

Calcul de déterminants

et

systèmes matriciels

Les déterminants : quelques rappels

Définition

Le **déterminant de n vecteurs** (V_1, \dots, V_n) de \mathbb{R}^n est défini comme la **forme n -linéaire alternée** de $(\mathbb{R}^n)^n$ dans \mathbb{R} telle que : $\det(I_d) = 1$

Forme n -linéaire :

$$\det(V_1, \dots, aV_i + bU_i, \dots, V_n) = a \cdot \det(V_1, \dots, V_i, \dots, V_n) + b \cdot \det(V_1, \dots, U_i, \dots, V_n)$$

Alternée :

$$\det(V_1, \dots, V_i, \dots, V_j, \dots, V_n) + \det(V_1, \dots, V_j, \dots, V_i, \dots, V_n) = 0$$

$$\text{D'où } \det(V_1, \dots, V_i, \dots, V_i, \dots, V_n) = 0$$

Déterminant d'une matrice A : c'est le déterminant des vecteurs composant la matrice noté $\det(A)$.

Calcul théorique

$$A = (V_1, \dots, V_n) \text{ et } \forall i, V_i = \sum_{j=1}^n a_{i,j} e_j$$

Comme le déterminant est une forme n-linéaire alternée :

$$\begin{aligned} \det(\sum_{j=1}^n a_{1,j} e_j, \dots, \sum_{j=1}^n a_{n,j} e_j) &= \\ \sum_{i_1=1, \dots, i_n=1}^{i_1=n, \dots, i_n=n} a_{i_1,1} \dots a_{i_n,n} \det(e_{i_1}, \dots, e_{i_n}) \\ &= \sum_{\sigma \in \mathcal{S}_n} \prod_{i=1}^n a_{\sigma(i),i} \times \det(e_{\sigma(1)}, \dots, e_{\sigma(n)}) \end{aligned}$$

\mathcal{S}_n représente les **permutations** de $(1, \dots, n)$

Comme le déterminant est alterné :

$\det(e_{\sigma(1)}, \dots, e_{\sigma(n)}) = \epsilon(\sigma) \det(I_d)$ où $\epsilon(\sigma) = \pm 1$ est la **signature** de la permutation σ (parité de la décomposition en **transpositions**).

Au final

$$\det(A) = \sum_{\sigma \in \mathcal{S}_n} \epsilon(\sigma) \prod_{i=1}^n a_{\sigma(i),i}$$

Quelques propriétés

L'ensemble des formes n -liéaires alternées est une droite vectorielle.

$$\det(A \times B) = \det(A) \times \det(B)$$

$$\det(\lambda A) = \lambda^n \det(A)$$

$$\det(A) = \det(A^t)$$

Développement selon une ligne ou une colonne

On définit le **mineur** $A_{i,j}$ d'une matrice A de dimension $n \times n$ comme la matrice de dimension $(n-1) \times (n-1)$ issue de la matrice A privée de sa i^{ieme} ligne et de sa j^{ieme} colonne.

Alors

$$\det(A) = \sum_{i=1}^n (-1)^{i+k} a_{i,k} \det(A_{i,k}) = \sum_{j=1}^n (-1)^{j+k} a_{k,j} \det(A_{k,j})$$

$(-1)^{i+j} \det(A_{i,j})$ est appelé le **cofacteur** de $a_{i,j}$

Un exemple

$$\begin{pmatrix} 6 & 2 & -3 \\ 2 & -1 & 0 \\ -4 & 2 & -1 \end{pmatrix}$$

Cas des matrice élémentaire de type 1

$$\text{Soit } P_{k,l} = \begin{matrix} & \dots & k & \dots & l & \dots \\ \vdots & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} & \end{matrix}$$

$$\text{Alors } \det(P_{k,l}) = -1$$

Cas des matrice élémentaire de type 2

$$\text{Soit } P_k = \begin{matrix} & & k \\ \vdots & \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & \lambda & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \end{matrix}$$

$$\text{Alors } \det(P_k) = \lambda$$

Cas des matrice élémentaire de type 3

$$\text{Soit } P_{k,l} = \begin{matrix} & & \dots & k & \dots & l & \dots \\ \vdots & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & \dots & -\lambda & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \end{matrix}$$

$$\text{Alors } \det(P_{k,l}) = 1$$

Fin

Calcul numérique du déterminant

Cas des matrices triangulaires

Considérons la **matrice triangulaire supérieure** :

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & & a_{2,n} \\ 0 & 0 & a_{3,3} & & a_{3,n} \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{pmatrix}$$

Alors

$$\det(A) = \prod_{i=1}^n a_{i,i}$$

Complexité de l'algorithme via les cofacteurs

$$\det(A) = \sum_{i=1}^n (-1)^{i+k} a_{i,k} \det(A_{i,k})$$

Soit D_n le nombre d'opération nécessaires pour le calcul d'un déterminant d'ordre n .

Alors

$$D_n = n \times D_{n-1} + 2 \times n - 1$$

Au final,

$$D_n = O(n!)$$

Un algorithme utilisable ?

