ulm university universität uulm

Fakultät für Mathematik und Wirtschaftswissenschaften

Institut für Numerische Mathematik

Cache-optimierte QR-Zerlegung

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Florian Krötz
florian.kroetz@uni-ulm.de

Gutachter:
Dr. Michael Lehn
Dr. Andreas Borchert

Betreuer:
Dr. Michael Lehn

2018

Fassung 16. Juni 2018

Satz: PDF-LaTeX 2_ε

Inhaltsverzeichnis

1 Einleitung

Wenn du dich mit dem Teufel einlässt, verändert sich nicht der Teufel, der Teufel verändert dich. (Alexander Schell)

Wozu dient die QR-Zerlegung?

Warum muss die QR-Zerlegung schnell sein?

1.1 Cache

1.2 Intel MKL

Kapitel über die Wichtigkeit der Intel MKL.

1.2.1 Anwendung der QR-Zerlegung

-LGS lösen
-Ausgleichsprobleme lösen
-QR-Verfahren Eigenwerte b erechnen.

2 BLAS

Die Abkürzung BLAS steht für Basic Linear Algebra Subprograms. BLAS-Bibliotheken enthalten elementare Operationen der linearen Algebra.

2.1 Datenstruktur für Matrizen

Vollbesetzte Matrizen werden bei BLAS entweder zeilen- oder spaltenweise abgespeichert. Das bedeutet, dass entweder die Zeilen- oder die Spalten der Matrix hintereinander im Speicher stehen.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Matrix A Zeilenweise gespeichert

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

*data incRow

Matrix A Spaltenweise gespeichert

$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{1,2}$	$a_{2,2}$	$a_{3,2}$	$a_{1,3}$	$a_{2,3}$	$a_{3,3}$

*data incCol

Eine Datenstruktur benötigt folgende Elemente:

- einen Zeiger auf eine Speicherfläche
- Informationen ob die Matrix zeilen- oder spaltenweise gespeichert ist

- die Dimension der Matrix.

Eine derartige Datenstruktur könnte in C so aussehen.

```
struct Matrix {
  double * data;
  std::ptrdiff_t incRow, incCol;
  std::size_t numRows, numCols;
}
```

Für Intel MKL Routinen müssen die Matrizen zeilenweise gespeichert sein.

2.2 Einige BLAS-Routinen

Im Folgenden werden einige BLAS-Routinen beschrieben, die bei der QR-Zerlegung benutzt werden. BLAS-Routinen werden nach folgendem Schema benannt: Der erste Buchstabe im Namen gibt an für welchen Datentype die Funktion implementiert wurde. Der Rest beschreibt die Funktion der Funktion.
Beispiel „dgemm“: Das „d“ zeigt, dass die Funktion ist für Doubles gilt und „gemm“ steht für „generel Matrix Matrix“. Die Funktion berechnet also das Matrix-Matrix Produkt für Matrizen, deren Einträge Doubles sind.

2.2.1 Matrix-Matrix Produkt (gemm)

Die Funktion „gemm“ berechnet das Matrix-Matrix Produkt. Der Funktion werden die Matrizen A, B und C und die Skalare α und β übergeben. Außerdem werden 2 Flags übergeben ob die Matrizen A und B transponiert werden sollen.
Die Funktion berechnet

$$C \leftarrow \beta C + \alpha AB \tag{2.1}$$

Falls $\beta = 0$ wird die Matrix C zuerst mit Nullen initialisiert. Falls C Einträge hat die NaN (Not a Number) sind, werden diese somit mit 0 überschrieben.

[8] Blas tecnicla forum netlib

2.2.2 Matrix-Vektor Produkt (gemv)

Die Funktion „gemv“ berechnet das Matrix-Vektor Produkt. Der Funktion werden die Matrix A, die Vektoren x und y, und die Skalare α und β übergeben. Außerdem wird ein Flag übergeben, das anzeigt ob die Matrix A transponiert werden soll.
Die Funktion berechnet

$$y \leftarrow \beta y + \alpha Ax \tag{2.2}$$

Falls $\beta = 0$ wird der Vektor y zuerst mit Nullen initialisiert. Falls y Einträge hat die NaN (Not a Number) sind, werden diese mit 0 überschrieben.

2.2.3 Rank1 update (ger)

Die Funktion „ger“ berechnet ein dyadisches Produkt aus den Vektoren x und y, skaliert die daraus resultierende Matrix mit α und addiert das Ergebnis auf A.
Der Funktion werden die Matrix A, die Vektoren x und y und das Skalar α übergeben.
Die Funktion berechnet

$$A \leftarrow A + \alpha x y^T \tag{2.3}$$

2.2.4 Matrix-Matrix Produkt (trmm)

Die Funktion „trmm“ berechnet das Matrix-Matrix Produkt einer Dreiecksmatrix mit einer voll besetzten Matrix. Der Funktion wird die Dreiecksmatrix A, die Matrix B und das Skalar α übergeben. Außerdem werden Flags mit übergeben, die anzeigen ob A eine obere oder untere Dreiecksmatrix ist, ob A eine strikte oder unipotente Dreiecksmatrix ist, ob A von links oder rechts auf B multipliziert werden soll und ob A transponiert werden soll. Diese Eigenschaften werden unten in $op(\cdot)$ zusammengefasst.
Die Funktion berechnet

$$B \leftarrow \alpha \cdot op(A) \cdot B \qquad \text{oder} \qquad B \leftarrow \alpha \cdot B \cdot op(A) \tag{2.4}$$

2.2.5 Matrix-Vektor Produkt (trmv)

Die Funktion „trmv“ berechnet das Matrix-Vektor Produkt für Dreiecksmatrizen. Die Funktion berechnet

$$x \leftarrow \alpha A x \tag{2.5}$$

3 QR-Zerlegung

3.1 Definition

Eine Matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$ besitzt eine eindeutige QR-Zerlegung

$$A = QR \tag{3.1}$$

mit einer orthogonalen Matrix $Q \in \mathbb{R}^{m \times m}$ und einer oberen Dreiecksmatrix $R \in \mathbb{R}^{n \times n}$ [4].

