KTH Engineering Sciences

Linjära ekvationssystem

1 Gausselimination

Vanlig gausselimination för det linjära ekvationssystemet $A\boldsymbol{x} = \boldsymbol{b}$ utgår från den utökade matrisen $[\,A \mid \boldsymbol{b}\,]$ och applicerar elementära radoperationer på denna för att transformera den till $[\,U \mid \tilde{\boldsymbol{b}}\,]$ där U är övertriangulär. Detta motsvarar ett nytt ekvivalent övertriangulärt ekvationssystem $U\boldsymbol{x} = \tilde{\boldsymbol{b}}$ som är enkelt att lösa med bakåtsubstitution.

Exempel 1: Vi vill lösa

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 9 \end{pmatrix}.$$

Vi bildar den utökade matrisen $[A \mid \boldsymbol{b}]$ och reducerar i två steg:

$$\left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 5 \\ 7 & 8 & 10 & 9 \end{array}\right) \xrightarrow[\substack{\text{Ny rad 2 = rad 2 - 4×rad 1} \\ \text{Ny rad 3 = rad 3 - 7×rad 1}}]{} \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & 1 \\ 0 & -6 & -11 & 2 \end{array}\right)$$

$$\xrightarrow[\text{Ny rad 3 = rad 3 - } \frac{-6}{-3}\text{×rad 2}]{} \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -6 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right).$$

Detta ger det ekvivalenta ekvationssystemet

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}.$$

I bakåtsubstitution startar vi på sista raden, vilken direkt ger att $z = 0$. Vi arbetar oss sedan bakåt rad för rad. Vi sätter in värdet för z i rad två och får

$$-3y - 6z = 1 \quad \Rightarrow \quad -3y - 6 \cdot 0 = 1 \quad \Rightarrow \quad y = -\frac{1}{3},$$

och sedan värdena för z och y i rad ett,

$$x + 2y + 3z = 1 \quad \Rightarrow \quad x + 2\left(-\frac{1}{3}\right) + 3 \cdot 0 = 1 \quad \Rightarrow \quad x = \frac{5}{3}.$$

2 LU-faktorisering

Resultatet av gausselimination kan också beskrivas i matristermer. Det man får fram är i själva verket en *faktorisering* av ursprungsmatrisen A i en produkt av två enklare matriser, nämligen $A = LU$ där U är övertriangulär och L är undertriangulär med ettor på diagonalen, dvs

schematiskt

$$L = \begin{pmatrix} 1 & & & \\ \times & 1 & & \\ \vdots & \vdots & \ddots & \\ \times & \times & \times & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} \times & \times & \times & \times \\ & \ddots & \vdots & \vdots \\ & & \times & \times \\ & & & \times \end{pmatrix}.$$

Detta kallas *LU-faktoriseringen* av A. Matrisen U är samma matris som står till vänster i den utökade matrisen efter gausselimination. Matrisen L utgörs av de faktorer som multiplicerar den rad som subtraheras bort i varje steg[1]. Dessa kan enkelt bokföras under gausseliminationen utan extra kostnad. För vektorn $\tilde{\boldsymbol{b}}$ i den utökade matrisen gäller slutligen att $L\tilde{\boldsymbol{b}} = \boldsymbol{b}$. Lösningsalgoritmen för $A\boldsymbol{x} = \boldsymbol{b}$ kan därför skrivas i matrisform som:

1. LU-faktorisera $A \;\Rightarrow\; LU\boldsymbol{x} = \boldsymbol{b}$,
2. Lös det undertriangulära systemet $L\tilde{\boldsymbol{b}} = \boldsymbol{b} \;\Rightarrow\; U\boldsymbol{x} = \tilde{\boldsymbol{b}}$,
3. Lös det övertriangulära systemet $U\boldsymbol{x} = \tilde{\boldsymbol{b}} \;\Rightarrow\; \boldsymbol{x}$.

Punkt 1) görs med gausselimination och löpande bokföring av L-matrisens element. Punkterna 2) och 3) är enkla triangulära ekvationssystem som löses med bakåtsubstitution (för övertriangulära) och framåtsubstitution (för undertriangulära).

En poäng med detta synsätt är att det visar hur man kan snabba upp beräkningar när man behöver lösa ett ekvationssystem med med många olika högerled, $A\boldsymbol{x} = \boldsymbol{b}_j$, $j = 1, 2, \ldots$ Se Exempel 4 nedan.