Soit le Summit (IBM) avec ses 122 millions de milliards d'opérations par seconde (122×10^{15})

Soit une matrice de rang 80, $80! > 10^{118}$

Admettons que l'âge de l'univers soit de 13,7 milliards d'années.

Depuis le début de l'univers, il s'est donc écoulé

$$13,7 \times 10^9 \times 365 \times 24 \times 3600s \approx 4,32 \times 10^{17} s.$$

La densité moyenne de l'univers est évaluée à 5 atomes par m^3 .

Le diamètre de l'univers observable est évalué à 93 milliards d'années-lumière soit $8,8 \times 10^{26} m$.

Le volume d'une sphère est $4 \times \pi \times R^3$.

Le nombre d'atomes dans l'univers est donc évalué à

$$20 \times \pi \times 85,18 \times 10^{78} \approx 5,35 \times 10^{81}.$$

Même si tous les atomes de l'univers avaient la puissance du Summit et travaillaient en parallèle, l'âge de l'univers ne suffirait pas pour faire le calcul.

Utilisation de l'algorithme de Gauß simple

Appliquons la descente de Gauß sans recherche partielle de pivot à une matrice A .

On obtient le théorème suivant.

Théorème : Si p_1, \dots, p_n sont les i pivots non nuls obtenus en appliquant la descente de Gauß sans recherche partielle de pivot à la matrice A alors

$$\det(A) = \prod_{i=1}^n p_i$$

Remarque : Nécessite que $\forall i, p_i \neq 0$.

En effet, en appliquant la descente de Gauß, on obtient

$$P_l.P_{l-1}....P_1.A = T$$

$$\text{avec } P_j = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & \dots & -\lambda & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \text{ ou } P_j = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & 1/p_k & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

$$\Rightarrow \prod_{i=1}^n 1/p_i \times \det(A) = 1 \text{ donc}$$

$$\det(A) = \prod_{i=1}^n p_i$$

La complexité est en $\frac{2}{3}n^3$.

Le code C

On part de :

```
void gauss(float *a, float *b, int n)
{
    int i,j,k,il;
    float aux, aux2;
    for(i=0;i<n-1;i++)
    {
        aux = *(a + i*n + i);
        for(k=i+1; k<n; k++)
            *(a + i*n + k) /= aux;
        *(b + i) /= aux;
        for(k=i+1;k<n;k++)
        {
            aux2 = *(a + k*n + i);
            for(j=i+1;j<n;j++)
                *(a + k*n + j) -= aux2 * *(a + i*n + j);
            *(b + k) -= aux2 * *(b+i);
        }
    }
    .....
}
```

Le code C

Au final :

```
void gauss_det(float *a, int n, float *det)
{
    int i,j,k,il;
    float aux, aux2;
    *det = 1;
    for(i=0;i<n-1;i++)
    {
        aux = *(a + i*n + i);
        *det *= aux;
        for(k=i+1; k<n; k++)
            *(a + i*n + k) /= aux;
        for(k=i+1;k<n;k++)
        {
            aux2 = *(a + k*n + i);
            for(j=i+1;j<n;j++)
                *(a + k*n + j) -= aux2 * *(a + i*n + j);
        }
    }
    *det *= *(a + n*n - 1);
}
```

Utilisation de l'algorithme de Gauß avec recherche de pivot maximum

Appliquons la descente de Gauß avec recherche partielle de pivot à une matrice A .

On obtient le théorème suivant.

Théorème : Si p_1, \dots, p_n sont les i pivots non nuls obtenus en appliquant la descente de Gauß avec recherche partielle de pivot à la matrice A alors

$$\det(A) = (-1)^k \times \prod_{i=1}^n p_i$$

où k est le nombre d'échanges de lignes effectués.

Remarque : Nécessite que $\det(A) \neq 0$

En effet, en appliquant la descente de Gauß avec recherche partiel de pivot, on obtient

$$P_l.P_{l-1}....P_1.A = T$$

$$\text{avec en plus } P_j = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \text{ pour chaque}$$

échange de lignes dont le déterminant vaut -1 .

Si on a fait k échanges de lignes

$$\Rightarrow (-1)^k \times \prod_{i=1}^n 1/p_i \times \det(A) = 1 \text{ donc}$$

$$\det(A) = (-1)^k \times \prod_{i=1}^n p_i$$

La complexité est toujours en $\frac{2}{3}n^3$.

