1 Die Methode der Konjugierten Gradienten

1.1 Problem

Lösung des linearen Gleichungssystems

$$Ax = b \tag{1.1}$$

mit $(N \times N)$–Matrix A und N–komponentigen Spaltenvektoren b und x.
N ist so gross, dass nur Vektoren, aber keine Matrizen gespeichert werden können. A wird implementiert durch eine *Vorschrift*, wie es auf einen Vektor wirkt, d.h. wie $v = Au$ zu bilden ist.
Die bekannten Eliminationsverfahren (Gauss, Householder) zur Lösung des linearen Systems sind nicht anwendbar. Stattdessen werden Wege gesucht, die Lösung aus einer Basis aufzubauen, die durch Anwendung der Matrix A auf *einen* Vektor u entsteht:

$$\{u, Au, A^2u, A^3u, \ldots, A^{n-1}u\}$$

Das nennt man einen **Krylov**–Raum.
Die Lösung x entsteht dann iterativ: $\{x_0 \to x_1 \to x_2 \ldots\}$, ihre Qualität beurteilt man anhand des Restvektors

$$r_k = b - Ax_k \tag{1.2}$$

Von jetzt an nehmen wir der Einfachheit halber an, dass die Matrix A **symmetrisch und positiv** ist. (Ansonsten könnte man statt $Ax = b$ einfach $A^T Ax = A^T b$ lösen.)
Das folgende Verfahren der **Konjugierten Gradienten** geht von einem Startvektor x_0 aus (womöglich $x_0 = 0$) und baut den Krylov–Raum über dem anfänglichen Restvektor $u = r_0$ auf.

1.2 Algorithmus

$$
\begin{array}{rcl}
\text{Initialisierung:} & & \\
\text{if} & & flag \\
 & & x_0 = 0 \\
 & & r_0 = b \\
\text{else} & & \\
 & & r_0 = b - Ax_0 \\
\text{if} & & r_0^T r_0 < tol \\
 & & \text{exit} \\
p_0 & = & r_0 \\
\text{Iteration:} & & \\
s & = & Ap_k \\
\alpha_k & = & \dfrac{p_k^T r_k}{p_k^T s} \\
x_{k+1} & = & x_k + \alpha_k p_k \\
r_{k+1} & = & r_k - \alpha_k s \\
\text{if} & & r_{k+1}^T r_{k+1} < tol \\
 & & \text{exit} \\
\beta_k & = & \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
p_{k+1} & = & r_{k+1} + \beta_k p_k
\end{array}
$$

Bemerkungen zur Implementierung:

- Die Iteration leistet $\{x_k r_k p_k\} \to \{x_{k+1} r_{k+1} p_{k+1}\}$, die Vektoren kann man gleich **überschreiben**. Man sollte sich auch noch $r_k^T r_k$ für die nächste Iteration merken, alle anderen Grössen haben nur temporäre Bedeutung.
- Es wird ein **Hilfsvektor** s eingeführt, damit man A nur einmal pro Iteration anwenden muss.
- Für α_k und β_k gibt es eine Reihe von **Varianten**, die mathematisch (aber nicht numerisch) äquivalent sind. Brauchbar ist vor allem $\alpha_k = (r_k^T r_k)/(p_k^T s)$, damit wird ein Skalarprodukt eingespart.
- Als **Abbruchbedingung** verlangt man sinnvollerweise, dass die Norm von r_k klein genug ist *im Vergleich zu* b, etwa $\|r_k\|/\|b\| < relerr$ mit einer *relativen* Genauigkeit $relerr = 10^{-8}...10^{-12}$.
- An **Speicherplatz** braucht man insgesamt 5 Vektoren, wenn man b behalten will, aber nur 4, wenn man b mit r überschreibt.

