TELECOM Nancy 1A 2025-26
Mathématiques Numériques MN1

Feuille 3 : systèmes linéaires, factorisations

Exercice 1. *Compléments sur les matrices triangulaires*

1. Soient A et B deux matrices triangulaires inférieures (resp. supérieures), montrer que la matrice AB est triangulaire inférieure (resp. supérieure) et que $(AB)_{ii} = a_{ii}b_{ii}$.
2. Soit T une matrice triangulaire inférieure (resp. supérieure) :
   (a) Calculer le déterminant de T ;
   (b) Quelles sont les valeurs propres de T ?
   (c) On suppose T inversible, montrer que T^{-1} est une matrice triangulaire inférieure (resp. supérieure) et que $(T^{-1})_{ii} = 1/t_{ii}$.

Exercice 2. *Equation de la chaleur en 1D et différences finies*

L'évolution de la température dans une barre de longueur L peut être modélisée par une équation de la chaleur en 1D d'espace en ne considérant que la dimension de la longueur de la barre. On choisit $L = 1$ et on note $u = u(x,t)$ la température en un point $x \in [0,1]$ de la barre et à l'instant t. La température vérifie le système suivant :

$$(P)\ \begin{cases} u_t(x,t) - \gamma u_{xx}(x,t) = f(x,t), \quad (x,t) \in]0,1[\times]0,T[& (1) \\ u(0,t) = u(1,t) = 0, \quad t \in]0,T[& (2) \\ u(x,0) = u_0(x), \quad x \in]0,1[& (3) \end{cases}$$

où on a noté les dérivées partielles $u_t = \dfrac{\partial u}{\partial t}$ et $u_{xx} = \dfrac{\partial^2 u}{\partial x^2}$. Dans (P), T est le temps final, $\gamma > 0$ est le coefficient de diffusion thermique (en m^2/s) et $f :]0,1[\times]0,T[\to \mathbb{R}$ est une fonction donnée représentant une source thermique appliquée à l'ensemble de la barre. Les conditions limites (2) en $x = 0$ et $x = 1$ sont fixées à 0 pour simplifier. Enfin dans (3), u_0 représente la température initiale donnée.

Dans cet exercice, on va construire le système linéaire résultant de la discrétisation de (P) pour en déterminer une solution approchée.

Discrétisation de (P) **par différences finies.** Soit deux entiers n et m donnés. On discrétise de façon uniforme les intervalles d'espace $[0,1]$ et de temps $[0,T]$ en introduisant les points

$$x_i = ih, \quad i \in [\![0, n+1]\!] \tag{4}$$
$$t^k = k\Delta t, \quad k \in [\![0, m]\!] \tag{5}$$

où h est le pas de discrétisation en espace donné par $h = 1/(n+1)$ et Δt est le pas de discrétisation en temps avec $\Delta t = T/m$.

On cherche alors une approximation $u_i^k \simeq u(x_i, t^k)$ de la solution exacte aux points $P_i^k = (x_i, t^k)$, en discrétisant les dérivées en espace et en temps, à l'instant $t = t^{k+1}$ de la façon suivante :

$$u_t(x_i, t^{k+1}) \simeq \frac{u(x_i, t^{k+1}) - u(x_i, t^k)}{\Delta t}, \quad u_{xx}(x_i, t^{k+1}) \simeq \frac{u(x_{i-1}, t^{k+1}) - 2u(x_i, t^{k+1}) + u(x_{i+1}, t^{k+1})}{h^2} \tag{6}$$

Le schéma numérique s'écrit alors, pour tout $k \in [\![0, m-1]\!]$ et pour tout $i \in [\![1, n]\!]$:

$$\frac{u_i^{k+1} - u_i^k}{\Delta t} = \gamma \frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{h^2} + f_i^{k+1} \tag{7}$$

où l'on a noté $f_i^{k+1} = f(x_i, t^{k+1})$. On impose aussi les conditions limites et initiale :

$$u_0^k = u_{n+1}^k = 0 \ \text{ pour tout } k \in [\![1, m]\!], \tag{8}$$
$$u_i^0 = u_0(x_i) \ \text{ pour tout } i \in [\![0, n+1]\!]. \tag{9}$$

