Cvičení 3 - zadání a řešení úloh

Základy numerické matematiky - NMNM201 Verze z 16. října 2024

---

1 LU rozklad

1.1 Opakování a LU rozklad v MATLABu

LU rozklad je maticový zápis Gaussovy eliminace, tedy převodu matice do odstupňovaného tvaru. Označíme-li E_k matici elementární transformace (vynásobení řádku nenulovým číslem, prohození dvou řádků, nebo přičtení násobku řádku k jinému), pak lze Gaussovu eliminaci schematicky zapsat jako

$$A \quad \to \quad A^{(1)} = E_1 A \quad \to \quad \dots \quad \to \quad U = A^{(n-1)} = \overbrace{E_{n-1} \dots E_1}^{L^{-1}} A, \tag{1}$$

kde U je matice v odstupňovaném tvaru a platí $LU = A$. (Připomínáme, že matice elementárních úprav jsou regulární.)

Pro tzv. silně regulární matice nepotřebujeme v Gaussově eliminaci prohazovat řádky. Pak je matice L dolní trojúhelníková, což vysvětluje název rozkladu (Lower–Upper). To, jestli je matice silně regulární, ale bohužel obvykle dopředu nevíme.

Úloha 1. *Pro vestavěnou funkci* `[L,U] = lu(A)`*:*

- *Zvolte* `A = wilkinson(6)` *a spočítejte její LU rozklad. Ověřte, že U je horní trojúhelníková. Je L dolní trojúhelníková?*
- *Prohoďte u matice A třetí a čtvrtý sloupec a zopakujte.*
- *(navíc) nastudujte si v nápovědě variantu volání vestavěné funkce jako* `[L,U,P] = lu(A)`*. Čemu odpovídá matice P? Jak se změní vlastnosti L oproti* `[L,U] = lu(A)`*?*

Řešení. V prvním případě L dolní trojúhelníková není.

Pro `[L,U] = lu(A(:,[1 2 4 3 5 6]))` už L dolní trojúhelníková je.

`[L,U,P] = lu(A)` odpovídá tomu, že elementární operace prohození řádků jsou uloženy v matici P, matice L je pak vždy dolní trojúhelníková. □

1.2 Řídké matice a LU rozklad

V řadě aplikací dostáváme matice, které mají na většině pozic nuly. Těmto maticím říkáme *řídké* a obvykle u nich ukládáme pouze nenulové prvky. (Matice, které nejsou řídké, označujeme jako *husté*.)

Úloha 2.
- *Vytvořte řídkou matici příkazem* `B = gallery('poisson',5);` *Jaký má rozměr? Jaký je rozdíl zadáte-li do příkazové řádky bez středníku* `A` *(z předchozí úlohy), resp.* `B`*?*
- *Pomocí příkazu* `whos` *srovnejte paměťové nároky na uložení řídké matice* `B = gallery('poisson',50);` *a náhodné (husté) matice stejného řádu.*
- *(navíc) Jakou největší matici lze pomocí* `B = gallery('poisson',N);` *sestavit a uložit? Odhadněte na základě několika voleb parametru* `N` *a ověřte.*

Řešení.
- U řídkých matic MATLAB vypisuje pozici nenulového prvku a jeho hodnotu; husté matice vypisuje po řádcích.

- Je to necelých 250 kB vs skoro 48 MB.

□

Úloha 3. *Zkonstruujte matici* `B = gallery('poisson',50);` *a matici C, která vznikne přidáním sloupcového vektoru samých jedniček (vhodné délky) zleva k matici B. Proveďte pro obě LU rozklad. Pomocí příkazu* `whos` *srovnejte paměťové nároky pro matice B a C a pro jejich LU faktory. Pomocí příkazu* `spy` *srovnejte zaplnění (tj. počet nenulových prvků) faktorů L pro matice B a C.*

Řešení.
```
C = [ones(2500,1),B];
[LB,UB] = lu(B);
[LC,UC] = lu(C);
spy(LB);
spy(LC);
```
LC je plná dolní trojúhelníková matice s 3 126 250 nenulovými prvky, LB jich má jen 125 049. I tak je to víc než desetkrát více, než kolik je nenul v matici B! □

Úloha 4. *(navíc) Zkonstruujte matici* `B = gallery('poisson',500);` *a spočítejte její LU rozklad. Jak dlouho to trvá? Kolik paměti faktory L a U zabírají?*

2 Stacionární iterační metody

Přímé metody (jako například LU rozklad) pro řešení soustav lineárních rovnic $Ax = b$ s regulární maticí, po nějaké době výpočtu, vydají jedno numerické řešení. Myšlenka iteračních metod je principálně odlišná, spočívá v konstrukci *posloupnosti* aproximací (přibližných řešení) $x_0, x_1, x_2, \ldots$, jež by se měla přibližovat skutečnému řešení x. Výhodou iteračních metod je, že (nějakou) aproximaci získáváme v každé iteraci, tj. kdykoli zastavíme výpočet.