1.3 Eigenschaften

Die Verschiebungen p_k sind A–konjugiert (daher der Name):

$$p_k^T A p_l \quad = \quad 0 \qquad \text{für} \quad k \neq l \tag{1.3}$$

Die Restvektoren sind orthogonal (aber nicht normiert):

$$r_k^T r_l \quad = \quad 0 \qquad \text{für} \quad k \neq l \tag{1.4}$$

Daraus folgt *mathematisch*, dass *spätestens* $r_N = 0$, d.h. die CG–Methode ist ein *N–Schritt–Verfahren*. Man beobachtet aber, dass wegen Rundungsfehlern oft erst $r_k = 0$ (Maschinengenauigkeit) für ein $k > N$ erreicht wird.
Die Konvergenz ist monoton in der Norm

$$r_k^T A^{-1} r_k \tag{1.5}$$

(Minimal–Eigenschaft).
Bei grossem N konvergiert das Verfahren exponentiell in dem Sinne, dass z.B.

$$|r_k| \sim e^{-\gamma k} \tag{1.6}$$

Die *Konvergenzrate* γ hängt von der **Konditionszahl** der Matrix ab:

$$\begin{aligned} \mathrm{cond}(A) \quad &\equiv \quad \frac{\lambda_{max}}{\lambda_{min}} && (1.7) \\ \tanh \gamma/2 \quad &= \quad \mathrm{cond}(A)^{-1/2} && (1.8) \\ & && (1.9) \end{aligned}$$

Die für eine gewisse Genauigkeit nötige Zahl von Iterationen ist also proportional zu $\mathrm{cond}(A)^{1/2}$.

1.4 Anwendung: Laplace–Operator auf einem kubischen Gitter

- Rechteckiges Gitter in d Dimensionen mit Gitterkonstante a
- Skalarfeld: $\Phi(x)$

Laplace–Operator:

$$\begin{aligned} -\Delta\Phi(x) &\to a^{-2}\sum_{k=1}^{d}\left[2\Phi(x)-\Phi(x+ae_k)-\Phi(x-ae_k)\right] && (1.10)\\ &= 2da^{-2}\Phi(x)-a^{-2}\sum_{k=1}^{d}\left[\Phi(x+ae_k)+\Phi(x-ae_k)\right] && (1.11)\end{aligned}$$

Diagonalisierung im Impulsraum:

$$\begin{aligned} -\Delta e^{ip\cdot x} &= a^{-2}\sum_{k=1}^{d}2(1-\cos p_k a)\,e^{ip\cdot x} && (1.12)\\ &= \sum_{k=1}^{d}4a^{-2}\sin^2(p_k a/2)\,e^{ip\cdot x} && (1.13)\end{aligned}$$

1.5 "Massiver" Fall

Einführung einer "Masse" m oder Abschirmlänge $\xi = m^{-1}$ führt auf den Operator

$$A = (-\Delta + m^2) \tag{1.14}$$

Bei periodischen Randbedingungen hat man $N_1 \times N_2 \times ... \times N_d$ Gitterpunkte und Impulse

$$p_k = \frac{2\pi n_k}{N_k a}, \qquad n_k = 0\ldots N_k - 1$$

Konditionszahl und Konvergenzrate:

$$\begin{aligned} \Rightarrow\quad \lambda_{min}(A) &= m^2\\ \lambda_{max}(A) &= 4da^{-2}+m^2\\ \Rightarrow\quad \mathrm{cond}(A) &= \frac{4d}{m^2a^2}+1\\ \Rightarrow\quad \sinh(\gamma/2) &= \frac{ma}{2\sqrt{d}}\\ \Rightarrow\quad \gamma &= \frac{ma}{\sqrt{d}}+\mathcal{O}\left((ma)^3\right) && (1.15)\end{aligned}$$

1.6 Laplace–Problem mit Dirichlet–Randbedingungen

Z.B. Gittergrösse $L_1 \times ... \times L_d$ mit $L_k = N_k a$.
Das Feld sei vorgegeben, wenn ein $x_k = 0$ oder $x_k = L_k$, es gibt also insgesamt $(N_1+1) \times ... \times (N_d+1)$ Punkte, davon aber nur $(N_1-1) \times ... \times (N_d-1)$ *innere* Punkte, auf denen das Feld bestimmt werden muss.
Diagonlisierung von $A = -\Delta$:

$$\begin{aligned}
\psi_p(x) &= \prod_{k=1}^{d} \sin p_k x_k \\
p_k &= \frac{\pi n_k}{L_k}, \qquad n_k = 1...(N_k-1) \qquad (1.16) \\
\lambda(p) &= \sum_{k=1}^{d} 4a^{-2} \sin^2(p_k a/2)
\end{aligned}$$

