Calcul formel

5. Algèbre linéaire

L'algèbre linéaire est la partie des mathématiques consacrée à l'étude des matrices et des structures vectorielles. Beaucoup de problèmes se ramènent ou s'approchent par des problèmes linéaires, pour lesquels il existe souvent des solutions efficaces.

5.1. Opérations de base

Travaux pratiques 1.
Soient les matrices :

$$A=\begin{pmatrix}1&2&3\\-1&0&1\\0&1&0\end{pmatrix} \quad B=\begin{pmatrix}2&-1&0\\-1&0&1\end{pmatrix} \quad I=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \quad u=\begin{pmatrix}1\\x\\x^2\end{pmatrix} \quad v=\begin{pmatrix}1\\0\\1\end{pmatrix}$$

1. Calculer tous les produits possibles à partir de A, B, u et v.
2. Calculer $(A-I)^7$ et en extraire le coefficient de la deuxième ligne, troisième colonne.
3. Calculer A^{-1}. Calculer la trace de A^{-1} (c'est-à-dire la somme des coefficients sur la diagonale).

1. On définit une matrice n lignes et p colonnes par la commande

   `matrix(n,p,[ [ligne1],[ligne2],... ])`

   En prenant les données de l'énoncé :

   Code 1 (*alglin.sage (1)*).
   ```
   A = matrix(3,3,[[1,2,3],[-1,0,1],[0,1,0]])
   B = matrix(2,3,[[2,-1,0],[-1,0,1]])
   I = identity_matrix(3)
   u = matrix(3,1,[1,x,x^2])
   v = matrix(3,1,[1,0,1])
   ```

   On multiplie deux matrices par `A*B`. Cette opération est possible seulement si le nombre de colonnes de A est égal au nombre de lignes de B. Les produits possibles sont donc ici : $B\times A$, $A\times u$, $A\times v$, $B\times u$ et $B\times v$.

2. Ayant défini la matrice identité I (par `I = identity_matrix(3)`), on calcule $M=(A-I)^7$ par `(A-I)^7`, ce qui renvoie

   $$(A-I)^7=\begin{pmatrix}-16&-46&11\\-25&-73&153\\32&57&-137\end{pmatrix}.$$

   Le coefficient en position $(2,3)$ d'une matrice M est celui sur la deuxième ligne et la troisième colonne. Donc ici c'est 153. Une matrice A se comporte en Sage comme une liste à double entrée et on accède aux coefficients par la commande `A[i,j]` (`i` pour les lignes, `j` pour les colonnes). Attention ! Les listes étant indexées à partir du rang 0, les premières valeurs des indices `i`, `j` sont 0. Le coefficient en position $(2,3)$ s'obtient donc par la commande :

```
((A-I)^7)[1,2]
```

Faites bien attention au décalage des deux indices.

3. L'inverse d'une matrice `A` se calcule par `A^-1` ou `A.inverse()`. Ce qui donne

$$A^{-1} = \begin{pmatrix} \frac{1}{4} & -\frac{3}{4} & -\frac{1}{2} \\ 0 & 0 & 1 \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{2} \end{pmatrix}.$$

La trace se calcule par une simple somme des éléments de la diagonale principale :

```
sum(A^(-1)[i,i] for i in range(3) )
```

et vaut ici $-\frac{1}{4}$. Mais on peut aussi calculer la trace avec la méthode `trace` :

```
A.inverse().trace()
```

qui donne encore $-\frac{1}{4}$.

Remarque.

- Noter encore une fois que pour une matrice de taille n, les indices des lignes et colonnes varient de 0 à $n-1$.
- On peut préciser le corps sur lequel on travaille, pour nous ce sera le plus souvent $\mathbb{K} = \mathbb{Q}$. Par exemple : `A = matrix(QQ, [[1,2],[3,4]])`.
- Avec Sage on peut aussi définir des vecteurs par `vector([x1,x2,...,xn])`. Un vecteur peut représenter soit un vecteur ligne, soit un vecteur colonne. On peut multiplier une matrice par un vecteur (à gauche ou à droite).
- Si on définit un vecteur, par exemple `v = vector([1, 2, 3, 4])` alors `L = matrix(v)` renvoie la matrice ligne correspondante. `C = L.transpose()` renvoie la matrice colonne correspondante.

