Iteratives Lösen linearer Gleichungssysteme - Übung 5

Tobias Danczul

18. Mai 2017

1. Beispiel

Ges.: Wie vereinfacht sich der GMRES Algorithmus, wenn die Input-Matrix A symmetrisch ist.
Der GMRES-Algorithmus besteht aus 2 verschiedenen Abschnitten.

- Im ersten Abschnitt wird die zu A gehörige Hessenbermatrix $\bar{H}_m$ aus der Arnoldi-Iteration berechnet. Diese ist für symmetrische A tridiagonal, da

  $$H_m = V_m^T A V_m = H_m^T$$

  und H_m Hessenberg-Gestalt besitzt. Daraus lässt sich eine 3-Term Rekursion, die Lanczos-Iteration, als effiziente Implementierung des Arnoldi-Algorithmus ableiten.

- Im zweiten Abschnitt muss ein lineares Ausgleichsproblem der Gestalt

  $$\|\beta e_1 - \bar{H}_m u_m\|_2 = \min!$$

  für u_m gelöst werden, was effizient über die QR-Zerlegung von $\bar{H}_m$ implementiert wird. Dafür betrachten wir die spezielle Form von R aus der QR-Zerlegung tridiagonaler Matrizen.
  Beh: Ist $A \in \mathbb{R}^{m \times m+1}$ tridiagonal, so hat die zugehörige Matrix R aus der QR-Zerlegung folgende Besetzungsstruktur:

  $$R = \begin{pmatrix} * & * & * & & & & \\ & * & * & * & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & * & * & * & \\ & & & & * & * & * \\ & & & & & * & * \\ & & & & & & * \end{pmatrix} \in \mathbb{R}^{m \times m}$$

  Sei dafür $m > 3$ vorausgesetzt, dann gilt:

  $$w_k := a_k - \sum_{i=1}^{k-1} \langle a_k, q_i \rangle q_i = a_k - \sum_{i=k-2}^{k-1} \langle a_k, q_i \rangle q_i \tag{1}$$

  da $a_i \perp a_j$ für $|i-j| > 2$ und $q_i \in [a_1 ... a_i]$. Damit folgt:

  $$q_k = \frac{w_k}{\|w_k\|_2}$$

  Da R sich aus den Gram-Schmidt Faktoren aus (1) zusammensetzt, folgt die Behauptung.

Ferner soll noch erwähnt werden, dass für eine effiziente Implementierung die Faktoren aus der QR-Zerlegung über die Lanczos-Iteration gewonnen werden können.

Übung 5 - Beispiel 2 (Daniel Herold)

Z.z.: Die Konstruktion aus Abschnitt 11.1 führt zu einer biorthonormalen Folge an v_i, w_i.
Beweis: Wir beweisen die Aussage mit vollständiger Induktion.

Induktionsanfang $(v_1, w_1) = 1$ nach Voraussetzung. Der Algorithmus liefert

$$\begin{aligned}
\hat{v}_2 &= Av_1 - (Av_1, w_1)v_1 \\
\hat{w}_2 &= A^T w_1 - (Av_1, w_1)w_1 \\
\delta_2 &= \sqrt{|(\hat{v}_2, \hat{w}_2)|} \\
\beta_2 &= \frac{(\hat{v}_2, \hat{w}_2)}{\delta_2} \\
w_2 &= \frac{\hat{w}_2}{\beta_2} \\
v_2 &= \frac{\hat{v}_2}{\delta_2}
\end{aligned}$$

Damit gilt

$$\begin{aligned}
(v_2, w_2) &= \frac{1}{\delta_2}\frac{1}{\beta_2}(\hat{v}_2, \hat{w}_2) = \frac{1}{(\hat{v}_2, \hat{w}_2)}(\hat{v}_2, \hat{w}_2) = 1 \\
(v_2, w_1) &= \frac{1}{\delta_2}(\hat{v}_2, w_1) = \frac{1}{\delta_2}(Av_1, w_1) - \frac{1}{\delta_2}(Av_1, w_1)\underbrace{(v_1, w_1)}_{=1} = 0 \\
(w_2, v_1) &= \frac{1}{\beta_2}(\hat{w}_2, v_1) = \frac{1}{\beta_2}\underbrace{(A^T w_1, v_1)}_{(w_1, Av_1)} - \frac{1}{\beta_2}(Av_1, w_1)\underbrace{(w_1, v_1)}_{=1} = 0
\end{aligned}$$

