Blockmatrizen

Arthur Wettschereck

11. April 2012, Bonn

Definition

Definition:
Eine Matrix, deren Zeilen und Spalten in Blöcken zusammengefasst werden, nennt man Blockmatrix. Einträge von Blockmatrizen werden mit Block oder Untermatrix (Submatrix) bezeichnet.

Bezeichnung: Sei $A \in \mathbb{R}^{m \times n}$

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1r} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qr} \end{pmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix}$$
$$\begin{matrix} n_1 & & n_r \end{matrix}$$

Dann gilt:

- $m_1 + ... + m_q = m$
- $n_1 + ... + n_r = n$
- Block $A_{\alpha\beta}$ hat Dimension $m_\alpha \times n_\beta$

Doppelpunkt Notation

Um Blockmatrix Algorithmen beschreiben zu können, benötigt man die Doppelpunkt Notation. Sei $A \in \mathbb{R}^{m \times n}$ und seien $i = (i_1, ..., i_r)$, sowie $j = (j_1, ..., j_c)$ Vektoren, für die gilt:

$$i_1, ..., i_r \in \{1, 2, ..., m\}$$

$$j_1, ..., j_c \in \{1, 2, ..., n\}$$

Dann bezeichnen wir mit A(i,j) die folgende r×c Untermatrix:

$$A(i,j) := \begin{pmatrix} A(i_1, j_1) & \cdots & A(i_1, j_c) \\ \vdots & & \vdots \\ A(i_r, j_1) & \cdots & A(i_r, j_c) \end{pmatrix}$$

Wobei gilt: $A(i_u, j_v) = A_{i_u, j_v}$

Doppelpunkt Notation

Die Doppelpunkt Notation kann benutzt werden, um A(i,j) mittels Skalareinträgen zu beschreiben. Wenn $1 \leq i_1 \leq i_2 \leq m$ und $1 \leq j_1 \leq j_2 \leq n$, so beschreibt $A(i_1 : i_2, j_1 : j_2)$ die Matrix, die aus A entsteht, wenn man nur die Zeilen i_1 bis i_2 und die Spalten j_1 bis j_2 extrahiert.

Beispiel

$$A(2:4, 5:6) = \begin{pmatrix} a_{25} & a_{26} \\ a_{35} & a_{36} \\ a_{45} & a_{46} \end{pmatrix}$$

Addition von Blockmatrizen

Blockmatrizen lassen sich genauso wie gewöhnliche Matrizen addieren, vorrausgesetzt die Dimension der Matrizen und einzelnen Blöcke ist identisch.
Sei A wie anfangs bestimmt und

$$B = \begin{pmatrix} B_{11} & \cdots & B_{1r} \\ \vdots & & \vdots \\ B_{q1} & \cdots & B_{qr} \end{pmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix}$$
$$\begin{matrix} n_1 & & n_r \end{matrix}$$

Dann gilt für die Summe C = A + B:

$$C = \begin{pmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{pmatrix} = \begin{pmatrix} A_{11} + B_{11} & \cdots & A_{1r} + B_{1r} \\ \vdots & & \vdots \\ A_{q1} + B_{q1} & \cdots & A_{qr} + B_{qr} \end{pmatrix}$$

Lemma 1.3.1

Lemma (1.3.1)

Sei $A \in \mathbb{R}^{m \times p}$ *und sei* $B \in \mathbb{R}^{p \times n}$.

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix} \qquad B = \begin{matrix} (B_1, \ldots, B_r) \\ n_1 \qquad n_r \end{matrix}$$

Dann gilt:

$$AB = C = \begin{matrix} n_1 & & n_r \\ \begin{pmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{pmatrix} & \begin{matrix} m_1 \\ \\ m_q \end{matrix} \end{matrix}$$

Dabei gilt: $C_{\alpha\beta} = A_\alpha B_\beta$ *mit* $\alpha \in \{1, \ldots, q\}$ *und* $\beta \in \{1, \ldots, r\}$.