Travaux pratiques 2.
Pour une matrice $A \in M_n(\mathbb{K})$ inversible et des vecteurs colonnes u et v à n lignes, on définit $\alpha \in \mathbb{K}$ par :

$$\alpha = 1 + v^T A^{-1} u.$$

La formule de Sherman-Morrison affirme que si $\alpha \neq 0$ alors

$$\left(A + uv^T\right)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{\alpha}$$

Prouver par le calcul formel cette formule pour les matrices de taille 2×2 et 3×3.

Connaissant l'inverse de A, cette formule permet de calculer à moindre coût l'inverse d'une déformation de A par une matrice de rang 1. Le code ne pose pas de problème.

Code 2 (*alglin.sage (2)*).

```
var('a,b,c,d,x,y,xx,yy')
A = matrix(2, 2, [[a, b], [c, d]])
u = matrix(2, 1, [x, y])
v = matrix(2, 1, [xx, yy])

Ainv = A^(-1)
alpha_matrice = 1 + v.transpose() * Ainv * u
alpha_reel = alpha_matrice[0,0]
B = (A + u * v.transpose())^(-1)
BB  = Ainv - 1/alpha_reel*( Ainv * u * v.transpose() * Ainv )
C = B - BB

for i in range(A.nrows()):
    for j in range(A.ncols()):
        print( "indices␣:",i,j,"coeff␣:", (C[i,j]).full_simplify() )
```

- On commence par définir les coefficients qui seront nos variables (ici pour $n = 2$).
- On calcule α par la formule $\alpha = 1 + v^T A^{-1} u$, qui est une matrice de taille 1×1 (`alpha_matrice`), identifiée à un réel (`alpha_reel`).
- On calcule ensuite les termes de gauche (`B`) et droite (`BB`) de la formule à prouver. Pour la matrice `C = B - BB`, on vérifie (après simplification) que tous ses coefficients sont nuls. Ce qui prouve l'égalité cherchée.

- On pourrait également obtenir directement le résultat en demandant à Sage : `C.simplify_rational() == matrix(2,2,0)`.

Pour une preuve à la main et en toute dimension, multiplier $\left(A+uv^T\right)$ par $A^{-1}-\frac{A^{-1}uv^TA^{-1}}{\alpha}$.

5.2. Réduction de Gauss, calcul de l'inverse

Le but de cette section est de mettre en œuvre la méthode de Gauss. Cette méthode est valable pour des systèmes linéaires (ou des matrices) de taille quelconque, mais ici on se limite au calcul de l'inverse d'une matrice (qui est obligatoirement carrée). N'hésitez pas à d'abord relire votre cours sur la méthode de Gauss.

Travaux pratiques 3.

1. Réaliser trois fonctions qui transforment une matrice en fonction des opérations élémentaires sur les lignes :
   - $L_i \leftarrow cL_i$ avec $c \neq 0$: on multiplie une ligne par un réel ;
   - $L_i \leftarrow L_i + cL_j$: on ajoute à la ligne L_i un multiple d'une autre ligne L_j ;
   - $L_i \leftrightarrow L_j$: on échange deux lignes.
2. À l'aide de ces transformations élémentaires, transformer par la méthode de Gauss une matrice inversible en une matrice échelonnée (c'est-à-dire ici, en partant d'une matrice carrée, obtenir une matrice triangulaire supérieure).
3. Toujours à l'aide de ces transformations et de la méthode de Gauss, transformer cette matrice échelonnée en une matrice échelonnée et réduite (c'est-à-dire ici, en partant d'une matrice inversible, obtenir l'identité).
4. La méthode de Gauss permet de calculer l'inverse d'une matrice A :
   - Partant de A, on pose $B = I$ la matrice identité.
   - Par la méthode de Gauss et des opérations élémentaires, on transforme A en la matrice I.
   - On applique exactement les mêmes transformations à la matrice B.
   - Lorsque A s'est transformée en I alors B s'est transformée en A^{-1}.

   Modifier légèrement vos deux fonctions (issues des questions 2. et 3.) afin de calculer l'inverse d'une matrice.