2.1 Klasické iterační metody

Klasické iterační metody jsou založeny na štěpení matice soustavy $A = M - N$, kde matice M je regulární a snadno invertovatelná. Dosazením do vztahu $Ax = b$ postupně dostáváme

$$\begin{aligned} (M-N)x &= b \\ Mx &= Nx + b \\ x &= M^{-1}Nx + M^{-1}b. \end{aligned}$$

Je-li dána počáteční aproximace řešení x_0, můžeme definovat iterační proces

$$x_k = M^{-1}Nx_{k-1} + M^{-1}b.$$

Pro analýzu stacionárních iteračních metod je důležitý následující vztah mezi chybami dvou následujících přibližných řešení x_{k-1} a x_k:

$$x - x_k = M^{-1}N\,(x - x_{k-1}) = (I - M^{-1}A)\,(x - x_{k-1}) = (I - M^{-1}A)^k\,(x - x_0)\,.$$

2.2 Příklady klasických iteračních metod

Klasické iterační metody jsou založeny na štěpení ve tvaru $A = D - L - U$, kde D je hlavní diagonála, $-L$ je striktně dolní trojúhelník matice A a $-U$ je striktně horní trojúhelník matice A. Jednotlivé metody pak lze odvodit z rovnice

$$(D - L - U)x = b.$$

- V Jacobiho metodě volíme $M = D$, $N = L + U$, a je tedy definována iterací

$$Dx_k = Lx_{k-1} + Ux_{k-1} + b,$$

- Gauss–Seidelova metoda používá $M = D - L$, $N = U$ a je definována jako

$$Dx_k = Lx_k + Ux_{k-1} + b.$$

2.3 Asymptotická konvergence

Z přednášky víme, že metoda je konvergentní právě tehdy, když $\rho(M^{-1}N) < 1$. Tím myslíme, že pro libovolný počáteční vektor chyba $x - x_k$ konverguje k nulovému vektoru.

Úloha 5. *Pro matici*

$$A = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{bmatrix} \qquad \textit{případně (navíc)} \quad A_2 = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & 2 & -3 \end{bmatrix}$$

odvoďte matici $M^{-1}N$ z Jacobiho a Gauss–Seidelovy metody a rozhodněte, zda metody budou konvergentní, nebo ne. Pro výpočty inverzí matice a vlastních čísel můžete využít MATLAB.

Řešení.

$$(M^{-1}N)_{\text{Jacobi}} = -\begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix} \qquad \text{sp}((M^{-1}N)_{\text{Jacobi}}) = \{-1, 1/2, 1/2\},$$

takže Jacobiho metoda konvergentní není, zatímco Gauss–Seidel ano, protože

$$(M^{-1}N)_{\text{GS}} = \begin{bmatrix} 0 & -1/2 & -1/2 \\ 0 & 1/4 & -1/4 \\ 0 & 1/8 & 3/8 \end{bmatrix} \qquad \text{sp}((M^{-1}N)_{\text{GS}}) = \{0, 1/16 \cdot (5 \pm i\sqrt{7})\}.$$

□

2.4 Přechodový jev

Zatímco vlastnost $\rho(M^{-1}N) < 1$ zaručuje, že chyba $x - x_k$ konverguje k nulovému vektoru, což také zaručuje $\|x - x_k\|_* \to^{k\to\infty} 0$ pro libovolnou vektorovou normu, pro popis $\|x - x_k\|_*$ v úvodních iteracích (pro malé k) nemusí být $\rho(M^{-1}N)$ vypovídající.

Situaci, kdy chyba $\|x - x_k\|_*$ roste před tím, než dosáhne asymptotického chování odpovídajícímu $(\rho(M^{-1}N))^k$, říkáme *přechodový jev.*

Úloha 6. *Naprogramujte Jacobiho a Gauss-Seidelovu metodu (doplňte předpřipravené skripty* `jacobi_errors.m` *pro Jacobiho a* `gs_errors.m` *pro Gauss-Seidelovu metodu) a vyzkoušejte jejich chování pomocí skriptu* `iteracni_metody_jac_gs.m`*.*

[Hint: Nastudujte si v nápovědě MATLABu funkce `diag` *(z matice „vyzobne“ diagonálu jako vektor, z vektoru vytvoří diagonální matici),* `tril` *a* `triu`*.]*

I na základě pozorování odpovězte na následující otázky, případně odkažte na konkrétní úlohu ze skriptu `iteracni_metody_jac_gs.m`*:*

- *Konverguje-li metoda například v Euklidovské normě, musí konvergovat i v jiných vektorových normách?*
- *Kdy máme zaručenu monotonní konvergenci (například v Euklidovské normě)?*

- *Souvisí přítomnost přechodového jevu (tj. jevu, kdy chyba na začátku výpočtu nejprve roste) s velikostí maticových norem či spektrálního poloměru iterační matice?*

- *Lze v plné obecnosti vzájemně porovnat Jacobiho a Gauss-Seidelovu metodu? (Porovnáním máme na mysli výpovědi typu: Gauss-Seidelova metoda má vždy/nikdy rychlejší konvergenci než Jacobiho metoda. Jacobiho metoda konverguje pouze když/právě když Gauss-Seidelova metoda, atp.)*