]
        M[j] = [v / piv for v in M[j]]
        for i in range(n):
            if i != j:
                c = M[i][j]
                M[i] = [M[i][k] - c * M[j][k] for k in range(2 * n)]
    return [ligne[n:] for ligne in M]
```

b) Première ligne de B fois les colonnes de B^{-1} : $-\frac{1}{6} + 1 + \frac{1}{6} = 1$, $-\frac{1}{4} + \frac{1}{2} - \frac{1}{4} = 0$, $\frac{7}{12} - \frac{1}{2} - \frac{1}{12} = 0$ ✓

c) On retrouve $B^{-1} = \frac{1}{12} \begin{pmatrix} -2 & -3 & 7 \\ 6 & 3 & -3 \\ -2 & 3 & 1 \end{pmatrix}$ (formule de la comatrice) ✓

d) Gauss directement : $\approx \frac{n^3}{3}$ opérations, contre $\approx n^3$ pour l'inverse, puis n^2 pour le produit :

$$\text{inverse : } \approx n^3 \quad \text{contre} \quad \text{Gauss : } \approx \frac{n^3}{3}$$

— On ne calcule A^{-1} que si l'on doit résoudre **beaucoup** de systèmes de même matrice (et même alors, on préfère garder la factorisation LU).

Réponse. $B^{-1} = \frac{1}{12} \begin{pmatrix} -2 & -3 & 7 \\ 6 & 3 & -3 \\ -2 & 3 & 1 \end{pmatrix}$; résoudre par Gauss plutôt qu'inverser.

Correction de l'exercice 4 - Calculer un rang

a)

```
def rang(A):
    M = [[Fraction(x) for x in ligne] for ligne in A]
    n, p = len(M), len(M[0])
    r = 0
    for j in range(p):
        piv = None
        for i in range(r, n):
            if M[i][j] != 0:
                piv = i
                break
        if piv is None:
            continue
        M[r], M[piv] = M[piv], M[r]
        for i in range(r + 1, n):
            c = M[i][j] / M[r][j]
            M[i] = [M[i][k] - c * M[r][k] for k in range(p)]
        r = r + 1
    return r
```

b) Première matrice : $L_1 + L_3 = 2L_2$, et L_1, L_2 non colinéaires : rang 2. Deuxième : $L_2 = 2L_1$, rang 1.

c) En flottants, une ligne qui devrait s'annuler peut valoir 10^{-16} au lieu de 0 : le test $!= 0$ la compte comme pivot et le programme renverrait 3. Il faut alors un **seuil** de tolérance, dont le choix est délicat.

d) Théorème du rang :

$$\dim \ker A = 4 - \text{rg} A = 2$$

Réponse. Rangs 2, 1, 3 ; en flottants, prévoir une tolérance ; $\dim \ker = 2$.

Correction de l'exercice 5 - Matrices de Hilbert : le piège des flottants

a) Même algorithme, même matrice : seule la représentation des nombres change. L'erreur croît de façon **explosive** avec n : à $n = 12$, la première décimale est déjà fautive.

b) H_n est inversible mais **presque singulière** : son déterminant est minuscule et décroît

très vite avec n . Les colonnes sont presque liées ; une infime perturbation du second membre déplace beaucoup la solution.

c) Chaque coefficient est entaché d'une erreur relative $\approx 10^{-16}$; amplifiée par κ :

$$\text{erreur relative sur } X \approx \kappa \times 10^{-16} \approx 1$$

— (l'erreur observée, 0,09, est de cet ordre).

d) Une réponse numérique n'a de sens qu'accompagnée d'une estimation du **conditionnement** ; pour les problèmes mal conditionnés, on reformule le problème (autre base, par exemple polynômes orthogonaux) ou on calcule en précision augmentée.

Réponse. Erreurs 10^{-13} , 10^{-7} , 10^{-1} en flottants pour $n = 4, 8, 12$; exacte en fractions ; cause : conditionnement $\approx 10^{16}$.

Correction de l'exercice 6 - Méthode de la puissance

a)