Voici une matrice A, sa forme échelonnée, sa forme échelonnée réduite (l'identité), et son inverse :

$$A=\begin{pmatrix}1&2&3\\-1&0&1\\0&1&0\end{pmatrix}\quad\begin{pmatrix}1&2&3\\0&2&4\\0&0&-2\end{pmatrix}\quad\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\quad A^{-1}=\begin{pmatrix}\frac{1}{4}&-\frac{3}{4}&-\frac{1}{2}\\0&0&1\\\frac{1}{4}&\frac{1}{4}&-\frac{1}{2}\end{pmatrix}$$

1. Voici la fonction pour la deuxième opération : $L_i \leftarrow L_i + cL_j$.

Code 3 (*gauss.sage (1)*).

```
def op2(A, i, j, c):
    ''' modifie la ligne i de la matrice A en lui ajoutant c fois la ligne j '''
    A[i, :] = A[i, :] + c*A[j, :]
```

`A[i,:]` correspond à la ligne `i` (les deux points signifiant de prendre tous les indices des colonnes). Lorsque l'on exécute cette fonction sur une matrice A cela modifie la matrice (voir la remarque plus bas). En particulier cette fonction ne renvoie aucune valeur.

Les autres opérations `op1(A, i, c)`, `op3(A, i, j)` se définissent de manière analogue.

2. Il s'agit de traduire la première partie de la méthode de Gauss. Pour chaque colonne p en partant de la plus à gauche :
   (a) on cherche un coefficient non nul dans cette colonne ;
   (b) on place ce coefficient en position de pivot (p,p), en permutant deux lignes par la troisième opération élémentaire ;
   (c) on annule, un par un, les coefficients qui sont sous ce pivot, par des opérations élémentaires : $L_i \leftarrow L_i + cL_p$ où $c = -\frac{a_{i,p}}{a_{p,p}}$.

Code 4 (*gauss.sage (2)*).

```
def echelonne(A):
```

```
    ''' échelonne la matrice A '''
    n = A.ncols()  # taille de la matrice
    for p in range(n):   # pivot en A[p,p]
        # a. On cherche un coeff non nul
        i = p
        while i < n and A[i, p] == 0:
            i = i + 1
        assert i < n, 'matrice non inversible'

        # b. On place la ligne avec coeff non nul en position de pivot
        op3(A, p, i)

        # c. On soustrait la ligne du pivot aux lignes dessous
        for i in range(p + 1, n):
            c = -A[i, p]/A[p, p]
            op2(A, i, p, c)
```

3. Pour passer à une forme réduite, on parcourt les colonnes en partant de la droite :
   (a) on commence par multiplier la ligne du pivot pour avoir un pivot valant 1 ;
   (b) on annule, un par un, les coefficients qui sont au-dessus de ce pivot.

Code 5 (*gauss.sage (3)*)**.**

```
def reduite(A):
    ''' réduit la matrice échelonnée A '''
    n = A.ncols()
    for p in range(n - 1, -1, -1):
        # a. On divise pour avoir un pivot valant 1
        c = 1/A[p, p]
        op1(A, p, c)

        # b. On élimine les coefficients au dessus du pivot
        for i in range(p - 1, -1, -1):
            c = -A[i, p]
            op2(A, i, p, c)
```

4. Pour calculer l'inverse, on définit deux nouvelles fonctions `echelonne_bis(A,B)` et `reduite_bis(A,B)` qui modifient chacune les deux matrices A et B. À chaque fois que l'on effectue une opération sur A, on effectue la même opération sur B. Par exemple, la dernière boucle de `echelonne_bis` contiendra les lignes `op2(A, i, p, c)` et `op2(B, i, p, c)`. C'est bien le même coefficient $c = -\frac{a_{i,p}}{a_{p,p}}$ pour les deux opérations. On obtient alors l'inverse comme ceci :

Code 6 (*gauss.sage (4)*)**.**

```
def inverse_matrice(A):
    ''' renvoie la matrice inverse de A '''
    n = A.ncols()
    AA = copy(A)
    B = identity_matrix(QQ, n)
    echelonne_bis(AA, B)
    reduite_bis(AA, B)
    return B
```

Remarque.
Si on pose `A = matrix( [[1,2],[3,4]] )`, puis `B = A`, puis que l'on modifie la matrice A par `A[0,0]= 7`. Alors la matrice `B` est aussi modifée !
Que se passe-t-il ?