1. Justifier les approximations (6) des dérivées.
2. On introduit le vecteur $\boldsymbol{u}^k = (u_1^k, \dots, u_n^k)^\top \in \mathbb{R}^n$. Ecrire le système (7),(8),(9) sous la forme matricielle

$$(I_d + \beta A)\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta t \boldsymbol{f}^{k+1} \text{ pour tout } k \in [\![0, m-1]\!], \tag{10}$$

$$\boldsymbol{u}^0 = (u_1^0, \dots, u_n^0)^\top \text{ donné} \tag{11}$$

avec $\beta = \gamma \frac{\Delta t}{h^2}$ et où A est une matrice de taille (n, n) qu'on explicitera et $\boldsymbol{f}^{k+1} = (f_1^{k+1}, \dots, f_n^{k+1})$.

On détermine ainsi $\boldsymbol{u}^{k+1}$ à partir de $\boldsymbol{u}^k$ en résolvant le système linéaire (10).

3. *Question subsidiaire.* Montrer que la matrice A est symétrique définie positive : pour tout $\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq 0$, on a $(A\mathbf{x} \,|\, \mathbf{x}) > 0$ où $(\cdot \,|\, \cdot)$ désigne le produit scalaire dans $\mathbb{R}^n$. Vous pourrez montrer que $(A\mathbf{x} \,|\, \mathbf{x}) = x_1^2 + x_n^2 + \sum_{i=2}^n (x_i - x_{i-1})^2$. En déduire que le système linéaire (10) admet une unique solution $\boldsymbol{u}^{k+1}$ quand $\boldsymbol{u}^k$ est connu.

Exercice 3*. *Equation de la chaleur : on passe au 2D !*

L'équation de la chaleur en dimension 2 d'espace consiste à chercher une température $u = u(x, y, t)$ en un point $(x, y) \in \Omega$ d'un domaine Ω et à l'instant $t > 0$. On choisit le domaine rectangulaire $\Omega =]0, 1[\times]0, 1[$ et la température u vérifie

$$(P) \begin{cases} u_t - \gamma(u_{xx} + u_{yy}) = f(x, y, t) & \text{pour } (x, y, t) \in \Omega \times]0, T[, \\ u = 0 & \text{sur le bord du rectangle } \partial\Omega, \\ u(x, y, 0) = u_0(x, y) & \text{pour } (x, y) \in \Omega. \end{cases}$$

Pour cela, on considère une discrétisation de $\overline{\Omega} = [0, 1] \times [0, 1]$ en introduisant les points $P_{i,j} = (x_i, y_j)$ avec

$$\begin{aligned} x_i &= i\,h, \quad i \in [\![0, n+1]\!] \\ y_j &= j\,h, \quad j \in [\![0, n+1]\!]. \end{aligned}$$

avec $h = 1/(n+1)$ le pas de discrétisation en espace. Comme dans l'exercice 1, on introduit aussi les instants $t^k = k\Delta t, \quad k \in [\![0, m]\!]$ avec le pas de discrétisation en temps $\Delta t = T/m$.

On cherche alors des approximations $u_{i,j}^k$ de la solution exacte u de (P) au point $P_{i,j}$ et à l'instant t^k, c'est-à-dire $u_{i,j}^k \simeq u(x_i, y_j, t^k)$. Pour déterminer $u_{i,j}^k$, on utilise les mêmes approximations des dérivées faites dans l'exercice 1. La dérivée seconde u_{yy} est approchée de la façon analogue à la dérivée u_{xx} (mais dans la direction y bien sûr, au lieu de x), ce qui conduit au schéma suivant :

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} - \gamma \left(\frac{u_{i-1,j}^{k+1} - 2u_{i,j}^{k+1} + u_{i+1,j}^{k+1}}{h^2} + \frac{u_{i,j-1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j+1}^{k+1}}{h^2} \right) = f_{i,j}^{k+1} \tag{1}$$

avec $f_{i,j}^{k+1} = f(x_i, y_j, t^{k+1})$.

