Solution of Linear Equations

Seminar – Algorithms from The BOOK
2022/23 WS

Jonathan Bimmüller

30. November 2022

Abstract

Das Lösen von Linearen Gleichungssystemen der Form $Ax = b$ ist eine der wichtigsten Anwendungen der Mathematik. Der vermutlich bekannteste Ansatz, ist das Lösen mithilfe der LU-Zerlegung, bei der die Matrix in zwei Dreiecksmatrizen zerlegt wird, für die nur noch triviale Substitutionen durchgeführt werden müssen. Die Cholesky-Zerlegung ist eine weitere Möglichkeit eine Matrix, unter der Bedingung, dass diese positiv definit ist, in zwei Dreiecksmatrizen zu zerlegen. Die QR-Zerlegung ist einer der Nachfolger der LU-Zerlegung, der zwar langsamer aber dafür numerisch stabiler ist. Hierbei wird die Matrix in eine unitäre und eine Dreiecksmatrix zerlegt, auch für diese reduziert sich der übrige Aufwand zum Lösen des Gleichungssystems. Außerdem können mithilfe der QR-Zerlegung leicht lineare Ausgleichsprobleme gelöst werden. Für große, dünn besetzte und positiv definite Matrizen wird meist das CG-Verfahren verwendet, das iterativ die Lösung des Gleichungssystems bestimmt.

1 Introduction

Jedes lineare Gleichungssystem

$$a_{1,1}x_1 + ... + a_{1,n}x_n = b_1 \tag{1}$$

$$a_{2,1}x_1 + ... + a_{2,n}x_n = b_2 \tag{2}$$

$$... \tag{3}$$

$$a_{m,1}x_1 + ... + a_{m,n}x_n = b_m \tag{4}$$

kann in der Form $Ax = b$ mit $A = (a_{i,j})_{i=1,..,m;j=1,...,n}$, $x = (x_j)_{j=1,...,n}$, $b = (b_i)_{i=1,..,m}$ geschrieben werden. Für $m = n$ kann dieses Gleichungssystem beispielsweise mit dem zum Teil schon in der Schule bekannten Gaußschen Eliminationsverfahren gelöst werden. Dieses bildet die Grundlage für unseren ersten Algorithmus.

2 LU Decomposition

2.1 Algorithm

Sei $A = (a_{i,j})_{i,j=1,...,n}$ eine $n \times n$ Matrix.
Die Grundlegende Idee hinter der LU-Zerlegung ist, die Matrix $A = A^0$ durch die Matrix $A^k = M_k A^{k-1}$ wobei $M_k = I_n - m_k e_k^*$ mit $m_k = (0, ..., 0, -a^k_{k+1,k}/a^k_{k,k}, ..., -a^k_{n,k}/a^k_{k,k})^T$ zu ersetzen und dadurch Spalte für Spalte in eine obere Dreiecksmatrix U zu überführen.

$$M_{n-1}...M_2M_1A = U \tag{5}$$

Da M_k untere Dreiecksmatrizen mit Determinante 1 sind, existieren auch ihre Inversen $M_k^{-1} = I_n + m_k e_k^*$ und das Produkt dieser ist wieder eine untere Dreiecksmatrix.

$$A = M_1^{-1}M_2^{-1}...M_{n-1}^{-1}U = LU \tag{6}$$

Problem: Für $a^k_{k,k} = 0$ oder sehr klein im Verhältnis zu den anderen Einträgen in dieser Spalte ist der Schritt nicht durchführbar beziehungsweise numerisch instabil.

Lösung: Pivotisierung, dass heißt in jedem Schritt werden die Zeilen so vertauscht, dass $a^k_{k,k}$ der betragsmäßig Größte der Einträge auf und unterhalb der Diagonale ist. Mit der entsprechenden dynamisch entstehenden Permutationsmatrix P ergibt sich:

$$PA = LU \tag{7}$$

Ein Gleichungssystem der Form $Ax = b \Leftrightarrow LUx = Pb = b'$ wird in zwei Schritten gelöst