Eine QR Zerlegung kann mit einer Householder-Transformation berechnet werden.

3.1.1 Beispiel

Lösung eines Minimierungsproblems

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 \tag{3.2}$$

mit Matrix $A \in \mathbb{R}^{m \times n}$ mit $rang(A) = n < m$ für die eine QR Zerlegung existiert. R besitzt die Gestalt

$$R = \begin{pmatrix} * & * & * \\ & * & * \\ & & * \\ \hline & 0 & \end{pmatrix} = \begin{pmatrix} \hat{R} \\ \hline 0 \end{pmatrix}$$

$\hat{R}$ stellt eine obere Dreiecksmatrix dar. Damit kann man das Minimierungsproblem wie folgt mit $A = QR$ modifizieren

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 = \min_{x \in \mathbb{R}^n} \|Q^T(Ax - b)\|^2 = \min_{x \in \mathbb{R}^n} \|Rx - Q^T b\|^2 \tag{3.3}$$

Also löst

$$Rx = Q^T b \tag{3.4}$$

das Minimierungsproblem (3.2). Da R eine Dreiecksmatrix ist, lässt sich (3.4) leicht mit Rückwärtseinsetzen lösen.

3.2 Householder-Transformation

Eine Matrix $H \in \mathbb{R}^{n \times n}$

$$H = I - 2\frac{vv^T}{v^T v} \tag{3.5}$$

wird als Householder-Transformation und der Vektor $v \in \mathbb{R}^n$ als Householder-Vektor bezeichnet. Eine Householder-Transformation $H = I - 2\frac{vv^T}{v^T v}$ ist orthogonal und symmetrisch [4].
Die Householder-Transformation spiegelt den Vektor x auf die Achse x_1. Dazu multipliziert man H von links auf x

$$Hx = \alpha e_1 \tag{3.6}$$

mit dem Skalar $\alpha \in \mathbb{R}$ und e_1 als ersten kanonischen Einheitsvektor. Der Householder-Vektor steht senkrecht auf der Ebene an welcher x gespiegelt wird.
Die Abbildung 3.1 veranschaulicht die Spiegelung des Vektors x an der gestrichelt eingezeichneten Ebene auf die Achse x_1.

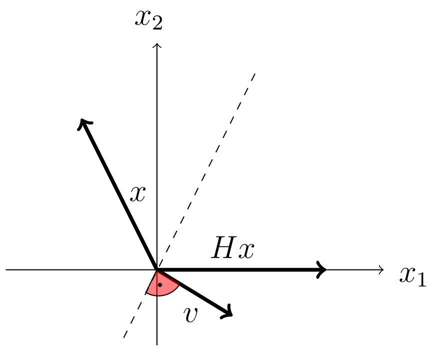

Abbildung 3.1: Beispiel Householder-Transformation mit $x = (-1, 2)^T$

3.2.1 Householder Vector

Damit (3.6) gilt, wird der Vektor berechnet, indem man (3.5) in (3.6) einsetzt

$$Hx = x - 2\frac{vv^T}{v^Tv}x = x - \underbrace{2\frac{v^Tx}{v^Tv}}_{\lambda} v = x - \lambda v \stackrel{!}{=} \alpha e_1$$
$$\implies v \in \text{span}\{x - \alpha e_1\}$$

Dadurch erhält man, dass v in dem Span $x - \alpha e_1$ liegt.[4]

Setzt man $v = t(x - \alpha e_1)$ in $Hx = \alpha e_1$(3.6) ein, dann erhält man

$$\begin{aligned} Hx =& x - \frac{2}{v^Tv}v(v^Tx) = x - 2\frac{v^Tx}{v^Tv}v \\ =& x - 2\frac{t(x-\alpha e_1)^Tx}{t(x-\alpha e_1)^Tt(x-\alpha e_1)}t(x-\alpha e_1) = x - 2\frac{(x-\alpha e_1)^Tx}{(x-\alpha e_1)^T(x-\alpha e_1)}(x-\alpha e_1) \\ =& x - \frac{(x-\alpha e_1)^Tx}{\|x-\alpha e_1\|_2^2}(x-\alpha e_1) = \underbrace{\left(1 - \frac{2(x-\alpha e_1)^Tx}{\|x-\alpha e_1\|_2^2}\right)}_{\stackrel{!}{=}0} x + \alpha e_1 \underbrace{\frac{2(x-\alpha e_1)^Tx}{\|x-\alpha e_1\|_2^2}}_{\stackrel{!}{=}1} \stackrel{!}{=} \alpha e_1 \end{aligned}$$

Damit das Letzte $=$ gilt muss gelten.

$$\begin{aligned} 1 &= \frac{2(x-\alpha e_1)^Tx}{\|x-\alpha e_1\|^2} \\ \Leftrightarrow (x-\alpha e_1)^T(x-\alpha e_1) &= 2x^Tx - 2\alpha x_1 \\ \Leftrightarrow x^Tx - 2\alpha x_1 + \alpha^2 &= 2x^Tx - 2\alpha x_1 \\ \Leftrightarrow \alpha &= \pm\sqrt{x^Tx} \end{aligned}$$

Das Vorzeichen von $\alpha = \pm\sqrt{x^Tx}$ kann man frei wählen, um $v = x - \alpha e_1$ zu berechnen.

