Beginn der Vorlesung zur Numerik I
(Wintersemester 2010/2011)

M. Sc. Frank Gimbel

---

1 Motivation

Ziel ist es, ein gegebenes lineares Gleichungssystem der Form

$$Ax = b \tag{1}$$

mit $x, b \in \mathbb{R}^n$ und $A \in \mathbb{R}^{n \times n}$ zu lösen.
Solche Systeme treten etwa bei der numerischen Simulation physikalischer und technischer Vorgänge (Finite Differenzen, Finite Elemente) oder bei Interpolationsproblemen auf.
Ein Beispiel ist die FE-Simulation der Laplace-Gleichung

$$\Delta u = -f$$

deren Diskretisierung mit Ansatzfunktionen φ_i auf Matrixeinträge

$$a_{ij} = \int \nabla\varphi_i \cdot \nabla\varphi_j dx \text{ und } b_i = \int f\varphi_i dx$$

führt.
Im eindimensionalen Fall linearer finiter Elemente mit konstanter Gitterweite h hat man konkret:

$$A = \frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}. \tag{2}$$

2 Gauß-Algorithmus

Ein bekanntes Lösungsverfahren für das lineare Gleichungssystem (1) ist der **Gaußsche Algorithmus**:

```
//I) Elimination auf Dreiecksform:
for i=1,...,n
  for j=i+1,...,n
    l(j,i) <- a(j,i)/a(i,i)
    for k=i+1,...,n
      a(j,k) <- a(j,k)-l(j,i)*a(i,k)
    b(j) <- b(j)-l(j,i)*b(i)

//II) Rueckwaertseinsetzen:
for i=n,...,1
  x(i) <- b(i)
  for j=i+1,...,n
    x(i) <- x(i)-a(i,j)*x(j)
  x(i) <- x(i)/a(i,i)
```

[Hierbei meint `a(i,j)` den Matrixeintrag a_{ij} und `x(j)` den Vektoreintrag x_j.] In Schritt I) wird das Gleichungssystem durch die Linearkombination der Gleichungen in Dreiecksgestalt überführt.
Mittels dieser Dreiecksgestalt lässt sich in Schritt II) die Lösung des Systems durch einfaches Rückwärtseinsetzen gewinnen.
Beispiel:

$$\begin{pmatrix} 2 & 3 & 4 & 9 \\ 6 & 0 & 2 & 0 \\ 1 & 3 & 2 & 8 \\ 6 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 0 \\ -3 \end{pmatrix}$$

Elimination					
2	3	4	9	56	
6	0	2	0	12	$-3\times$ 1. Zeile
1	3	2	8	45	$-1/2\times$ 1. Zeile
6	0	-1	1	7	$-3\times$ 1. Zeile
2	3	4	9	56	
0	-9	-10	-27	-156	
0	3/2	0	7/2	17	$+1/6\times$ 2. Zeile
0	-9	-13	-26	-161	$-$ 2. Zeile
2	3	4	9	56	
0	-9	-10	-27	-156	
0	0	$-5/3$	-1	-9	
0	0	-3	1	-5	$-9/5\times$ 3. Zeile
2	3	4	9	56	
	-9	-10	-27	-156	
		$-5/3$	-1	-9	
			14/5	56/5	
Rückwärtseinsetzen				b	x
2	3	4	9	56	x_1
	-9	-10	-27	-156	x_2
		$-5/3$	-1	-9	x_3
			14/5	56/5	4
2	3	4	9	56	x_1
	-9	-10	-27	-156	x_2
		$-5/3$	-1	-9	3
			14/5	56/5	4
2	3	4	9	56	x_1
	-9	-10	-27	-156	2
		$-5/3$	-1	-9	3
			14/5	56/5	4
2	3	4	9	56	1
	-9	-10	-27	-156	2
		$-5/3$	-1	-9	3
			14/5	56/5	4

