Iterative Methoden zur Lösung von Gleichungssystemen

Matthias Heming

Bergische Universität Wuppertal

29. Januar 2009

Zielsetzung des Vortrags

Die Teilnehmer sollen . . .

- Iterationsverfahren in die Kategorien stationär und nicht stationär zuordnen können.
- ein intuitives Verständnis des Verfahrens der konjugierten Gradienten (CG) entwickeln.
- iterative Methoden mit Sparse-Matrizen nutzen.
- ein Gefühl für die Leistungsfähigkeit der dargestellten Methoden auch im Vergleich zu Direktlösern bekommen.

Inhalt und Ablauf

Das Problem

- Betrachte das Gleichungsystem über $\mathbb{R}$:

$$Ax = b \qquad x, b \in \mathbb{R}$$

Das Problem

- Betrachte das Gleichungsystem über $\mathbb{R}$:

$$Ax = b \qquad x, b \in \mathbb{R}$$

- Erste Einschränkung: Nur quadratische Matrizen!

$$Ax = b \qquad A \in \mathbb{R}^{n \times n} \qquad x, b \in \mathbb{R}$$

Das Problem

- Betrachte das Gleichungsystem über $\mathbb{R}$:

$$Ax = b \qquad x, b \in \mathbb{R}$$

- Erste Einschränkung: Nur quadratische Matrizen!

$$Ax = b \qquad A \in \mathbb{R}^{n \times n} \qquad x, b \in \mathbb{R}$$

- Zweite Einschränkung: Nur reguläre Matrizen!

Es gibt eine eindeutige Lösung.

Überblick

Herstellung einer Iterationsvorschrift

Wähle Zerlegung: $A = M - N$ mit regulärem M

$$(M - N)x = b$$
$$x = M^{-1}Nx + M^{-1}b$$

Herstellung einer Iterationsvorschrift

Wähle Zerlegung: $A = M - N$ mit regulärem M

$$(M - N)x = b$$
$$x = M^{-1}Nx + M^{-1}b$$

Erzeugt Interationsvorschrift:

$$x^{(i+1)} = M^{-1}Nx^{(i)} + M^{-1}b$$

Fehlerbetrachtung

- Der Fehler im i-ten Iterationsschritt ist

$$e^{(i)} = x^{(i)} - x.$$

Fehlerbetrachtung

- Der Fehler im i-ten Iterationsschritt ist
$$e^{(i)} = x^{(i)} - x.$$
- Damit verändert sich der Fehler durch
$$e^{(i+1)} = M^{-1}Ne^{(i)}.$$

Fehlerbetrachtung

- Der Fehler im i-ten Iterationsschritt ist
$$e^{(i)} = x^{(i)} - x.$$
- Damit verändert sich der Fehler durch
$$e^{(i+1)} = M^{-1}Ne^{(i)}.$$
- Ein Iterationsverfahren ist also sinnvoll, wenn gilt
$$\rho(M^{-1}N) < 1.$$

Abbruchbedingung

- Abweichung in Eingangswerten nicht feststellbar.

Abbruchbedingung

- Abweichung in Eingangswerten nicht feststellbar.
- Abweichung in Ausgangswerten ist das Residuum

$$r^{(i)} = b - Ax^{(i)}$$

Abbruchbedingung

- Abweichung in Eingangswerten nicht feststellbar.
- Abweichung in Ausgangswerten ist das Residuum

$$r^{(i)} = b - Ax^{(i)}$$

- Sei $\varepsilon > 0$ geforderte Genauigkeit. Abbruch, falls

$$||r^{(i)}|| \leq \varepsilon ||b||$$

Abbruchbedingung

- Abweichung in Eingangswerten nicht feststellbar.
- Abweichung in Ausgangswerten ist das Residuum

$$r^{(i)} = b - Ax^{(i)}$$

- Sei $\varepsilon > 0$ geforderte Genauigkeit. Abbruch, falls

$$||r^{(i)}|| \leq \varepsilon ||b||$$

- Matrix schlecht konditioniert? Die Hoffnung stirbt zuletzt!