Induktionsvoraussetzung $\{v_1, ..., v_n\}$ und $\{w_1, ..., w_n\}$ bilden biorthonormales System.
Induktionsschritt $n \to n+1$

$(v_{n+1}, w_{n+1}) =$

$$\frac{1}{\delta_{n+1}}\frac{1}{\beta_{n+1}}(\hat{v}_{n+1}, \hat{w}_{n+1}) = \frac{1}{(\hat{v}_{n+1}, \hat{w}_{n+1})}(\hat{v}_{n+1}, \hat{w}_{n+1}) = 1$$

$(v_{n+1}, w_j) =$

$$= \frac{1}{\delta_{n+1}}\left[(Av_n, w_j) - \alpha_n(v_n, w_j) - \beta_n(v_{n-1}, w_j)\right]$$

$$= \frac{1}{\delta_{n+1}} \cdot \begin{cases} (Av_n, w_n) - \underbrace{\alpha_n}_{(Av_n, w_n)}\underbrace{(v_n, w_n)}_{=1} - \beta_n\underbrace{(v_{n-1}, w_n)}_{=0} & j = n \\ (v_n, \underbrace{A^T w_j}_{\hat{w}_{j+1} + \alpha_j w_j + \beta_j w_{j-1}}) - \alpha_n(v_n, w_j) - \beta_n(v_{n-1}, w_j) & j < n \end{cases}$$

$$= \frac{1}{\delta_{n+1}} \cdot \begin{cases} 0 & j = n \\ \underbrace{(v_n, \hat{w}_n)}_{\beta_n\underbrace{(v_n, w_n)}_{=1}} + \alpha_{n-1}\underbrace{(v_n, w_{n-1})}_{=0} + \beta_{n-1}\underbrace{(v_n, w_{n-2})}_{=0} - \alpha_n\underbrace{(v_n, w_{n-1})}_{=0} - \beta_n\underbrace{(v_{n-1}, w_{n-1})}_{=1} & j = n-1 \\ \underbrace{(v_n, \hat{w}_{j+1})}_{=0} + \alpha_j\underbrace{(v_n, w_j)}_{=0} + \beta_j\underbrace{(v_n, w_{j-1})}_{=0} - \alpha_n\underbrace{(v_n, w_j)}_{=0} - \beta_n\underbrace{(v_{n-1}, w_j)}_{=0} & j \leq n-2 \end{cases}$$

$= 0$

$$(w_{n+1}, v_j) =$$

$$\frac{1}{\beta_{n+1}} \left[\underbrace{(A^T w_n, v_j)}_{(w_n, A v_j)} - \alpha_n (w_n, v_j) - \beta_n (w_{n-1}, v_j) \right]$$

$$= \frac{1}{\beta_{n+1}} \cdot \begin{cases} (w_n, Av_n) - \underbrace{\alpha_n}_{(Av_n, w_n)} \underbrace{(w_n, v_n)}_{=1} - \beta_n \underbrace{(w_{n-1}, v_n)}_{=0} & j = n \\ (w_n, \underbrace{Av_j}_{\hat{v}_{j+1} + \alpha_j v_j + \beta_j v_{j-1}}) - \alpha_n (w_n, v_j) - \beta_n (w_{n-1}, v_j) & j < n \end{cases}$$

$$= \frac{1}{\delta_{n+1}} \cdot \begin{cases} 0 & j = n \\ \underbrace{(v_n, \hat{w}_n)}_{\beta_n \underbrace{(v_n, w_n)}_{=1}} + \alpha_{n-1} \underbrace{(v_n, w_{n-1})}_{=0} + \beta_{n-1} \underbrace{(v_n, w_{n-2})}_{=0} - \alpha_n \underbrace{(v_n, w_{n-1})}_{=0} - \beta_n \underbrace{(v_{n-1}, w_{n-1})}_{=1} & j = n-1 \\ \underbrace{(v_n, \hat{w}_{j+1})}_{=0} + \alpha_j \underbrace{(v_n, w_j)}_{=0} + \beta_j \underbrace{(v_n, w_{j-1})}_{=0} - \alpha_n \underbrace{(v_n, w_j)}_{=0} - \beta_n \underbrace{(v_{n-1}, w_j)}_{=0} & j \leq n-2 \end{cases}$$

Damit ist die Biorthonormalität gezeigt.

Iteratives Lösen linearer Gleichungssysteme - Übung 5

Michael Neunteufel

26. Mai 2017

3. Beispiel

Z.z.: Der Algorithmus 12.2 ist korrekt.