1. Vorwärtssubstitution: Lösen der Gleichung $Lz = b'$
   $z_1 = l_{1,1}^{-1} b'_1$
   $z_j = l_{j,j}^{-1}[b'_j - \sum_{k=1}^{j-1} l_{j,k} z_k]$, $j = 2, ..., n$
2. Rückwärtssubstitution: Lösen der Gleichung $Ux = z$
   $x_n = u_{n,n}^{-1}$
   $x_j = u_{j,j}^{-1}[z_j - \sum_{k=j+1}^{n} u_{j,k} x_k]$, $j = (n-1), ..., 1$

2.2 Some details

(i) Die LU-Zerlegung benötigt ungefähr $\frac{2}{3}n^3$ Rechenoperationen

(ii) Die Vorwärts- und Rückwärtssubstitution benötigen je $O(n^2)$ Rechenoperationen

(iii) $det(A) = \pm \prod_{i=1}^{n} u_{i,i}$, wobei $u_{i,i}$ die Diagonaleinträge von U sind

2.3 Pseudo-code

Algorithm 1: LUdecomposition

```
Data: (a_{i,j})_{i,j=1,...,n}
Result: (lu_{i,j})_{i,j=1,...,n}, (perm_i)_{i=1,...,n}
begin
    for i = 1 to n do
        perm_i = i
    end
    for k = 1 to (n - 1) do
        p = k
        for i = (k + 1) to n do
            if |a_{i,k}| > |a_{p,k}| then
                p = i
            end
        end
        swap perm_k and perm_p
        for j = 1 to n do
            swap a_{p,j} and a_{k,j}
        end
        for i = (k + 1) to n do
            a_{i,k} = a_{i,k} / a_{k,k}
        end
        for i = (k + 1) to n do
            for j = (k + 1) to n do
                a_{i,j} = (a_{i,j} - (a_{i,k} · a_{k,j}))
            end
        end
    end
end
```

Algorithm 2: LUsolve

```
Data: (lu_{i,j})_{i,j=1,...,n}, (perm_i)_{i=1,...,n}, (b_i)_{i=1,...,n}
Result: (x_j)_{j=1,...,n}
begin
    x, z = 0⃗_n
    for i = 1 to n do
        z_i = b_(perm_i)
        for j = 1 to (i − 1) do
            z_i = z_i − lu_{i,j} z_j
        end
    end
    for i = (n − 1) to 1 do
        x_i = z_i
        for j = i + 1 to n do
            x_i = x_i − lu_{i,j} x_j
        end
        x_i = x_i / lu_{i,i}
    end
end
```

3 Cholesky Decomposition

3.1 Algorithm

Sei $A = (a_{i,j})_{i,j=1,...,n}$ eine positiv definite $n \times n$ Matrix.
Dann bildet die Cholesky-Zerlegung L von A eine untere Dreiecksmatrix, so dass:

$$A = LL^* \tag{8}$$

Der Induktionsbeweis, dass die Cholesky-Zerlegung existiert und eindeutig ist, führt direkt zum Code.
Im Induktionsanfang beginnt man mit:

$$A = LL^* \Leftrightarrow \begin{pmatrix} a_{1,1} & a^* \\ a & A_{2,2} \end{pmatrix} = \begin{pmatrix} l_{1,1} & 0^* \\ l & L_{2,2} \end{pmatrix} \begin{pmatrix} l_{1,1} & l^* \\ 0 & L_{2,2}^* \end{pmatrix} \tag{9}$$

$$\Downarrow \tag{10}$$

$$l_{1,1} = \sqrt{a_{1,1}} \tag{11}$$

$$l = l_{1,1}^{-1} a \tag{12}$$

$$L_{2,2} L_{2,2}^* = A_{2,2} - a_{1,1}^{-1} a a^* \tag{13}$$

Da $a_{1,1} = e_1^* A e_1 > 0$ sind $l_{1,1}$ und l eindeutig bestimmt.

Im Induktionsschritt zeigt man, dass $A_{2,2} - a_{1,1}^{-1} a a^*$ positiv definit und somit $L_{2,2}$ existiert und eindeutig bestimmt ist.

3.2 Some details

(i) Die Cholesky-Zerlegung benötigt ungefähr $\frac{1}{3} n^3$ Rechenoperationen

(ii) Gleichungssysteme der Form $Ax = b \Leftrightarrow LL^* x = b$ können analog zur LU-Zerlegung mit Vorwärts- und Rückwärtssubstitutionen gelöst werden.