Konvergenzrate:

$$\begin{aligned}
\lambda_{min}(A) &= \sum_{k=1}^{d} 4a^{-2} \sin^2 \frac{\pi}{2N_k} \\
\lambda_{max}(A) &= 4da^{-2} \\
\Rightarrow \quad \mathrm{cond}(A) &= d/\sum_{k=1}^{d} \sin^2 \frac{\pi}{2N_k} \\
\Rightarrow \quad \tanh \gamma/2 &= \left[d^{-1} \sum_{k=1}^{d} \sin^2 \frac{\pi}{2N_k} \right]^{1/2} \qquad (1.17)
\end{aligned}$$

Speziell für $N_k = N$:

$$\begin{aligned}
\tanh \frac{\gamma}{2} &= \sin \frac{\pi}{2N} \\
&= \sin \frac{\pi a}{2L} \qquad (1.18) \\
\Rightarrow \quad \gamma &= \frac{\pi a}{L} + \mathcal{O}\left((a/L)^3\right) \qquad (1.19)
\end{aligned}$$

1.7 Implementierung der Gittergeometrie

Auf einem d–dimensionalen Gitter (Gitterkonstante $a = 1$) der Größe $L_1 \times L_2 \times .. \times L_d$ soll ein Feld $\Phi(x)$ als *eindimensionaler* Array implementiert werden. Man muss die Gitterpunkte also irgendwie durchnummerieren und dann dafür sorgen, dass man die Nachbarpunkte wiederfindet, ggf. unter Berücksichtigung der Randbedingungen.
Dazu folgende Idee: man durchläuft das Gitter systematisch, x_1 läuft am schnellsten, dann x_2 usw. Wenn man die Koordinaten als $x_k = 0..(L_k - 1)$ ansetzt, dann lässt sich die Nummerierung schreiben als

$$\begin{aligned} i &= x_1 + L_1 x_2 + L_1 L_2 x_3 + ... \\ i &= 0..(V-1) \end{aligned}$$

Das hat eine interessante Interpretation: die x_k erscheinen als *Ziffern* in einem Zahlensystem mit (positionsabhängiger) *Basis*

$$\begin{aligned} b_1 &= 1 \\ b_2 &= L_1 \\ b_3 &= L_1 L_2 \\ &... \\ \Rightarrow \quad i &= \sum_{k=1}^{d} b_k x_k \\ b_k &= \prod_{l=1}^{k-1} L_l \end{aligned} \tag{1.20}$$

und der sich so ergebende *Zahlenwert* ist der gesuchte Index. Natürlich braucht man Koordinaten und Index nicht von 0 an zu zählen, aber damit ist es am einfachsten zu erklären.