```
A = [[2, 1], [1, 2]]
x = [1.0, 0.0]
for k in range(1, 21):
    y = [A[0][0] * x[0] + A[0][1] * x[1], A[1][0] * x[0] + A[1][1] * x[1]]
    lam = max(abs(y[0]), abs(y[1]))
    x = [y[0] / lam, y[1] / lam]
```

b) $A(1, 0) = (2, 1) : \lambda = 2, x = (1; 0,5)$. $A(1; 0,5) = (2,5; 2) : \lambda = 2,5, x = (1; 0,8) \checkmark$

c) $(1, 1)$ et $(1, -1)$ sont propres pour 3 et 1 :

$$A^k(1, 0) = \frac{3^k}{2}(1, 1) + \frac{1}{2}(1, -1) = \frac{3^k}{2}[(1, 1) + 3^{-k}(1, -1)]$$

— La composante parasite décroît comme $\left(\frac{1}{3}\right)^k = \left|\frac{\lambda_2}{\lambda_1}\right|^k$: à $k = 20$, $3^{-20} \approx 3 \times 10^{-10}$, cohérent avec l'écart $5,7 \times 10^{-10}$ affiché pour x_2 .

d) Elle n'utilise que des **produits matrice-vecteur** (coût proportionnel au nombre de coefficients non nuls), sans jamais modifier ni factoriser la matrice : c'est le principe de PageRank (exercice 15).

Réponse. $\lambda \rightarrow 3, x \rightarrow (1, 1)$; vitesse $\left(\frac{1}{3}\right)^k = \left|\frac{\lambda_2}{\lambda_1}\right|^k$.

Correction de l'exercice 7 - Méthode itérative de Jacobi

a) $y = 1 - 4x$, puis $2x + 5 - 20x = 2 : x = \frac{1}{6}, y = \frac{1}{3}$.

b)

```
x, y = 0.0, 0.0
k = 0
while True:
    nx, ny = (1 - y) / 4, (2 - 2 * x) / 5
    k = k + 1
    if abs(nx - x) + abs(ny - y) < 1e-10:
        x, y = nx, ny
        break
    x, y = nx, ny
```

— $|4| > |1|$ et $|5| > |2|$: matrice à **diagonale strictement dominante**, ce qui garantit la convergence de Jacobi.

c) $e_{k+1}^x = -\frac{1}{4}e_k^y$ et $e_{k+1}^y = -\frac{2}{5}e_k^x$, donc

$$e_{k+2}^x = -\frac{1}{4}\left(-\frac{2}{5}e_k^x\right) = \frac{1}{10}e_k^x$$

— Facteur $\sqrt{0,1} \approx 0,32$ par itération, soit 0,5 décimale : ≈ 20 itérations pour 10^{-10} ✓

d) Pour de très grands systèmes **creux** (discretisations d'équations aux dérivées partielles) : chaque itération coûte peu et ne remplit pas la matrice, alors que Gauss coûte $\frac{n^3}{3}$ et crée des coefficients non nuls.

Réponse. $(x, y) = (\frac{1}{6}, \frac{1}{3})$; 21 itérations ; erreur divisée par 10 tous les deux pas.

Correction de l'exercice 8 - Droite des moindres carrés

a)

```
X = [1, 2, 3, 4, 5]
Y = [2, 3, 5, 4, 6]
n = len(X)
mx = sum(X) / n
my = sum(Y) / n
cov = sum((X[i] - mx) * (Y[i] - my) for i in range(n)) / n
vx = sum((X[i] - mx) ** 2 for i in range(n)) / n
a = cov / vx
b = my - a * mx
print(a, b, sum((Y[i] - a * X[i] - b) ** 2 for i in range(n)))
```

b) $\bar{x} = 3, \bar{y} = 4$; $\text{cov} = \frac{4+1+0+0+4}{5} = \frac{9}{5}$, $V(x) = 2$, $a = 0,9$, $b = 4 - 2,7 = 1,3$. Résidus : $-0,2, -0,1, 1, -0,9, 0,2$; carrés : $0,04 + 0,01 + 1 + 0,81 + 0,04 = 1,9$ ✓

c) $\frac{\partial S}{\partial b} = -2 \sum (y_i - ax_i - b) = 0$ donne $b = \bar{y} - a\bar{x}$; $\frac{\partial S}{\partial a} = -2 \sum x_i(y_i - ax_i - b) = 0$ donne alors

$$a \sum (x_i - \bar{x})^2 = \sum (x_i - \bar{x})(y_i - \bar{y})$$

— (S est un trinôme convexe en (a, b) : ce point critique est le minimum.)

d) $y \approx 0,9 \times 6 + 1,3 = 6,7$: extrapolation hors des données, à prendre avec prudence (cinq points, résidus d'amplitude 1).

Réponse. $y = 0,9x + 1,3$, somme des carrés 1,9 ; prévision 6,7 pour $x = 6$.

Correction de l'exercice 9 - Puissances d'une matrice de transition

a)