(iii) Die Lösung Linearer Ausgleichsprobleme der Form $||b - Ax||_2^2 = ||b - LL^* x||_2^2 = min_y ||b - Ay||_2^2$ können durch Vorwärts- und Rückwärtssubstitution aus der Gleichung $LL^* x = A^* b$ bestimmt werden.

3.3 Pseudo-Code

Algorithm 3: Choleskydecomposition

```
Data: (a_{i,j})_{i,j=1,...,n}
Result: (l_{i,j})_{i,j=1,...,n}
begin
    for k = 1 to n do
        a_{k,k} = sqrt(a_{k,k})
        for j = (k+1) to n do
            a_{j,k} = a_{j,k} / a_{k,k}
        end
        for i = (k+1) to n do
            a_{i:n,i} = a_{i:n,i} - a_{i,k} a_{i:n,k}
        end
    end
end
```

4 QR Decomposition and Gram-Schmidt Orthogonalization

4.1 Algorithm

Sei $A = (a_{i,j})_{i=1,...,n;j=1,...,p}$ eine $n \times p$ Matrix, wobei $n \geq p$.
Die QR-Zerlegung von A kann unter anderem aus dem Gram-Schmidt Orthogonalisierungsverfahren gewonnen werden, dabei wird aus den Spaltenvektoren $a_1, ..., a_p$ ein orthonormaler Satz von Vektoren $q_1, ..., q_p$ generiert, der den gleichen Spaltenraum aufspannt.
$A = QR$
mit einer in den Spalten orthogonalen $n \times p$ Matrix $Q = (q_1, ..., q_p)$

$$q_1 = \frac{1}{||a_1||_2} a_1 \tag{14}$$

Für $j = 2, ..., p$ (Gram-Schmidt Orthogonalisierung)

$$q_j = a_j - \sum_{i=1}^{j-1} q_i^* a_j q_i \tag{15}$$

und einer oberen Dreiecksmatrix $R = (r_{j,k})_{j,k=1,...,p}$

$$r_{j,j} = ||q_j||_2 \tag{16}$$

Für $1 \leq j < p$, $j < k \leq p$

$$r_{j,k} = q_j^* a_k \tag{17}$$

Um das Verfahren, vor allem bei nahezu kollinearen Spalten von A, numerisch stabiler zu machen eignet es sich in Gleichung (15) die Projektionen einzeln abzuziehen.

Lineare Gleichungssysteme der Form $Ax = b$ werden folgendermaßen gelöst

1. Berechne $z = (Q^T b)_{i=1,...,p}$
2. Löse $Rx = z$ mit der Rückwärtssubstitution

Im Fall $n = p$ ergibt sich eine numerisch stabilere Alternative zur LU-Zerlegung.
Im Fall $n > p$ ergibt sich die Lösung des Ausgleichsproblem nach der Methode der kleinsten Fehlerquadrate $||b - Ax||_2^2 = min_y ||b - Ay||_2^2$

4.1.1 Some details

(i) Die QR-Zerlegung benötigt für $n = p$ ungefähr $\frac{4}{3}n^3$ und für $n \gg p$ ungefähr $4n^2m$ Rechenoperationen [2]

(ii) Gilt $Rang(A) = p$ ist die QR-Zerlegung, für fest gewählte Vorzeichen der Diagonalelemente von R, eindeutig [3]

(iii) Die QR-Zerlegung kann unter anderem auch durch numerisch stabilere Householdertransformationen bestimmt werden. Dieses Verfahren wird in Algorithms from The BOOK Kapitel 8 behandelt.

4.1.2 Pseudo Code

Algorithm 4: QRdecomposition

```
Data: (a_{i,j})_{i=1,...,n;j=1,...,p}
Result: (q_{i,j})_{i=1,...,n;j=1,...,p}, (r_{j,k})_{j,k=1,...,p}
begin
    (r_{i,j})_{:,:} = 0
    (q_{i,j})_{:,:} = (a_{i,j})_{:,:}
    for j = 1 to p do
        r_{j,j} = ||q_{:,j}||_2
        q_{:,j} = q_{:,j} / r_{j,j}
        for k = j + 1 to p do
            r_{j,k} = q_{:,j} ∘ q_{:,k}
            q_{:,k} = q_{:,k} − r_{j,k} q_{:,j}
        end
    end
end
```