1.7.1 Periodische Randbedingungen

Nun muss man noch das Indexfeld konstruieren, mit dem man Nachbarn findet. Für periodische Randbedingungen wird das durch das folgende Geometrieprogramm erledigt, das einmal das gesamte Gitter durchläuft, dabei sowohl die x_k als auch den Index i im Auge behält und so für jeden Punkt die Indizes der Nachbarn bestimmen kann:

```
#include <stdlib.h>
#include "global.h"
/*
   ueber global.h werden folgende globale Variablen eingebunden:
   int   ndim, nvol, *lsize, **nn;
*/

void geom_pbc(){
```

```
/*
Folgende Groessen muessen gesetzt sein:              B Bunk 12/2005
    Dimension     ndim                               rev     4/2013
    Gittergroesse lsize[k], k=1..ndim

Angelegt und berechnet wird
    Volumen       nvol
    NN-Indexfeld  nn[k][i], k=0..2*ndim, i=0..(nvol-1)

nn[k][i] gibt den Index des Nachbarn von i in Richtung +k,
    unter Beruecksichtigung periodischer Randbedingungen.
    Fuer einen Schritt in Richtung -k setze man den Index auf (ndim+k).
    nn[0][i] ist reserviert.
*/
int   i, k;
int *ibase, *ix;

ibase = (int *) malloc((ndim+2) * sizeof(int));
ix = (int *) malloc((ndim+1) * sizeof(int));

/* Basis fuer Punktindizes */
ibase[1] = 1;
for (k=1; k<=ndim; k++) ibase[k+1] = ibase[k]*lsize[k];
nvol = ibase[ndim+1];

if (nn) free(nn[0]);
free(nn);
nn = (int **) malloc((2*ndim+1) * sizeof(int *));
nn[0] = (int *) malloc(nvol*(2*ndim+1) * sizeof(int));
for (k=1; k<=2*ndim; k++) nn[k] = nn[0] + nvol*k;

for (k=1; k<=ndim; k++) ix[k] = 0;   /* Koord. des Anfangspunkts */

for (i=0; i<nvol; i++){           /* Schleife ueber Punkte */
  for (k=1; k<=ndim; k++){
    nn[k][i] = i + ibase[k];      /* Nachbar x + e_k */
    if (ix[k] == (lsize[k]-1)) nn[k][i] -= ibase[k+1];

    nn[ndim+k][i] = i - ibase[k]; /* Nachbar x - e_k */
    if (ix[k] == 0) nn[ndim+k][i] += ibase[k+1];
  }

  for (k=1; k<=ndim; k++){        /* Koord. des naechsten Punkts */
    ix[k]++;
    if (ix[k] < lsize[k]) break;
    ix[k] = 0;
  }
}
free(ibase); free(ix);
```

```
58 }
```

Wenn dann das Feld wirklich bearbeitet wird, beispielsweise im CG, werden nur noch die Indizes i durchlaufen, von den $\{x_k\}$ ist nicht mehr die Rede.

1.7.2 Dirichlet–Randbedingen

Hier sieht man für die festgehaltenen Werte von $\Phi(x)$ auf dem Rand ebenfalls Feldkomponenten vor, die anfangs auf die vorgegebenen Werte gesetzt werden und dann nicht mehr verändert werden dürfen, sondern nur als Nachbarn der inneren (aktiven) Punkte ins Spiel kommen. Der Einfachheit halber werden auch alle anderen Felder entsprechend erweitert. Man implementiert nun den Laplace–Operator so, dass er (bei Anwendung auf irgendein Feld) am Rand Nullen zurückgibt. Dann muss bei einem Problem wie $(-\Delta+m^2)\Phi(x) = \eta(x)$ das Quellfeld ergänzt werden um $\eta(x) = m^2\Phi(x)$ auf dem Rand. Im CG–Verfahren werden damit alle Shift– und Restvektoren auf den Randpunkten ebenfalls verschwinden, und die Gleichungen für $\Phi(x)$ auf den inneren Punkten werden konsistent mit den Randbedingungen gelöst.
Genauer: der anfängliche Restvektor

$$r_0 = \eta - (-\Delta + m^2)\phi_0 \tag{1.21}$$

muss auf dem Rand verschwinden. Dann kann man anhand der Rekursion zeigen, dass *alle* Rest- und Shiftvektoren am Rand verschwinden, wenn eine der beiden Varianten implementiert wird:

$$\Delta \;\rightarrow\; 0 \qquad \eta \;=\; m^2\phi_0 \quad \text{auf dem Rand} \tag{1.22}$$

oder

$$(-\Delta + m^2) \;\rightarrow\; 0 \qquad \eta \;=\; 0 \quad \text{auf dem Rand} \tag{1.23}$$

Bei $m = 0$ fällt beides zusammen.
Um die Unterscheidung zwischen inneren und Randpunkten schnell anhand des Index treffen zu können, führt man ein Feld von Flags ein. Dazu kann das Geometrieprogramm z.B. die Komponente $nn(i, 0)$ verwenden – sie wird ja ansonsten nicht benutzt. Im Übrigen wird $nn(i, k)$ für $k \neq 0$ wie bei periodischen Randbedingungen gesetzt.