UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Numerické metody řešení soustav lineárních rovnic

Václav Harapes

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2010/2011

ZADANÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Václav HARAPES**
Osobní číslo:	**I08048**
Studijní program:	**B2648 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Numerické metody pro řešení soustav lineárních rovnic**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je popis a naprogramování algoritmů pro řešení soustav lineárních rovnic. Soustavy rovnic lze řešit přímo (např. Gaussovou eliminací, Choleského, LU, LUP a QR rozklad), nebo iteračním způsobem. Známými příklady jsou stacionární metody (Jacobiho metoda. Gauss-Seidelova metoda a relaxační metody). V současné době jsou nejpoužívanější takzvané projektivní metody. Mezi mě patří mimo jiné následující metody: Metoda sdružených gradientů (conjugate gradient method - CGM), metoda bikonjugovaných gradientů (biconjugate gradient method - BiCGM) a metoda minimalizace reziduí (generalized minimal residual method - GMRES)

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***E. Vitásek: Numerické metody. SNTL, Praha 1987.**
***M. Fiedler: Speciální matice a jejich použití v numerické matematice, SNTL, Praha, 1981 (vybrané části)**

Vedoucí bakalářské práce: **RNDr. Josef Rak**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

prof. Ing. Simeon Karamasov, Dr.
děkan

L.S.

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Cerhýnkách dne 8. srpna 2011 Václav Harapes

Poděkování

V první řadě bych chtěl poděkovat vedoucímu mé bakalářské práce, RNDr. Josefu Rakovi, za jeho cenné připomínky, a čas strávený konzultacemi. Jako dalšímu bych rád poděkoval svému otci, který mi po celou dobu mého studia poskytoval materiální i psychickou podporu. V neposlední řadě děkuji i všem svým přátelům, kteří mi byli po boku v dobrých i špatných chvílích mého života i studia.

Anotace

Tato práce se zaměřuje na různé algoritmy pro řešení soustav lineárních rovnic. Cílem práce je porovnat jednotlivé algoritmy z hlediska typů úloh, které mohou řešit a také z hlediska nároků na paměť a výpočetní čas.

Klíčová slova

soustavy lineárních rovnic, přímé metody, Gaussova eliminace, Choleského rozklad, LU rozklad, LUP rozklad, QR rozklad, iterační metody, stacionární metody, Jacobiho metoda, Gauss-Seidelova metoda, projektivní metody, CGM, metoda sdružených gradientů, BiCGM, metoda bikonjugovaných gradientů, GMRES, metoda minimalizace reziduí

Title

Numerical Methods for Solving Systems of Linear Equations

Annotation

This work focuses on the different algorithms for solving systems of linear equations. The aim is to compare different algorithms in terms of types of jobs that can solve and also in terms of demands on memory and time.

Keywords

linear equations systems, direct methods, Gauss elimination, Cholesky decomposition, LU decomposition, LUP decomposition, QR decomposition, iterative methods, stationary methods, Jacobi method, Gauss-Seidel method, projection methods, CGM, conjugate gradient method, BiCGM, biconjugate gradient method, GMRES, generalized minimal residual method

Obsah

Základní pojmy

Prvek matice – a_{ij}, případně $a_{i,j}$
Prvek ležící na í-tém řádku v j-tém sloupci matice $\boldsymbol{A}$.

Prvek vektoru – b_i
I-tý prvek vektoru $\boldsymbol{b}$.

Dolní trojúhelníková matice – L
Matice, jejíž prvky nad hlavní diagonálou jsou rovny nule ($l_{ij} = 0$, pro $i < j$).

Dolní striktně trojúhelníková matice – L^*
Podobně jako dolní trojúhelníková matice, ale má nulové prvky i na hlavní diagonále ($l_{ij} = 0$, pro $i \leq j$).

Jednotková matice
Jednotková matice I má prvky na hlavní diagonále rovny jedné, a všechny ostatní prvky nulové. Platí podmínky

- $i_{ii} = 1$ pro $i = 0,1, \dots, \min(m-1, n-1)$ a zároveň
- $i_{ij} = 0$ pro $i \neq j$, $i = 0, \dots, n-1$, $j = 0, \dots, m-1$).

Horní trojúhelníková matice – U
Matice, která má všechny prvky pod dolní diagonálou rovny nule ($u_{ij} = 0$, pro $i > j$).

Horní striktně trojúhelníková matice – U^*
Podobně jako horní trojúhelníková matice, ale navíc ještě s nulovými prvky na diagonále ($u_{ij} = 0$, pro $i \geq j$).

Hodnost matice
Počet vzájemně lineárně nezávislých řádků, případně sloupců, matice.

V trojúhelníkové matice je to počet nenulových prvků na hlavní diagonále.

Numerická stabilita
Numericky nestabilní algoritmus se vyznačuje tím, že relativně malé chyby v jednotlivých krocích způsobí tak velké chyby ve výsledku, že se nedá považovat za přesný. Numericky stabilní algoritmus je pravý opak.

Ortogonální matice
Ortogonální matice je taková matice, jejíž inverzní i transponovaná matice jsou shodné, platí $\boldsymbol{A} \cdot \boldsymbol{A}^T = \boldsymbol{I}$.

Regulární matice
Čtvercová matice, jejíž hodnost je rovna počtu řádků, respektive sloupců.

Řídká matice
Řídká matice je taková matice, která má většinu prvků rovnou 0. Udává se také definice, že řídká matice má maximálně 5 % nenulových prvků.

Singulární matice
Matice, jejíž hodnost je menší než menší z počtu řádků a sloupců této matice.

Spektrální poloměr matice
Spektrální poloměr matice je v absolutní hodnotě nejvyšší ze všech vlastních čísel dané matice. Značí se symbolem $\varrho(\boldsymbol{A})$.

Symetrická pozitivně definitní matice
Matice, která je symetrická ($\boldsymbol{A} = \boldsymbol{A}^T$), a zároveň všechny její vlastní čísla jsou kladná.

Vlastní číslo matice
Vlastní číslo matice je takové číslo λ, pro které platí, že $\boldsymbol{A} \cdot \boldsymbol{u} = \lambda \cdot \boldsymbol{u}$, kde $\boldsymbol{u}$ je tzv. vlastní vektor matice a λ je vlastní číslo přidružené k tomuto vektoru.

Vlastní vektor matice
Vlastní vektor matice $\boldsymbol{A}$ je takový vektor, jehož směr se po aplikaci transformace $\boldsymbol{A}$ nezmění.

Seznam obrázků

Seznam zkratek

OOP	objektově orientované programování
SPD	symetrická, pozitivně definitní (matice)

Úvod

Soustavy lineárních rovnice se vyskytují v nejrůznějších oborech matematiky. Typicky je lze nalézt lineární algebře, analytické geometrii, fyzice, ekonomii, chemii a v některých oblastech operačního výzkumu, např. v lineárním programování.

Obecným požadavkem všech algoritmů pro řešení soustav lineárních rovnic je, aby matice koeficientů byla regulérní.

Při volbě konkrétního algoritmu řešení soustavy je nutné přihlédnout k řadě faktorů, např.:

- typ matice (obecná, symetrická, pozitivně definitní, pásová, atd.),
- předpodmíněnost soustavy,
- počet rovnic,
- kolik se bude řešit rovnic se stejnou maticí,
- požadovaná přesnost.

Jednotlivé algoritmy se liší v

- paměťové náročnosti,
- časové složitosti,
- přesnosti.

Cílem mé bakalářské práce je srovnat jednotlivé metody pro řešení soustav lineárních rovnic z hlediska jejich použitelnosti pro různé typy úloh a jejich časové a paměťové náročnosti.