Zu dieser Version des Gauß-Algorithmus ist anzumerken, daß es bei der Berechnung der Koeffizienten `l(j,i) <- a(j,i)/a(i,i)` zu Problemen kommt, wenn `a(i,i)` Null ist. Dieser Fall wird zunächst ausgeschlossen, um die weiteren Untersuchungen übersichtlich zu gestalten. Außerdem kann man unter gewissen Voraussetzungen zeigen, daß während des gesamten Algorithmus $a_{ii} \neq 0$, $i = 1, \ldots, n$ gilt, etwa im Fall diagonal-dominanter oder positiv definiter Matrizen:

Definition
Eine Matrix $A \in \mathbb{R}^{n \times n}$ heißt

- diagonal-dominant, wenn

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \quad i = 1, \ldots, n$$

- streng diagonal-dominant, wenn

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \quad i = 1, \ldots, n$$

- symmetrisch, wenn $a_{ij} = a_{ji}$ für $i, j = 1, \ldots, n$
- positiv semi-definit, wenn sie symmetrisch ist, und für beliebige Vektoren $x \in \mathbb{R}^n$ gilt

$$x^T A x \geq 0$$

- positiv definit, wenn sie positiv semi-definit und $x^T A x = 0$ nur für $x = 0$ gilt.
- negativ (semi-)definit, wenn $-A$ positiv (semi-)definit ist.

Übung:
Zeigen Sie:

a) Eine streng diagonaldominante Matrix ist regulär.

b) Eine symmetrische, streng diagonaldominante Matrix mit positiven Diagonaleinträgen ist positiv definit.

Es stellt sich die Frage, wie teuer dieser Algorithmus ist. Um diese Kosten quantifizieren zu können, bedient man sich der folgenden Notation:

Definition: Für eine Funktion

$$f : \mathbb{R} \to \mathbb{R}$$

gilt

$$f = \mathcal{O}(g) \quad (x \to \infty),$$

sprich „f ist von der Ordnung g“, wenn Konstanten $K, M \in \mathbb{R}$ existieren, mit denen

$$\forall x \geq M \qquad |f(x)| \leq K \cdot g(x)$$

ist.

Wenn wir nun voraussetzen, daß eine algebraische Operation (im vorliegenden Code alle Stellen, an denen `<-` auftritt) konstanten Aufwand (Rechenzeit) W benötigt, lässt sich die Komplexität berechnen:

- Die Kosten der Schleife über `k` sind $W_k = (n - i) \cdot W$.

- Die Kosten jedes einzelnen `j`-Schleifendurchlaufes in (I) sind $W + W_k + W = 2W + (n-i) \cdot W = (n-i+2)W$. Die gesamte Schleife hat also die Kosten $W_i = (n-i) \cdot (n-i+2)W = W \cdot [i^2 - (2n+2)i + n(n+2)]$.

- Die Kosten des gesamten Eliminationsschrittes (I) sind demzufolge

$$\begin{aligned} W_E &= \sum_{i=1}^{n} W_i = W \sum_{i=1}^{n} i^2 - W(2n+2) \sum_{i=1}^{n} i + Wn^2(n+2) \\ &= W \left(\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{n}{6} \right) - 2W(n+1)\frac{n(n-1)}{2} + Wn^2(n+2) \\ &= \frac{W}{3}n^3 + \frac{5}{2}Wn^2 + \frac{7}{6}Wn \end{aligned}$$

Also ist der Aufwand von Schritt I) $\mathcal{O}(n^3)$, etwa mit $M = 1$ und $K = 4$. Der Aufwand des Rückwärtseinsetzens (Schritt II) berechnet sich ähnlich zu

$$W_R = \sum_{i=1}^{n} (W + (n-i) \cdot W) = W \cdot \left(n + n^2 - \frac{n(n-1)}{2} \right) = \mathcal{O}(n^2).$$