Exempel 2: LU-faktoriseringen av matrisen

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}$$

ges direkt av gausseliminationen i Exempel 1 ovan. Den är

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{pmatrix}.$$

Notera hur matriselementen i L dyker upp i gausseliminationens olika steg. Verifiera att $A = LU$.

Kommentar: En komplikation är att LU-faktorisering och gausselimination inte fungerar för alla icke-singulära matriser. Det finns problem $A\boldsymbol{x} = \boldsymbol{b}$ som har en väldefinierad lösning men där vår vanliga lösningsalgoritm inte går att använda, tex.

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Lösningen är $x = y = 1$ men gausselimination bryter samman direkt i steg ett eftersom faktorn som ska multiplicera rad ett blir "1/0". Följande sats ger de precisa villkoren för när A kan LU-faktoriseras, dvs när gausselimination fungerar.

[1] Detta är inte helt uppenbart! På sid 80-81 i Sauer ges en informell härledning.

Theorem 1 *Låt A_j vara den ledande principala undermatrisen av storlek $j \times j$ till $A \in \mathbb{R}^{n\times n}$, med $j \leq n$. Med detta menas det övre vänstra hörnet av A, i* MATLAB*-notation $A_j =$*`A(1:j,1:j)`*. Matrisen A har en unik LU-faktorisering (med ettor på diagonalen till L) om och endast om alla A_j är icke-singulära, $j = 1, \ldots, n$.*

Ett exempel på matriser som uppfyller villkoren i satsen är symmetrisk positivt definita matriser (dvs $A = A^T$ och $\boldsymbol{x}^T A\boldsymbol{x} > 0$ för alla $\boldsymbol{x} \neq 0$).

Att standard gausselimination inte fungerar för alla matriser är dock i praktiken ett mindre problem, eftersom det enkelt går att åtgärda genom så kallad *pivotering*, där man systematiskt kastar om raderna i varje steg för att undvika division med små tal. Läs mer om pivotering i Sauer kap 2.4. Gausselimination med pivotering ger den modifierade LU-faktoriseringen $PA = LU$, där P är en permutationsmatris (identitetsmatrisen med omkastade rader). Man kan visa att denna modifierade LU-faktorisering existerar för alla icke-singulära matriser. Med pivotering undviks också problem med utskiftning som kan uppstå till följd av avrundningsfel, se Sauer kap 2.3.2.

3 Beräkningskostnader

Beräkningskostnaden för en numerisk algoritm brukar räknas som antalet enkla flyttalsoperationer (flop) den inbegriper, där

1 flop = 1 multiplikation, division, addition eller subtraktion.

Det ger ofta en god bild av hur snabb en algoritm är, men man ska samtidigt vara medveten om att många andra faktorer också spelar in, t.ex. antal minnesaccesser, hur mycket data som måste skickas mellan processorer i en multicore-dator, storleken på cache-minnet, etc. Speciellt intressant är hur kostnaden beror på problemets storlek, typiskt antal obekanta, eller storleken på en matris/vektor. Detta brukar också kallas problemets komplexitet.

Exempel 3: Matris-vektormultiplikation. Låt $A \in \mathbb{R}^{n\times n}$ och $\boldsymbol{b} \in \mathbb{R}^n$. Om $A\boldsymbol{x} = \boldsymbol{b}$,

$$b_i = \sum_{j=1}^{n} a_{ij}x_j, \qquad i = 1, \ldots, n.$$

För varje element b_i krävs alltså n multiplikationer och $n-1$ additioner. Totala kostnaden för att beräkna samtliga element blir då

$$n(2n-1) = 2n^2 - n \text{ flops.}$$

Normalt är vi bara intresserade av den högsta potensen (n^2) eftersom den helt dominerar när n är stort. Konstanten framför (2) är ibland också av intresse, men ofta bortser vi även från den och säger att komplexiteten i detta fall är $\mathcal{O}(n^2)$ för matris-vektormultiplikation.

I listan nedan sammanfattas beräkningskostnader för några vanliga operationer i linjär algebra.

1. Multiplikation matris-vektor: $\mathcal{O}(n^2)$
2. Multiplikation matris-matris: $\mathcal{O}(n^3)$
3. Lösa övertriangulärt system med bakåtsubstitution: $\mathcal{O}(n^2)$
4. Lösa undertriangulärt system med framåtsubstitution: $\mathcal{O}(n^2)$
5. LU-faktorisera en matris med gausselimination: $\mathcal{O}(n^3)$

Punkterna 3-5) visar att kostnaden att lösa ekvationssystemet $A\boldsymbol{x} = \boldsymbol{b}$ är $\mathcal{O}(n^3)$. Se kaptiel 3.1.2 i Sauer för en härledning av detta viktiga resultat.