Vzhledem k mému oboru nebudu podrobně probírat teorii jednotlivých metod. Spíše bych se chtěl zaměřit na jejich aplikace a konkrétní implementaci pomocí programovacího jazyka – nebudu probírat všechny možné typy předpodmiňovacích matic, rovinných rotací, atd.

1 Přímé metody

Mezi přímé metody pro výpočet soustav lineárních rovnic lze zařadit například Gaussovu eliminační metodu a její varianty. Nejznámějšími variantami Gaussovy eliminační metody jsou např. trojúhelníkový rozklad, Choleského rozklad a další.

Díky jednoduchosti těchto algoritmů jsou vhodné pro ruční počítání.

Velkou nevýhodou těchto metod je jejich rychlost, obecně dosahují asymptotické složitosti $O(n^3)$. Na druhou stranu tyto algoritmy mívají nízké paměťové nároky, většina algoritmů si vystačí jen s vlastní maticí a několika málo uloženými výsledky výpočtů (VITÁSEK, 1987). Další výhodou je, že upravenou matici soustavy lze použít opakovaně pro různé tvary vektoru pravých stran.

1.1 Gaussova eliminace

Gaussova eliminace je algoritmus, který danou matici převede na horní trojúhelníkovou matici, případně diagonální matici.

Mezi přednosti této metody patří její univerzálnost. Metodou lze vypočítat

- soustavu lineárních rovnic,
- inverzní matici,
- determinant.

Algoritmus

Mějme maticově zapsanou soustavu rovnic:

$$\boldsymbol{A}^{(0)} = \left(\begin{array}{cccc|c} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} & b_1^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & a_{23}^{(0)} & \cdots & a_{2n}^{(0)} & b_2^{(0)} \\ a_{31}^{(0)} & a_{32}^{(0)} & a_{33}^{(0)} & \cdots & a_{3n}^{(0)} & b_3^{(0)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & a_{n3}^{(0)} & \cdots & a_{nn}^{(0)} & b_n^{(0)} \end{array}\right).$$

Tuto matici se snažíme převést na matici

$$\boldsymbol{A}^{(n-1)} = \left(\begin{array}{cccc|c} a_{0,0}^{(n-1)} & a_{0,1}^{(n-1)} & a_{0,2}^{(n-1)} & \cdots & a_{0,n-1}^{(n-1)} & b_0^{(n-1)} \\ 0 & a_{1,1}^{(n-1)} & a_{1,2}^{(n-1)} & \cdots & a_{1,n-1}^{(n-1)} & b_1^{(n-1)} \\ 0 & 0 & a_{2,2}^{(n-1)} & \cdots & a_{2,n-1}^{(n-1)} & b_2^{(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1,n-1}^{(n-1)} & b_{n-1}^{(n-1)} \end{array}\right),$$

a to pomocí takových úprav, aby se nezměnilo řešení soustavy.

Celý algoritmus v pseudokódu vypadá následujícím způsobem:

1. krok: pro $k = 0, \dots, n-1$
   2. krok: když $a_{kk}^{(k)} = 0$
      3. krok: pro $l = k+1, \dots, n-1$
         4. krok: když $a_{lk}^{(k)} \neq 0$
            prohoď řádky k a l matice $\boldsymbol{A}$
            prohoď prvky k a l vektoru $\boldsymbol{b}$
   5. krok: když $a_{kk}^{(k)} = 0$
      metoda selže z důvodu singulárnosti matice
   6. krok: pro $i = k+1, \dots, n-1$
      7. krok: pro $j = i, \dots, n-1$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \cdot a_{kj}^{(k)}$$

   8. krok: $b_i^{(k+1)} = b_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \cdot b_k^{(k)}$.

Pokud by byl člen $a_{kk}^{(k)}$ v některém kroku nulový, tak použijeme operací záměna řádků, případně sloupců. Ačkoli obě operace jsou ekvivalentní, častěji se používá záměna řádků. Při operaci nestačí prohodit příslušné řádky matice $\boldsymbol{A}$, ale musí být prohozeny i příslušné prvky vektoru $\boldsymbol{b}$. Pokud se nám nepodaří ani tak získat nenulové $a_{kk}^{(k)}$, znamená to, že matice je singulární a tím pádem má daná soustava rovnic nekonečně mnoho řešení (KOLDA, a další, 2004), (VITÁSEK, 1987).

A nakonec postupně od konce dopočítáme hodnoty vektoru $\boldsymbol{x}$ pomocí algoritmu:

1. krok: pro $i = n-1, \dots, 0$

$$x_i = \frac{1}{a_{ii}^{(i-1)}} \cdot \left(a_{i,n+1}^{(i-1)} - \sum_{j=i+1}^{n} \left(a_{ij}^{(i-1)} \cdot x_j \right) \right).$$

Příklad

Vzhledem k tomu, že je tato metoda vhodná i pro ruční počítání, tak bych rád uvedl i kompletní příklad.

Mějme soustavu rovnic

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 9, \\ x_1 + x_2 + x_3 &= 3, \\ 2x_1 - x_2 - x_3 &= 6. \end{aligned}$$

Případně může být vyjádřena maticově jako:

$$\begin{pmatrix} 1 & 2 & -1 \\ 1 & 1 & 1 \\ 2 & -1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 9 \\ 3 \\ 6 \end{pmatrix}.$$

Následujícími kroky převedeme matici $\boldsymbol{A}$ na horní trojúhelníkovou matici.

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 9 \\ 1 & 1 & 1 & 3 \\ 2 & -1 & -1 & 6 \end{array}\right)$$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 9 \\ 0 & -1 & 2 & -6 \\ 0 & -5 & 1 & -12 \end{array}\right)$$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 9 \\ 0 & -1 & 2 & -6 \\ 0 & 0 & -9 & 18 \end{array}\right)$$

Tuto soustavu můžeme též zapsat jako

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 9 \\ -x_2 + 2x_3 &= -6 \\ -9x_3 &= 18 \end{aligned}$$

V posledním kroku postupně od konce vypočítáváme hodnoty koeficientů vektoru $\boldsymbol{x}$.

$$\begin{aligned} -9x_3 &= 18 \\ \underline{\underline{x_3 &= -2}} \end{aligned}$$

$$\begin{aligned} -x_2 + 2x_3 &= -6 \\ -x_2 - 4 &= -6 \\ -x_2 &= -2 \\ \underline{\underline{x_2 &= 2}} \end{aligned}$$

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 9 \\ x_1 + 4 + 2 &= 9 \\ \underline{\underline{x_1 &= 3}} \end{aligned}$$

Výsledek můžeme zapsat jako

$$\boldsymbol{x} = \begin{pmatrix} 3 \\ 2 \\ -2 \end{pmatrix}.$$

Vlastnosti

Metoda má jedinou podmínku konvergence, a to regulárnost matice $\boldsymbol{A}$.

Výhodou metody je, že nepotřebuje žádné další místo v paměti, všechny operace se provádí přímo na matici $\boldsymbol{A}$.

Nevýhodu této metody je asymptotická složitost $O(n^3)$ a nízká numerická stabilita.

1.2 Choleského rozklad

Choleského rozklad je první metoda trojúhelníkového rozkladu matic, kterou ve své práci popíši.

Metoda rozkládá matici $\boldsymbol{A}$ na součin

$$\boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{L^T}.$$

Jak lze z tohohle vzorce zpozorovat, tak metoda má oproti jiným menší nároky na paměť, stačí v paměti uložit jen jednu trojúhelníkovou matici. Bohužel z toho plyne i nevýhoda – algoritmus vyžaduje symetrickou, pozitivně definitní matici, tudíž ho nelze použít na libovolnou matici $\boldsymbol{A}$ (VITÁSEK, 1987), (WEISSTEIN, 2011).