Wählt man das Vorzeichen positiv, kann Auslöschung auftreten, falls x annähernd ein positives Vielfaches von e_1 ist.

LAPACK [7] vermeidet die Auslöschung, indem das Vorzeichen entgegengesetzt gewählt wird. Das bedeutet x wird immer auf die gegenüberliegende Seite gespiegelt.

Im Skript von Numerik 1 [4] wird das Vorzeichen immer positiv gewählt:

$\alpha = |\sqrt{x^T x}| = \|x\|_2$. Eine mögliche Auslöschung im Fall $x_1 > 0$ wird hier durch die folgende Umformung vermieden.

$$v_1 = x_1 - \|x\|_2 = \frac{x_1^2 - \|x\|_2^2}{x_1 + \|x\|_2} = \frac{-(x_2^2 + ... + x_n^2)}{x_1 + \|x\|_2}$$

Um den Vektor v später auf der frei werdenden Diagonalen von A speichern zu können, wird er auf $v_1 = 1$ normiert. Dies geschieht mit

$$v = \frac{x - \alpha e_1}{x_1 - \alpha} \tag{3.7}$$

Mit der Normierung kann man den Faktor $\tau = \frac{2}{v^T v}$ berechnen. Setze dazu (3.7) in die Definition von τ ein.

$$\tau = \frac{2}{v^T v} = \frac{2(x_1 - \alpha)^2}{(x - \alpha e_1)^T (x - \alpha e_1)} = \frac{2(x_1 - \alpha)^2}{\|x\|_2^2 - 2\alpha x^T e_1 + \alpha^2} = \frac{2(x_1 - \alpha)^2}{2\alpha(\alpha - x_1)} = \frac{x_1 - \alpha}{\alpha}$$

Mit dem Faktor $\tau = \frac{2}{v^T v}$ kann man die Householder-Transformation schreiben als

$$H = I - 2\frac{vv^T}{v^T v} = I - \tau vv^T$$

Algorithmus 1 Housholder-Vector(LAPACK DLARFG)

Input: $x \in \mathbb{R}^n$
$\alpha = -1 * \text{sign}(x_1)\|x\|_2$
$\tau = \frac{x_1 - \alpha}{\alpha}$
$v = \frac{x - \alpha e_1}{x_1 - \alpha}$
Output: Householder-Vektor v, τ

3.2.2 Householder-Transformation anwenden

Ein aufwändiges Matrix-Matrix-Produkt kann bei der Anwendung einer Housholder-Transformation $H = I - \tau vv^T$ auf die Matrix A umgangen werden, indem man geschickt klammert.

$$HA = (I - \tau vv^T)A = A - \tau(vv^T)A = A - \tau v(v^T A)$$

Statt eines Matrix-Matrix-Produkts muss man nur ein Matrix-Vektor-Produkt und ein dyadisches Produkt berechnen.

3.2.3 QR-Zerlegung mittels Housholder-Transformationen

Um A in eine obere Dreiecksmatrix R zu transformieren, wird eine Folge von Housholder-Transformationen auf A angewendet.

Zuerst wird aus der ersten Spalte der Matrix A ein Householder-Vektor berechnet, dann wird die Householder-Transformationen auf die Matrix angewandt. Diese Housholder-Transformation erzeugt Nullen in der ersten Spalte unterhalb es ersten Eintrags. Damit eine obere Dreiecksmatrix entsteht, wird als nächstes die Matrix A ohne die erste Zeile und Spalte betrachtet. Aus der ersten Spalte der neu betrachteten Matrix wird wieder ein Householder-Vektor berechnet und die Householder-Transformationen auf die Matrix angewandt. Fährt man nach diesem Schema immer weiter fort, entsteht eine obere Dreiecksmatrix.

$$H_1 A = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} \quad , \quad H_2 H_1 A = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix}$$

So erhält man die Faktorisierung

$$R = H_{n-1} H_{n-2} \cdot ... \cdot H_1 A \Leftrightarrow A = (H_1 \cdot ... \cdot H_{n-1}) R \Rightarrow Q = H_1 \cdot ... \cdot H_{n-1}$$

Q ist das Produkt aller Householder-Transformationen. Diese Vorgehensweise führt zum Algorithmus 2.

Algorithmus 2 Ungeblockte Housholder-Transformation.
Zur übersichtlicheren Beschreibung des Algorithmus werden die Bezeichnungen A_i und $\hat{a}_i$ eingeführt. A_i zeigt auf einen Matrixblock der am i-ten Diagonalelement beginnt. $\hat{a}_i$ zeigt auf die i-te Spalte unterhalb der Diagonalen. Matrizen sind 0-indiziert notiert.