Exempel 4: Antag att vi behöver lösa samma ekvationssystem med många högerled. Mer precist, antag att vi har m högerled och att $A \in \mathbb{R}^{n\times n}$,

$$A\boldsymbol{x}_j = \boldsymbol{b}_j, \qquad j = 1, \ldots, m.$$

Ett naivt angreppssätt vore att lösa dessa system med gausselimination ett efter ett. Den totala kostnaden skulle då bli $\mathcal{O}(mn^3)$ eftersom varje problem kostar $\mathcal{O}(n^3)$ flops. Ett bättre sätt vore istället att

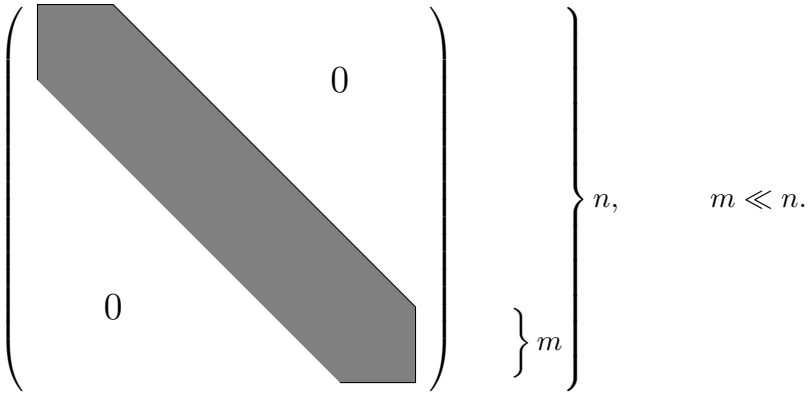
1. LU-faktorisera A,
2. För varje j lös de triangulära systemen:

$$L\tilde{\boldsymbol{b}}_j = \boldsymbol{b}_j, \qquad U\boldsymbol{x}_j = \tilde{\boldsymbol{b}}_j.$$

Kostnaden för steg 1) är $\mathcal{O}(n^3)$. Kostnaden för steg 2) är $\mathcal{O}(n^2)$ per j eftersom systemen är triangulära. Totalkostnaden blir alltså $\mathcal{O}(n^3 + mn^2)$, vilket är betydligt mindre än $\mathcal{O}(mn^3)$ när n och antalet högerled m är stora.

4 Glesa matriser

Glesa matriser (*sparse matrices*) är matriser som nästan bara innehåller nollor. En mycket vanlig typ av glesa matriser är *bandade* matriser där alla element utanför ett smalt band runt diagonalen är noll:



Här kallas m bandbredden. För tridiagonala matriser är tex $m = 2$. Matris-vektor- och matris-matris-multiplikationer med glesa matriser kan göras betydligt snabbare än för vanliga fulla matriser genom att utnyttja strukturen av nollor. För vissa typer av glesa matriser (men inte alla!) kan även gausselimination och LU-faktorisering göras snabbare, tex för bandade matriser. För dessa blir en avsevärd andel av mellanresultaten bara noll och behöver inte räknas ut.

För bandade matriser är komplexiteten för alla de fem operationerna som listas i Sektion 3 ovan tex bara $\mathcal{O}(n)$, om vi antar att m är fixt när n varierar. Att lösa ett tridiagonalt ekvationssystem med en bandad version av gausselimination har alltså kostnaden $\mathcal{O}(n)$ istället för $\mathcal{O}(n^3)$. (Se tex kap 1.6 i NAM.)

I MATLAB kan matriser och vektorer lagras och hanteras i ett speciellt "sparse"-format. MATLAB kan då utnyttja bättre, snabbare algoritmer och använda mindre minne. Se tex `help sparse`.

Exempel 5: Antag att matrisen $A \in \mathbb{R}^{n\times n}$ är tridiagonal med elementen α_j, β_j och γ_j,

$$A = \begin{pmatrix} \beta_1 & \gamma_1 & & & \\ \alpha_2 & \beta_2 & \gamma_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} \\ & & & \alpha_n & \beta_n \end{pmatrix}.$$

