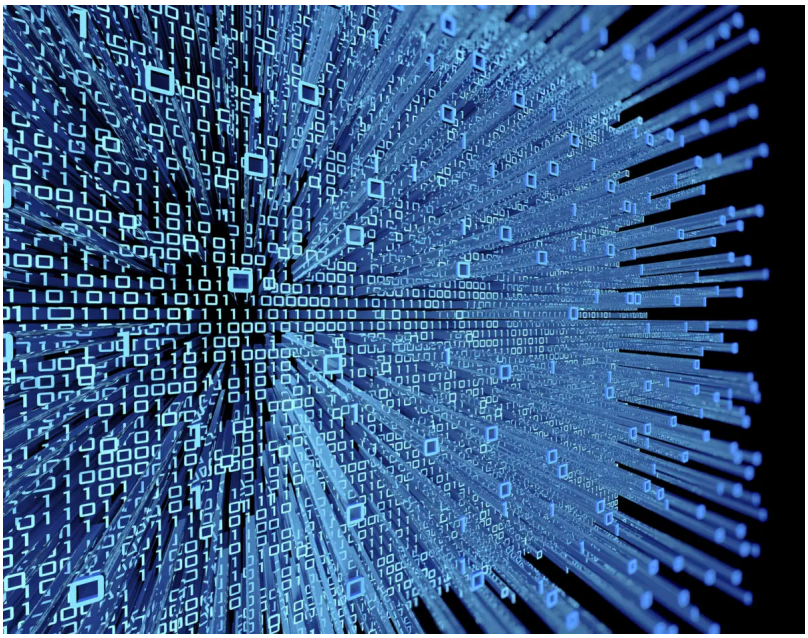

LINMA1170 Analyse numérique

Simon Desmidt

Année académique 2023-2024 - Q2

UCLouvain

Table des matières

Factorisation QR

L'objectif de la factorisation QR est de trouver une sélection de vecteurs q_i orthonormés et générant successivement les sous-espaces de la matrice A.

1.1 Projecteurs

Un projecteur est une matrice carrée P telle que $P^2 = P$. Il s'agit donc d'une matrice idempotente. Soit un vecteur $v \in \mathbb{R}^n$ et soit un vecteur quelconque $X \in \mathbb{R}^n$. La projection du vecteur X sur l'espace engendré par le vecteur v est $P_x = \frac{vv^T}{v^Tv}X = PX$, avec P le projecteur. La projection $\hat{P}_x$ du vecteur X sur l'espace engendré par $v^\perp$ est $\hat{P}_x = X - P_x = (I - P)X$, et on appelle $I - P$ le projecteur complémentaire.

→ Remarque : La propriété $P = P^2$ est intéressante, car cela signifie que $Px = x$ si $x \in range(P)$. Cela implique que l'ensemble des projections $v - Pv \in \mathbb{N}(P)$: $P(v - Pv) = Pv - P^2v = Pv - Pv = 0$.

Le lien entre les projecteurs P et $I - P$ est que $range(I - P) = nullspace(P)$. Un projecteur sépare donc l'espace dans lequel il existe en deux sous-espaces disjoints : son image et son nullspace. Tout vecteur appartient donc forcément à l'un de ces deux espaces, mais jamais aux deux. De plus, si $v_1 = Pv$ et $v_2 = (I - P)v$, alors $v_1 + v_2 = v$.

1.1.1 Projecteur orthogonal

Soit un projecteur P. Il sépare l'espace en deux sous-espaces, que l'on note S_1 et S_2, avec S_1 l'ensemble des vecteurs v tels que $Pv \neq 0$ et S_2 tel que $PV = 0$. Un projecteur est dit orthogonal si les espaces S_1 et S_2 sont orthogonaux.

Propriété :

- Un projecteur P est orthogonal ssi $P = P^*$, avec P^* la transposée de P où toutes les entrées sont conjuguées.