1: Input: $A \in \mathbb{R}^{m \times n}$
2: **for** i = 0,1,2,..., n-1 **do**
3:     $(v_i, \tau_i) \leftarrow$ householdervector($\hat{a}_i$)
4:     $w \leftarrow v^T * A_i$ (dgemv)
5:     $A_i \leftarrow \tau * v * w + A_i$ (dger)
6:     **if** i < m **then**
7:         $\hat{a}_i \leftarrow v$
8:     **end if**
9: **end for**
10: Output: A QR zerlegt, Vektor $\tau \in \mathbb{R}^n$

Der Algorithmus 2 überschreibt die Matrix A mit R. Aufgrund der Dreiecksstruktur von R, können unter der Diagonale die Housholder-Vektoren gespeichert werden. Die Householder-Vektoren haben die Form

$$v^{(j)} = (\underbrace{0, ..., 0}_{j-1}, 1, v_{j+1}^{(j)}, ..., v_m^{(j)})$$

Da die ersten $j-1$ Einträge Null sind und der Vektor so normiert wurde, dass der j Eintrag gleich 1 ist, müssen die ersten j Einträge nicht gespeichert werden. Die Householder-Vektoren können somit unterhalb der Diagonalen gespeichert werden. Das geschieht im Algorithmus 2 in Zeile 7. Die Matrix A hat somit die Form

$$A = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ v_2^{(1)} & r_{2,2} & r_{2,3} \\ v_3^{(1)} & v_3^{(2)} & r_{3,3} \\ v_4^{(1)} & v_4^{(2)} & v_4^{(3)} \end{pmatrix}$$

3.3 Geblockte QR-Zerlegung

Ein geblockter Algorithmus ist sinnvoll, um bei großen Matrizen den Cache optimal zu nutzen.

Im Folgenden wird ein geblockter Algorithmus beschrieben wie er auch von LAPACK verwendet wird. Die entsprechende Funktion bei LAPACK heißt „DGEQRF“ [6].

Die Idee beim geblockten Algorithmus ist, die Matrix in Blöcke aufzuteilen, die geblockte QR-Zerlegung für die Blöcke zu berechnen und die dabei verwendeten Householder-Transformationen auf den Rest der Matrix anzuwenden.

Betrachte dazu die Matrix $A \in \mathbb{R}^{m \times n}$ geblockt, mit einer geeigneten Blockgröße bs.

$$A = \left(\begin{array}{c|c} A_{0,0} & A_{0,\text{bs}} \\ \hline A_{\text{bs},0} & A_{\text{bs},\text{bs}} \end{array} \right) \tag{3.8}$$

Die Abbildung 3.2 zeigt schematisch die Partitionierung von A.

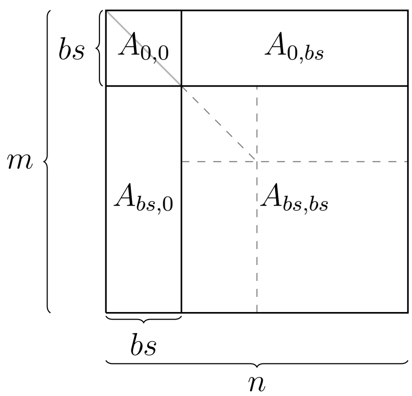

Abbildung 3.2: Aufteilung der Matrix A

Die Blockgröße bs wird so gewählt, dass die Geschwindigkeit der ungeblockten QR-Zerlegung für den Block $\left(\begin{array}{c} A_{0,0} \\ \hline A_{\text{bs},0} \end{array} \right)$ optimal ist.

Für diesen Block wird die QR-Zerlegung mit dem ungeblockten Algorithmus (Algorithmus 2) berechnet.

$$\left(\begin{array}{c} A_{0,0} \\ \hline A_{\text{bs},0} \end{array} \right) \leftarrow \left(\begin{array}{c} Q_{0,0} \backslash R_{0,0} \\ \hline Q_{\text{bs},0} \end{array} \right) \tag{3.9}$$

Im Block $A_{0,0}$ steht auf und über der Diagonalen $R_{0,0}$. Unterhalb der Diagonalen und im Block $A_{\text{bs},0}$ stehen die Householder-Vektoren.

Nun muss man die bei der ungeblocketn QR-Zerlegung verwendeten Housholder-Transformationen auf die restliche Matrix $\left(\frac{A_{0,\text{bs}}}{A_{\text{bs},\text{bs}}}\right)$ anwenden.

Das Produkt mehrerer Householder-Transformationen kann geschrieben werden als:

$$H_1 H_2 \cdots H_k = I - VTV^T \qquad \text{mit} \qquad H_i = I - \tau_i v_i v_i^T$$

[2]

Die Anwendung der Householder-Transformationen $I - V * T * V^T$ auf $\left(\frac{A_{\text{bs},\text{bs}}}{A_{\text{bs},\text{bs}}}\right)$ erfolgt in 2 Schritten. Zuerst wird die Matrix T berechnet. Dann wird $I - V * T * V^T$ auf $\left(\frac{A_{\text{bs},\text{bs}}}{A_{\text{bs},\text{bs}}}\right)$ angewandt.

$$\left(\frac{A_{0,\text{bs}}}{A_{bs,\text{bs}}}\right) \leftarrow H^T \left(\frac{A_{0,\text{bs}}}{A_{bs,\text{bs}}}\right) \tag{3.10}$$

Der Block $A_{bs,\text{bs}}$ wird erneut aufgeteilt. Das ist in Abbildung 3.2 gestrichelt dargestellt. Dies wird solange fortgesetzt, bis $A_{bs,\text{bs}}$ gleich der Blockgröße ist.