Das Jacobiverfahren

- Gewählte Zerlegung: $A = D - B$
- Daraus folgende Iterationsvorschrift:

$$x^{(i+1)} = D^{-1}Bx^{(i)} + D^{-1}b$$

Das Jacobiverfahren

- Gewählte Zerlegung: $A = D - B$
- Daraus folgende Iterationsvorschrift:

$$x^{(i+1)} = D^{-1}Bx^{(i)} + D^{-1}b$$

- Komponentenweise Berechnung der Iterierten

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^k \right)$$

Das Jacobiverfahren

- Gewählte Zerlegung: $A = D - B$
- Daraus folgende Iterationsvorschrift:

$$x^{(i+1)} = D^{-1}Bx^{(i)} + D^{-1}b$$

- Komponentenweise Berechnung der Iterierten

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^k \right)$$

- Optimierte Matrix/Vektor-Schreibweise

$$x^{(i+1)} = x^{(i)} + D^{-1} r^{(i)}$$

Das Gauß-Seidel-Verfahren

- Gewählte Zerlegung: $A = (D + L) - (-U)$, also $M = D + L$ und $N = -U$
- Optimierte Iterationsvorschrift:

$$x^{(i+1)} = x^{(i)} + (D + L)^{-1} r^{(i)}$$

Das Gauß-Seidel-Verfahren

- Gewählte Zerlegung: $A = (D + L) - (-U)$, also $M = D + L$ und $N = -U$
- Optimierte Iterationsvorschrift:

$$x^{(i+1)} = x^{(i)} + (D + L)^{-1} r^{(i)}$$

- Vorwärtssubstitution anstelle der expliziten Inversenberechnung In Matlab mit \-Operator

Matlab-Implementierung des Jacobi-Verfahrens

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
    $iter = 0;$
    $res = b - A \cdot x;$
    $resnorm = norm(res);$
    $bnorm = norm(b);$
    $D = diag(A);$
    solange $(iter < imax)$ *und* $(resnorm/bnorm > tol)$ **tue**
        $x = x + res./D;$
        $res = b - A \cdot x;$
        $resnorm = norm(res);$
        $iter = iter + 1;$
    Ende
Ende

Matlab-Implementierung des Gauß-Seidel-Verfahrens

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
    $iter = 0;$
    $res = b - A \cdot x;$
    $resnorm = norm(res);$
    $bnorm = norm(b);$
    $DL = tril(A);$
    solange $(iter < imax)$ *und* $(resnorm/bnorm > tol)$ **tue**
        $x = x + DL \backslash res;$
        $res = b - A \cdot x;$
        $resnorm = norm(res);$
        $iter = iter + 1;$
    Ende
Ende

Vergleiche Geschwindigkeit und Genauigkeit

Beispielmatrix: Modifizierte (diagonaldominante) Variante der Matrix 924 der *UF Matrix Collection*, vgl. Dav (2008)

Laufzeitverhalten:

- Jacobi VS: argh!
- Jacobi: $\approx 0.8s$
- Gauß-Seidel: $\approx 0.5s$

Genauigkeit $||x - x^*||$:

- Jacobi: $6.2504 \cdot 10^{-14}$
- Gauß-Seidel: $1.3073 \cdot 10^{-13}$

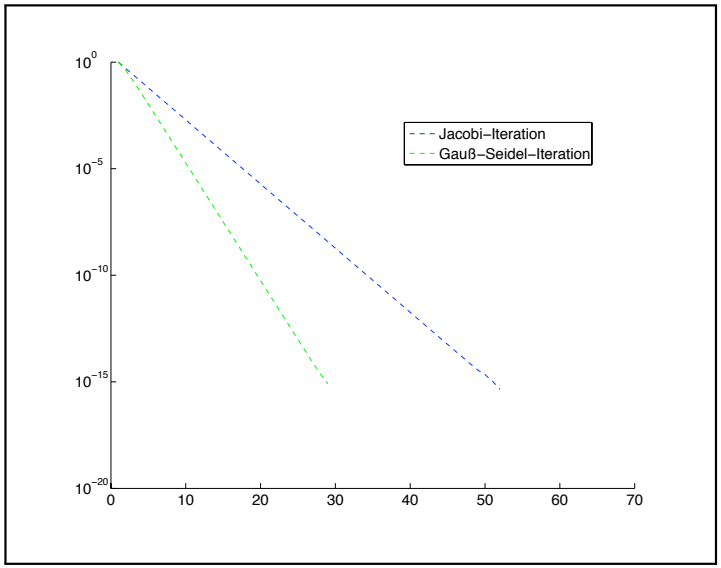


Abbildung: Jacobi- und Gauß-Seidel-Verfahren, Vergleich der Iterationsgeschwindigkeit

Überblick

Eine Quadratische Form

Aus der quadratischen Form

$$f : \mathbb{R}^n \to \mathbb{R} \qquad f(x) = \frac{1}{2} x^T A x - b^T x + c, \quad c \in \mathbb{R}$$

Eine Quadratische Form

Aus der quadratischen Form

$$f : \mathbb{R}^n \to \mathbb{R} \qquad f(x) = \frac{1}{2} x^T A x - b^T x + c, \quad c \in \mathbb{R}$$

wird durch Ableitung:

$$f'(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b$$

Eine Quadratische Form

Aus der quadratischen Form

$$f : \mathbb{R}^n \to \mathbb{R} \qquad f(x) = \frac{1}{2} x^T A x - b^T x + c, \quad c \in \mathbb{R}$$

wird durch Ableitung:

$$f'(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b$$

Also für symmetrische Matrizen:

$$f'(x) = Ax - b$$

Visualisierung für spd-Matrizen

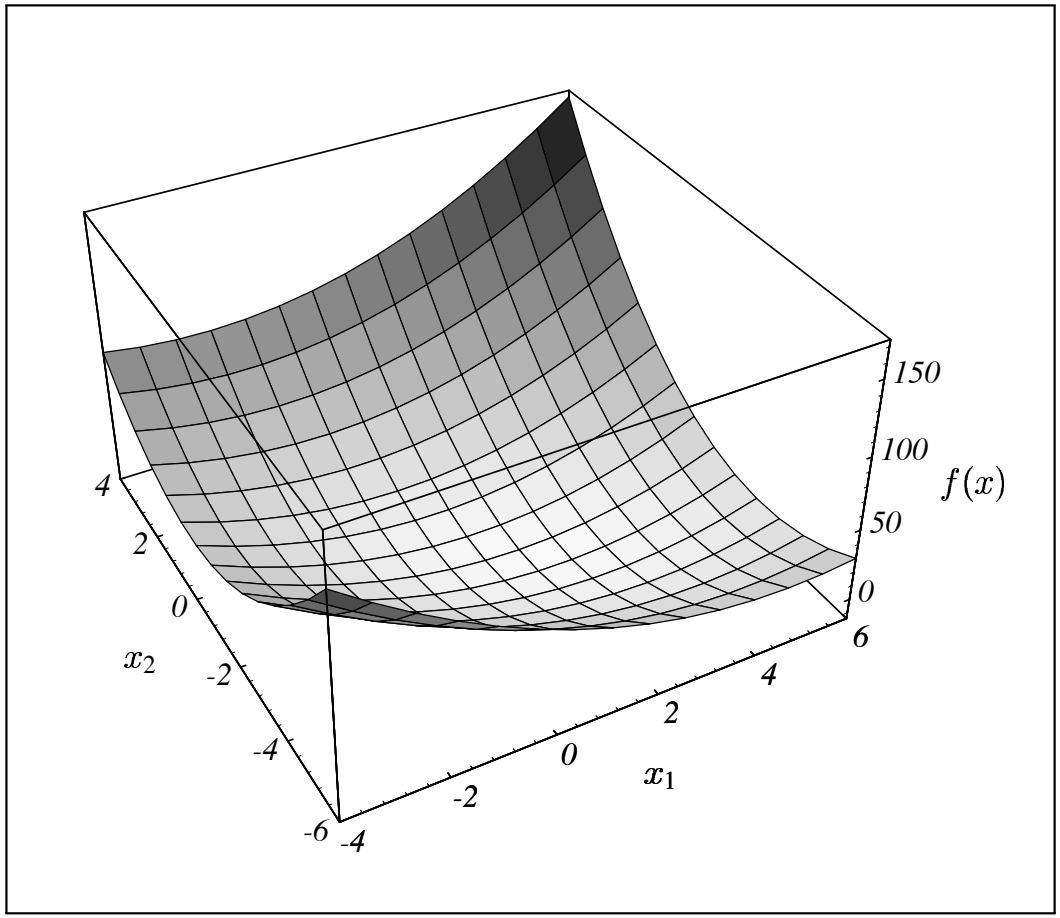


Abbildung: Graph einer quadratischen Form, (Shewchuk, 1994, S. 3)

Auf dem Weg nach unten . . .

- Residuum $r_{(i)} = b - Ax_{(i)}$ ist der negative Gradient.
- Das Minimum ist unten
  $\Rightarrow$ Residuum als Suchrichtung scheint passend.
- Lineare Suche mit Iterationsschritt

$$x_{(i+1)} = x_{(i)} + \alpha r_{(i)}$$

- Optimales α in Suchrichtung:

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

Matlab-Implementierung des Steepest-Decent-Verfahrens

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
  $iter = 0;$
  $res = b - A \cdot x;$
  $resnorm = norm(res);$
  $bnorm = norm(b);$
  solange $(iter < imax)$ *und* $(resnorm/bnorm > tol)$ **tue**
    $alpha = (res' \cdot res)/(res' \cdot A \cdot res);$
    $x = x + alpha \cdot res;$
    $res = b - A \cdot x;$
    $resnorm = norm(res);$
    $iter = iter + 1;$
  Ende
Ende

Ein paar Iterationsschritte am Beispiel

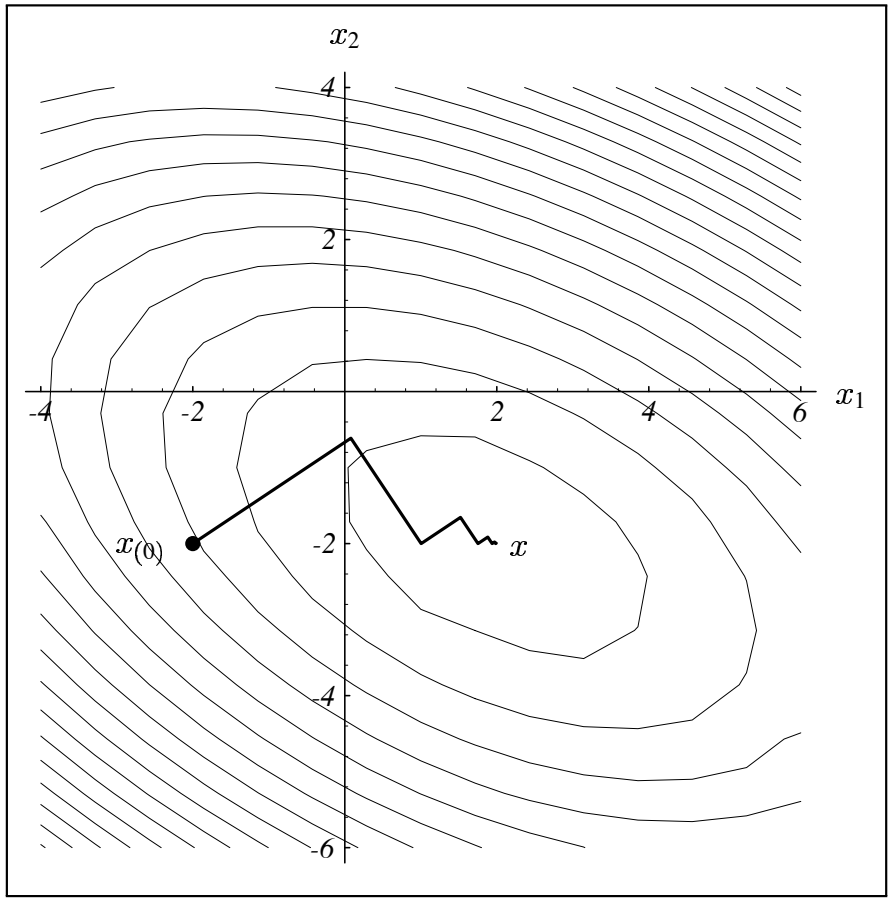


Abbildung: Verlaufbeispiel *Steepest Decent*, (Shewchuk, 1994, S. 8)