Algoritmus

V prvním kroku se musí vypočítat koeficienty matice $\boldsymbol{L}$, např. pomocí následujícího algoritmu:

1. krok: pro $i = 0, \dots, n-1$

$l_{ii} = \sqrt{a_{ii} - \sum_{j=1}^{i-1} l_{ij}^2}$

2. krok: pro $j = i, \dots, n-1$

$l_{ij} = \frac{1}{l_{ii}} \cdot \left(a_{ij} - \sum_{k=1}^{j-1} l_{jk} \cdot l_{ik}\right).$

Jakmile podle těchto vzorců spočítáme koeficienty matice $\boldsymbol{L}$, můžeme už snadno dopočítat koeficienty vektoru $\boldsymbol{x}$, a to podle vztahů

$$\boldsymbol{L} \cdot \boldsymbol{y} = \boldsymbol{b} \text{ a}$$
$$\boldsymbol{L^T} \cdot \boldsymbol{x} = \boldsymbol{y},$$

což můžeme přepsat do následujícího algoritmu:

1. krok: pro $i = 0, \dots, n-1$

$y_i = \frac{1}{l_{ii}} \cdot \left(b_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j\right)$

2. krok: pro $i = n-1, \dots, 0.$

$x_i = \frac{1}{l_{ii}} \cdot \left(y_i - \sum_{j=i+1}^{n} l_{ji} \cdot x_j\right).$

Vlastnosti

Metoda má opět asymptotickou složitost $O(n^3)$, ale s relativně malou asymptotickou konstantou, protože vnitřní cykly mají malý počet opakování.

Co se týče nároků na paměť, algoritmu stačí ukládat do paměti pouze jednu trojúhelníkovou matici, což při vhodné implementaci může ušetřit téměř 50 % potřebné paměti.

Jak lze z předchozích dvou odstavců vyčíst, algoritmus je vcelku nenáročný, ale na druhou stranu umožňuje řešit pouze soustavy rovnic s SPD maticí $\boldsymbol{A}$.

1.3 LU rozklad

LU rozklad je další metoda, která je založena na trojúhelníkovém rozkladu, a to na takovém, že platí vztah

$$\boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}.$$

Na rozdíl od Choleského rozkladu si zde nevystačíme už s jedinou trojúhelníkovou maticí (VITÁSEK, 1987), (WEISSTEIN, 2011).

Algoritmus

Existují různé metody, jak rozložit matici $\boldsymbol{A}$ na matice $\boldsymbol{L}$ a $\boldsymbol{U}$. Nejčastěji se používá tzv. Croutova metoda, kterou lze vyjádřit algoritmem:

1. krok: pro $r = 0, \dots, n-1$
   2. krok: pro $i = 0, \dots, r-1$
      $u_{ir} = a_{ii} - \sum_{s=1}^{i-1}(l_{is} \cdot u_{sr})$
   3. krok: pro $i = r, \dots, n-1$
      $l_{ir} = \frac{1}{u_{rr}} \cdot \left(a_{ir} - \sum_{s=1}^{i-1} l_{is} \cdot u_{sr}\right)$.

Jakmile spočítáme koeficienty matic $\boldsymbol{L}$ a $\boldsymbol{U}$, tak už můžeme pokračovat v podstatě stejnými vzorci jako v Choleského dekompozici, a to:

$$\boldsymbol{L} \cdot \boldsymbol{y} = \boldsymbol{b} \text{ a}$$

$$\boldsymbol{U} \cdot \boldsymbol{x} = \boldsymbol{y},$$

resp.

1. krok: pro $i = 0, \dots, n-1$
   $y_i = \frac{1}{l_{ii}} \cdot \left(b_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j\right)$
2. krok: pro $i = n-1, \dots, 0$.
   $x_i = \frac{1}{l_{ii}} \cdot \left(y_i - \sum_{j=i+1}^{n} l_{ji} \cdot x_j\right)$.

Vlastnosti

Metoda má téměř stejné vlastnosti jako Choleského rozklad. Na rozdíl od něj pro konvergenci nevyžaduje SPD, ale na druhou stranu má vyšší paměťové nároky. Také je časově náročnější, musí se spočítat koeficienty dvou trojúhelníkových matic.

1.4 LUP rozklad

LUP rozklad je velmi podobný LU rozkladu, ale matici $\boldsymbol{A}$ dekomponujeme na matice $\boldsymbol{L}$, $\boldsymbol{U}$ a $\boldsymbol{P}$, tak, že platí

$$\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U},$$

kde $\boldsymbol{P}$ je tzv. permutační matice, která má v každém řádku a sloupci pouze jeden nenulový prvek. Zavedení matice $\boldsymbol{P}$ umožňuje vyhnout se dělení nulou a malými čísly, což zvyšuje numerickou stabilitu této metody (VITÁSEK, 1987), (WEISSTEIN, 2011)..

Mezi nejznámější algoritmy pro LUP rozklad patří

- Doolitlova metoda a
- Croutova metoda (kde $\boldsymbol{U} = \boldsymbol{I}$ – horní trojúhelníková matice je jednotková).

Algoritmus

Algoritmus je ve své další části velice podobný LU rozkladu, s tím rozdílem, že položky vektoru y se počítají ze vztahů:

$\boldsymbol{c} = \boldsymbol{P} \cdot \boldsymbol{b}$

1. krok: pro $i = 0, \ldots, i - 1$

$y_i = c_i - \sum_{j=1}^{i-1} l_{ij} \cdot y_j$.

·Vlastnosti

Metoda má téměř shodné vlastnosti jako LU rozklad, ale za cenu mírného zvýšení paměťových nároků (matici $\boldsymbol{P}$ lze uložit jako pole o rozměru n) poskytuje mnohem vyšší numerickou stabilitu, a tím i přesnost výsledků.

1.5 QR rozklad

Cílem této metody je rozložit matici $\boldsymbol{A}$ na ortogonální matici $\boldsymbol{Q}$ a horní trojúhelníkovou matici $\boldsymbol{R}$ tak, aby platilo:

$$\boldsymbol{A} = \boldsymbol{Q} \cdot \boldsymbol{R}.$$

Existuje několik algoritmů rozkladu, mezi nejznámější patří

- Gram-Schmidtova metoda,
- Householderova transformace,
- Givensova rotace.

Popis jednotlivých metod je mimo rozsah mé práce, bližší informace o jednotlivých metodách nalezne čtenář např. v (WEISSTEIN, 2011).

Poté co máme vypočítané matice $\boldsymbol{Q}$ a $\boldsymbol{R}$ můžeme už snadno vypočítat např. determinant matice $\boldsymbol{A}$, ale pro řešení soustavy lineárních rovnic použijeme následující substituci:

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \boldsymbol{Q} \cdot \boldsymbol{R} \cdot \boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \boldsymbol{R} \cdot \boldsymbol{x} = \boldsymbol{Q}^* \cdot \boldsymbol{b} \Leftrightarrow \boldsymbol{R} \cdot \boldsymbol{x} = \boldsymbol{c}.$$

Vzhledem k faktu, že $\boldsymbol{R}$ je horní trojúhelníková matice, můžeme složky vektoru $\boldsymbol{x}$ dopočítat podobně jako v Gaussově eliminační metodě pomocí vzorce:

1. krok: pro $i = n-1, \dots, 0$

$x_i = \frac{1}{r_{ii}} \cdot \left(c_i - \sum_{j=i+1}^{n} \left(r_{ij} \cdot x_j\right)\right)$ (GANDER, 1980), (WEISSTEIN, 2011).

Vlastnosti

Z hlediska časové náročnosti závisí rychlost algoritmu převážně na zvolené metodě dekompozice. Obecně lze říci, že časová složitost je přibližně $O(n^3)$.

Co se týče paměti, tak algoritmus potřebuje uložit v paměti navíc matice $\boldsymbol{Q}$ a $\boldsymbol{R}$.