Algorithm 5: QRsolve

```
Data: (q_{i,j})_{i=1,...,n;j=1,...,p}, (r_{i,j})_{i,j=1,...,p}, (b_j)_{j=1,...,p}
Result: (x_j)_{j=1,...,p}
begin
    x = Q^T b
    for i = (p − 1) to 1 do
        for j = i + 1 to p do
            x_i = x_i − r_{i,j} x_j
        end
        x_i = x_i / r_{i,i}
    end
end
```

5 Conjugate Gradient Method

Sei $A = (a_{i,j})_{i,j=1,...,n}$ eine große, dünn besetzte positiv definite $n \times n$ Matrix.
Die Grundlegende Idee hinter dem CG-Verfahren ist, dass das Lösen von $Ax = b$ äquivalent zum minimieren der Funktion $f(x) = \frac{1}{2}x^*Ax - b^*x + c$ ist. Dabei ist der Gradient von $f(x)$ an der Stelle x^k gleich minus das Residuum r^k

$$\nabla f(x)|_{x^k} = Ax^k - b = -r^k \tag{18}$$

Der Code beginnt mit dem Startwert $x^0 = \vec{0}$ und der Suchrichtung $d^0 = b$
Das Minimieren von $h(t) = f(x^k + td^k)$ ergibt die Schrittweite

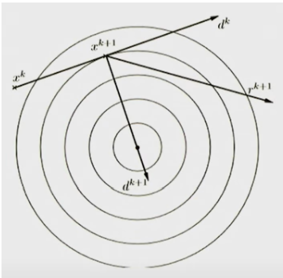
$$t^k = -\frac{(Ax^k - b)^*d^k}{(d^k)^*Ad^k} = \frac{(r^k)^*r^k}{(d^k)^*Ad^k} \tag{19}$$

Setze

$$x^{k+1} = x^k + t^k d^k \tag{20}$$

$$r^{k+1} = b - Ax^{k+1} = r^k - t^k A d^k \tag{21}$$

$$d^{k+1} = r^{k+1} + \frac{(r^{k+1})^* r^{k+1}}{(r^k)^* r^k} d^k \tag{22}$$



[4]

5.1 Some Details

(i) Ohne Rundungsfehler konvergiert das CG-Verfahren in maximal n Schritten

(ii) Fehler fallen monoton und können durch weitere Schritte eliminiert werden [5]

5.2 Pseudo Code

Algorithm 6: conjugategradients

Data: $(a_{i,j})_{i,j=1,\dots,n}$; $(b_i)_{i=1,\dots,n}$
Result: $(q_{i,j})_{i,j=1,\dots,n}$, $(r_{i,j})_{i,j=1,\dots,n}$
begin
  if $n == 1$ **then**
    return $b_1/a_{1,1}$
  end
  else
    $\vec{x} = \vec{0}$
    $\vec{d} = \vec{b}$
    $\vec{r} = \vec{b}$
    $s = ||\vec{r}||_2^2$
    for $k = 1$ **to** n **do**
      $\vec{ad} = A\vec{d}$
      $c = \vec{ad} \circ \vec{d}$
      if $c == 0$ **then**
        return $\vec{x}$
      end
      $t = \frac{s}{c}$
      $\vec{x} = \vec{x} + t\vec{d}$
      $\vec{r} = \vec{r} - t\vec{ad}$
      $e = ||\vec{r}||_2^2$
      if $\sqrt{e} < tol$ **then**
        return $\vec{x}$
      end
      $\vec{d} = \vec{r} + \frac{\vec{d}}{s} \times \vec{d}$ $s = e$
    end
  end
end

6 Implementation

GitLab

References

[1] Kenneth Lange; Algorithms from the Book; SIAM; Philadelphia; 12. Mai 2022.

[2] Helmut Abels; Skript zur Vorlesung Numerik I; Regensburg; 6. Februar 2019.

[3] https://de.wikipedia.org/wiki/QR-Zerlegung; 18. November 2022.

[4] Daniel Weiß; 06: cg-Verfahren für lineare Gleichungssysteme, Minimierung eines Funktionals, Energienorm; https://www.youtube.com/watch?v=UBxXJK9DHBo; Karlsruhe; 25. Februar 2015.

[5] https://de.wikipedia.org/wiki/CG-Verfahren 18. November 2022.