3.3.1 Berechnung der Matrix T

Die Matrix T wird in LAPACK von der Funktion „DLARFT“ berechnet [5].

Sie bekommt eine Dreiecksmatrix $V \in \mathbb{R}^{m \times k}$, einen Vektor $\tau \in \mathbb{R}^k$ und eine Matrix $T \in \mathbb{R}^{k \times k}$ übergeben.

In der Dreiecksmatrix V stehen die Householder-Vektoren, im Vektor τ die zu den Householder-Vektoren gehörende τ_i.

Die Funktion berechnet eine obere Dreiecksmatrix T so, dass

$$H_1 H_2 ... H_k = I - VTV^T \qquad \text{mit} \qquad H_i = I - \tau_i v_i v_i^T$$

Warum und wie das Verfahren funktioniert, wird hier beschreiben [2].

3.3.2 Anwenden von $I - VTV^T$

Die Anwendung der Householder-Transformationen auf eine Matrix C wird in LAPACK von der Funktion „LARFB" implementiert.

Die Funktion bekommt eine untere Dreiecksmatrix $V \in \mathbb{R}^{m \times k}$, eine obere Dreiecksmatrix $T \in \mathbb{R}^{k \times k}$ und eine Matrix $C \in \mathbb{R}^{m \times n}$ übergeben.

In der Dreiecksmatrix V stehen die Householder-Vektoren und T ist die zuvor berechnete Matrix. Die Matrix C wird upgedatet, indem die Matrix $I - VTV^T$ von rechts auf die Matrix C angewendet wird.

Ein weiterer Übergabeparameter gibt an, ob die Matrix $I - VTV^T$ transponiert werden soll. Die Funktion berechnet also

$$C \leftarrow HC = C - VTV^TC \quad \text{oder} \quad C \leftarrow H^TC = C - VT^TV^TC \tag{3.11}$$

Der Zweck der Funktion ist es, die Householder-Transformationen die bei der Bereicherung der QR-Zerlegung für einen Block entstanden sind, auf die restliche Matrix anzuwenden. Die Abbildung 3.3 zeigt, wie die Matrix A für die Funktion partitioniert wird.

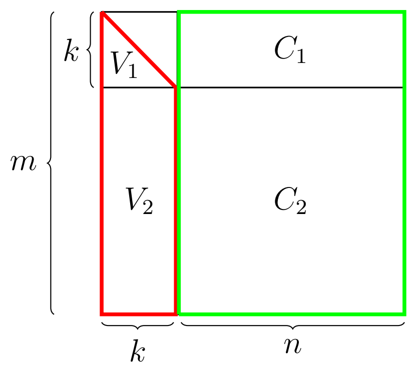

Abbildung 3.3: Partitionierung vom A für larfb

alls $m > k$ werden die Matrizen V und C aufgeteilt in $V = \begin{pmatrix} V_1 \\ \hline V_2 \end{pmatrix}$ und $C = \begin{pmatrix} C_1 \\ \hline C_2 \end{pmatrix}$.
Dabei wird V genau so geteilt, dass $V_1 \in \mathbb{R}^{k \times k}$ der quadratisch Dreiecksteil der Matrix ist und $V_2 \in \mathbb{R}^{m-k \times k}$ der Rest der Matrix. Die Matrix C wird in $C_1 \in \mathbb{R}^{k \times n}$ und $C_2 \in \mathbb{R}^{m-k \times n}$ aufgeteilt. Die Aufteilung ist so gewählt, dass das Matrix-Matrix-Produkt $V_1 \cdot C_1$ und $V_2 \cdot C_2$ möglich ist.
Diese Aufteilung ist notwendig, da die BLAS-Funktion trmm (matrix-matrix product

where one input matrix is triangular) nur für quadratische Dreiecksmatrizen implementiert ist.

Im Fall $m = k$ ist die Aufteilung nicht notwendig, da V quadratisch ist.

(3.11) kann mit der Umformung

$$\begin{aligned} C &\leftarrow C - VTV^TC \\ C &\leftarrow C - (VTV^TC)^{TT} \\ C &\leftarrow C - (C^TVT^TV^T)^T \end{aligned}$$

mit BLAS-Routinen wie in Algorithmus 3 berechnet werden.

Algorithmus 3 $I - VTV^T$ auf C anwenden.
Die Matrix W ist ein Workspace

Input: $V \in \mathbb{R}^{m \times k}$, $T \in \mathbb{R}^{k \times k}$, $C \in \mathbb{R}^{m \times n}$
$W \leftarrow C_1^T$ (copy)
$W \leftarrow W * V_1$ (trmm)
if m > k **then**
    $W \leftarrow W + C_2^T * V_2$ (gemm)
end if
$W \leftarrow W * T^T$ or $W * T$ (trmm)
if m > k **then**
    $C_2 \leftarrow C_2 - V_2 * W^T$ (gemm)
end if
$W \leftarrow W * V_1^T$ (trmm)
$C_1 \leftarrow C_1 - W^T$

4 Implementierung und Benchmarks

Die verwendete Bibliothek wurde in der Vorlesung High Performance Computing 1 entwickelt [3].