Výhoda oproti Gaussově eliminační metodě je vyšší numerická stabilita, a také to, že existuje modifikace pro obdélníkové matice.

2 Stacionární iterační metody

Iterační metody se od přímých metod liší tím, že nedávají přesné výsledky po určitém počtu kroků. Namísto toho každý průběh iteračního algoritmu vezme výsledek z předchozího kroku, a ten zpřesní. Počet kroků závidí na požadované přesnosti, ale může být omezen i jinak – např. maximálním počtem iterací.

U iteračních metod musí být splněna tzv. podmínka konvergence.

Pokud není potřeba přesný výsledek, jsou iterační metody oproti přímým mnohem rychlejší.

Iterační metody pro řešení soustav lineárních rovnic jsou založeny na vztahu

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{B} \cdot \boldsymbol{x}^{(i)} + \boldsymbol{C} \cdot \boldsymbol{b}, \quad i = 0, 1, \ldots,$$

kde $\boldsymbol{B}$ a $\boldsymbol{C}$ jsou takové matice, že platí

$$\boldsymbol{B} + \boldsymbol{C} \cdot \boldsymbol{A} = \boldsymbol{I}.$$

Nutnou a postačující podmínkou konvergence je, aby platilo $\varrho(\boldsymbol{B}) < 1$, tedy aby spektrální poloměr matice $\boldsymbol{B}$ byl menší než 1.

Jednotlivé metody se liší pouze volbou matice $\boldsymbol{B}$ (VITÁSEK, 1987).

2.1 Jacobiho metoda

V této metodě volíme matice $\boldsymbol{B}$ a $\boldsymbol{C}$ následujícím způsobem:

$$\boldsymbol{B} = -\boldsymbol{D}^{-1} \cdot (\boldsymbol{L}^* + \boldsymbol{U}^*) \text{ a}$$

$$\boldsymbol{C} = \boldsymbol{D}^{-1}.$$

Algoritmus:

Vlastní výpočetní algoritmus můžeme zjednodušit pouze na následující krátký kus pseudokódu:

1. krok: pro $i = 0, 1, \ldots$
   2. krok: pro $j = 0, \ldots, n-1$
      $x_j^{(i+1)} = \frac{1}{a_{jj}} \cdot \left(b_j - \sum_{k=1}^{j-1} a_{jk} \cdot x_k^{(i)} - \sum_{k=j+1}^{n} a_{jk} \cdot x_k^{(i)} \right)$.

Vlastnosti

Metoda může být velmi rychlá i velmi pomalá, záleží, jakou požadujeme přesnost. Obecně se dá říci, že platí asymptotická složitost $O(n^3)$.

Metoda potřebuje v paměti udržovat složky vektoru $\boldsymbol{x}^{(i)}$, tak i $\boldsymbol{x}^{(i+1)}$, což je sice nevýhoda, ale nijak velká, vektory jsou oproti maticím nenáročné na paměť.

Požadavky na vstupní matici jsou jako u většiny ostatních metod – regulární matice $\boldsymbol{A}$. Dalším požadavkem je, aby byla i matice $\boldsymbol{D}$ regulární, nicméně pokud je matice $\boldsymbol{A}$

regulární, lze toto zařídit vhodnou permutací řádků, respektive sloupců matice $\boldsymbol{A}$ (VITÁSEK, 1987).

2.2 Gauss-Seidelova metoda

Tato metoda je velmi podobná Jacobiho metodě. Jedinou výhodou je, že si nemusíme pamatovat všechny složky vektoru $\boldsymbol{x}^{(i)}$. V této metodě volíme matice $\boldsymbol{B}$ a $\boldsymbol{C}$ takto:

$$\boldsymbol{B} = (-\boldsymbol{D} + \boldsymbol{L}^*)^{-1} \cdot \boldsymbol{U}^* \text{ a}$$

$$\boldsymbol{C} = (\boldsymbol{D} + \boldsymbol{L}^*)^{-1}.$$

Algoritmus:

Vlastní algoritmus je opět jednoduchý, zapsaný v pseudokódu vypadá následujícím způsobem:

1. krok: pro $i = 0, 1, \ldots$
   2. krok pro $j = 0, \ldots, n-1$
   $x_j^{(i+1)} = \frac{1}{a_{jj}} \cdot \left(b_j - \sum_{k=1}^{j-1} a_{jk} \cdot x_k^{(i+1)} - \sum_{k=j+1}^{n} a_{jk} \cdot x_k^{(i)} \right)$.

Vlastnosti

Metoda má téměř stejné vlastnosti jako Jacobiho metoda, jediná výhoda je o něco menší paměťová náročnost, stačí si pamatovat jen jeden vektor $\boldsymbol{x}$ (VITÁSEK, 1987).

2.3 Superrelaxační metoda

Tato metoda se řadí mezi relaxační metody. Vychází z předchozí Gauss-Seidelovy metody. Jako matice $\boldsymbol{B}$ a $\boldsymbol{C}$ se volí

$$\boldsymbol{B} = (\boldsymbol{D} + \omega \cdot \boldsymbol{L}^*)^{-1} \cdot [-\omega \cdot \boldsymbol{U}^* + (1 - \omega) \cdot \boldsymbol{D}] \text{ a}$$

$$\boldsymbol{C} = (\boldsymbol{D} + \boldsymbol{L}^*)^{-1}.$$

V rovnicích se nově objevuje parametr ω. Parametr ω je tzv. relaxační faktor a volí se z rozsahu $(0; 2)$. Jeho přesná hodnota závidí na konkrétním typu matice $\boldsymbol{A}$ a jejím spektrálním poloměru. Parametr ω zvyšuje rychlost konvergence metody (VITÁSEK, 1987).

Algoritmus

Vlastní algoritmus je opět jednoduchý:

1. krok: pro $i = 0, 1, \ldots$
   2. krok: pro $j = 0, \ldots, n-1$
   $x_j^{(i+1)} = \frac{\omega}{a_{jj}} \cdot \left(b_j - \sum_{k=1}^{j-1} a_{jk} \cdot x_k^{(i+1)} - \sum_{k=j+1}^{n} a_{jk} \cdot x_k^{(i)} \right) + (1 - \omega) \cdot x_j^{(i)}$.

Vlastnosti

Superrelaxační metoda je pouze modifikací Gauss-Seidelovy metody, a tudíž má téměř stejné vlastnosti, s výjimkou rychlosti konvergence, která je mnohem vyšší. Nicméně k časové složitosti by bylo nutné ještě přičíst výpočet faktoru ω.

3 Projektivní metody

Největší nevýhoda všech předchozích metod je, že ke své funkci potřebují operace nad maticemi. Většina algoritmů, jako např. sčítání a odčítání matic, má nepříznivou asymptotickou složitost $O(n^2)$. Násobení matic dle definice má dokonce asymptotickou složitost $O(n^3)$. Ačkoli pro násobení existují i rychlejší algoritmy (Strassenův algoritmus s časovou složitostí přibližně $O(n^{2,81})$ a nejrychlejší zatím známý Coppersmith-Winogradův algoritmus se složitostí $O(n^{2,37})$) (HRIC), pro lineární systémy o řádově stovkách rovnic by byly klasické algoritmy stále nepoužitelné.

Proto bych chtěl se v této části práce zaměřit hlavně na metody využívající hledání řešení v Krylovově podprostoru. Krylovův podprostor se dá vyjádřit jako množina vektorů:

$$K_r(\boldsymbol{A}, \boldsymbol{b}) = \{\boldsymbol{b}, \boldsymbol{A} \cdot \boldsymbol{b}, \boldsymbol{A}^2 \cdot \boldsymbol{b}, \dots, \boldsymbol{A}^{r-1} \cdot \boldsymbol{b}\}.$$

Výhoda při využití Krylovova podprostoru je vyhnutí se násobení matic – násobí se pouze vektory s maticemi, což je oproti násobení dvou matic podstatně rychlejší operace (složitost $O(n^2)$) (BARRET, a další, 1994).