- `A` ne contient pas les coefficients de la matrice, mais l'adresse où sont stockés ces coefficients (c'est plus léger de manipuler l'adresse d'une matrice que toute la matrice).
  Comme `A` et `B` pointent vers la même zone qui a été modifiée, les deux matrices `A` et `B` sont modifiées.
- Pour pouvoir définir une copie de `A`, avec une nouvelle adresse mais les mêmes coefficients, on écrit `AA = copy(A)`. Les modifications sur `AA` ne changeront pas `A`.

5.3. Application linéaire, image, noyau

Travaux pratiques 4.
Soit $f : \mathbb{R}^3 \to \mathbb{R}^4$ l'application linéaire définie par

$$f : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \longmapsto \begin{pmatrix} x+2z \\ 5x+2y+2z \\ 2x+y \\ x+y-2z \end{pmatrix}.$$

1. Expliciter la matrice A de f (dans les bases canoniques). Comment s'écrit alors l'application f ?
2. Calculer une base du noyau de f. L'application linéaire f est-elle injective ?
3. Calculer une base de l'image de f. L'application linéaire f est-elle surjective ?

1. La matrice associée est

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 5 & 2 & 2 \\ 2 & 1 & 0 \\ 1 & 1 & -2 \end{pmatrix}.$$

L'application linéaire f est alors définie par $X \mapsto Y = AX$. Voici l'implémentation avec un exemple de calcul.

Code 7 (*matlin.sage (1)*).
```
K = QQ
A = matrix(K,4,3,[[1,0,2],[5,2,2],[2,1,0],[1,1,-2]])
X = vector(K,[2,3,4])
Y = A*X
```

2. et 3. Le noyau s'obtient par la commande `right_kernel()` et l'image par la commande `column_space()` car l'image est exactement le sous-espace vectoriel engendré par les vecteurs colonnes de la matrice.

Code 8 (*matlin.sage (2)*).
```
Ker = A.right_kernel()
Ker.basis()
Im = A.column_space()
Im.basis()
```

- Le noyau est un espace vectoriel de dimension 1 dans $\mathbb{R}^3$ engendré par $\begin{pmatrix} 2 \\ -4 \\ -1 \end{pmatrix}$. Le noyau n'étant pas trivial, l'application f n'est pas injective.
- L'image est un espace vectoriel de dimension 2 dans $\mathbb{R}^4$, dont une base est :

$$\left(\begin{pmatrix} 2 \\ 0 \\ -1 \\ -3 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 1 \\ 1 \end{pmatrix} \right).$$

L'image n'étant pas $\mathbb{R}^4$ tout entier, l'application f n'est pas non plus surjective.

- On peut tester si un vecteur donné est dans un sous-espace. Si on pose par exemple `X = vector(K,[1,5,2,1])` alors le test « `X in Im` » renvoie vrai. Ce vecteur est donc bien un élément de l'image de f.

Attention ! Il ne faut pas utiliser directement les méthodes `kernel()` et `image()` car Sage préfère travailler avec les vecteurs lignes et donc ces méthodes calculent le *noyau à gauche* (c'est-à-dire l'ensemble des X tels que $XA = 0$) et l'*image à gauche* (c'est-à-dire l'ensemble des $Y = XA$). Si vous souhaitez utiliser ces méthodes pour calculer le noyau et l'image avec notre convention habituelle (c'est-à-dire à droite) il faut le faire sur la transposée de A :

Code 9 (*matlin.sage (3)*).
```
AA = A.transpose()
Ker = AA.kernel()
Ker.basis()
Im = AA.image()
Im.basis()
```

5.4. Méthodes des moindres carrés

Si on veut résoudre un système linéaire avec plus d'équations que d'inconnues alors il n'y a pas de solution en général. On aimerait pourtant parfois trouver ce qui s'approche le plus d'une solution. Formalisons ceci : soit $A \in M_{n,p}(\mathbb{R})$ avec $n \geqslant p$ une matrice à coefficients réels, X un vecteur inconnu de taille p et B un vecteur de taille n. Il n'y a en général pas de solution X au système $AX = B$, mais on aimerait que $AX - B$ soit le plus proche du vecteur nul. Ainsi, une ***solution des moindres carrés*** est un vecteur X tel que $\|AX - B\|$ soit minimale, où la norme considérée est la norme euclidienne. Cette solution des moindres carrées est donnée par la formule :