Le code C

```
void gauss_piv_det(float *a, int n, float *det)
{
    int i,j,k,il;
    float aux, aux2;
    *det = 1;
    for(i=0;i<n-1;i++)
    {
        aux=fabs(*(a + i*n + i));
        il=i;
        for(k = i+1; k < n; k++)
            if (aux<fabs(*(a + k*n + i)))
            {
                aux = fabs(*(a + k*n + i));
                il = k;
            };
        if (il != i)
        {
            *det = - *det;
            for(j=i;j<n;j++)
            {
                .....
            }
            aux = *(a + i*n + i);
            *det *= aux;
            for(k=i+1; k<n; k++)
                *(a + i*n + k) /= aux;
            for(k=i+1;k<n;k++)
            {
                .....
            }
        }
        *det *= *(a + n*n - 1);
    }
}
```

Fin

La méthode de Gauß-Jordan

On souhaite résoudre les systèmes :

$$\begin{pmatrix} a_{1,1} & .. & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & .. & a_{n,n} \end{pmatrix} \cdot \begin{pmatrix} x_{1,i} \\ \vdots \\ x_{n,i} \end{pmatrix} = \begin{pmatrix} b_{1,i} \\ \vdots \\ b_{n,i} \end{pmatrix}$$

pour $i = 1, \dots, k$.

Ce qui revient à résoudre le système matriciel :

$$\begin{pmatrix} a_{1,1} & .. & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & .. & a_{n,n} \end{pmatrix} \cdot \begin{pmatrix} x_{1,1} & .. & x_{1,k} \\ \vdots & & \vdots \\ x_{n,1} & .. & x_{n,k} \end{pmatrix} = \begin{pmatrix} b_{1,1} & .. & b_{1,k} \\ \vdots & & \vdots \\ b_{n,1} & .. & b_{n,k} \end{pmatrix}.$$

Première idée :

On applique la méthode de Gauß avec ou sans recherche de pivot « en parallèle » sur les B_i . On obtient :

$$\begin{pmatrix} 1 & .. & t_{1,n} \\ \vdots & \ddots & \vdots \\ 0 & .. & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{1,1} & .. & x_{1,n} \\ \vdots & & \vdots \\ x_{n,1} & .. & x_{n,n} \end{pmatrix} = \begin{pmatrix} c_{1,1} & .. & c_{1,n} \\ \vdots & & \vdots \\ c_{n,1} & .. & c_{n,n} \end{pmatrix}$$

Deuxième idée : on effectue la remontée « sur place » i.e. on continue jusqu'à ce que la matrice A deviennent I_d .

C'est la méthode de Gauß-Jordan

Example

$$\begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \end{pmatrix} \cdot X = \begin{pmatrix} 3 & -1 & 8 & 2 \\ 1 & 0 & 7 & 2 \\ 0 & 1 & 9 & 0 \end{pmatrix}$$

\Rightarrow

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 1 & 2 & 1 & 1 & 0 & 7 & 2 \\ 2 & 1 & 1 & 0 & 1 & 9 & 0 \end{array} \right)$$

Example

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 1 & 2 & 1 & 1 & 0 & 7 & 2 \\ 2 & 1 & 1 & 0 & 1 & 9 & 0 \end{array} \right)$$

\Rightarrow

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 0 & 1 & -1 & -2 & 1 & -1 & 0 \\ 0 & -1 & -3 & -6 & 3 & -7 & -4 \end{array} \right)$$

Example

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 1 & 2 & 1 & 1 & 0 & 7 & 2 \\ 2 & 1 & 1 & 0 & 1 & 9 & 0 \end{array} \right)$$

\Rightarrow

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 0 & 1 & -1 & -2 & 1 & -1 & 0 \\ 0 & -1 & -3 & -6 & 3 & -7 & -4 \end{array} \right)$$

\Rightarrow

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 3 & 5 & -2 & 9 & 2 \\ 0 & 1 & -1 & -2 & 1 & -1 & 0 \\ 0 & 0 & -4 & -8 & 4 & -8 & -4 \end{array} \right)$$

Example

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 1 & 2 & 1 & 1 & 0 & 7 & 2 \\ 2 & 1 & 1 & 0 & 1 & 9 & 0 \end{array} \right)$$

\Rightarrow

$$\left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 3 & -1 & 8 & 2 \\ 0 & 1 & -1 & -2 & 1 & -1 & 0 \\ 0 & -1 & -3 & -6 & 3 & -7 & -4 \end{array} \right)$$

\Rightarrow

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 3 & 5 & -2 & 9 & 2 \\ 0 & 1 & -1 & -2 & 1 & -1 & 0 \\ 0 & 0 & -4 & -8 & 4 & -8 & -4 \end{array} \right)$$

\Rightarrow

$$X = \begin{pmatrix} -1 & 1 & 3 & -1 \\ 0 & 0 & 1 & 1 \\ 2 & -1 & 2 & 1 \end{pmatrix}$$

Le code C

```
void gauss_jordan(float *a, float *b, int n)
{
    int i,j,k,il;
    float aux;
    for(i = 0; i < n; i++)
    {
        aux = *(a + i*n + i);
        for(j = i+1; j < n; j++)
            *(a + i*n + j) /= aux;
        for(j = 0; j < n; j++)
            *(b + i*n + j) /= aux;
        for(k=0;k<n;k++)
            if (k != i)
            {
                aux = *(a + k*n + i);
                for(j = i+1; j < n; j++)
                    *(a + n*k + j) -= aux * *(a + n*i + j);
                for(j = 0; j < n; j++)
                    *(b + n*k + j) -= aux * *(b + n*i + j);
            }
    }
}
```

Fin