Das bedeutet, daß eine Verdopplung der Anzahl an Unbekannten n den Aufwand des Eliminierens verachtfacht, während es den Aufwand des Einsetzens lediglich vervierfacht. Bedenkt man, daß in Anwendungen durchaus Gleichungssysteme mit mehreren Tausend oder gar Millionen Unbekannten auftreten und dies mehrfach für dieselbe Matrix A und unterschiedliche rechte Seiten b (Zum Beispiel in Newton-Iterationen oder Uzawa-Algorithmen), ist es sinnvoll, den Eliminationsschritt (I) nur möglichst selten durchzuführen. Dies gelingt, indem man ihn zunächst aufspaltet:

```
//Ia) Elimination auf Dreiecksform:
for i=1,...,n
  for j=i+1,...,n
    l(j,i) <- a(j,i)/a(i,i)
    for k=i+1,...,n
      a(j,k) <- a(j,k)-l(j,i)*a(i,k)
//Ib) Transformation der rechten Seite
for j=1,...,n
  for i=1,...,j-1
    b(j) <- b(j)-l(j,i)*b(i)

//II) Rueckwaertseinsetzen:
for i=n,...,1
  x(i) <- b(i)
  for j=i+1,...,n
    x(i) <- x(i)-a(i,j)*x(j)
  x(i) <- x(i)/a(i,i)
```

Nun muss man jedoch die Koeffizienten des Eliminationsschrittes `l(i,j)` abspeichern um sie im Transformationsschritt (Ib) verwenden zu können.
Dies ermöglicht auch, die Reihenfolge der beiden Schleifen der `b`-Transformation zu vertauschen.
Das Abspeichern der `l(j,i)` kann ohne zusätzlichen Speicherplatzbedarf geschehen, wenn man `l(j,i)` in den Speicher von `a(j,i)` schreibt, da dieser ohnehin nicht mehr benötigt

wird.
Nach dem Eliminationsschritt (I) hat man dann eine Matrix der Form

$$\begin{pmatrix} \tilde{a}_{11} & \dots & \dots & \tilde{a}_{1n} \\ l_{21} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ l_{n1} & \dots & l_{n,n-1} & \tilde{a}_{nn} \end{pmatrix} \tag{3}$$

Im obigen Beispiel, hat die Matrix dann die Gestalt

$$\begin{array}{|cccc|} \hline 2 & 3 & 4 & 9 \\ \hline 3 & -9 & -10 & -27 \\ \hline 1/2 & -1/6 & -5/3 & -1 \\ \hline 3 & 1 & 9/5 & 14/5 \\ \hline \end{array}$$

Um das Gleichungssystem (1) nun für unterschiedliche rechte Seiten zu lösen, muss man den Schritt (Ia) mit Aufwand $\mathcal{O}(n^3)$ einmal ausführen und danach nur noch für jede rechte Seite die beiden Schritte (Ib) und (II) mit Aufwand $\mathcal{O}(n^2)$.
Formal entspricht Schritt (Ib) dem Lösen des Systemes $Ly = b$ und Schritt (II) dem Lösen des Systemes $Rx = y$.

Übung:
Ermitteln Sie unter Nutzung der LR-Zerlegung der Beispielmatrix die Lösung des Systems $Ax = b'$ mit $(b')^T = (34, 28, 25, 23)$.

Bei allen Betrachtungen des Aufwandes der Algorithmen ist zu beachten, daß bereits der Aufruf `b(i)` in der Regel den Aufwand $\mathcal{O}(\log(n))$ benötigt. Dieser lässt sich jedoch durch die Verwendung von Iteratoren auf $\mathcal{O}(1)$ reduzieren.
Um den neuen Algorithmus formal zu untersuchen, spalten wir die Matrix (3) auf in

$$C = \begin{pmatrix} 0 & \dots & \dots & 0 \\ l_{2,1} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ l_{n,1} & \dots & l_{n,n-1} & 0 \end{pmatrix} \quad \text{und} \quad R = \begin{pmatrix} \tilde{a}_{1,1} & \dots & \dots & \tilde{a}_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \tilde{a}_{n,n} \end{pmatrix}. \tag{4}$$