On introduit le vecteur $\mathbf{u}^k = \left(u_{11}^k, u_{12}^k, \cdots, u_{1n}^k,\ u_{21}^k, \cdots, u_{2n}^k,\ \cdots, u_{n1}^k, \cdots, u_{nn}^k\right)^\top \in \mathbb{R}^{n^2}$, c'est-à-dire qu'on ordonne les noeuds du maillage intérieur *de bas en haut puis de la gauche vers la droite.*

1. Ecrire les conditions limites pour les approximations $u_{i,j}^k$.
2. Ecrire alors le système sous la forme matricielle :

$$(I_d + \beta A)\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta t \boldsymbol{f}^{k+1} \text{ pour tout } k \in [\![0, m-1]\!], \tag{2}$$

$$\boldsymbol{u}^0 \text{ donné} \tag{3}$$

avec $\beta = \gamma \frac{\Delta t}{h^2}$ et $\boldsymbol{f}^{k+1} \in \mathbb{R}^{n^2}$. La matrice A de taille (n^2, n^2), est *tridiagonale par blocs* :

$$A = \begin{pmatrix} D_n & E_n & & 0 \\ E_n & \ddots & \ddots & \\ & \ddots & \ddots & E_n \\ 0 & & E_n & D_n \end{pmatrix} \tag{4}$$

où les matrices D_n et E_n de taille (n, n) sont à déterminer.

Exercice 4. *Méthode de Gauss-LU à la main !*

Résoudre le système linéaire $Ax = b$ par la méthode de Gauss (sans échange de lignes) avec :

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{pmatrix}, \; b = \begin{pmatrix} 2 \\ 10 \\ 44 \\ 190 \end{pmatrix}.$$

On mettra bien en évidence les 3 matrices d'élimination successives $M^{(1)}$, $M^{(2)}$ et $M^{(3)}$ et leurs inverses $L^{(k)}$. On rappelle qu'à l'étape k on a $A^{(k)} = M^{(k)}A^{(k-1)}$ et de même $b^{(k)} = M^{(k)}b^{(k-1)}$ avec :

$$M^{(k)} = I - z^{(k)}(e^k)^\top, \quad L^{(k)} = I + z^{(k)}(e^k)^\top$$

où le vecteur $z^{(k)}$ contient les coefficients permettant à l'étape k d'éliminer la variable x_k des équations $k+1, \ldots, n$.

Pour être proche des algorithmes effectivement utilisés, on remarquera qu'on peut enregistrer au fur et à mesure ces coefficients à la place des zéros obtenus, ce qui permet d'obtenir la matrice L de la factorisation LU de la matrice A sans calculs supplémentaires.

Dans un second temps vérifier que le vecteur $b^{(3)}$ (obtenu suite aux 3 étapes d'élimination de Gauss) est bien égal à la solution y du système linéaire $Ly = b$.

Exercice 5. *Unicité de la factorisation $A = LU$*

Soit A une matrice inversible qui admet une factorisation $A = LU$ (c'est à dire L est triangulaire inférieure à diagonale unité (tous les éléments diagonaux sont égaux à 1) et U est une matrice triangulaire supérieure. Démontrer l'unicité de la factorisation (aide : on suppose donc qu'il existe une autre factorisation $A = \tilde{L}\tilde{U}$ puis on montre que $\tilde{L} = L$ et $\tilde{U} = U$; pour cela on utilisera les résultats de l'exercice précédent sur les matrices triangulaires.).

Exercice 6*. *Compte d'opérations*

1. écrire un algorithme effectuant la factorisation $A = LU$ en place (cf cours) et calculer le nombre d'opérations (la complexité de cet algorithme) en fonction de l'ordre n de la matrice.
2. étant donné une factorisation "en place" écrire l'algorithme de "descente remontée" permettant de résoudre un système linéaire $Ax = b$. De même calculer le nombre d'opérations.

Exercice 7. *Importance du pivotage dans la méthode de Gauss et stratégie de pivot partiel : la décomposition $PA = LU$.*

Montrer sur l'exemple ci-dessous que la méthode de Gauss sans pivotage donne des résultats catastrophiques si l'on considère une machine hypothétique travaillant avec l' ensemble de flottants $\mathbb{F}(10, 4, ., .)$ et une arithmétique de type IEEE (c-a-d que si u et v sont 2 flottants alors $u \odot v = fl(u \cdot v)$, tant que l'on ne rencontre pas d'*overflow* ou d'*underflow*, $\cdot$ désignant l'une des 4 opérations et $\odot$ l'opération machine correspondante).