Beweis

Skalareinträge im Block $C_{\alpha\beta}$ mit denen in C in Verbindung bringen:
$(C_{\alpha\beta})_{ij} = c_{\lambda+i,\mu+j}$ mit:

$\lambda = m_1 + \cdots + m_{\alpha-1}$

$\mu = n_1 + \cdots + n_{\beta-1}$

$$c_{\lambda+i,\mu+j} = \sum_{k=1}^{p} a_{\lambda+i,k} b_{k,\mu+j}$$

$$= \sum_{k=1}^{p} (A_\alpha)_{ik} (B_\beta)_{kj}$$

$$= (A_\alpha B_\beta)_{ij}$$

Lemma 1.3.2

Lemma (1.3.2)

Sei $A \in \mathbb{R}^{m \times p}$ *und sei* $B \in \mathbb{R}^{p \times n}$.

$$A = \underset{p_1}{(A_1, \ldots, A_s)} \underset{p_s}{} \qquad B = \begin{pmatrix} B_1 \\ \vdots \\ B_s \end{pmatrix} \begin{matrix} p_1 \\ \\ p_s \end{matrix}$$

Dann gilt:

$$AB = C = \sum_{\gamma=1}^{s} A_\gamma B_\gamma$$

Beweis

$p_\gamma := p_1 + \cdots + p_{\gamma-1}$ mit $\gamma \in \{1, \ldots, s+1\}$

Wissen dass: $c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$

$$= \sum_{k=p_1+1}^{p_2} a_{ik} b_{kj} + \sum_{k=p_2+1}^{p_3} a_{ik} b_{kj} + \cdots + \sum_{k=p_s+1}^{p_{s+1}} a_{ik} b_{kj}$$

$$= \sum_{\gamma=1}^{s} \sum_{k=p_\gamma+1}^{p_{\gamma+1}} a_{ik} b_{kj}$$

$$= [A_1 B_1]_{ij} + [A_2 B_2]_{ij} + \cdots + [A_s B_s]_{ij}$$

$$= \sum_{\gamma=1}^{s} [A_\gamma B_\gamma]_{ij}$$

Matrixmultiplikation in der Theorie

Theorem (1.3.3)

Sei

$$A = \begin{matrix} \\ m_1 \\ \\ m_q \end{matrix} \overset{\begin{matrix} p_1 & & p_s \end{matrix}}{\begin{pmatrix} A_{11} & \cdots & A_{1s} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qs} \end{pmatrix}}, \quad B = \begin{matrix} \\ p_1 \\ \\ p_s \end{matrix} \overset{\begin{matrix} n_1 & & n_r \end{matrix}}{\begin{pmatrix} B_{11} & \cdots & B_{1r} \\ \vdots & & \vdots \\ B_{s1} & \cdots & B_{sr} \end{pmatrix}}$$

Wir unterteilen die Matrix $C := AB$ wie folgt:

$$C = \begin{matrix} \\ m_1 \\ \\ m_q \end{matrix} \overset{\begin{matrix} n_1 & & n_r \end{matrix}}{\begin{pmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{pmatrix}}$$

Dann gilt: $C_{\alpha\beta} = \sum_{\gamma=1}^{s} A_{\alpha\gamma} B_{\gamma\beta}$

Beweis

$$A_\gamma := \begin{pmatrix} A_{1\gamma} \\ \vdots \\ A_{q\gamma} \end{pmatrix}, B_\gamma := \begin{pmatrix} B_{\gamma 1} & \dots & B_{\gamma r} \end{pmatrix}$$