Die Bibliothek ist in C++ geschrieben. Es sind Klassen für Matrizen und Vektoren implementiert, sowie einige BLAS-Routinen.

Die Matrix-Klassen erlauben den zugriff auf Matrixblöcke.

eventuell Beispiel

4.1 Aufwand

D

4.2 Fehlerschätzer

Es wurde der Fehlerschätzer von ATLAS[1] verwendet.

$$err = \frac{\|A - QR\|_i}{\|A\|_i \cdot \min(m, n) \cdot \varepsilon} \tag{4.1}$$

$\|\cdot\|_i$ ist eine passende Norm. Die Matrizen Q und R sind die QR-Zerlegung der Matrix $A \in \mathbb{R}^{m \times n}$. ε ist die kleinste darstellbare Zahl.
Die QR-Zerlegung ist gut genug falls der Fehler kleiner 1 ist $err < 1$.

Als Norm wurde die Zeilensummennorm $\|\cdot\|_\infty$ gewählt. Diese ist für eine Matrix $A \in \mathbb{R}^{m \times n}$ gegeben durch

$$\|A\|_\infty = \max_{i=1,\ldots,m} \sum_{j=1}^{n} |a_{ij}|$$

ϵ ist auf dem Test-System $2.220446 \cdot 10^{-16}$

4.3 Benchmarks

4.3.1 Test System

Getestet wurde auf einem System mit einer Intel(R) Core(TM) i5-3470 CPU mit 3.20GHz.

Die Theoretische peak performance errechnet sich aus der Taktrate mal die Registerbreite mal 2.

Die CPU des Test Systems hat eine Taktrate von 3.20GHz. Die AVX-Register sind 256-Bit groß. Darin haben 4 double Platz.

$$\text{Taktrate} \cdot \text{Registerbreite} \cdot 2 = 3,20\ \text{GHz} \cdot 4 \cdot 2 = 25,6\ \text{GFLOPs}$$