1.2 Factorisation QR

Soient les a_i les colonnes de A. On cherche les q_i tels que $\langle q_1, \ldots, q_j \rangle = \langle a_1, \ldots, a_j \rangle$. On peut écrire cela sous la forme suivante équivalente :

$$\begin{cases} a_1 = r_{11} q_1 \\ a_2 = r_{12} q_1 + r_{22} q_2 \\ \vdots a_j = \sum_{i=1}^{j} r_{ij} q_i \end{cases} \tag{1.1}$$

Que l'on peut écrire sous forme matricielle :

$$\begin{pmatrix} a_1 & a_2 & \ldots & a_n \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & \ldots & q_n \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ & r_{22} & \ldots & \vdots \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix} \Longrightarrow A = \hat{Q}\hat{R} \tag{1.2}$$

On appelle cette factorisation la factorisation QR réduite, car la matrice Q n'est pas carrée. Nous voulons qu'elle le soit afin de pouvoir simplifier les systèmes linéaires $QRx = b$ en inversant Q. Pour cela, nous allons remplir artificiellement la matrice Q avec des vecteurs linéairement indépendants, tout en préservant l'orthonormalité jusqu'à arriver à une matrice $m \times m$. Si on ajoute des colonnes à droite dans Q, on peut annuler leur effet sur la factorisation $A = QR$ si on ajoute le même nombre de lignes contenant uniquement des 0 en-dessous de la matrice R. La matrice Q devient alors unitaire.

La méthode finale pour résoudre $Ax = b$ est donc

- Calculer $A = QR$
- Calculer $y = Q^* b$
- Résoudre par substitution $Rx = y$

1.2.1 Théorèmes

Si $A \in \mathbb{C}^{m \times n}$, $m \geq n$, alors A possède une factorisation complète réduite.

Si $A \in \mathbb{R}^{m \times n}$, $m \geq n$ est de rang plein, alors sa factorisation QR est unique et $r_{ij} > 0$ $\forall i, j$.

1.3 Triangularisation de Householder

1.3.1 Householder et Gram-Schmidt

La méthode de Gram-Schmidt permet de trouver la factorisation QR en appliquant une série de matrices élémentaires triangulaires R_k à droite de A, tel que $AR_1R_2 \ldots R_n = \hat{Q}$, avec $R_1R_2 \ldots R_n = \hat{R}^{-1}$ triangulaire supérieure. Cette méthode fonctionne, mais elle

n'est pas unique.

La méthode de Householder fonctionne dans le sens inverse : on applique une succession de matrices élémentaires triangulaires Q_k à gauche de A tel que

$$Q_n \dots Q_2 Q_1 A = R \tag{1.3}$$

On appelle la méthode de Gram-Schmidt l'orthogonalisation triangulaire, tandis que Householder est une triangularisation orthogonale.

1.3.2 Méthode de Householder

Il s'agit ici de trouver une suite de matrices unitaires telles que $Q_n \dots Q_2 Q_1 A$ est triangulaire supérieure, l'objectif étant de faire apparaître des 0 dans la $i$ème colonne lors de la multiplication par Q_i, sans faire disparaitre ceux déjà existant. L'approche standard est de choisir

$$Q_k = \begin{pmatrix} I & 0 \\ 0 & F \end{pmatrix} \tag{1.4}$$

- I est la matrice identité de taille $(k-1) \times (k-1)$.
- F est une matrice unitaire de taille $(m-k+1) \times (m-k+1)$.

La multiplication par F place des 0 dans la $k$ème colonne, on appelle cela un réflecteur de Householder.

1.3.3 Réflecteur de Householder

Un réflecteur est un opérateur matriciel $H(v)$ tel que, si appliqué à un vecteur X, le vecteur résultant est obtenu par symétrie orthogonale dans $\mathbb{R}^n$, d'axe de symétrique $span\{v^\perp\}$. Soit P le projecteur lié au vecteur v, alors $H(v) = I - 2P$. Cette matrice est unitaire, i.e. $H^2(v) = I$ et ses valeurs propres sont donc toutes -1 ou 1.

Le but de F est de "réfléchir" l'espace $\mathbb{C}^{m-k+1}$ à travers l'hyperplan H, orthogonal à $v = \|x\| e\_x$. On design donc F de telle manière que $x \to \|x\| e_1$:

$$F = I - 2P_v = I - 2\frac{vv^*}{v^*v} \tag{1.5}$$

Il est toutefois possible de faire la réflexion vers $-\|x\| e_1$. Ce choix a de l'importance quant à la stabilité de la méthode numérique. Il est préférable de prendre la réflexion la plus loin possible de x. Cela est dû au fait que si v est de faible norme, il est plus probable d'avoir des erreurs d'arrondis. On prend donc le vecteur v suivant : $v = x + sign(x_1)\|x\| e_1$, x_1 étant la première composante de x.

$\to$ Remarque : on utilise la convention que si $x_1 = 0$, $sign(x_1) = 1$.