Lemma 1.3.2 $\implies$ C$= \sum_{\gamma=1}^{s} A_\gamma B_\gamma$

$$c_{ij} = (\sum_{\gamma=1}^{s} A_\gamma B_\gamma)_{ij}$$

$$= \sum_{\gamma=1}^{s} (A_\gamma B_\gamma)_{ij}$$

$$= \text{(Lemma 1.3.1)} \sum_{\gamma=1}^{s} A_{i\gamma} B_{\gamma j}$$

Blockmatrix mal Vektor

Aus Theorem 1.3.3 folgt die Multiplikation einer Blockmatrix mit einem Vektor. Wir werden uns nun den gaxpy y=Ax+y anschauen, wobei $A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n$, y $\in \mathbb{R}^m$, und:

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix} \qquad y = \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix}$$

Dabei bezeichnet A_i die i.te Blockzeile. Der Vektor $m.vec = (m_1, \ldots, m_q)$ bezeichne im Folgenden die Zeilenhöhe der einzelnen Blöcke.

Blockmatrix mal Vektor

Dann induziert der gaxpy

$$\begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix} = \begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix} \text{x} + \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix}$$

den folgenden Algorithmus:

last = 0
for i = 1:q
first = last + 1
last = frist + m.vec(i) - 1
y(first:last) = A(first:last,:)x + y(first:last)
end

Bei jedem Schleifendurchlauf wird ein normaler, auf Skalareinträgen basierender gaxpy ausgeführt. Dies kann z.B. mit Algorithmus 1.1.3 oder 1.1.4 durchgeführt werden.

Blockmatrix mal Vektor

Es ist auch möglich, die Matrix mittels Blockspalten zu beschreiben:

$$\underset{n_1}{\mathrm{A}} = \underset{n_1}{(A_1} \underset{n_r}{, \ldots, A_r)} \qquad \mathrm{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_r \end{pmatrix} \begin{matrix} n_1 \\ \\ n_r \end{matrix}$$

In diesem Fall bezeichnet A_j den j.ten Spaltenblock von A. Der Vektor $n.vec = (n_1, \ldots, n_r)$ bezeichne die Spaltenbreite.

Blockmatrix mal Vektor

Dann induziert der gaxpy:

$$y = (A_1, \ldots, A_r) \begin{pmatrix} x_1 \\ \vdots \\ x_r \end{pmatrix} + y = \sum_{j=1}^{r} A_j x_j + y$$

den folgenden Algorithmus:

last = 0
for j = 1:r
    first = last + 1
    last = frist + n.vec(j) - 1
    y = A(:,first:last)x(first:last) +y
end

Matrixmultiplikation

Je nach Blockung der Matrizen A, B und C lässt sich die Blockmatrixmultiplikation folgendermaßen durchführen:

- Skalarprodukt
- saxpy
- dyadisches Produkt

Seien $A = (A)_{\alpha\beta}$, $B = (B)_{\alpha\beta}$ und $C = (C)_{\alpha\beta} \in \mathbb{R}^{n\times n}$ mit N×N Blöcken der Dimension $\ell \times \ell$.

Dann folgt aus Theorem 1.3.3:

$$C_{\alpha\beta} = \sum_{\gamma=1}^{N} A_{\alpha\gamma} B_{\gamma\beta} + C_{\alpha\beta} \text{ mit } \alpha = 1 : N, \beta = 1 : N$$

Matrixmultiplikation

Daraus resultiert der folgende Algorithmus, in Anlehnung an Algorithmus 1.1.5 (ijk Variante)

Algorithmus 1.3.3:

for $\alpha = 1 : N$
  $i = (\alpha - 1)l + 1 : \alpha\ell$
  for $\beta = 1 : N$
    $j = (\beta - 1)l + 1 : \beta\ell$
    for $\gamma = 1 : N$
      $k = (\gamma - 1)l + 1 : \gamma\ell$
      $C(i,j) = A(i,k)B(k,j) + C(i,j)$
    end
  end
end

Im Falle von ℓ=1, ist α = i, β = j und γ = k und wir haben wieder den Algorithmus 1.1.5.