3.1 CGM – metoda sdružených gradientů

Metoda patří mezi jednu z nejjednodušších, ale bohužel je v základní formě použitelná pouze pro symetrické pozitivně definitní matice. Nicméně existují metody, které libovolnou regulární matici dokáží převést na symetrickou pozitivně definitní matici. Nejjednodušší způsob je převést rovnici $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$ na $\boldsymbol{A}^T \cdot \boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{A}^T \cdot \boldsymbol{b}$. Bohužel tento postup má nevýhodu v tom, že rychlost konvergence metody závidí na koeficientech matice $\boldsymbol{A}$ (BARRET, a další, 1994).

Pokud $\boldsymbol{A}$ je SPD, potom řešení soustavy rovnic je jediným bodem, ve kterém má funkce

$$F(x) = \frac{1}{2} \cdot x^T \cdot \boldsymbol{A} \cdot x - x^T \cdot b$$

minimum. Tato metoda se snaží vyhledat toto minimum (ČERNÁ, 2010).

Zda už je výsledek dostatečně přesný lze poznat z vektoru reziduí $\boldsymbol{r}$. Platí, že čím menší reziduum (resp. jeho norma), tím je výsledek přesnější. Pokud $\|\boldsymbol{r}\|_2 = 0$, pak je výsledek přesný. Tato poučka platí i u následujících metod (BARRET, a další, 1994).

Algoritmus

Na začátku je potřeba předat algoritmu matici $\boldsymbol{A}$, vektor $\boldsymbol{b}$ a vektor $\boldsymbol{x}$ – odhad řešení, který nejčastěji bývá nulový. Následuje popis algoritmu:

$\boldsymbol{v}^{(0)} = \boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}^{(0)}$

1. krok: pro $i = 0,1,\ldots$

$\alpha = \frac{\boldsymbol{v}^{(i)T} \cdot \boldsymbol{r}^{(i)}}{\boldsymbol{v}^{(i)T} \cdot \boldsymbol{A} \cdot \boldsymbol{v}^{(i)}}$

$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \alpha \cdot \boldsymbol{v}^{(i)}$

$\boldsymbol{r}^{(i+1)} = \boldsymbol{r}^{(i)} + \alpha \cdot \boldsymbol{A} \cdot \boldsymbol{v}^{(i)}$

zkontroluji $\boldsymbol{r}^{(i+1)}$, a pokud už je dostatečně malé, mohu algoritmus ukončit

$\beta = -\frac{\boldsymbol{v}^{(i)T} \cdot \boldsymbol{A} \cdot \boldsymbol{r}^{(i+1)}}{\boldsymbol{v}^{(i)T} \cdot \boldsymbol{A} \cdot \boldsymbol{v}^{(i)}}$

$\boldsymbol{v}^{(i+1)} = \boldsymbol{r}^{(i+1)} + \beta \cdot \boldsymbol{v}^{(i)}$.

Vlastnosti

Algoritmus je velmi rychlý, lze dokázat, že po n krocích vede k přesnému řešení (pokud se nedopouštíme zaokrouhlovacích chyb). Z toho vyplývá výhodná asymptotická složitost $O(n^3)$.

Z hlediska paměťové náročnosti je potřeba navíc uchovávat v paměti vektory $\boldsymbol{r}$ a $\boldsymbol{v}$ o velikosti n.

Další nevýhoda je, že metoda potřebuje SPD matici $\boldsymbol{A}$, jinak nezkonverguje.

3.2 BiCGM – metoda bikonjugovaných gradientů

BiCGM je jeden z odvozených algoritmů od metody sdružených gradientů. Výhodou je, že umí řešit i nesymetrické lineární systémy, ale pro určité kombinace koeficientů matic může selhat. Tomu se lze v určitých případech vyhnout restartem metody těsně před krokem, ve kterém selže. V ostatních případech je potřeba sáhnout po jiných metodách – např. GMRES (BARRET, a další, 1994).

V případě symetrických soustav dává stejné výsledky jako CGM, ale za cenu zhruba dvakrát vyšší asymptotické konstanty (BARRET, a další, 1994).

Tato metoda využívá předpodmiňovací matici. Nejčastěji se používá velmi jednoduchá Jacobiho předpodmiňovací matice. Platí, že

$$\boldsymbol{M} = \boldsymbol{D}^{-1}.$$

Vypočítat inverzní matici k diagonální matici je snadné, protože stačí spočítat převrácené hodnoty čísel k číslům nacházejících se na diagonále, neboli

1. krok: pro $i = 0, \ldots, n-1$

$m_{ii} = d_{ii}^{-1}$.

Algoritmus

Algoritmus pro svou správnou funkci potřebuje stejné parametry jako metoda CGM. Pseudokód má následující podobu:

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}^{(0)}$
$\boldsymbol{r}^{\sim(0)} = \boldsymbol{r}^{(0)}$ ($\boldsymbol{r}^{\sim(0)}$ může být zvoleno i jinak)
1. krok: pro $i = 0,1,\ldots$
vyřeš $\boldsymbol{M} \cdot \boldsymbol{z}^{(i-1)} = \boldsymbol{r}^{(i-1)}$
vyřeš $\boldsymbol{M}^T \cdot \boldsymbol{z}^{\sim(i-1)} = \boldsymbol{r}^{\sim(i-1)}$
$\varrho_{i-1} = \boldsymbol{z}^{(i-1)T} \cdot \boldsymbol{r}^{\sim(i-1)}$
2. krok: když $\varrho_{i-1} = 0$
metoda selže
3. krok: když $i = 1$
$\boldsymbol{p}^{(i)} = \boldsymbol{z}^{(i-1)}$
$\boldsymbol{p}^{\sim(i)} = \boldsymbol{z}^{\sim(i-1)}$
4. krok: jinak
$\beta = \frac{\varrho_{i-1}}{\varrho_{i-2}}$
$\boldsymbol{p}^{(i)} = \boldsymbol{z}^{(i-1)} + \beta \cdot \boldsymbol{p}^{(i-1)}$
$\boldsymbol{p}^{\sim(i)} = z^{\sim(i-1)} + \beta \cdot p^{\sim(i-1)}$
$\boldsymbol{q}^{(i)} = \boldsymbol{A} \cdot \boldsymbol{p}^{(i)}$
$\boldsymbol{q}^{\sim(i)} = \boldsymbol{A}^T \cdot \boldsymbol{p}^{\sim(i)}$
$\alpha = \frac{\varrho_{i-1}}{\boldsymbol{p}^{\sim(i)T} \cdot \boldsymbol{q}^{(i)}}$
$\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i-1)} + \alpha \cdot \boldsymbol{p}^{(i)}$
$\boldsymbol{r}^{(i)} = \boldsymbol{r}^{(i-1)} + \alpha \cdot \boldsymbol{q}^{(i)}$
$\boldsymbol{r}^{\sim(i)} = \boldsymbol{r}^{\sim(i-1)} + \alpha \cdot \boldsymbol{q}^{\sim(i)}$
zkontroluji $\boldsymbol{r}^{(i)}$, a pokud už je dostatečně malé, mohu algoritmus ukončit.

V této a následující metodě se objevuje funkce nazvaná „vyřeš $\boldsymbol{M} \cdot \boldsymbol{x} = \boldsymbol{b}$“. Pro diagonální předpodmiňovací matici $\boldsymbol{M}$ se jedná o následující algoritmus:

1. krok: pro $i = 1, \ldots, n$
$x_i = b_i \cdot m_{ii}$.