Idee

- Nutzung von orthogonalen Suchrichtungen $d_{(i)}$
- Iteration mit

$$x_{(i+1)} = x_{(i)} + \alpha r_{(i)}$$

- Einteilung der Fehler in orthogonale Komponenten
- Iteration jeweils auf diesen orthogonalen Komponenten
  $\Rightarrow$ es wird jeweils eine Fehlerkomponente komplett eliminiert

Idee

- Nutzung von orthogonalen Suchrichtungen $d_{(i)}$
- Iteration mit

$$x_{(i+1)} = x_{(i)} + \alpha r_{(i)}$$

- Einteilung der Fehler in orthogonale Komponenten
- Iteration jeweils auf diesen orthogonalen Komponenten
  $\Rightarrow$ es wird jeweils eine Fehlerkomponente komplett eliminiert
- Aber: Optimales α nicht berechenbar:

$$\alpha = -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}}$$

Geht's nicht so brauch ich Gewalt...

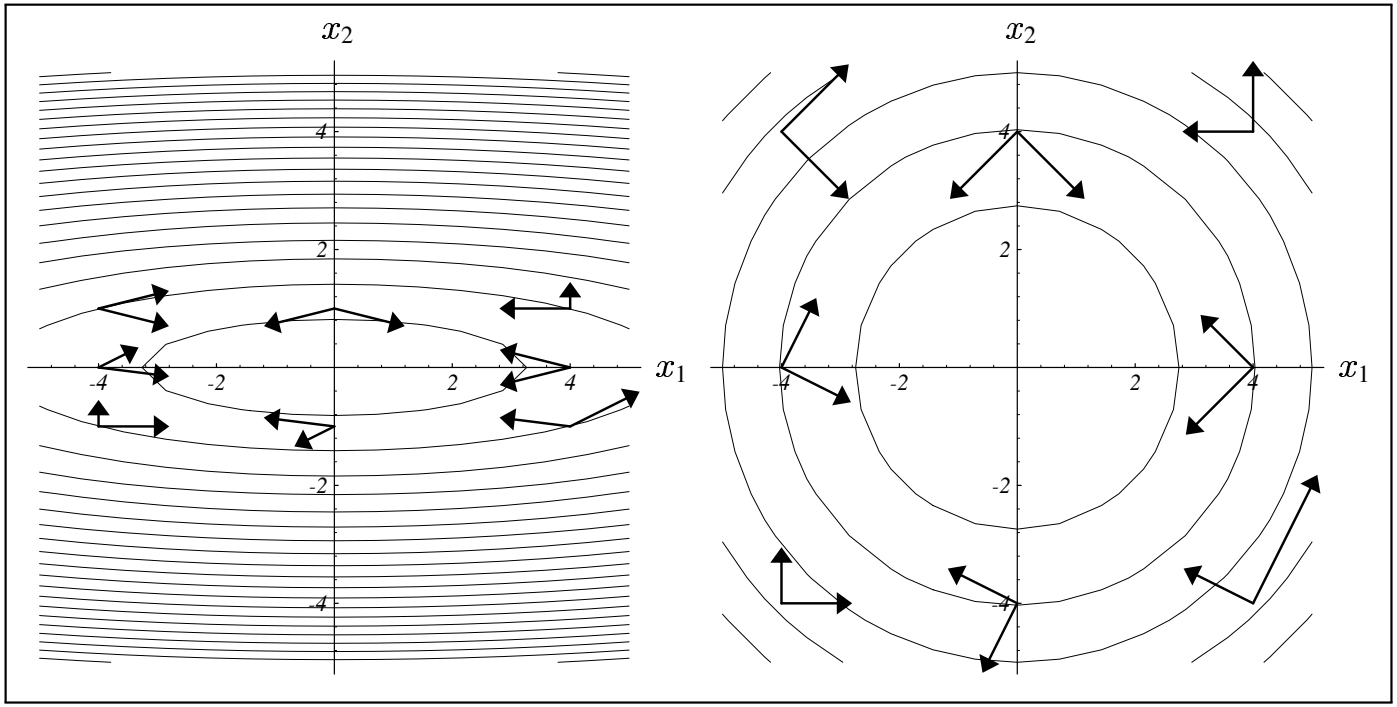


Abbildung: *A*-orthogonale Vektoren (links) werden durch *Ziehen und Zerren* zu orthogonalen Vektoren (rechts) modifiziert, (Shewchuk, 1994, S. 23)

orthogonal wird A-orthogonal

- Optimales α nun berechenbar:

$$\alpha = -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} = -\frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}$$

- Aber: Woher nimmt man die A-orthogonalen Suchvektoren?

Gram-Schmidt-Verfahren

- Man nehme n linear unabhängige Vektoren u_i
- Es ist $d_{(0)} = u_0$
- Von alle weiteren Vektoren u_i werden jeweils die Anteile entfernt, die zu den bereits gefundenen Vektoren $d_{(j)}$, $j < i$, nicht A-orthogonal sind:

$$d_{(i)} = u_i - \sum_{j=0}^{i-1} \beta_{i,j} d_{(j)} \quad \text{mit} \quad \beta_{ij} = -\frac{u_i^T A d_{(j)}}{d_{(j)}^T A d_{(j)}}$$