Matrixmultiplikation variabler Größe

Algorithmus 1.3.3 für variable Blockgrößen
Im folgenden sei:

- m.vec(i) die Zeilengröße des i.ten Blocks von A
- n.vec(j) die Spaltengröße des j.ten Blocks von B
- o.vec(k) die Spaltengröße des k.ten Blocks von A und somit auch die Zeilengröße des k.ten Blocks von B

Matrixmultiplikation variabler Größe

$i_{first} = 1 \quad i_{last} = \mathrm{m.vec}(1)$
for $\alpha = 1 : N$
  $j_{first} = 1 \quad j_{last} = \mathrm{n.vec}(1)$
  $i = i_{first} : i_{last}$
  for $\beta = 1 : N$
    $k_{first} = 1 \quad k_{last} = \mathrm{o.vec}(1)$
    $j = j_{first} : j_{last}$
    for $\gamma = 1 : N$
      $k = k_{first} : k_{last}$
      $C(i,j) = A(i,k)B(k,j) + C(i,j)$
      $k_{first} = k_{last} + 1 \quad k_{last} = k_{last} + o.vec(\gamma)$
    end
      $j_{first} = j_{last} + 1 \quad j_{last} = j_{last} + n.vec(\beta)$
  end
  $i_{first} = i_{last} + 1 \quad i_{last} = i_{last} + m.vec(\alpha)$
end

Block saxpy - Matrixmultiplikation

Es ist auch möglich, die Blockmatrizmultiplikation als saxpy auszuführen.
Dafür blocken wir C = AB + C wie folgt:

$$\begin{pmatrix} C_1 & \dots & C_N \end{pmatrix} = \begin{pmatrix} A_1 & \dots & A_N \end{pmatrix} \begin{pmatrix} B_{11} & \dots & B_{1N} \\ \vdots & \ddots & \vdots \\ B_{N1} & \dots & B_{NN} \end{pmatrix} + \begin{pmatrix} C_1 & \dots & C_N \end{pmatrix}$$

mit $A_\alpha, C_\alpha \in \mathbb{R}^{n \times \ell}$ und $B_{\alpha\beta} \in \mathbb{R}^{\ell \times \ell}$
Daraus folgt die Blockmatrix Version des Algorithmus 1.1.7.

Block saxpy - Matrixmultiplikation

Algorithmus 1.3.4:

$j_{first} = 1$
$j_{last} = \text{n.vec}(1)$
for $\beta = 1 : n$
  $k_{first} = 1$
  $k_{last} = \text{o.vec}(1)$
  $j = j_{first} : j_{last}$
  for $\gamma = 1 : n$
    $k = k_{first} : k_{last}$
    $C(:, j) = A(:, k)B(k, j) + C(:, j)$
    $k_{first} = k_{last} + 1$
    $k_{last} = k_{last} + o.vec(\gamma)$
  end
  $j_{first} = j_{last} + 1$
  $j_{last} = j_{last} + n.vec(\beta)$
end

dyadisches Produkt - Matrixmultiplikation

Einen auf dem dyadischen Produkt basierenden Algorithmus erhalten wir, indem wir die Matrizen A und B wie folgt blocken:

$A = \begin{pmatrix} A_1, \ldots, A_N \end{pmatrix}$ $B = \begin{pmatrix} B_1^T \\ \vdots \\ B_N^T \end{pmatrix}$ mit $A_\gamma, B_\gamma \in \mathbb{R}^{n \times \ell}$. Aus Lemma 1.3.2 wissen wir:

$$C = \sum_{\gamma=1}^{N} A_\gamma B_\gamma^T + C$$

dyadisches Produkt - Matrixmultiplikation

Algorithmus 1.3.5:

$k_{first} = 1$
$k_{last} = o.vec(1)$
for $\gamma = 1 : N$
  $k = k_{first} : k_{last}$
  $C = A(:, k)B(k, :) + C$
  $k_{first} = k_{last} + 1$
  $k_{last} = k_{last} + o.vec(\gamma)$
end

Strassenmultiplikation

Als nächstes betrachten wir die Strassenmultiplikation:

- herkömmliche Multiplikation 2er 2x2 Blockmatrizen benötigt 8 Multiplikationen und 4 Additionen
- herkömmliche Multiplikation hat eine Komplexität von $\mathcal{O}(n^3)$
- bei der Stassenmultiplikation werden nur 7 Multiplikationen, aber dafür 15 Additionen benötigt
- Strassens Algorithmus lässt sich rekursiv anwenden
- Strassens Algorithmus hat eine Komplexität von $\mathcal{O}(n^{2,807})$

Strassenmultiplikation

Es ist jedoch auch möglich, die Matrix C auf folgende Art zu berechnen:

$$
\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_2 &= (A_{21} + A_{22})B_{11} \\
P_3 &= A_{11}(B_{12} - B_{22}) \\
P_4 &= A_{22}(B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{12})B_{22} \\
P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 \\
C_{22} &= P_1 + P_3 - P_2 + P_6
\end{aligned}
$$

Strassenmultiplikation

Strassens Idee lässt sich nun auf jeder der Teilmatrizen A_{ij}, B_{ij} mit $1 \leq i, j \leq 2$ wieder anwenden. Ab einer gewissen Rekursionstiefe sind die nxn-Teilmatrizen so klein, dass sich die herkömmliche Matrixmultiplikation lohnt ($n \leq n_{min}$). Daraus folgt der Algorithmus 1.3.1:

Strassenmultiplikation

function: $C = \textbf{strass}(A, B, n, n_{min})$

if $n \leq n_{min}$

$C = AB$

else

$m = n/2; u = 1 : m; v = m + 1 : n$

$P1 = strass(A(u,u) + A(v,v), B(u,u) + B(v,v), m, n_{min}$

$P2 = strass(A(u,u) + A(v,v), B(u,u), m, n_{min}$

$P3 = strass(A(u,u), B(u,v) - B(v,v), m, n_{min}$

$P4 = strass(A(v,v), B(v,u) - B(u,u), m, n_{min}$

$P5 = strass(A(u,u) + A(u,v), B(v,v), m, n_{min}$

$P6 = strass(A(v,u) - A(u,u), B(u,u) + B(u,v), m, n_{min}$

$P7 = strass(A(u,v) - A(v,v), B(v,u) + B(v,v), m, n_{min}$

$C(u,u) = P_1 + P_4 - P_5 + P_7$

$C(u,v) = P_3 + P_5$

$C(v,u) = P_2 + P_4$

$C(v,v) = P_1 + P_3 - P_2 + P_6$

end

Strassenmultiplikation - Komplexität

Strassens Algorithmus hat Komplexität $\mathcal{O}(n^{2,807})$.
Da die Matrixmultiplikation teurer ist als die Addition, werden wir uns nur erstere anschauen. Dafür müssen wir die tiefste Rekursionsebene betrachten:
Dort finden 7^{q-d} Matrix Multiplikationen der Größe 2^d statt. Daraus folgt:

$s = (2^d)^3 7^{q-d}$ Multiplikationen

Für die herkömmliche Multiplikation gilt:

$c = (2^q)^3$

Daraus folgt:

$\frac{s}{c} = (\frac{2^d}{2^q})^3 7^{q-d} = (\frac{7}{8})^{q-d}$

Wenn wir d = 0 setzen, also nur Matrizen der Größe 1 multiplizieren, folgt daraus:

$s = (\frac{7}{8})^q c = 7^q = n^{log_2 7} \approx n^{2,807}$

Strassenmultiplikation

Der vorgestellte Algorithmus funktioniert nur für quadratische Matrizen der Größe 2^k. Um quadratische Matrizen beliebiger Größe multiplizieren zu können, muss man folgende Hilfsfunktionen einführen:
AddDim(A)
DeleteDim(A)

Wenn die aktuelle Matrix eine ungerade Zeilen/Spaltenzahl hat, würde man mittels AddDim eine 0 Zeile und Spalte hinzufügen, so dass wieder eine Matrix mit gerader Zeilen/Spaltenzahl entsteht. Nachdem man die Matrix C in der Rekursionsebene bestimmt und befüllt hat, würde man mittels DeleteDim die letzte Zeile und Spalte wieder löschen.
Es lässt sich nachrechnen, dass die Nullzeile auf das Ergebnis der Multiplikation und Addition der anderen Zeilen/Spalten keinen Einfluss hat.