Dazu vergleichen wir das CG-Verfahren mit dem M-Skalarprodukt und den entsprechnenden Algorithmus:

CG:

1: $r_0 = b - Ax_0$, $\hat{r}_0 = M^{-1}\hat{r}_0$, $\hat{d}_0 = \hat{r}_0$
2: **for** $k = 0, 1, \dots$ until convergence **do**
3: $\quad \alpha_k = \frac{\langle \hat{r}_k, \hat{r}_k \rangle_M}{\langle M^{-1} A\hat{d}_k, \hat{d}_k \rangle_M}$
4: $\quad x_{k+1} = x_k + \alpha_k \hat{d}_k$
5: $\quad \hat{r}_{k+1} = \hat{r}_k - \alpha_k M^{-1} A\hat{d}_k$
6: $\quad \beta_k = \frac{\langle \hat{r}_{k+1}, \hat{r}_{k+1} \rangle_M}{\langle \hat{r}_k, \hat{r}_k \rangle_M}$
7: $\quad \hat{d}_{k+1} = \hat{r}_{k+1} + \beta_k \hat{d}_k$
8: **end for**

Algorithmus 12.2:

Algorithm 1 Left-preconditioned Conjugate Gradient method

1: Compute $r_0 = b - Ax_0$, $\hat{r}_0 = M^{-1}\hat{r}_0$, $\hat{d}_0 = \hat{r}_0$
2: **for** $k = 0, 1, \dots$ until convergence **do**
3: $\quad \alpha_k = \frac{\langle r_k, \hat{r}_k \rangle}{\langle A\hat{d}_k, \hat{d}_k \rangle}$
4: $\quad x_{k+1} = x_k + \alpha_k \hat{d}_k$
5: $\quad r_{k+1} = r_k - \alpha_k A\hat{d}_k$
6: $\quad \hat{r}_{k+1} = M^{-1} r_{k+1}$
7: $\quad \beta_k = \frac{\langle r_{k+1}, \hat{r}_{k+1} \rangle}{\langle r_k, \hat{r}_k \rangle}$
8: $\quad \hat{d}_{k+1} = \hat{r}_{k+1} + \beta_k \hat{d}_k$
9: **end for**

Durch Einführen des Residuums r_k mit der Identität $\hat{r}_k = M^{-1} r_k$ wurden die M-Skalarprodukte aufgelöst. Nachrechnen ergibt

$$\frac{\langle \hat{r}_k, \hat{r}_k \rangle_M}{\langle M^{-1} A\hat{d}_k, \hat{d}_k \rangle_M} = \frac{\langle M\hat{r}_k, \hat{r}_k \rangle}{\langle M^{-1} MA\hat{d}_k, \hat{d}_k \rangle} = \frac{\langle r_k, \hat{r}_k \rangle}{\langle A\hat{d}_k, \hat{d}_k \rangle}$$

$$r_{k+1} = \hat{r}_{k+1} = M\hat{r}_k - \alpha_k M M^{-1} A\hat{d}_k = r_k - \alpha_k A\hat{d}_k$$

$$\frac{\langle \hat{r}_{k+1}, \hat{r}_{k+1} \rangle_M}{\langle \hat{r}_k, \hat{r}_k \rangle_M} = \frac{\langle M\hat{r}_{k+1}, \hat{r}_{k+1} \rangle}{\langle M\hat{r}_k, \hat{r}_k \rangle} = \frac{\langle r_{k+1}, \hat{r}_{k+1} \rangle}{\langle r_k, \hat{r}_k \rangle}$$

und damit die Korrektheit des Algorithmus.

4. Beispiel

Geg.: Lineares Gleichungssystem $Ax = b$. ein zugehöriger SPD Preconditioner M mit Choleskyzerlegung $M = LL^T$.
Z.z.: Die Iterierten $x_m = L^{-T}u_m$ des auf die vorkonditionierte Gleichung $L^{-1}AL^{-T}u = L^{-1}b$ angewandten CG-Verfahrens stimmen mit jenen des linkskonditionierten CG-Verfahrens $MAx = b$ überein.
Wir definieren die folgenden Residuen

$$\bar{r} := L^{-1}(b - AL^{-T}u) = L^{-1}(b - Ax) = L^{-1}r$$
$$\tilde{r} := M^{-1}r = L^{-T}\bar{r}$$