$$\begin{cases} 5 \cdot 10^{-5}x + y &= 1 \\ x - y &= 0 \end{cases}$$

Remarque : cet exemple explique pourquoi la factorisation standard est $PA = LU$ et non $A = LU$; la factorisation $PA = LU$ est obtenue en cherchant à chaque étape d'élimination le pivot le plus grand en valeur absolue. Une étape s'écrit alors matriciellement :

$$A^{(k)} = M^{(k)}P^{(k)}A^{(k-1)}$$

où $P^{(k)}$ est une matrice de permutation élémentaire (c-a-d une permuation qui n'échange que 2 éléments, ici k avec $i \geq k$ où i est l'indice du pivot choisi). On a vu dans le cours que si A est inversible alors une telle factorisation existe toujours. Son utilisation pour résoudre un système linéaire est toujours simple : on procède comme avec une factorisation $A = LU$ mais en l'appliquant sur le second membre $b' = Pb$.

Exercice 8. *Factorisation PLU en python.*

Le module `scipy` de `python` dispose d'une fonction pour déterminer une factorisation de type LU d'une matrice A, plus précisément pour calculer la décomposition $A = PLU$ où P est une matrice de permutation provenant d'une stratégie de *pivot partiel**. La fonction `lu` du sous-module `linalg` de `scipy` calcule une telle décomposition. Le script ci-dessous détermine la factorisation d'une matrice A déclarée comme un tableau *array* du module `numpy`†.

```
import numpy as np
import scipy
import scipy.linalg   # SciPy Linear Algebra Library

A = np.array([ 15, 14], [5,4]])
# Factorisation A = PLU
P, L, U = scipy.linalg.lu(A)

# Vérification et calcul du résidu (l'opérateur @ entre deux matrices calcule leur produit)
residu = np.max(np.abs(A - P@L@U))  # coefficient le plus grand de la matrice de la différence
print('résidu=',residu)
```

Calculer les factorisations $A = PLU$ des matrices suivantes et vérifier que la stratégie de *pivot partiel* s'applique bien c'est-à-dire que la matrice P est différente de la *matrice identité*.

$$i)\ A = \begin{bmatrix} 5 & 4 \\ 15 & 14 \end{bmatrix} \qquad ii)\ A = \begin{bmatrix} 5\cdot 10^{-5} & 1 \\ 1 & -1 \end{bmatrix} \text{ (cf. ex. 7)}$$

$$iii)\ A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{bmatrix} \text{ (cf. ex. 4)}$$

Remarque. On ne cherche pas ici à résoudre de système linéaire. Si c'est le cas, il vaut mieux utiliser la fonction `scipy.linalg.lu_factor()` qui retourne une factorisation "en place" c'est-à-dire avec en retour une seule matrice lu qui contient les éléments non nuls de U dans sa partie triangulaire supérieure et ceux de L sans les "1" de la diagonale dans sa partie triangulaire inférieure. A partir de ce tableau lu, Il faut alors d'utiliser la fonction `lu_solve` pour résoudre un système linéaire.

Exercice 9*. *Décomposition LU d'une matrice tridiagonale*

Dans cet exercice, on va déterminer la factorisation LU d'une matrice A tridiagonale en tenant compte de la structure particulière de A. Le coût de calcul correspondant est beaucoup moins important qu'une factorisation "standard" (pivot de Gauss) de A. Soit A la matrice carrée d'ordre n définie ci-dessous, que l'on veut décomposer en $A = LU$ avec :

$$A = \begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix},\ L = \begin{pmatrix} 1 & & & & 0 \\ a_2^* & 1 & & & \\ & \ddots & \ddots & & \\ & & a_{n-1}^* & 1 & \\ 0 & & & a_n^* & 1 \end{pmatrix},\ U = \begin{pmatrix} b_1^* & c_1^* & & & 0 \\ & b_2^* & c_2^* & & \\ & & \ddots & \ddots & \\ & & & b_{n-1}^* & c_{n-1}^* \\ 0 & & & & b_n^* \end{pmatrix}.$$

On veut déterminer les coefficients a_i^*, b_i^*, c_i^* en identifiant le produit matriciel LU avec A.

---

*. Comme dans l'exercice 7, la stratégie de pivot partiel conduit à une décomposition de la forme $PA = LU$ où P est une matrice de permutation qui prend en compte la permutation des lignes de la matrice A où à chaque étape d'élimination de Gauss, le coefficient de la matrice servant de pivot est choisi comme le plus grand en valeur absolue dans la colonne. La matrice de permutation P est orthogonale ($P^{-1} = P^\top$) d'où la décomposition $A = P^\top LU$.