Winograd

Es gibt noch einige andere Algorithmen als den von Strassen, zB einen von Winograd. Die Idee ist:
f(x): $= \sum_{i=1}^{\frac{n}{2}} x_{2i-1}x_{2i}$
Dann gilt:
$x^T y = \sum_{i=1}^{\frac{n}{2}}(x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - f(x) - f(y)$

Diese Summe lässt sich verwenden, um die Matrizenmultiplikation A+B = C zu vereinfachen:

- f(x) wird auf jede Zeile von A angewendet ($\frac{n^2}{2}$ Operationen)
- f(x) wird auf jede Spalte von A angewendet ($\frac{n^2}{2}$ Operationen)
- Anstelle der üblichen Multiplikation von Zeile und Spalte wird nun die obige Summe benutzt ($\frac{n^3}{2}$ Operationen)
- Gesamt Komplexität: $\frac{n^3}{2} + n^2$

Komplexe Matrix Multiplikation

Betrachten wir das komplexe Matrixmultiplikationsupdate

$$C_1 + iC_2 = (A_1 + iA_2)(B_1 + iB_2) + (C_1 + iC_2),$$

wobei alle Matrizen reellwertig sind und i die imaginäre Einheit ist. Vergleicht man den reellen und imaginären Teil, so folgt

$$C_1 = A_1B_1 - A_2B_2 + C_1$$
$$C_2 = A_1B_2 - A_2B_1 + C_2.$$

Dies lässt sich auch in folgender Weise ausdrücken:

$$\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} + \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

Daraus lässt sich ansatzweise erkennen, wie komplexe Matrizen in Software gehandhabt werden. Der einzige Nachteil ist, dass die Matrizen A_1 *und* A_2 doppelt gespeichtert werden müssen.

Transponierte Blockmatrizen

Sei $A = \begin{pmatrix} A_{11} & \dots & A_{1r} \\ \vdots & \ddots & \vdots \\ A_{q1} & \dots & A_{qr} \end{pmatrix}$

Dann ist das die transponierte Matrix $A^T = \begin{pmatrix} A_{11}^T & \dots & A_{q1}^T \\ \vdots & \ddots & \vdots \\ A_{1r}^T & \dots & A_{qr}^T \end{pmatrix}$

Schurkomplement

Definition:

Sei $A = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$ eine Blockmatrix.

A\E $:= H - GE^{-1}F$ heißt Schurkomplement von E in A.

Aufgabe

Sei A eine quadratische Matrix: $A = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$

E,F,G,H seien selbst Matrizen, wobei det E $\neq 0$.

Zeigen Sie, dass A genau dann regulär ist, wenn die Matrix A\E = H - GE^{-1}F regulär ist, und es gilt det A = det E * det A\E

Lösung

Sei A nun regulär.

$\Longrightarrow$ det(A) $\neq 0$ und damit A invertierbar

Führe Zeilenoperation aus:

Addiere $-GE^{-1}$-fache der ersten Zeile zur zweiten:

$\Longrightarrow$ A= $\begin{pmatrix} E & F \\ 0 & H - GE^{-1}F \end{pmatrix}$

$\Longrightarrow$ det A = det E * det A\E

Aus det A, det E $\neq 0$ folgt det A\E $\neq 0$

Sei nun A\E regulär.

Wie oben gesehen gilt auch hier:

det A = det E * det A\E $\neq 0$.

Positiv definite Matrizen

Definition:
Eine Matrix A heißt positiv definit, wenn für alle Vektoren $x \neq 0$ (mit $||x||_2 = 1$) gilt: $x^T * A * x > 0$.