Pro jiné předpodmiňovací matice je algoritmus jiný.

Vlastnosti

Metoda má oproti CGM téměř dvakrát větší asymptotickou konstantu. Tento problém se dá z velké části potlačit na více procesorových systémech, protože hodnoty jednotlivých dvojic vektorů (např. $\boldsymbol{x}^{(i)}$ a $\boldsymbol{x}^{\sim(i)}$) lze počítat paralelně.

Stejně tak paměťová náročnost je oproti CGM mnohem vyšší.

3.3 GMRES – metoda minimalizace reziduí

GMRES je další z projektivních metod, kterou bych chtěl popsat ve své bakalářské práci. Metoda je relativně nová, byla objevena Y. Saadem and M. H. Schultzem v roce 1986. Existují 2 verze tohoto algoritmu – restartovaný a nerestartovaný. Níže uvedený algoritmus je restartovaný, ale volbou dostatečně velkého m ho lze snadno převést na nerestartovaný. Restartovaná verze se používá z důvodu snížení paměťových nároků, ale pokud je koeficient m příliš malý, pak metoda nemusí zkonvergovat. Je dokázáno, že metoda zkonvertuje v méně než n krocích (BARRET, a další, 1994).

Algoritmus

vyřeš $\boldsymbol{M} \cdot \boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}$

1. krok: pro $j = 0,1, \ldots$

$\boldsymbol{v}^{(0)} = \frac{\boldsymbol{r}}{\|\boldsymbol{r}\|_2}$

$s_0 = \|\boldsymbol{r}\|_2$

2. krok: pro $i = 0,1, \ldots, m-1$

vyřeš $\boldsymbol{M} \cdot \boldsymbol{w} = \boldsymbol{A} \cdot \boldsymbol{v}^{(i)}$

3. krok: pro $k = 0,1, \ldots, i$

$h_{k,i} = \boldsymbol{w}^T \cdot \boldsymbol{v}^{(k)}$

$\boldsymbol{w} = \boldsymbol{w} - h_{k,i} \cdot \boldsymbol{v}^{(k)}$

$h_{i+1,i} = \|\boldsymbol{w}\|_2$

$\boldsymbol{v}^{(i+1)} = \frac{\boldsymbol{w}}{h_{i+1,i}}$

4. krok: pro $k = 0, \ldots, i-1$

použij rovinnou rotaci J_k na $h_{k+1,i}$

sestav rovinnou rotaci J_i z hodnot h_{ii} a $h_{i+1,i}$

použij rovinnou rotaci J_i na h_{ii} a $h_{i+1,i}$

použij rovinnou rotaci J_i na s_i a s_{i+1}

pokud s_{i+1} je dostatečně malé, pak aktualizuj řešení a ukonči

aktualizuj řešení

vyřeš $\boldsymbol{M} \cdot \boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}$

pokud $\|\boldsymbol{r}\|_2$ je dostatečně malé, pak ukonči.

V hlavním kódu algoritmu je použito několik podprogramů, které si zaslouží vysvětlit. Nejdříve rovinné rotace – použil jsem z důvodu jednoduchosti Givensovu rotaci, která se generuje následujícím způsobem z hodnot dx a dy:

1. krok: když $dy = 0$
   $c = 1$
   $s = 0$
2. krok: jinak když $|dy| > |dx|$
   $s = \frac{1}{\sqrt{1+p^2}}$
   $p = \frac{dx}{dy}$
   $c = p \cdot s$
3. krok: jinak
   $p = \frac{dy}{dx}$
   $c = \frac{1}{\sqrt{1+p^2}}$
   $s = p \cdot c.$

Spočítané parametry c a s jsou použity v následujícím podprogramu, který použije danou rotaci na koeficienty matice:

$t = c \cdot dx + s \cdot dy$
$dy = -s \cdot dx + c \cdot dy$
$dx = t.$

Poslední podprogram, který není rozepsaný v hlavním algoritmu je „aktualizuj řešení“. Pseudokód metody vypadá následovně:

$\boldsymbol{y = s}$
1. krok: pro $i = n, \dots 0$
   2. krok: pro $j = i - 1, \dots, 0$
      $y_j = y_j - h_{ji} \cdot y_i$
3. krok: pro $i = 0, \dots, n$
   $\boldsymbol{x = x + v^{(i)} \cdot y_i}$

Vlastnosti

Tato metoda je jedna z nejsložitějších projektivních metod, které zatím existují, a je také nejsložitější, která je popsána v této bakalářské práci. To je vyváženo tím, že je vhodná pro jakékoliv typy úloh.

Asymptotická složitost je přibližně $O(n^3)$. Sice na první pohled se může zdát, že při jedné aktualizaci se dostáváme až k $O(n^4)$, ale tato část kódu proběhne za celou dobu jen jednou, takže se příliš výrazně neprojevuje na celkové asymptotické složitosti algoritmu.

Nevýhoda metody je velká náročnost na pomocnou paměť. Je potřeba navíc uložit Hessenbergovu matici $\boldsymbol{H}$, předpodmiňovací matici $\boldsymbol{M}$, až m rotací J a několik vektorů. Množství použité paměti lze omezit volbou relativně malého koeficientu m, který udává počet iterací, po kterých se metoda restartuje, ale ten nesmí být zas příliš malý, jinak by metoda nezkonvergovala.

4 Praktická část

Cílem praktické části mé bakalářské práce bylo vytvoření aplikace pro řešení soustav lineárních rovnic pomocí projektivních metod.

4.1 Úvod do moderních programovacích jazyků

V první řadě jsem musel zvolit vhodný programovací jazyk. Dle svých znalostí nabitých před i během jsem si vytyčil několik bodů, které by měl daný programovací jazyk splňovat:

- objektově orientovaný programovací jazyk,
- programovací jazyk by měl mít silný Framework, který poskytuje základní funkcionalitu.

Objektově orientovaný programovací jazyk

OOP je moderní, v současné době nejpoužívanější přístup programování. Vznikl v 70. letech jako nástupce tehdy široce používaného strukturálního programování (typickým zástupcem strukturálních jazyků je klasické C). V OOP se jednotlivé objekty reálného světa reprezentují třídami[1], resp. objekty[2]. OOP, na rozdíl od strukturálního programování, zapouzdřuje data a metody[3] s nimi pracující do jediného celku (DAŘENA, © 2004).

Nejznámější objektové jazyky jsou:

- C++,
- C#,
- Java,
- a další…

Mezi nejdůležitější vlastnosti OOP patří:

- zapouzdření,
- dědičnost,
- polymorfismus.

[1] Třída je předpis, který určuje, jaké má mít daný modelovaný objekt vlastnosti (např. kniha má vlastnosti autor, název, …)
[2] Objekt je instance třídy (objekt už je konkrétní kniha s konkrétním názvem, autorem, …)
[3] Metoda je konkrétní funkcionalita (např. kniha by měla metody „otevři knihu“, „přejdi na stránku“, …)

Framework

Spousta komponent jednotlivých programů je stejná. Většina programů má grafické uživatelské rozhraní, využívá kolekce, … Cílem frameworku je poskytnout kvalitní implementaci základních komponent programu. Mezi nejznámější frameworky patří:

- QT (C++),
- Swing, AWT (Java),
- .NET (C#, Visual Basic).

4.2 Požadavky na aplikaci

Požadavky, které jsem si vytyčil pro mnou implementovanou aplikaci:

- aplikace by měla mít jak grafickou verzi, tak verzi ovládanou z příkazové řádky,
- aplikace by měla mít vstupy a výstupy výhradně ve formátu textových souborů.

4.3 Volba programovacího jazyka

Vzhledem k mým zkušenostem jsem nakonec vybral 2 kandidáty jako implementační jazyky a to:

- Javu a
- C++.