†. Le module `numpy` ne propose pas de décomposition LU mais la fonction `np.solve(A,b)` pour résoudre le système linéaire $Ax = b$ utilise une factorisation LU, non directement accessible.

1. On suppose dans tout l'exercice que A est inversible. En supposant que la décomposition précédente $A = LU$ existe, montrer que $b_k^* \neq 0$, $\forall k \in [\![1, n]\!]$.
2. Soit $k \in [\![1, n-1]\!]$. Calculer le coefficient $(LU)_{k,k+1}$ et en identifiant avec $A_{k,k+1}$ en déduire que $c_k^* = c_k$.
3. Soit $k \in [\![2, n]\!]$. Calculer le coefficient $(LU)_{k,k-1}$ et en déduire une relation entre les coefficients a_k^*, b_{k-1}^* et a_k.
4. Calculer le coefficient $(LU)_{1,1}$ et en déduire b_1^* en fonction des coefficients de A.
   Soit $k \in [\![2, n]\!]$. Calculer le coefficient $(LU)_{k,k}$. En déduire une relation entre a_k^*, b_k^* et les coefficients de A.
5. Ecrire l'ensemble des relations obtenues et montrer qu'elles permettent de déterminer tous les coefficients de L et U.
6. Ecrire un algorithme pour calculer tous les coefficients a_i^*, b_i^*, c_i^*. Calculer le nombre d'opérations et le comparer au nombre d'opérations d'une factorisation LU standard.

Exercice 10. *SVD et compression d'images.*

Dans cette exercice, on applique la décomposition en valeurs singulières (SVD) d'une matrice pour comprimer une image en niveau de gris ou en couleur (format RGB). Une image en niveau de gris est représentée par une matrice dont les coefficients sont des nombres réels entre 0 (noir) et 1 (blanc) correspondants au niveau de gris de chaque pixel. Une image couleur en format RGB (Red/Green/Blue) est constituée de 3 matrices (pour chacune des 3 couleurs).

On rappelle le procédé de compression par SVD (cf. Cours).

• *Décomposition SVD.* Pour une matrice $A \in \mathbb{R}^{m\times n}$, il existe deux matrices orthogonales $U \in \mathbb{R}^{m\times m}$ et $V \in \mathbb{R}^{n\times n}$ telles que $A = U\Sigma V^\top$ où $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$, $p = \min(m, n)$ et avec les valeurs singulières ordonnées de façon décroissante $\sigma_1 \geq \cdots \geq \sigma_p \geq 0$. La décomposition SVD de A peut s'écrire aussi

$$A = \sum_{i=1}^{p} \sigma_i U^i {V^i}^\top \tag{5}$$

où U^i (resp. V^i) désigne la i-ème colonne de U (resp. V).

• *Approximation de matrice.* Pour $1 \leq k \leq p = \min(m, n)$, on introduit l'approximation matricielle A_k obtenue en ne retenant que les k premières valeurs singulières dans (5) :

$$A_k = \sum_{i=1}^{k} \sigma_i U^i {V^i}^\top \tag{6}$$

La matrice A_k contient l'image compressée.

• *Python et SVD*. En Python, on obtient la décomposition SVD d'une matrice `A` avec la fonction `svd` du module `linalg` de `numpy` :

```
U, S, V = numpy.linalg.svd(A, full_matrices=True)
```

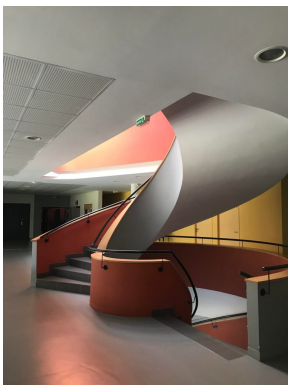
Le produit vecteur × vecteur qui intervient dans (5) et (6) retourne une matrice. Pour deux vecteurs $u \in \mathbb{R}^m$ et $v \in \mathbb{R}^n$, on a $M := u\,v^\top \in \mathbb{R}^{m\times n}$ avec

$$(u\,v^\top)_{ij} = u_i v_j \tag{7}$$

Le calcul de $u\,v^\top$ en Python peut se faire directement avec la fonction `outer` de `numpy`. L'instruction s'écrit `M=np.outer(u, v)`