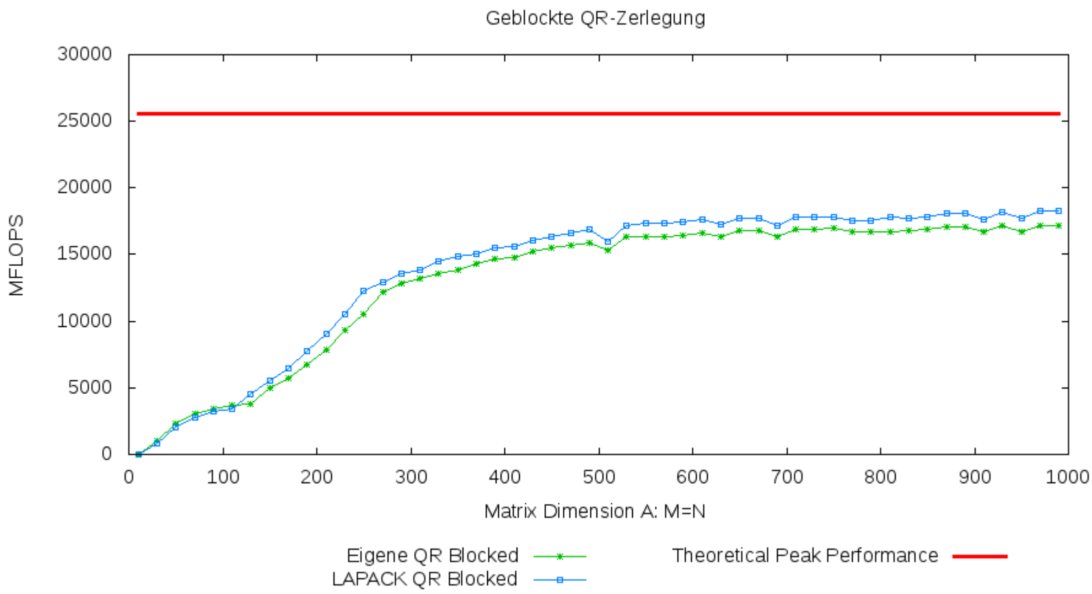

Abbildung 4.1: Benchmark geblockte QR-Zerlegung

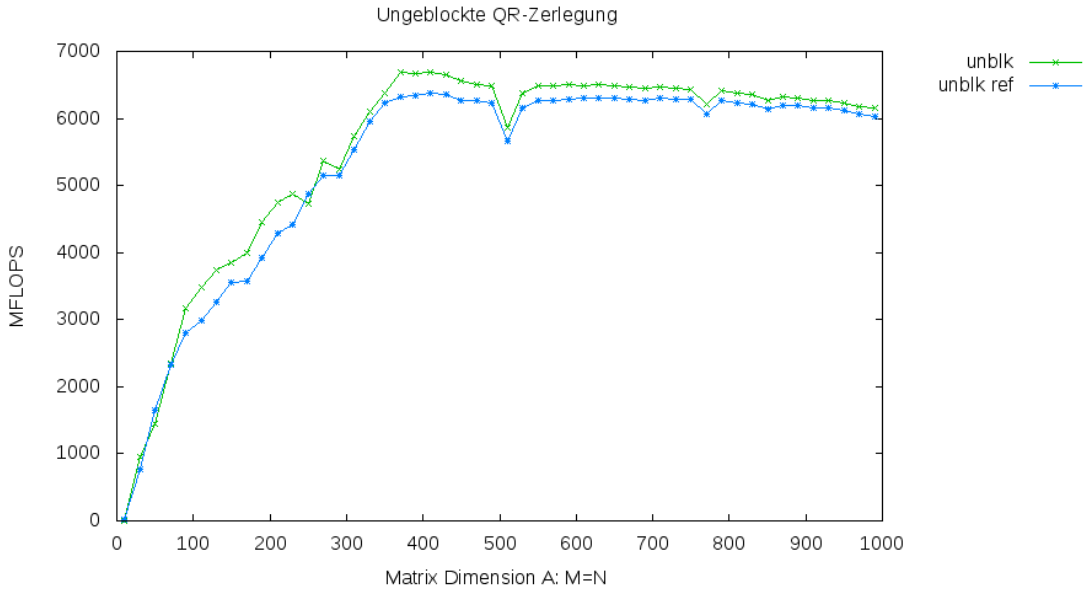

Abbildung 4.2: Benchmark ungeblockte QR-Zerlegung

A Block Reflector

Das Produkt aus Householder-Transformationen $H_1 \cdot ... \cdot H_n$ lässt sich schreiben als

$$H_1 \cdot ... \cdot H_n = I - VTV^T$$

mit einer unteren Dreiecksmatrix $V \in \mathbb{R}^{m \times n}$ die die Housholder-Vektoren enthält und eine oberen Dreiecksmatrix $T \in \mathbb{R}^{n \times n}$ [2]
Beweis:
n=2 Vorwärts

$$\begin{aligned} H_1 H_2 x = &(I - \tau_1 v_1 v_1^T)(I - \tau_2 v_2 v_2^T)x \\ =&(I - \tau_1 v_1 v_1^T - \tau_2 v_2 v_2^T + \tau_1 v_1 v_1^T \tau_2 v_2 v_2^T)x \\ =&x - \tau_1 v_1 v_1^T x - \tau_2 v_2 v_2^T x + \tau_1 \tau_2 v_1 (v_1^T v_2) v_2^T x \\ =&x - \tau_1 v_1 v_1^T x - \tau_2 v_2 v_2^T x + \tau_1 \tau_2 (v_1^T v_2) v_1 v_2^T x \end{aligned}$$

Rückwärts

$$\begin{aligned} H_{1,2} x &= (I - VTV^T)x = x - VTV^T x \\ &= x - (v_1, v_2) \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \end{pmatrix} x \\ &= x - (v_1, v_2) \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \begin{pmatrix} v_1^T x \\ v_2^T x \end{pmatrix} \\ &= x - (v_1, v_2) \begin{pmatrix} a v_1^T x + b v_2^T x \\ c v_2^T x \end{pmatrix} \\ &= x - v_1 (a v_1^T x + b v_2^T x) - v_2 (c v_2^T x) \\ &= x - a v_1 v_1^T x - b v_1 v_2^T x - c v_2 v_2^T x \end{aligned}$$

Koeffizienten Vergleich

$$\begin{aligned}
a &= \tau_1 \\
b &= -\tau_1\tau_2(v_1^T v_2) \\
c &= \tau_2 \\
T &= \begin{pmatrix} \tau_1 & -\tau_1\tau_2(v_1^T v_2) \\ 0 & \tau_2 \end{pmatrix}
\end{aligned}$$

n=3

Vorwärts

$$
\begin{aligned}
H_1H_2H_3x = (I - \tau_1 v_1 v_1^T)(I - \tau_2 v_2 v_2^T)(I - \tau_3 v_3 v_3^T)x \\
=&(I - \tau_1 v_1 v_1^T - \tau_2 v_2 v_2^T + \tau_1 v_1 v_1^T \tau_2 v_2 v_2^T)(I - \tau_3 v_3 v_3^T)x \\
=&(I - \tau_1 v_1 v_1^T - \tau_2 v_2 v_2^T - \tau_3 v_3 v_3^T \\
&+ \tau_1 v_1 v_1^T \tau_2 v_2 v_2^T + \tau_1 v_1 v_1^T \tau_3 v_3 v_3^T + \tau_2 v_2 v_2^T \tau_3 v_3 v_3^T \\
&- \tau_1 v_1 v_1^T \tau_2 v_2 v_2^T \tau_3 v_3 v_3^T)x \\
=&x - \tau_1 v_1 v_1^T x - \tau_2 v_2 v_2^T x - \tau_3 v_3 v_3^T x \\
&+ \tau_1 \tau_2 (v_1^T v_2) v_1 v_2^T x + \tau_1 \tau_3 (v_1^T v_3) v_1 v_3^T x + \tau_2 \tau_3 (v_2^T v_3) v_2 v_3^T x \\
&- \tau_1 \tau_2 \tau_3 (v_1^T v_2 v_2^T v_3) v_1 v_3^T x
\end{aligned}
$$