1.3.4 Codes

Voici un pseudo-code permettant de calculer R :

```
    for k = 1 to n :
        a_k = A(k:m,k)
        v_k = a_k + sign(a_{k1}) norm(a_k) e_1 # Calcul du vecteur v
        v_k = v_k/norm(v_k) # Normalisation
        V[k] = v_k # Sauvegarde pour augmenter l'efficacite
        A(k:m,k:n) -= 2v_k(v_k^* A(k:m,k:n))
    R = A
```

Dans la résolution du système $Ax = b$, la mateice Q n'apparait que dans le terme Q^*b; on ne doit donc pas calculer explicitement la matrice. Voici un algorithme calculant ce produit Q^*b :

```
    for k = 1 to n :
        b[k:m] = b[k:m] - 2v_k(v_k^* b[k:m])
```

Cout des opérations

L'opération principale du premier algorithme est

```
    A[k:m,j] - 2v_k(v_k^*A[k:m,k]
```

Soit $l = m - k + 1$ la longueur de ce vecteur. Chaque actualisation nécessite $4l - 1$ (arrondi à $4l$) opérations : l soustractions et multiplications, plus $2l - 1$ opérations pour le produit scalaire. En supprimant les termes -1 car d'ordre inférieur, on trouve le nombre d'opérations global :

$$4\sum_{k=1}^{n}(m-k)(n-k) = 2mn^2 - \frac{2}{3}n^3 \tag{1.6}$$

1.4 Problème des moindres carrés

On étudie ici les systèmes linéaires surdéterminés (pas de sens sinon). On a donc $m > n$. Soit le système tel que $A \in \mathbb{C}^{m\times n}$ et $b \in \mathbb{C}^m$. On cherche $x \in \mathbb{C}^m$ tel que $Ax = b$. Pour rappel, si $b \in range(A)$, alors x n'existe pas. On définit donc le résidu $M = b - Ax \in \mathbb{C}^m$. La solution serait exacte si $M = 0$, l'intuition est donc de minimiser ce résidu M. On cherche le point $x \in range(A)$ le plus proche de b selon une certaine norme (prenons $\|\cdot\|_2$).

$$Ax = Pb \tag{1.7}$$

avec $P \in \mathbb{C}^{m\times m}$ le projecteur orthogonal projetant l'espace $\mathbb{C}^m$ sur $range(A)$.

Si $A \in \mathbb{C}^{m\times n}$, $m \geq n$ et $b \in \mathbb{C}^m$ sont donnés et $x \in \mathbb{C}^n$ minimise $\|M\|_2$ ssi $M \perp range(A)$. On a donc

$$A^*M = 0 \Longrightarrow A^*Ax = A^*b \tag{1.8}$$

1.5 Orthogonalisation de Gram-Schmidt

Soient n vecteurs $a_1, \ldots, a_n \in \mathbb{C}^m$ avec $m \geq n$. On souhaite créer une base orthonormée pour l'espace engendré par cet ensemble : elle sera

$$\hat{q}_k = a_k - \sum_{i=1}^{k-1} (q_i^* a_i) q_i \tag{1.9}$$

$$q_k = \hat{q}_k / \|\hat{q}_k\| \tag{1.10}$$

A partir de cette base, on peut faire apparaitre la décomposition QR de la matrice A : soit les a_k les colonnes de A. On a

$$a_k = \sum_{i=1}^{k} \underbrace{(q_i^* a_k)}_{r_{ik}} \tag{1.11}$$

Sous forme matricielle, cela donne

$$\begin{pmatrix} a_1 & a_2 & \ldots & a_n \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & \ldots & q_n \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ & r_{22} & \ldots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix} \tag{1.12}$$

$\rightarrow$ Remarque : $Q \in \mathbb{R}^{m \times n}$. On a $Q^*Q = I_{n \times n}$, mais $QQ^* \neq I$.

1.5.1 Deux algorithmes

L'algorithme de la méthode ci-dessus est le suivant :

```
for j = 1 to n:
    v[j] = a[j]
    for i = 1 to j-1:
        r[i][j] = a[i].t @ a[j]
        v[j] = v[j] - r[i][j]*q[i]
    r[j][j] = norm(v[j])
    q[j] = v[j]/r[j][j]
```

Une autre manière de procéder est de projeter a_k sur l'espace orthogonal à $q_1, \ldots, q_{k-1}$: $\hat{q}_k = P_k a_k$, avec P_k une matrice de projection. Ou en projetant successivement : $\hat{q}_k = P_{\perp q_{k-1}} \ldots P_{\perp q_1} a_k$.

```
for k=1 to n:
    Hatq[k] = a[k]
for k = 1 to n :
    #Hatq[k] est orthogonal a q[1]->q[k-1]
    q[k] = Hatq[k]/norm(Hatq[k])
    for j =k+1 to n :
        Hatq[j] = Hatq[j] - (q[k].T @ Hatq[j])q[k]
```

1.5.2 Conditionnement

Le conditionnement concerne les perturbations et le comportement en réaction à ces perturbations d'un problème mathématique.