und kennzeichnen die auftretenden Koeffizienten α und β, sowie die Suchrichtung d des beidseitig vorkonditionierten Verfahrens mit $\bar{\cdot}$.
Da die Matrix $L^{-1}AL^{-T}$ wieder SPD ist, kann das klassische CG-Verfahren darauf angewendet werden (mit der neuen rechten Seite $L^{-1}b$).
Wir beweisen nun folgende Aussagen per Induktion: $\bar{\alpha}_k = \alpha_k$, $\bar{\beta}_k = \beta_k$, $\tilde{d}_k = L^{-T}\bar{d}_k$, $\tilde{r}_k = L^{-T}\bar{r}_k$ und $x_k = L^{-T}u_k$.
Zur besseren Übersicht wird vorab gezeigt, unter der Voraussetzung der Gültigkeit der obigen Identitäten für d_k und r_k, dass $\bar{\alpha}_k = \alpha_k$ und $\bar{\beta}_k = \beta_k$:

$$\bar{\alpha}_k = \frac{\langle \bar{r}_k, \bar{r}_k \rangle}{\langle L^{-1}AL^{-T}\bar{d}_k, \bar{d}_k \rangle} = \frac{\langle LL^T\tilde{r}_k, \tilde{r}_k \rangle}{\langle A\tilde{d}_k, \tilde{d}_k \rangle} = \alpha_k$$
$$\bar{\beta}_k = \frac{\langle \bar{r}_{k+1}, \bar{r}_{k+1} \rangle}{\langle \bar{r}_k, \bar{r}_k \rangle} = \beta_k$$

IA: Nach Voraussetzung gilt für die Anfangswerte $x_0 = L^{-T}u_0$, $\tilde{r}_0 = L^{-T}\bar{r}_0$, sowie $\tilde{d}_0 = L^{-T}\tilde{d}_0$.
IV: Es gelte $\tilde{d}_k = L^{-T}\bar{d}_k$, $\tilde{r}_k = L^{-T}\bar{r}_k$ und $x_k = L^{-T}u_k$.
IS: Wir verwenden dieselbe Reihenfolge wie im Algorithmus 12.2:

$$L^{-T}u_{k+1} = L^{-T}u_k + \bar{\alpha}_k L^{-T}\bar{d}_k \overset{IV}{=} x_k + \alpha_k\tilde{d}_k = x_{k+1}$$
$$L^{-T}\bar{r}_{k+1} = L^{-T}\bar{r}_k - \bar{\alpha}_k \underbrace{L^{-T}L^{-1}}_{=M^{-1}} AL^{-T}\bar{d}_k \overset{IV}{=} \tilde{r}_k - \alpha_k M^{-1}A\tilde{d}_k = M^{-1}r_{k+1} = \tilde{r}_{k+1}$$
$$L^{-T}\bar{d}_{k+1} = L^{-T}\bar{r}_{k+1} + \bar{\beta}_k L^{-T}\bar{d}_k \overset{IV}{=} \tilde{r}_{k+1} + \beta_k\tilde{d}_k = \tilde{d}_{k+1}$$

Damit wurde die Gleichheit gezeigt.

Iterative Lösung großer Gleichungssysteme,
Übung 5, Aufgabe 5
Lukas Kogler

Die Aufgabe war es, mehrere Schritte von SGS (symmetric Gauß-Seidel) als Vorkonditionierer für PCG für die $2d$-Poisson-Matrix zu benutzen.
Wir hatten ja schon in der Vorlesung gesehen, dass eine Stationäre Iteration für das lineare GS $Ax = b$ geschrieben werden kann als:

$$x_{k+1} = Mx_k + Nb = (I - NA)x_k + Nb = x_k + N(b - A_x)$$

Das heißt, die stationäre Iteration kann als Richardson-Iteration mit Vorkonditionierer N interpretiert werden. Anwenden eines Vorkonditionierers $y \to C^{-1}y$ kann also auch als ein Schritt solch einer Richardson-Iteration mit Startvektor $x_0 = 0$ gesehen werden, wodurch auch klar wird, was mit "k Schritte SGS" als Vorkonditionierer gemeint ist, nämlich k Schritte der Richardson-Iteration mit SGS (bzw. äquivalent k Schritte des stationären SGS Verfahrens) angewandt auf die Gleichung $Ax = y$.