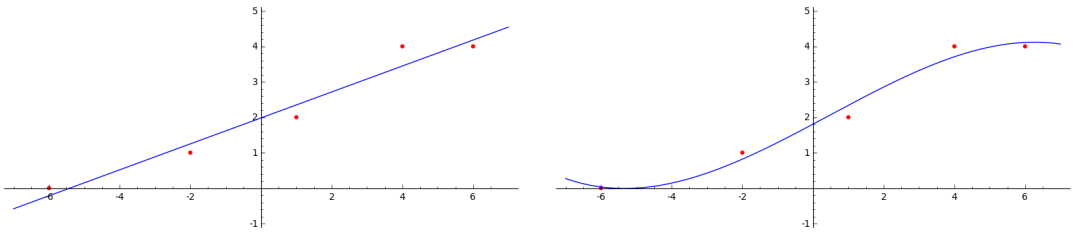
$$X = (A^T A)^{-1} A^T B \qquad (\dagger)$$

(si $n \geqslant p$ et A est de rang maximal p, ce qu'on suppose, alors on peut montrer que la matrice $A^T A$ est inversible).

Travaux pratiques 5.

1. **Régression linéaire.** On considère les points $\{(-6,0),(-2,1),(1,2),(4,4),(6,4)\}$. Trouver la droite qui approche au mieux ces points (au sens des moindres carrés).
   Indications. On pose $f(x) = a + bx$. On aimerait trouver $a, b \in \mathbb{R}$ tels que $f(x_i) = y_i$ pour tous les points (x_i, y_i) donnés. Transformer le problème en un problème des moindres carrés : quel est le vecteur inconnu X ? quelle est la matrice A? quel est le second membre B ?
2. **Interpolation polynomiale.** Pour ces mêmes points, quel polynôme de degré 3 approche au mieux ces points (au sens des moindres carrés) ?
   Indication. Cette fois $f(x) = a + bx + cx^2 + dx^3$.

Voici la droite demandée (à gauche) ainsi que le polynôme de degré 3 (à droite).



1. Bien sûr les points ne sont pas alignés, donc il n'existe pas de droite passant par tous ces points. On cherche une droite d'équation $y = a + bx$ qui minimise (le carré de) la distance entre les points et la droite. Posons $f(x) = a + bx$.
   Alors pour nos n points (x_i, y_i) donnés, on voudrait $f(x_i) = y_i$.

$$\begin{cases} f(x_1) = y_1 \\ f(x_2) = y_2 \\ \vdots \quad \vdots \\ f(x_n) = y_n \end{cases} \iff \begin{cases} a + bx_1 = y_1 \\ a + bx_2 = y_2 \\ \vdots \quad \vdots \\ a + bx_n = y_n \end{cases} \iff \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \iff AX = B.$$

On résout (de façon non exacte) notre problème $AX = B$, par la formule des moindres carrés $X = (A^T A)^{-1} A^T B$. Comme $X = \binom{a}{b}$, on obtient l'équation $y = a + bx$ de la droite des moindres carrés (voir la figure ci-dessus).

La fonction `moindres_carres` résout le problème général des moindres carrés. Il reste ensuite à définir la matrice A et le vecteur B en fonction de notre liste de points afin de trouver X. Le code est le suivant :

Code 10 (*moindres_carres.sage (1)*).

```
def moindres_carres(A, B):
    ''' renvoie le vecteur X tel que ||AX - B|| soit minimal'''
    tA = A.transpose()
    return (tA * A).inverse() * tA * B

points = [(-6, 0), (-2, 1), (1, 2), (4, 4), (6, 4)]

A = matrix([[1, p[0]] for p in points])
B = vector(p[1] for p in points)
X = moindres_carres(A, B)
```