Corollar

Positiv definite Matrizen sind regulär.

Beweis:
Sei A eine spd Matrix. Angenommen A wäre nicht regulär: Dann existiert $x \neq 0$ so, dass x^T * A $= 0$, da der Zeilenrang nicht voll ist. Daraus folgt: $x^T * A * x = 0$. Dies steht im Widerspruch dazu, dass A positiv definit ist.

Aufgabe

Sei A $= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ eine symmetrisch positiv definite Matrix (spd-Matrix). Man zeige:

a) A_{22} ist regulär.

b) Sei $\vec{x} = \begin{pmatrix} \vec{x_1} \\ \vec{x_2} \end{pmatrix}$. Dann gilt $(S\vec{x_1}, \vec{x_1}) = min_{\vec{x_2}}(A\vec{x}, \vec{x})$ mit dem Schur-Komplement S $= A \backslash A_{22} = A_{11} - A_{12}A_{22}^{-1}A_{21}$

c) Das Schur-Komplement ist positiv definit.

Lösung Teil a

zZ: A_{22} ist regulär.
Wir werden zeigen: A_{22} ist spd und somit regulär.
Sei im folgenden $A_{11} \in \mathbb{R}^{m \times m}, A_{12} \in \mathbb{R}^{m \times n}, A_{21} \in \mathbb{R}^{n \times m}, A_{22} \in \mathbb{R}^{n \times n}$

zZ: $\vec{x}^T * A_{22} * \vec{x} > 0, \quad \forall \vec{x} \neq 0, \vec{x} \in \mathbb{R}^n$
$\vec{u} := \begin{pmatrix} 0 \\ \vec{x} \end{pmatrix}, \vec{u} \in \mathbb{R}^{m+n}$
Dann gilt: $u^T * A * u > 0$ (weil A positiv definit ist)
Nachrechnen zeigt: $\vec{x}^T * A_{22} * \vec{x} = u^T * A * u$
$\Longrightarrow$ A_{22} ist spd und somit regulär.

Lösung Teil b

zZ: $(S\vec{x_1}, \vec{x_1}) = min_{\vec{x_2}}(A\vec{x}, \vec{x})$

Es gilt: $(S\vec{x_1}, \vec{x_1}) = -\vec{x_1}^T A_{11}\vec{x_1} - \vec{x_1}^T A_{12}A_{22}^{-1}A_{12}^T\vec{x_1}$
$(A\vec{x}, \vec{x}) = (\vec{x_2}+A_{22}^{-1}A_{21}\vec{x_1})^T A_{22}(\vec{x_2}+A_{22}^{-1}A_{21}\vec{x_1})+\vec{x_1}^T(A_{11}-A_{12}A_{22}^{-1}A_{21})\vec{x_1}$
Da A_{22} positiv definit ist, ist der Term
$(\vec{x_2} + A_{22}^{-1}A_{21}\vec{x_1})^T A_{22}(\vec{x_2} + A_{22}^{-1}A_{21}\vec{x_1})$ größer Null für alle
$\vec{x_2} \neq -A_{22}^{-1}A_{21}\vec{x_1}$ und für diesen gleich Null. Somit haben wir das Minimum gefunden
Explizites Nachrechnen zeigt, dass die Terme gleich sind.

Lösung Teil c

zZ: Das Schur-Komplement ist positiv definit.

Aus Teil b) wissen wir:
$(S\vec{x_1}, \vec{x_1}) = min_{\vec{x_2}}(A\vec{x}, \vec{x})$

Definiere $\vec{x}$ wie in Teil b). Wähle $\vec{x_2}$ so, dass die Minimalitätseigenschaft von Teil b) erfüllt ist und $\vec{x_1}$ beliebig, aber ungleich 0. Dann gilt:
$\vec{x_1}^T * S * \vec{x_1} = \vec{x}^T * A * \vec{x} > 0$
$\Longrightarrow$ Behauptung.