Gram-Schmidt-Verfahren

- Man nehme n linear unabhängige Vektoren u_i
- Es ist $d_{(0)} = u_0$
- Von alle weiteren Vektoren u_i werden jeweils die Anteile entfernt, die zu den bereits gefundenen Vektoren $d_{(j)}$, $j < i$, nicht A-orthogonal sind:

$$d_{(i)} = u_i - \sum_{j=0}^{i-1} \beta_{i,j} d_{(j)} \quad \text{mit} \quad \beta_{ij} = -\frac{u_i^T A d_{(j)}}{d_{(j)}^T A d_{(j)}}$$

- Aber: Aufwand vergleichbar mit Gauss-Elimination!

It's magic...

MMM[1] hat erwirkt...

- Neues Residuum $r_{(i+1)}$ bereits orthogonal zu den vorherigen
  $\Rightarrow$ Residuen als u_i im Gram-Schmidt-Verfahren geeignet
- $r_{(i+1)} \perp_A d_{(j)}$ für $j < i+1$

Da das meiste 0 ist, ergibt sich mit einer Prise Zaubersalz für das Gram-Schmidt-Verfahren:

$$d_{(i)} = r_{(i)} + \beta_{(i)} d_{(i-1)} \quad \text{mit} \quad \beta_{(i)} = \frac{r_{(i)}^T r_{(i)}}{r_{(i-1)}^T r_{(i-1)}}$$

[1] My Mathematical Magician

Matlab-Implementierung des CG-Verfahrens

Beginn
$iter = 0;$
$res = b - A \cdot x;$
$resnorm = norm(res);$
$bnorm = norm(b);$
$d = res;$
solange $(iter < imax)$ und $(resnorm/bnorm > tol)$ **tue**
$alpha = (res' \cdot res)/(d' \cdot A \cdot d);$
$x = x + alpha \cdot d;$
`%Vorbereitung für nächste Iteration: Berechnung von neuem` d
$resneu = b - A \cdot x;$
$beta = (resneu' \cdot resneu)/(res' \cdot res);$
$d = resneu + beta \cdot d;$
`%Abbruchbedingung berücksichtigen`
$res = resneu;$
$resnorm = norm(res);$
$iter = iter + 1;$
Ende
Ende