2. L'idée est la même, mais cette fois les inconnues sont les coefficients de la fonction $f(x) = a + bx + cx^2 + dx^3$:

$$\begin{cases} f(x_1) &= y_1 \\ f(x_2) &= y_2 \\ \vdots & \vdots \\ f(x_n) &= y_n \end{cases} \iff \begin{cases} a + bx_1 + cx_1^2 + dx_1^3 &= y_1 \\ a + bx_2 + cx_2^2 + dx_2^3 &= y_2 \\ \vdots & \vdots \\ a + bx_n + cx_n^2 + dx_n^3 &= y_n \end{cases}$$

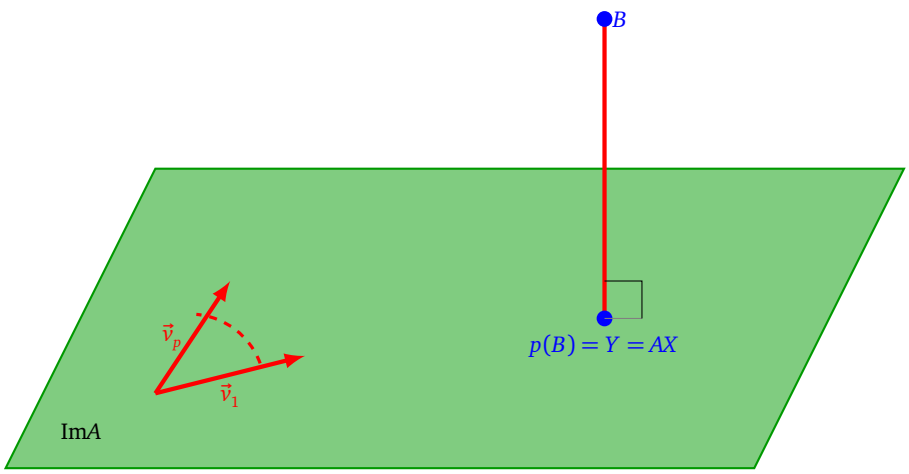
$$\iff \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \iff AX = B$$

Voici le code pour les mêmes points et une interpolation polynomiale de degré d :

Code 11 (*moindres_carres.sage (2)*).

```
d = 3   # degré
A = matrix([ [p[0]^k for k in range(d+1)]  for p in points ])
B = vector(p[1] for p in points)
X = moindres_carres(A,B)
```

Pour conclure, voici la preuve de la formule (†) des moindres carrés. Caractérisons géométriquement la condition « $\|AX - B\|$ minimale ». On note $\mathrm{Im}A$ l'image de A, c'est-à-dire le sous-espace vectoriel engendré par les vecteurs colonnes de la matrice A. Notons $p : \mathbb{R}^n \to \mathbb{R}^n$ la projection orthogonale sur $\mathrm{Im}A$. Enfin, notons $Y = p(B)$ le projeté orthogonal de B sur l'image de A.



Comme Y est dans l'image de A alors il existe X tel que $AX = Y$ et X est notre « solution » cherchée. En effet, c'est l'une des caractérisations du projeté orthogonal : $Y = p(B)$ est le point de $\mathrm{Im}A$ qui minimise la distance entre B et un point de $\mathrm{Im}A$.
Que Y soit le projeté orthogonal de B sur $\mathrm{Im}A$ signifie que le vecteur $B - Y$ est orthogonal à tout vecteur de $\mathrm{Im}A$:

$$\begin{aligned}
& \langle AV \mid B - Y \rangle = 0 \quad \forall V \in \mathbb{R}^p \\
\iff \quad & \langle AV \mid B - AX \rangle = 0 \quad \forall V \in \mathbb{R}^p \quad \text{pour } Y = AX \\
\iff \quad & (AV)^T \cdot (B - AX) = 0 \quad \forall V \in \mathbb{R}^p \quad \text{car } \langle u \mid v \rangle = u^T \cdot v \\
\iff \quad & V^T A^T (B - AX) = 0 \quad \forall V \in \mathbb{R}^p \\
\iff \quad & A^T (B - AX) = 0 \\
\iff \quad & A^T AX = A^T B \\
\iff \quad & X = (A^T A)^{-1} A^T B.
\end{aligned}$$

Remarque.
Dans la dernière ligne de cette preuve, nous supposons (et admettons) que la matrice $A^T A$ est inversible, ce qui est toujours vrai lorsque la matrice A est de rang maximal (donc de rang p, car ici $p \leqslant n$).