Ačkoli Java má několik nevýhod oproti C++ (např. chybějící přetěžování operátorů, pomalejší a paměťově náročnější programy), tak nakonec jsem kvůli svým zkušenostem zvolil právě Javu.

Java nemá žádnou vlastní knihovnu pro práci s maticemi, tak jsem měl 2 možnosti – buď si danou knihovnu naprogramovat sám, nebo najít a použít již existující knihovnu. Knihovna 3. strany má několik výhod:

- je nepravděpodobné, že by algoritmy obsahovaly velký počet chyb,
- algoritmy jsou s největší pravděpodobností kvalitně optimalizované.

Pro Javu existuje několik knihoven pro práci s maticemi. Mezi nejznámější patří:

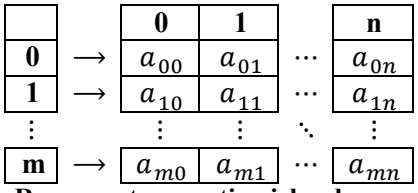
- MTJ – Matrix Toolkit for Java <http://code.google.com/p/matrix-toolkits-java/>,
- JAMA – A Java Matrix Package <http://math.nist.gov/javanumerics/jama/>,
- Colt <http://acs.lbl.gov/software/colt/>.

Nakonec jsem zvolil knihovnu MTJ, a to hlavně kvůli její komplexnosti.

4.4 Metody uložení řídkých matic v paměti

4.4.1 Dvojrozměrné pole

Tento formát uložení matice v paměti je jeden z nejjednodušších. Formát je znázorněn na ilustračním obrázku 1Obrázek 1.



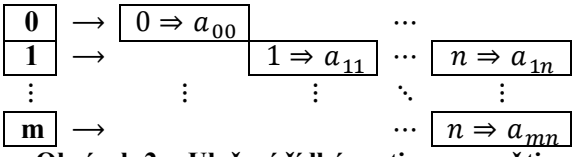
Obrázek 1 – Reprezentace matice jako dvourozměrné pole

Mezi výhody tohoto formátu patří nízký přístupový čas ke konkrétnímu členu matice. Pole jsou v paměti souvislé bloky dat s pevně stanovenou délkou prvku, takže adresa konkrétního prvku se zjistí přičtením koeficientu i k adrese prvního prvku matice $\boldsymbol{A}$, následnou dereferencí a k takto získané hodnotě se přičte koeficient j a dereferencí se získá vlastní hodnota prvku a_{ij}. Operace jsou to velmi rychlé, navíc jejich počet a rychlost je zcela nezávislý na velikosti matice. Asymptotická složitost přístupu je $O(1)$.

Rychlost přístupu je vyvážena tou nevýhodou, že tento formát neumí využít řídkost matice. Pro uložení matice o rozměrech $m \times n$ je potřeba $m \cdot n \cdot o$, kde o je velikost prvku matice, nejčastěji 4 nebo 8 bajtů. To pro matici o rozměru např. $10^5 \times 10^5$ dává dohromady minimálně $4 \cdot 10^{10}$ B $= 40$ GB $\cong 37{,}25$ GiB potřebné paměti.

4.4.2 Další formáty

Všechny další formáty byly objeveny z důvodu úspory místa pro uložení řídkých matic. Nejčastěji se vyskytují implementace typu pole × seznam nebo pole × mapa. Na ilustračním obrázku 2Obrázek 2 je vidět příklad takové struktury.



Obrázek 2 – Uložení řídké matice v paměti

Tento formát je vhodný hlavně pro řídké matice z důvodu, že u každého prvku musí být ještě uložen celočíselný index. Opět uvedu příklad. Mějme řídkou matici o rozměru $10^5 \times 10^5$ s 5 % nenulových prvků. Velikost indexu i vlastního prvku vezměme 4 B. To nám dává potřebu paměti $(4 + 4) \cdot 10^5 \cdot 10^5 \cdot 0{,}05 = 4 \cdot 10^9$ B $= 4$ GB $\cong 3{,}73$ GiB paměti. V tomto konkrétním případě jsou paměťové nároky oproti předchozímu formátu přibližně o 90 % nižší.

Bohužel snížené nároky na paměť jsou vykoupeny zvýšenou asymptotickou složitostí přístupu k prvku. V nejhorším případě při implementaci nesetříděný seznam × nesetříděný seznam je asymptotická složitost až $O(n^2)$. Vzhledem k četnosti této operace je tato implementace nepoužitelná.

Mnohem lepší výsledky dává implementace pole × mapa. Nebudu se zabývat konkrétní vnitřní implementací mapy, důležité je pouze to, že pro přístup k prvku mapy podle indexu se používá binární vyhledávání s asymptotickou složitostí $O(\log_2 n)$, kde n není rozměr matice, ale mnohem menší počet prvků v matici. Přístup k prvku pole má konstantní asymptotickou složitost $O(1)$, to nám dává celkovou asymptotickou složitost $O(\log_2 n)$ – např. pro projití 10^5 prvků je potřeba pouze 17 operací.

4.5 Metody uložení řídkých matic do souboru

V této kapitole se nebudu zabývat otázkami, zda je výhodnější textový soubor, binární soubor, nebo postup ukládání do souboru poskytovaný vyšším programovacím jazykem (serializace). Spíš bych na tomto místě chtěl probrat formáty textových souborů.

4.5.1 Formát dvourozměrného pole

Tento formát je podobný formátu uložení do paměti popsanému v kapitole 4.4.1, s tím rozdílem, že si zachovává si své nevýhody a ztrácí své přednosti, čtení většího souboru bude trvat déle než čtení menšího. Formát je uveden na obrázku 3Obrázek 3.

m	n		
a_{00}	0	$\cdots$	0
0	a_{11}	$\cdots$	a_{1n}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
0	0	$\cdots$	a_{mn}

Obrázek 3 – Uložení dvojrozměrného pole do souboru

Formát je použitelný pouze pro malé nebo husté matice.

4.5.2 Harwell-Boeingův formát

Harwell-Boeing je vcelku oblíbený formát souboru pro uložení matic. Výstupní soubor je velmi malý, ale vlastní formát je relativně složitý. Mezi další nevýhody patří, že je uzpůsoben pro programovací jazyk Fortran, který již v dnešní době není tak významný.

Formát souboru popíši jen stručně, podrobné informace čtenář najde v (National Institute of Standards and Technology, 2011). První čtyři, resp. pět řádků je hlavička souboru, která udává informace o vlastní matice, případně i informaci o vektoru pravých stran. Na následujících řádcích jsou potom uvedeny jednotlivé prvky matice (National Institute of Standards and Technology, 2011).

4.5.3 Formát koordinátů

m	n	4
0	0	a_{00}
1	1	a_{11}
1	n	a_{1n}
m	n	a_{mn}

Obrázek 4 – Formát koordinátů

Tento formát patří mezi nejjednodušší. Jak je ukázáno na obrázku 4, v prvním řádku je uveden počet řádků, počet sloupců a počet nenulových hodnot v matici. Na každém dalším řádku je uvedeno vždy číslo řádku, číslo sloupce a hodnota, a to pro každý nenulový prvek matice. Mezi výhody patří i to, že soubor nemusí být seřazený.

Výstupní souboru je oproti Harwell-Boeingově formátu asi o 30 % větší, nicméně pokud se na výsledný soubor použije nějaký standartní komprimační algoritmus (např. zip apod.), pak jsou soubory zhruba stejně velké. Další nevýhodou je, že nedokáže efektivně uložit symetrickou matici (National Institute of Standards and Technology, 2011).

4.5.4 Matrix market formát

Na začátku souboru jsou hlavičky začínající znakem „%“.