```
def prod(A, B):
    return [[sum(A[i][k] * B[k][j] for k in range(len(B))) for j in range(len(B[0]))]
            for i in range(len(A))]
```

b) Probabilité qu'il fasse beau dans **deux** jours sachant qu'il fait beau aujourd'hui : $0,8 \times 0,8 + 0,2 \times 0,4 = 0,72$ ✓ (probabilités totales sur le temps du lendemain).

c) Au bout de 20 jours, la probabilité de beau temps vaut $\frac{2}{3}$ **quel que soit** l'état initial : la chaîne a oublié son point de départ.

d) $\left(\frac{2}{3} \times 0,8 + \frac{1}{3} \times 0,4, \frac{2}{3} \times 0,2 + \frac{1}{3} \times 0,6\right) = \left(\frac{2}{3}, \frac{1}{3}\right)$ ✓ Trace $1,4 = 1 + 0,4$ et déterminant $0,4 = 1 \times 0,4$:

$$P^n = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{2}{3} & \frac{1}{3} \end{pmatrix} + 0,4^n (...)$$

— $0,4^{20} \approx 1,1 \times 10^{-8}$: cohérent avec l'écart $\approx 4 \times 10^{-9}$ observé.

Réponse. $P_{11}^2 = 0,72$; $P^n \rightarrow \begin{pmatrix} 2/3 & 1/3 \\ 2/3 & 1/3 \end{pmatrix}$ à la vitesse $0,4^n$.

Correction de l'exercice 10 - Interpolation de Lagrange

a)

```
from fractions import Fraction
```

```
def lagrange(points, x):
    s = 0
    for i, (xi, yi) in enumerate(points):
        L = 1
        for j, (xj, yj) in enumerate(points):
            if j != i:
                L = L * Fraction(x - xj, xi - xj)
        s = s + yi * L
    return s
```

b) $P(4) = 16 - 8 + 3 = 11$ ✓ ; $P(0) = 3$ ✓ ; aux nœuds, $L_i(x_j) = 1$ si $i = j$, 0 sinon : on retrouve les y_i .

c) Deux boucles imbriquées de n tours :

coût $\approx n^2$ opérations par évaluation

d) Avec beaucoup de points équidistants, le polynôme interpolateur peut **osciller violemment** près des bords (exemple de Runge : $\frac{1}{1+25x^2}$ sur $[-1, 1]$) : l'erreur augmente avec le degré. On préfère des points de Tchebychev ou des splines (polynômes par morceaux).

Réponse. $P(4) = 11$, $P(0) = 3$; coût n^2 ; attention aux oscillations de Runge.

Correction de l'exercice 11 - Rotation répétée et dérive numérique

a) R^{12} est la rotation d'angle $12 \times \frac{\pi}{6} = 2\pi$.

$$R^{12} = \begin{pmatrix} \cos 2\pi & -\sin 2\pi \\ \sin 2\pi & \cos 2\pi \end{pmatrix} = I_2$$

b) $\cos \frac{\pi}{6}$ et $\sin \frac{\pi}{6}$ sont arrondis ($\sqrt{3}$ irrationnel), et chaque produit ajoute une erreur $\approx 10^{-16}$; 12 produits donnent $\approx 10^{-15}$: dérive faible mais **non nulle**.

c) Chaque étape ajoute une erreur relative $\approx 10^{-16}$; après 10^6 étapes, l'erreur peut atteindre 10^{-10} (accumulation linéaire dans le pire cas, plutôt $\sqrt{10^6} \times 10^{-16} = 10^{-13}$ si les erreurs se compensent au hasard). La matrice cesse d'être exactement orthogonale : les longueurs dérivent.

d) Recalculer l'angle total et construire directement la rotation d'angle $k\theta$ (pas d'accumulation) ; ou **réorthonormaliser** périodiquement la matrice (Gram-Schmidt), comme en infographie 3D.

Réponse. $R^{12} = I_2$; erreur flottante 10^{-15} ; dérive croissante sur de longues simulations, à corriger.

Correction de l'exercice 12 – Fibonacci : matrice exacte contre formule de Binet

a)

```
def puiss(A, n):
    R = [[1, 0], [0, 1]]
    while n > 0:
        if n % 2 == 1:
            R = prod(R, A)
        A = prod(A, A)
        n = n // 2
    return R