Soit un algorithme classique à une entrée y et une sortie $f(y)$:

$$y \longrightarrow f(y) \tag{1.13}$$

Le conditionnement cherche à savoir comment $f(y)$ est affecté par un petit changement en y. Appliquons cela à un système linéaire $Ax = b$. Les entrées sont A, b et la sortie est $x = A^{-1}b$. Si on applique une variation δA à A, que vaut la variation δx?

$$(A + \delta A)(x + \delta x) = b \Longrightarrow Ax + (\delta A)x + A(\delta x) + (\delta A)(\delta x) = b \tag{1.14}$$

Le terme $(\delta A)(\delta x)$ est négligeable et on peut simplifier les termes Ax et b. On a donc

$$\delta x = -(A^{-1})(\delta A)x \Longrightarrow \|\delta x\| \leq \|A^{-1}\|\|\delta A\|\|x\| \Longleftrightarrow \frac{\|\delta x\|/\|x\|}{\|\delta A\|\|A\|} \leq \|A^{-1}\|\|A\| \tag{1.15}$$

La variation de la solution en fonction de la variation de l'entrée est donc bornée supérieurement.

1.6 Stabilité

Le calcul numérique n'est pas exact car le rpz d'un ordinateur est discret, alors qu'on étudie généralement des phénomènes continus. La notion de stabilité est donc la façon standard de caractériser ce qui est possible numériquement. Il existe des exigences concernant la qualité d'une réponse discrète (fournie par un ordinateur) à un problème continu.

Soit f un problème et soit un ordinateur dont le système de calcul vérifie la condition suivante :

$$x * y = (x * y)(1 + \varepsilon) \qquad |\varepsilon| \leq \varepsilon_{\text{machine}} \tag{1.16}$$

Un algorithme peut être vu comme $\tilde{f} : X \to Y$, avec $\tilde{f}$ le "computed analog" du problème continu f. On dira ici que $\tilde{f}$ est un algorithme précis de f. Nous avons donc la condition pour chaque $x \in X$:

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = \mathcal{O}(\varepsilon_{\text{machine}}) \tag{1.17}$$

L'ordinateur travaille en réalité comme suit :

$$y \longrightarrow \tilde{y} \longrightarrow \tilde{f}(y) \tag{1.18}$$

On voit apparaître une erreur créée avant même que l'algorithme ne soit initié. Il n'est donc pas raisonnable de demander qu'un algorithme soit précis, on va plutôt parler de stabilité.

1.6.1 Stabilité inverse

La condition de stabilité , malgré qu'elle soit bien plus raisonnable que la précision, reste tout de même assez forte. Toutefois, beaucoup d'algorithmes respectent la stabilité inverse, une propriété à la fois plus forte et plus simple que la stabilité elle-même. On dit d'un algorithme $\tilde{f}$ pour un problème f qu'il est "backward stable" si

$$\forall x \in X \qquad \exists \tilde{x} \text{ tel que } \frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(\varepsilon_{\text{machine}}) \Longrightarrow \tilde{f}(x) = f(\tilde{x}) \tag{1.19}$$

Cela signifie qu'un algorithme inversement stable donne la réponse exacte à une question qui est environ celle posée.

Elimination gaussienne

L'élimination gaussienne est la méthode la plus simple, aussi bien numérique qu'analytique, pour résoudre $Ax = b$.

2.1 Factorisation LU

La factorisation LU consiste à appliquer des applications linéaires à la matrice A pour obtenir U.

Supposons la matrice A carrée $A \in \mathbb{R}^{m\times m}$. L'objectif est d'introduire des 0 sous la diagonale dans chacune des colonnes les unes après les autres (dans l'ordre croissant). L'application linéaire liée à la colonne i se note L_i.

La factorisation LU de la matrice A est donc $L_{m-1} \dots L_1 A = U \iff A = L_1^{-1} \dots L_{m-1}^{-1} U$. La matrice L est triangulaire inférieure avec une diagonale unitaire, tandis que U est triangulaire supérieure avec une diagonale quelconque.