Implementation von SGS:
Version 1:
Einen einzelnen GS-Schritt kann man in folgender Art und Weise umschreiben:

$$x_{k+1} = x_k + (L + D)^{-1}(b - Ax_k) \quad (1)$$
$$(L + D)(x_{k+1} - x_k) = b - Ax_k \quad (2)$$
$$D(x_{k+1} - x_k) = b - (L^T + D)x_k - Lx_{k+1} \quad (3)$$
$$x_{k+1} = x_k + D^{-1}\left(b - (L^T + D)x_k - Lx_{k+1}\right) \quad (4)$$

An (4) kann man nun sehen, dass man, um die j-te Komponente von x_{k+1} auszurechnen nur die ersten $j-1$ Komponenten von x_{k+1} sowie die letzten $n-j+1$ Komponenten von x_k braucht. Für einen Rückwärts-GS-Schritt braucht man nur L und L^T zu vertauschen und man kann die j-te Komponente mit den letzten $j-1$ Komponenten des "neuen" und den ersten $n-j+1$ Komponenten des "alten" Vectors berechnen - man kann den neuen Vektor also immer noch komponentenweise berechnen, muss aber von hinten anfangen. Damit kann man k SGS-Schritte mit Start-Vector $x_0 = 0$ wie in Algorithmus (1) schreiben.
Man beachte dabei zwei Feinheiten:

- Die j-Schleife läuft von 1 bis n und dann wieder von n zurück bis 1.
- Es wird das Produkt einer **Spalte** von A mit x berechnet, eigentlich sollte hier eine **Zeile** benutzt werden, da in diesem fall A aber symmetrisch ist, ist dies äquivalent. Der Vorteil hier besteht darin, dass Matrizen in MATLAB spaltenweise gespeichert werden und daher der Zugriff auf eine Spalte schneller als der auf eine Zeile ist!

Algorithm 1 SGS

```
1: procedure SGS(A, k, b)
2:     x := 0
3:     for l = 1 ... k do
4:         for j = [1 ... n, n ... 1] do
5:             x_k = x_k + D_k^{-1} (b_k − A_{:,k} x)
6:         end for
7:     end for
8: end procedure
```

Version 2:
Eine weitere Möglichkeit SGS zu schreiben ist, wie wir ebenfalls in der VO gesehen hatten, ist:

$$x_{k+1} = x_k + (L^T + D)^{-1} D (L + D)^{-1} (b - Ax) \tag{5}$$

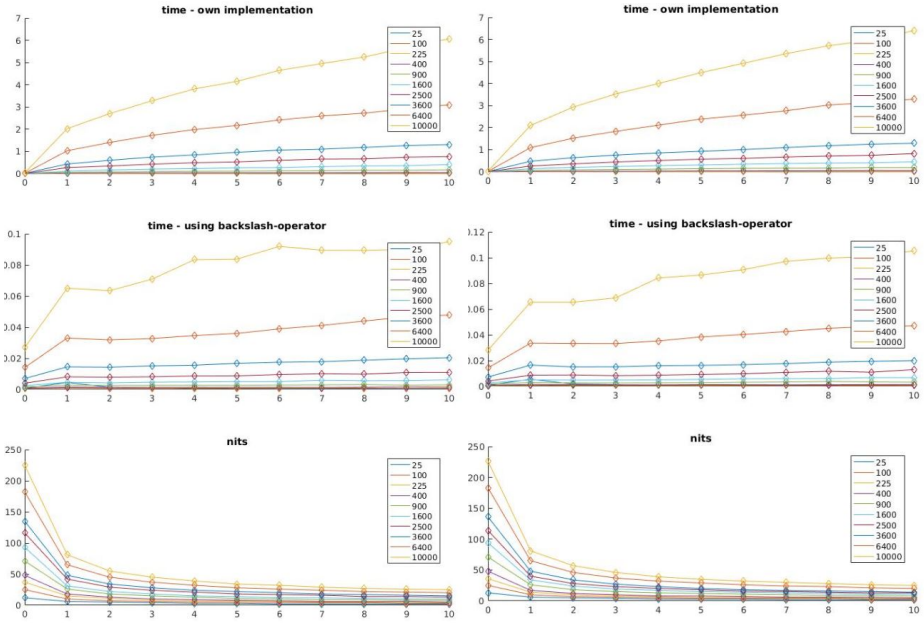
Wenn man dies anstatt (4) als Grundlage des Algorithmus nimmt, kann man k-mal SGS auch wie in Algorithmus (2) schreiben, wobei man zum Lösen der Gleichungssysteme in MATLAB den "\"-Operator benutzen kann.