F = puiss([[1, 1], [1, 0]], 100)
print(F[0][1])
```

b) $F_{n+2} = F_{n+1} + F_n$ a pour équation caractéristique $r^2 = r + 1$, de racines $\varphi = \frac{1+\sqrt{5}}{2}$ et $\psi = \frac{1-\sqrt{5}}{2}$; les conditions $F_0 = 0, F_1 = 1$ donnent

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}, \quad |\psi^n| < \frac{1}{2} \implies F_n = \text{arrondi}\left(\frac{\varphi^n}{\sqrt{5}}\right)$$

c) La formule est exacte, mais $\varphi^{100} \approx 7,9 \times 10^{20}$ calculé en flottants n'a que ≈ 16 chiffres significatifs : les 5 ou 6 derniers chiffres sont faux. Les entiers Python, eux, sont exacts.

d) $100 = 1100100_2$ (7 chiffres) : 7 élévations au carré et 3 produits pour les chiffres 1, soit 10 produits 2×2 .

Réponse. $F_{100} = 354\,224\,848\,179\,261\,915\,075$; Binet en flottants faux sur les derniers chiffres ; 10 produits.

Correction de l'exercice 13 – Système tridiagonal : l'algorithme de Thomas

a) Matrice tridiagonale 9×9 : 2 sur la diagonale, -1 sur les deux diagonales voisines (la matrice $D_n = n + 1$ de la plaquette de déterminants), second membre $h^2 = 0,01$.

b) Une boucle de descente et une de remontée, chacune avec un nombre fixe d'opérations par ligne :

$$\text{coût} \approx 8n \text{ opérations, contre } \frac{n^3}{3} \text{ pour Gauss général}$$

$$\text{c) } y(0,5) = \frac{0,5 \times 0,5}{2} = 0,125 \checkmark ; y(0,1) = \frac{0,1 \times 0,9}{2} = 0,045 \checkmark$$

d) Pour un polynôme de degré ≤ 3 , $\frac{y(x+h) - 2y(x) + y(x-h))}{h^2} = y''(x)$ **exactement** (Taylor s'arrête : le terme d'erreur fait intervenir $y^{(4)} = 0$). La solution exacte, de degré 2, vérifie donc exactement le schéma, qui a une unique solution ($\det \neq 0$).

Réponse. Matrice tridiagonale ; coût $O(n)$; $y_5 = 0,125$; schéma exact pour une solution polynomiale de degré 2.

Correction de l'exercice 14 – Coût du pivot et temps de calcul

a) $\approx \frac{n^3}{3}$ multiplications et autant d'additions, soit $\frac{2}{3}n^3$ opérations.

b)

$$n = 100 : \approx 7 \times 10^5 \text{ (0,7 ms)}; \quad n = 1000 : \approx 7 \times 10^8 \text{ (0,7 s)}; \quad n = 10^4 : \approx 7 \times 10^{11} \text{ (} \approx 11 \text{ min)}$$

— Multiplier n par 10 multiplie le temps par 1000.

c) $\frac{2}{3}(10^8)^3 \approx 7 \times 10^{23}$ opérations, soit $\approx 2 \times 10^7$ ans. Mais la matrice est **creuse** (chaque inconnue n'interagit qu'avec ses voisines) : on utilise des méthodes itératives (Jacobi, gradient conjugué, multigrille) en $O(n)$ par itération.

d) $(10^5)^2 \times 8 = 8 \times 10^{10}$ octets, soit 80 Go : la matrice pleine ne tient même pas en mémoire d'un ordinateur courant. On ne stocke que les coefficients non nuls.

Réponse. 0,7 ms, 0,7 s, 11 min pour $n = 10^2, 10^3, 10^4$; $n = 10^8$ impossible en plein ; 80 Go pour $n = 10^5$.

Correction de l'exercice 15 - Synthèse : le classement PageRank

a)

```
liens = {0: [1, 2], 1: [2], 2: [0], 3: [2]}
n, d = 4, 0.85
r = [1 / n] * n
for it in range(100):
    nr = [(1 - d) / n] * n
    for i in liens:
        for j in liens[i]:
            nr[j] = nr[j] + d * r[i] / len(liens[i])
    r = nr
print(r)
```

b) Aucune page ne pointe vers 3 : seule la part « saut au hasard » lui revient, $\frac{0,15}{4} = 0,0375$ ✓

c) La page 1 ne reçoit qu'un lien, de 0 qui en a deux : $0,0375 + 0,85 \times \frac{0,3725}{2} \approx 0,0375 + 0,1583 = 0,1958$ ✓ La page 2, pointée par trois pages, est la mieux classée ; la page 0 hérite de **tout** le score de 2.

d) Le vecteur r est vecteur propre, pour la valeur propre 1, d'une matrice stochastique G : l'itération $r \leftarrow Gr$ est la méthode de la puissance. Le facteur $d < 1$ rend G à coefficients **strictement positifs**, ce qui garantit l'unicité du score et une convergence à la vitesse d^k (ici $0,85^{100} \approx 10^{-7}$).

Réponse. $r \approx (0,373; 0,196; 0,394; 0,0375)$: page 2 en tête ; méthode de la puissance, convergence en $0,85^k$.