Wir untersuchen die drei Teile des Algorithmus separat:

- (Ia): Das Aufstellen der Matrizen C und R, das im ersten Schritt des Algorithmus geschieht, wird nicht näher umformuliert und untersucht.
- (Ib): Die neuen Einträge $\tilde{b}_j$ berechnen sich zu

$$\begin{aligned} \tilde{b}_j =& b_j - \sum_{i=1}^{j-1} l_{ji}\tilde{b}_i = b_j - \sum_{i=1}^{n} l_{ji}\tilde{b}_i = b_j - (C\tilde{b})_j, \quad j = 1, \dots, n \\ \Rightarrow \qquad \tilde{b} =& (I + C)^{-1} b \end{aligned}$$

- (II): Das Ergebnis $x \in \mathbb{R}^n$ berechnet sich mit diesem $\tilde{b}$ zu

$$x_i = \frac{1}{\tilde{a}_{ii}} \left(\tilde{b}_i - \sum_{j=i+1}^{n} \tilde{a}_{ij} x_j \right) = \frac{1}{\tilde{a}_{ii}} \left(\tilde{b}_i - ((R - D)x)_i \right), \text{ mit } D = \begin{pmatrix} \tilde{a}_{11} & & 0 \\ & \ddots & \\ 0 & & \tilde{a}_{nn} \end{pmatrix}$$

$$\begin{aligned} \Rightarrow \quad Dx =& \tilde{b} - (R - D)x \\ \Rightarrow \quad x =& R^{-1}\tilde{b} = R^{-1}(I + C)^{-1} b \end{aligned} \tag{5}$$

Gleichung (5) lässt sich umformen zu

$$(I + C)Rx = b = Ax.$$

Diese Gleichung gilt für alle x. Falls also A regulär ist, gilt

$$A = (I + C)R.$$

Man hat also im Gauß-Algorithmus eine Zerlegung der Matrix $A = LR$ in ein Produkt aus der oberen Dreiecksmatrix R und der unteren Dreiecksmatrix

$$L = I + C = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n1} & \dots & l_{n,n-1} & 1 \end{pmatrix}$$

gewonnen.
Das Invertieren jeder einzelnen dieser beiden Matrizen L und R, das in den Schritten (Ib) und (II) erledigt wird, kostet einen Aufwand von $\mathcal{O}(n^2)$.
Dieser Aufwand lässt sich verringern, wenn L und R dünn besetzte Matrizen sind, wenn also in jeder Zeile der beiden Matrizen maximal β Einträge ungleich Null sind, wobei β als unabhängig von n vorausgesetzt wird. Zum einen benötigt man dann nur noch $\beta \cdot n$ Speicherplätze, um die einzelnen Matrizen zu speichern. Zum anderen ist dann der Aufwand der `i`-Schleife in Schritt (Ib) durch $\beta \cdot W = \mathcal{O}(1)$ beschränkt und damit der Gesamtaufwand für (Ib) von der Ordnung $\mathcal{O}(n)$ und genauso der Aufwand für das Rückwärtseinsetzen (Schritt (II)) $\mathcal{O}(n)$.
Dieser Aufwand ist aber asymptotisch optimal, da bereits das Speichern oder auch Ausgeben des Lösgunsvektors `x` einen Aufwand $\mathcal{O}(n)$ benötigt.
Insbesondere bei Gleichungssystemen, die in Finite-Elemente-Simulationen aufgestellt werden, hat man in der Regel dünn besetzte Matrizen. Etwa für lineare Finite Elemente in zwei Dimensionen hat man $\beta = 9$ und noch einfacher in einer Dimension für die Matrix (2) $\beta = 1$.