Algorithm 2 SGSV2

```
1: procedure SGS(A, k, b)
2:     x := 0
3:     for l = 1 ... k do
4:         Löse (L + D)y = b − Ax
5:         Löse (L^T + D)w = Dy
6:         x = x + w
7:     end for
8: end procedure
```

Ergebnisse:
Aus den Ergebnissen (Grafik (1)) können wir 3 Aussagen ableiten:

- Für diesen Fall scheint keine Anzahl an SGS-Schritten einen Zeitgewinn zu erzielen, allerdings sinkt die Anzahl an benötigten Interationen
- Starten eines "parallel pools" bringt kaum Vorteile mit sich. Das war eigentlich zu erwarten, da, wie man in Algorithmu(1) sehr gut sehen kann, eine Komponente **nach der anderen** geupdated wird, das heißt SGS ist ein inherent **sequentieller** Algorithmus! Dies könnte mit einem coloring auf Basis der Besetzungsstruktur von A verbessert werden.
- Algorithmus (2) ist viel schneller als (1). MATLAB-interne Funktionen sind einfach viel besser optimiert als eine naive Implementation wie in Algorithmus (1).



(a) without parallel pool (b) with parallel pool

Fig. 1: Reihe 1: runtime vs nr of alg-1-steps, Reihe2: runtime vs nr of alg-2-steps, Reihe 3: nr its vs nr of SGS steps.

```
% ----------------b)-----------------
c = @(x) x + 1;
mean = integral(c,0,1);

%Mesh
n = 10;
h = 1/n;
mesh = 0:h:1;

%Diskretisierung
N = 2*eye(n-1) + diag(-ones(n-2,1),1) + diag(-ones(n-2,1),-1);
N = mean/h^2*N;

%Rechte Seite
func = @(x) x.^0;
f = transpose(func(mesh([2:n])));

%LÃ¶se finite Differenzen
u = N\f;


%Exakte Matrix
D = diag(c(mesh([2:n])- h/2*ones(1,n-1)) + c(mesh([2:n]) + h/2*ones(1,↙
n-1)));
LD = diag(-c(mesh([3:n]) - h/2*ones(1,n-2)),-1);
UD = diag(-c(mesh([3:n])+ h/2*ones(1,n-2)),1);

A = 1/h^2*(D + LD + UD);


%StationÃ¤re Iteration: x(k+1) = Mx(k) + Nb
N = inv(N);
M = eye(n-1) - N*A;

%Maximale Iterationsanzahl
iter = 100;

%Residuum
Res = norm(f);
res1 = zeros(iter,1);

%Toleranz
tol = 1e-5*Res;
```

```
%Startwert
x1 = zeros(n-1,1);
tic
for k = 1:iter
    x2 = M*x1 + N*f;
    if norm(f - A*x1) < tol
        disp('Anzahl der Iterationen stationÃ¤re Iteration:')
        disp(k)
        break;
    end
    res1(k) = norm(f - A*x2);
    %Update
    x1 = x2;
end
disp('Zeit: stationÃ¤re Iteration:')
toc

figure(1)
semilogy(res1)
hold on



%--------------------c)--------------------
x0 = zeros(n-1,1);
r0 = f - A*x0;
Minv = N;
R0 = Minv*r0;
D0 = R0;

%Residuum
res2 = zeros(iter,1);

tic
[~,~,~,iter,resvec] = pcg(A,f,tol,1000,inv(N));
disp('Iterations PCG:')
disp(iter)
disp('Zeit: Preconditioner')
toc

%
% for i = 1:iter
%    rSkp = transpose(r0)*R0;
%     dSkp = transpose(A*D0)*D0;
```

```
%     alpha = rSkp/dSkp;
%     x1 = x0 + alpha*D0;
%     r1 = r0 - alpha*A*D0;
%     R1 = Minv*r1;
%
%     Residuum
%     res2(i) = norm(f - A*x1);
%
%     if norm(f - A*x1) < tol
%         disp('Anzahl der Iterationen Preconditioner:')
%         disp(k);
%         break;
%     end
%
%     r1Skp = transpose(r1)*R1;
%     r0Skp = transpose(r0)*R0;
%     beta = r1Skp/r0Skp;
%     D1 = R1 + beta*D0;
%
%     Update
%     D0 =D1;
%     r0 = r1;
%     R0 = R1;
%     x0 = x1;
% end

%figure(2)
semilogy(resvec)
legend('StationÃ¤re Iteration','PCG')
hold off
```