Rückwärts

$$
\begin{aligned}
H_{1,2,3}x = (I - VTV^T)x = x - VTV^Tx \\
= x - (v_1, v_2, v_3) \begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ v_3^T \end{pmatrix} x \\
= x - (v_1, v_2, v_3) \begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix} \begin{pmatrix} v_1^T x \\ v_2^T x \\ v_3^T \end{pmatrix} \\
= x - (v_1, v_2, v_3) \begin{pmatrix} a v_1^T x + b v_2^T x + c v_3^T \\ d v_2^T x + e v_3^T \\ f v_3^T \end{pmatrix} \\
=&x - v_1(a v_1^T x + b v_2^T x + c v_3^T x) \\
&- v_2(d v_2^T x + e v_3^T x) \\
&- v_3(f v_3^T) \\
=&x - a v_1 v_1^T x - b v_1 v_2^T x - c v_1 v_3^T x \\
&- d v_2 v_2^T x - e v_2 v_3^T \\
&- f v_3 v_3^T
\end{aligned}
$$

Koeffizienten Vergleich

$$
\begin{aligned}
a =& \tau_1 \\
b =& -\tau_1\tau_2(v_1^T v_2) \\
c =& -\tau_1\tau_2\tau_3(v_1^T v_2 v_2^T v_3) + \tau_1\tau_3(v_1^T v_3) \\
d =& \tau_2 \\
e =& -\tau_2\tau_3(v_2^T v_3) \\
f =& \tau_3 \\
T =& \begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix} = \begin{pmatrix} \tau_1 & -\tau_1\tau_2(v_1^T v_2) & -\tau_1\tau_2\tau_3(v_1^T v_2 v_2^T v_3) + \tau_1\tau_3(v_1^T v_3) \\ 0 & \tau_2 & -\tau_2\tau_3(v_2^T v_3) \\ 0 & 0 & \tau_3 \end{pmatrix}
\end{aligned}
$$

Mit Induktion kann man zeigen... siehe paper Im Paper wird gezeit wie man das verallgemeinern kann.

A.0.1 Orthogonal

Eine quadratische Matrix $Q \in \mathbb{R}^{n \times n}$ ist orthogonal, dann gilt

$$QQ^T = Q^T Q = I$$

Produkt orthogonaler Matrizen ist orthogonal. Sei $A^{-1} = A^T, B^{-1} = B^T$

$$(AB)^{-1} = B^{-1}A^{-1} = B^T A^T = (AB)^T$$

Die Househlder-Transformation $H = I - 2\frac{vv^T}{v^T v}$ ist symmetrisch und orthogonal das heißt $H^{-1} = H^T$
Da vv^T symmetrisch ist ($(vv^T)^T = vv^T$), folgt

$$H^T = \left(I - 2\frac{vv^T}{v^T v}\right)^T = I - 2\frac{vv^T}{v^T v} = H$$

Orthogonalität

$$HH^T = \left(I - 2\frac{vv^T}{v^T v}\right)\left(I - 2\frac{vv^T}{v^T v}\right) = I - 2\frac{vv^T}{v^T v} - 2\frac{vv^T}{v^T v} + \underbrace{4\frac{vv^T vv^T}{(v^T v)^2}}_{=4\frac{(v^T v)vv^T}{(v^T v)^2} = 4\frac{vv^T}{v^T v}} = I$$

$\Rightarrow H = I - VTV^T$ und Q sind orthogonal

Literaturverzeichnis

[1] *Automatically Tuned Linear Algebra Software (ATLAS)*. http://math-atlas.sourceforge.net/, . – [Online; zugegriffen 12-06-2018]

[2] JOFFRAIN, Thierry ; LOW, Tze M. ; QUINTANA-ORTÍ, Enrique S. ; GEIJN, Robert van d. ; ZEE, Field G. V.: Accumulating Householder Transformations, Revisited. In: *ACM Trans. Math. Softw.* 32 (2006), Juni, Nr. 2, 169–179. http://dx.doi.org/10.1145/1141885.1141886. – DOI 10.1145/1141885.1141886. – ISSN 0098–3500

[3] LEHN MICHAEL, Borchert A.: *Vorlesung High Performance Computing 1*. http://www.mathematik.uni-ulm.de/numerik/hpc/ws17/, 2017. – [Online; zugegriffen 31-05-2018]

[4] STEFAN A. FUNKEN, Karsten U.: *Einführung in die Numerische Lineare Algebra*. Ulm, Germany, 2016

[5] TENNESSEE, Univ. of California B. o. ; LTD.., NAG: *DLARFT forms the triangular factor T of a real block reflector H of order n, which is defined as a product of k elementary reflectors.* http://www.netlib.org/lapack/explore-3.1.1-html/dgeqrf.f.html#DGEQRF.1, 2006. – [Online; zugegriffen 12-06-2018]

[6] TENNESSEE, Univ. of California B. o. ; LTD.., NAG: *LAPACK blocked QR*. http://www.netlib.org/lapack/explore-3.1.1-html/dgeqrf.f.html, 2006. – [Online; zugegriffen 31-01-2018]

[7] TENNESSEE, Univ. of California B. o. ; LTD.., NAG: *LAPACK unblocked QR*. http://www.netlib.org/lapack/explore-3.1.1-html/dgeqr2.f.html, 2006. – [Online; zugegriffen 31-01-2018]

[8] TENNESSEE, University of: *BLAS Technical Forum*. http://www.netlib.org/blas/blast-forum/, 2001. – [Online; zugegriffen 03-06-2018]

Name: Florian Krötz Matrikelnummer: 884948

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ....

Florian Krötz