Vergleiche von Laufzeit und Genauigkeit

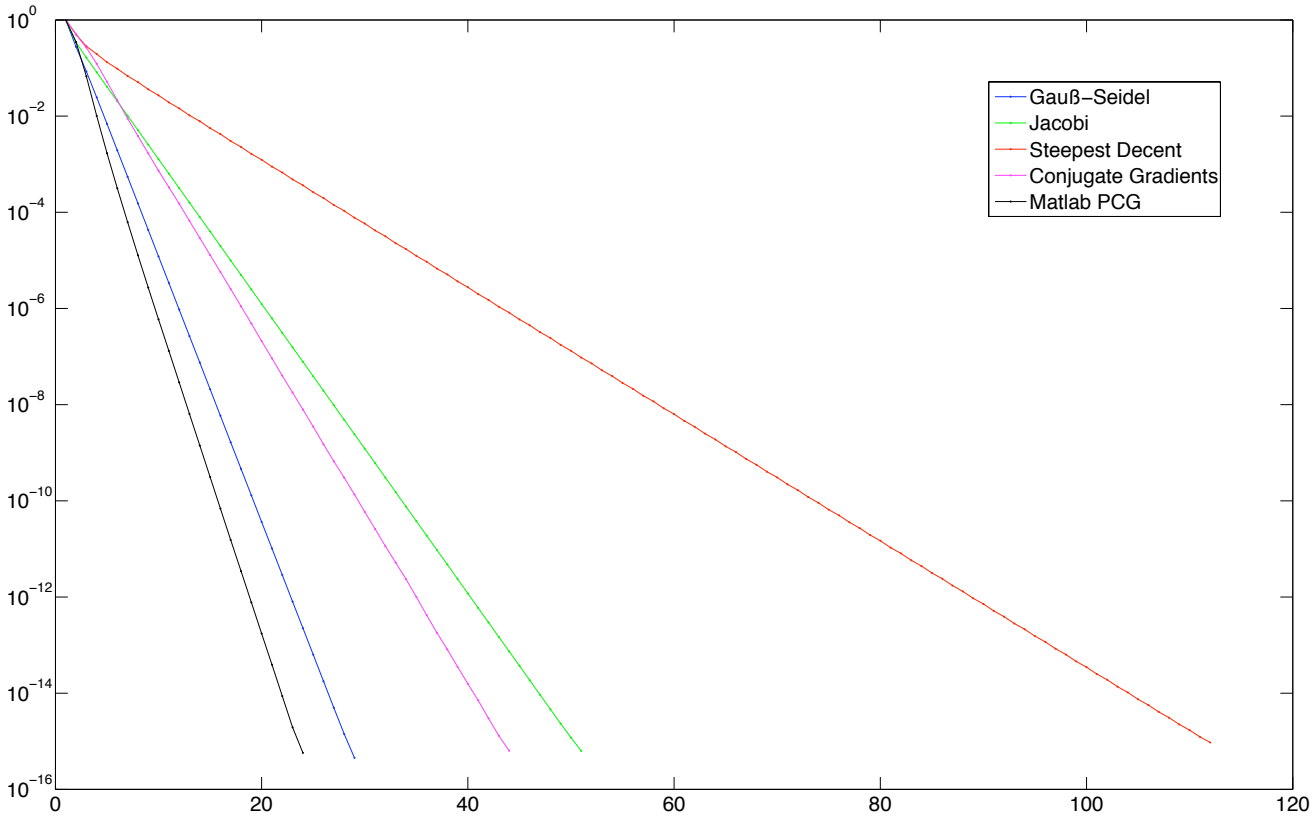
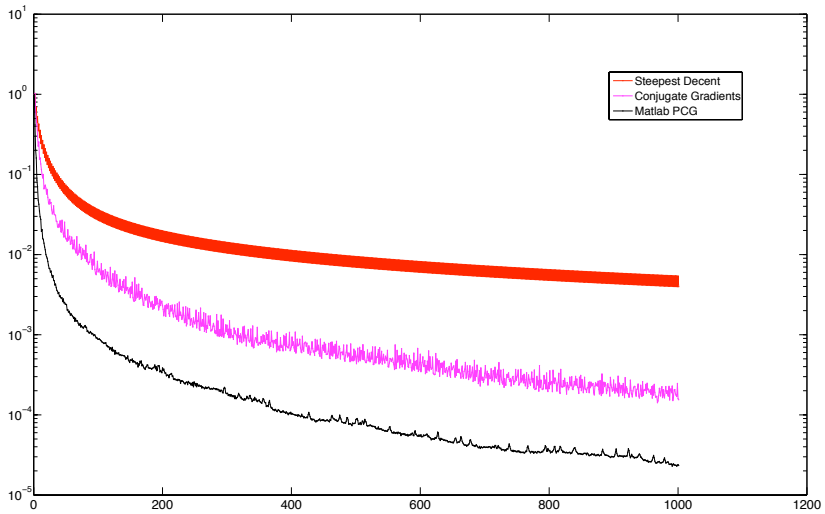