První řádek souboru má následující tvar:

```
%%MatrixMarket <struktura> <formát> <typ hodnot> <symetričnost>.
```

Kde

- „struktura“ značí jaký typ struktury je uložen v souboru. Může nabývat hodnot:
  - matrix nebo
  - vector,
- „formát“ značí, jak jsou hodnoty uloženy v souboru, a může být:
  - coordinate (shodné s formátem popsaným v kapitole 4.5.3) nebo
  - array (shodné s formátem uvedeným v kapitole 4.5.1),
- „typ hodnot“ může nabývat jednu z následujících hodnot:
  - real,
  - complex,
  - integer,
  - pattern,
- a nakonec „symetričnost“ značí typ symetričnosti matice, může nabývat hodnot:
  - general (nesymetrická),
  - symetric (symetrická),
  - skew-symetric (záporně symetrická ($a_{ij} = -a_{ji}$),
  - hermitian (koeficienty a_{ij} a a_{ji} jsou komplexně sdružené).

V hlavičce ještě mohou být komentáře, vždy za znakem „%“ (National Institute of Standards and Technology, 2011).

Po hlavičce následují vlastní data matice ve shodném formátu, jako bylo uvedeno v kapitolách 4.5.1 resp. 4.5.3.

Formát odstraňuje nevýhody předchozího, a zanechává si jeho výhody.

Závěr

Cílem teoretické části této bakalářské práce bylo srovnat některé známé algoritmy pro řešení soustav lineárních rovnic. Nejprve jsem se zaměřil na přímé metody, pokračoval jsem stacionárními iteračními metodami a v poslední řadě jsem zhodnotil některé projektivní iterační metody. Nedá se jednoduše říci, která z těchto metod je nejlepší, výhody a nevýhody jsem rozebral v diskuzi v příslušných kapitolách.

Obecně by se dalo říci, že pro malé soustavy rovnic je jedno, jaké metody použijeme, rozdíly ve výpočetním čase a potřebné paměti budou minimální. U velkých neřídkých systémů bych doporučoval použití iteračních stacionárních metod, a u řídkých systémů zas projektivní metody, které plně umožňují využít řídkosti.

Cílem praktické části bylo vytvoření aplikace pro řešení soustav lineárních rovnic projektivními metodami. V textu jsem opět uvedl diskuzi jaký použít systém pro uložení matice v paměti a v souboru. Uživatelská příručka aplikace je uvedena v příloze. Zdrojové kódy aplikace v jazyce Java jsou uloženy na přiloženém disku CD.

Bibliografie

BARRET, R., a další. 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* Philadelphia : SIAM, 1994. Dostupné z WWW: <www.netlib.org/linalg/html_templates/Templates.html>.

ČERNÁ, Dana. 2010. *Numerické metody lineární algebry.* Liberec : Technická univerzita v Liberci, 2010. Dostupné z WWW: <www.kmd.fp.tul.cz/lide/cerna/MA4/num_lin_alg.pdf>.

DAŘENA, František. © 2004. Objektově orientované programování. *František Dařena.* [Online] © 2004. [Citace: 24. 06 2011.] <https://akela.mendelu.cz/~darena/Perl/objekty.html>.

FIEDLER, Miroslav. 1981. *Speciální matice a jejich použití v numerické matematice.* Praha : SNTL – Nakladatelství trchnické literatury, n. p., 1981.

GANDER, Walter. 1980. Algorithms for the QR-Decomposition. Curych : Eidgenoessische technische hochschule, 1980. Dostupné z WWW: <www.inf.ethz.ch/personal/gander/papers/qrneu.pdf>.

HRIC, Jan. Algoritmy a datové struktury. [Online] [Citace: 24. 06 2011.] <www.mff.cz/data/ADS_slidy.pdf>.

KOLDA, Stanislav a MILADA, Černá. 2004. *Matematika – Úvod do lineární algebry a analytické geometrie.* Pardubice : Univerzita Pardubice, 2004. 80-7194-709-1.

National Institute of Standards and Technology. 2011. Text File Formats. *Matrix Market.* [Online] 17. 06 2011. [Citace: 30. 06 2011.] <www.math.nist.gov/MatrixMarket/formats.html>.

TAUFER, Ivan, KOTYK, Josef a Javůrek, Milan. 2009. *Jak psát a obhajovat závěrečnou práci bakalářskou, diplomovou, rigorózní, disertační, habilitační.* Pardubice : Univerzita Pardubice, 2009. 978-80-7395-157-3.

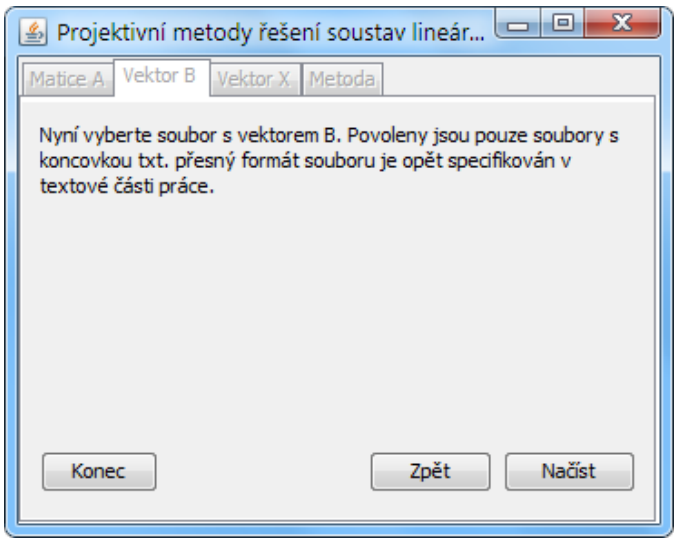
VITÁSEK, Emil. 1987. *Numerické metody.* Praha : SNTL – Nakladatelství technické literatury, n. p., 1987.

WEISSTEIN, Eric. 2011. Wolfram MathWorld: The Web's Most Extensive Mathematics Resource. *Wolfram MathWorld.* [Online] Wolfram Research, 16. 03 2011. [Citace: 31. 03 2011.] <http://mathworld.wolfram.com/>.

Přílohy

Příloha A – Uživatelská příručka

Na obrázku 5 je vidět grafické rozhraní aplikace.



Obrázek 5 – Grafické rozhraní aplikace

Rozhraní programu je řešeno formou průvodce.

Po tisku tlačítka „Načíst“ se zobrazí dialogové okno pro výběr souboru. Povolené formáty souborů jsou popsány v kapitolách 4.5.3 a 4.5.4. Pokud bude soubor úspěšně načten, průvodce přejde do dalšího kroku. Jednotlivé kroky jsou blíže popsány přímo v aplikaci.

Funkce tlačítek „Zpět“ a „Konec“ je zřejmá z názvu.

Aplikace má i rozhraní ovládané z příkazové řádky. Jednotlivé parametry jsou:

- -? zobrazí tuto nápovědu
- -a <cesta k souboru s maticí *A*>,
- -b <cesta k souboru s vektorem *b*>,
- -x <cesta k souboru s vektorem *x*>,
- -m <metoda>, kde metoda může být jednou z následujících hodnot:
  - cg,
  - bicg,
  - gmres.

Pokud není některá hodnota zadána správně, je spuštěna grafická verze aplikace.

Příloha B – Obsah přiloženého CD

V adresáři „Aplikace“ se nachází praktická část mé bakalářské práce – aplikace pro řešení úloh projektivními metodami.

V adresáři „Zdrojový kód“ se nachází projekt pro vývojové studio Netbeans,

V adresáři „Matice“ se nachází příklady matic pro výše uvedenou aplikaci.

V adresáři „Teoretická část“ se nachází tento dokument ve formátech Microsoft Word 2007 (docx) a pdf.