1. Récupérer sur ARCHE l'image `telecomnancy.png`. Il s'agit d'une image couleur au format RGB qui peut être lue en Python avec le module `image` de `matplotlib` :

```
import matplotlib.image as mpimg
imgRGB=mpimg.imread("telecomnancy.png")
```



L'image contenue dans `imgRGB` est un tableau (`ndarray`) de `numpy` de taille `(m,n,3)` avec les 3 matrices `R=imgRGB(m,n,0)`, `G=imgRGB(m,n,1)`, `B=imgRGB(m,n,2)`. On obtient une image `(m,n)` en niveau de gris (une seule matrice `A`) en faisant la moyenne des 3 matrices contenues dans imgRGB avec :

```
A=numpy.mean(imgRGB,axis=-1)
```

L'argument `axis=-1` signifie que la moyenne (arithmétique) est faite selon la dernière dimension.

Compléter le script `compression_svd_squelette.py` pour calculer `Ak` selon la formule (6). Faire varier le nombre de valeurs singulières k pour observer la variation de qualité obtenue. Calculer à chaque fois le taux de compression τ=(taille de `Ak`)/(taille de `A`).

2. Modifier le script précédent pour effectuer les décompositions SVD sur les 3 matrices R, G et B de l'image couleur et déterminer les approximations R_k, G_k et B_k correspondantes qui définissent l'approximation "couleur" de l'image initiale.

Exercice 11*. *Matrice de Householder*

Pour obtenir une factorisation $A = QR$ d'une matrice réelle carrée A avec Q une matrice orthogonale et R une matrice triangulaire supérieure, une méthode consiste à utiliser des matrices de Householder (cf. cours). La matrice de Householder associée à un vecteur $v \in \mathbb{R}^n$, $v \neq 0$ est définie par

$$H(v) = I_d - 2\frac{vv^\top}{\|v\|^2}.$$

Le but de cet exercice est de montrer qu'avec deux vecteurs non-colinéaires quelconques, on peut trouver un vecteur v tel que la matrice $H(v)$ transforme l'un des vecteurs en un vecteur colinéaire à l'autre. Plus précisément, soit $x, y \in \mathbb{R}^n$, $x \neq 0$, $y \neq 0$ deux vecteurs non-colinéaires. On veut alors montrer qu'il existe un vecteur $v \neq 0$ et $\alpha \in \mathbb{R}$ tels que $H(v)x = \alpha y$.

1. Montrer qu'on peut toujours se ramener au cas où $\|y\| = 1$.
2. On suppose désormais que $\|y\| = 1$. Soit $v^+ = x + \|x\|y$ et $v^- = x - \|x\|y$.
   (a) Calculer $\|v^+\|^2$ et $v^{+\top}x = (v^+ \mid x)$ en fonction de x et y. Même chose avec v^-.
   (b) En déduire le vecteur $H(v^+)x$ (resp. $H(v^-)x$) et montrer qu'il possède bien la propriété attendue.
3. *Application à la factorisation QR : une itération sur un exemple.* Soit la matrice

$$A = \begin{pmatrix} 12 & -51 & 4 \\ 6 & 167 & -68 \\ -4 & 24 & -41 \end{pmatrix}$$

   On note A^1 le vecteur de la première colonne de A et $e^1 = (1, 0, 0)^\top$. Transformer A pour mettre des zéros sous le coefficient de la diagonale de la première colonne revient à rendre A^1 colinéaire à e^1. Calculer la matrice de Householder $H^{(1)}$ qui réalise cela, puis calculer la matrice $A^{(1)}$ telle que $(A^{(1)})^i = H^{(1)}A^i$ pour tout i (on applique à chacun des vecteurs colonnes de A la matrice $H^{(1)}$). Une deuxième itération consisterait à procéder de façon analogue avec la sous-matrice carrée $(2, 2)$ et pour obtenir au final la matrice R (ne pas faire cette étape).
4. Le module `numpy` de `python` contient une fonction pour la décomposition QR d'une matrice `A` (définie comme un `array` de `numpy`) :

   ```
   Q,R = numpy.linalg.qr(A)
   ```

   Déterminer la factorisation QR de A avec la fonction `qr` de `numpy`.
5. Indiquer comment résoudre un système linéaire à partir d'une factorisation QR.