Abbildung: Modifizierte (diagonaldominante) Variante der Matrix 924 der *UF Matrix Collection*, vgl. Dav (2008), Relative Fehler im Residuum

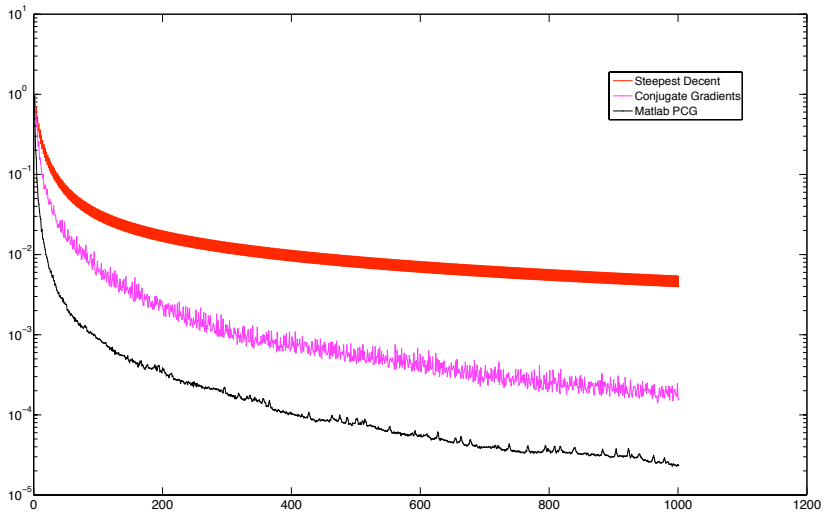
Vergleiche von Laufzeit und Genauigkeit



Matrix 362 der *UF Matrix Collection*, vgl. Dav (2008), Relative Fehler im Residuum

Steepest Decent: 15s
Conjugate Gradients: 15s
Matlab PCG: 30s

Vergleiche von Laufzeit und Genauigkeit



Matrix 362 der *UF Matrix Collection*, vgl. Dav (2008), Relative Fehler im Residuum

Steepest Decent: 15s
Conjugate Gradients: 15s
Matlab PCG: 30s

Aber: Direkte Lösung mit $A\backslash b$ in 0.7s!

Tolles Verfahren nicht so toll?

Note that no incomplete factorizations, sparse approximate inverses, algebraic multigrid, or other powerful preconditioners were used. This comparison is using PCG as black box solver, which is admittedly unfair to PCG. I used a random right-hand side, which is also unfair to PCG. The primary point of this comparison is to show that PCG doesn't make a good black box solver. You really need to use problem-specific knowledge for iterative methods to work well. *Davis (2007)*

Literatur I

[Dav 2008] Davis, Tim (Hrsg.): *The University of Florida Sparse Matrix Collection*. `http://www.cise.ufl.edu/research/sparse/matrices/`. Version: 2008, Abruf: 12. Januar 2009

[Davis 2007] Davis, Tim: *PCG vs backslash in MATLAB*. online. `http://www.cise.ufl.edu/research/sparse/pcg_compare/`. Version: 2007, Abruf: 18. Januar 2009

[Shewchuk 1994] Shewchuk, Jonathan R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. online. `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps`. Version: August 1994, Abruf: 04. Dezember 2009