→ Remarque : Il n'y a pas de L_m, car il n'est pas possible d'introduire des 0 en-dessous de la dernière ligne.

2.1.1 Formulation générale

Soit la matrice $A \in \mathbb{R}^{m\times m}$. On note x_k sa colonne k. Soit l_k un vecteur défini pour calculer L_k, tel que

$$l_{k,j} = \frac{x_{jk}}{x_{kk}} \qquad j \in]k, m] \tag{2.1}$$

On peut maintenant définir la matrice L_k :

$$L_k = I - l_k e_k^* \tag{2.2}$$

avec e_k le conjugué[1] du vecteur de base tel que chacun de ses éléments est nul, sauf l' $k$ème valant 1. On peut facilement montrer que $L_k^{-1} = I + l_k e_k^T$.

De plus, la somme des inverses des matrices de transformation se simplifie fortement :

$$L_k^{-1} L_{k+1}^{-1} = I + l_k e_k^* + l_{k+1} e_{k+1}^* \tag{2.3}$$

Voici un algorithme pour effectuer la factorisation LU de la matrice A

[1] Equivalent à la transposée si $in\mathbb{R}^m$.

```
    U = A
    L=I
    for k =1 to m-1:
        for j =k+1 to m:
            l[j][k] = u[j][k]/u[k][k]
            u[j][k:m] = u[j][k:m] - l[j][k] u[k][k:m]
```

La complexité de cet algorithme est $\frac{2}{3}m^3$.

2.1.2 Pivots

L'élimination gaussienne telle que vue précédemment n'est pas stable. On peut toutefois l'améliorer en pivotant les lignes de la matrice. A l'étape k, on enlève des multiples de la ligne k aux lignes $k+1, \ldots, m$ pour faire apparaître des 0 dans la colonne k. On appelle l'élément a_{kk} pivot. On pourrait en réalité le faire avec n'importe quelle ligne plutôt que la k. On peut donc effectuer des permutations des lignes avant l'étape d'échelonnement, i.e. multiplier par une matrice d'échelonnement P_k. On échelonne donc en fait la matrice PA, avec $P = P_{m-1} \ldots P_1$.

Il y a toutefois un coup de chance : on sait que $P_k P_k^{-1}$, on a donc que $P_k L_{k-1} P_k^{-1}$ est donc aussi une matrice d'échelonnement! On a donc la factorisation $PA = LU$, avec L une matrice différente que la factorisation $A = LU$, mais aussi une transformée de Gauss.

2.1.3 Alogithme de factorisation avec pivots

```
int LU (double ** a, int n, i++){
    for (int i = 0, i<n, i++) piv[i] = i;
    for (int k = 0, i<n, i++){
        double rpiv = fabs(a[k][k]);
        int kpiv = k;
        for (int i = k, i<n, i++){
            if (fabs(a[i][k]>rpiv){
            rpiv = fabs(a[i][k]);
            kpiv = i;
            }
        }
        for (int i = k, i<n, i++){
            double temp = a[k][i];
            a[k][i] = a[kpiv][i];
            a[kpiv][i] = temp;
        }
        int tmp = piv[k];
        piv[k] = piv[kpiv];
        for (int i = k+1, i<n, i++){
            a[i][k] = a[i][k]/a[k][k];
            for (int j=k+1, j<n, j++){
                a[i][j] = a[i][j] - a[i][k]*a[k][j];
            }
        }
    }
}
```

2.1.4 Stockage efficace de la matrice

La majorité du temps dans les systèmes réels, les matrices sont très creuses. On peut donc les stocker beaucoup plus efficacement dans des `arrays` avec les indices plutôt qu'avec des tableaux à deux dimensions.

```
void vec_mat (double* x, double* a, int* ia, int* ja, int* res){
    for (int i=0, i<n, i++){
    res[i] = 0;
    for (int j = ia[i], j <ia[i+1], j++){
        res[i] += x[ja]*a[j];
        }
    }
}
```

2.2 Résolution de systèmes linéaires

Partons du système $Ax = b$. Par factorisation LU, on obtient deux systèmes linéaires triangulaires :

$$Ux = y \tag{2.4}$$
$$Ly = b \tag{2.5}$$
$$\tag{2.6}$$

On les résout par substitution.

→ Remarque : la résolution par élimination gaussienne n'est pas stable : une petite perturbation sur les valeurs de la matrice A peut, dans certains cas, changer fortement le résultat de la factorisation.