Om vi vill lösa $A\boldsymbol{x} = \boldsymbol{b}$ med $\boldsymbol{b} = (b_1, \ldots, b_n)^T \in \mathbb{R}^n$ innebär gausselimination följande två steg. Först reducerar vi till triangulär form. Vi låter $\beta_1' = \beta_1$, $b_1' = b_1$ och

$$\beta_j' = \beta_j - \frac{\gamma_{j-1}\alpha_j}{\beta_{j-1}'}, \qquad b_j' = b_j - \frac{b_{j-1}\alpha_j}{\beta_{j-1}'}, \qquad j = 2, \ldots, n.$$

Subtraktion av rader blir här alltså bara subtraktion av skalärer. Vi får

$$\begin{pmatrix} \beta_1' & \gamma_1 & & & \\ & \beta_2' & \gamma_2 & & \\ & & \ddots & \ddots & \\ & & & \beta_{n-1}' & \gamma_{n-1} \\ & & & & \beta_n' \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \\ \vdots \\ b_{n-1}' \\ b_n' \end{pmatrix}$$

Lösningen ges sedan av bakåtsubstitutionen $x_n = b_n'/\beta_n'$ och

$$x_j = \frac{1}{\beta_j'}(b_j' - \gamma_j x_{j+1}), \qquad j = n-1, n-2, \ldots, 1.$$

I första reduktionen görs 2×3 operationer för varje rad $2, \ldots, n$ (en multiplikation, en division och en subtraktion). I bakåtsubstitutionen görs 3 operationer för varje rad $n-1, \ldots, 1$ och en operation på rad n. Totalt har vi $9(n-1)+1$ operationer, dvs $\mathcal{O}(n)$ operationer.

5 Vektor- och matrisnormer

För att mäta storleken av en vektor eller matris använder vi *normer*. Det kan ses som en generalisering av absolutbeloppet för skalära tal. Normen av vektorn $\boldsymbol{x}$ skrivs $||\boldsymbol{x}||$. Flera olika val av normer är möjliga för vektorer. Vanligast är den *euklidiska* normen (även kallad 2-normen),

$$||\boldsymbol{x}||_2 := \sqrt{\sum_{j=1}^{n} x_j^2}, \qquad \boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n,$$

som mäter "längden" av $\boldsymbol{x}$ i $\mathbb{R}^n$. Ibland används också

$$\begin{aligned} &\text{1-normen} && ||\boldsymbol{x}||_1 := \textstyle\sum_{j=1}^{n} |x_j|, \\ &\text{max-normen} && ||\boldsymbol{x}||_\infty := \max_j |x_j|. \end{aligned}$$

Alla normer uppfyller

1. $||\boldsymbol{x}|| \geq 0$ för alla vektorer $\boldsymbol{x}$,
2. $||\boldsymbol{x}|| = 0$ om och endast om $\boldsymbol{x} = 0$,

3. $||\alpha \boldsymbol{x}|| = |\alpha| ||\boldsymbol{x}||$ för alla skalära tal $\alpha \in \mathbb{R}$,

4. $||\boldsymbol{x} + \boldsymbol{y}|| \leq ||\boldsymbol{x}|| + ||\boldsymbol{y}||$ för alla vektorer $\boldsymbol{x}$ och $\boldsymbol{y}$ (triangelolikheten).

(Dessa fyra villkor kan även ses som axiomen vilka definierar en norm i ett vektorrum.)

Matrisnormer definieras vanligtvis i termer av vanliga vektornormer enligt

$$||A|| := \sup_{\boldsymbol{x} \neq 0} \frac{||A\boldsymbol{x}||}{||\boldsymbol{x}||}. \tag{1}$$

Notera att normerna i högerledet båda är vektornormer. För varje vektornorm får man således en motsvarande matrisnorm. Låt a_{ij} vara elementet på rad i och kolumn j i A. Man kan visa att 1-, 2-, och max-normen ger följande uttryck för matrisnormerna:

1-normen	$\|\|A\|\|_1 = \max_j \sum_{i=1}^n \|a_{ij}\|$, (dvs "max kolumnsumma"),
2-normen	$\|\|A\|\|_2 = \max_j \sqrt{\|\lambda_j(A^T A)\|}$, (där $\lambda_j(A^T A)$ är egenvärde j till matrisen $A^T A$),
max-normen	$\|\|A\|\|_\infty = \max_i \sum_{j=1}^n \|a_{ij}\|$, (dvs "max radsumma").

Matrisnormerna definierade enligt (1) uppfyller automatiskt punkterna 1-4 ovan. Efter multiplikation med $||\boldsymbol{x}||$ på båda sidor ger dessutom (1) denna viktiga olikhet:

$$\boxed{||A\boldsymbol{x}|| \leq ||A||\,||\boldsymbol{x}||}. \tag{2}$$