FAKULTA STROJNÍHO INŽENÝRSTVÍ

SAMOSTATNÁ PRÁCE Z PŘEDMĚTU

Numerické metody

NÁZEV PRÁCE: ŘEŠENÍ SYSTÉMŮ ROVNIC S TŘÍDIAGONÁLNÍMI MATICEMI

vypracoval: Lukáš Lev, 256660

Anotace

Na základě zadání poskytnutého vyučujícím byla navržena a implementována metoda obdobná Gaussově eliminační metodě pro řešení třídiagonálních matic. Tato metoda byla testována na vzorovém příkladě (taktéž poskytnutém vyučujícím) a několika vlastních příkladech.

Obsah

1 Zadání

Navrhněte a implementujte obdobu Gaussovy eliminační metody, která řeší soustavy s třídiagonálními maticemi. Svůj program otestujte na soustavě (pro $n = 100$):

$$\begin{bmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 3 & -1 \\ & & & -1 & 3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ \vdots \\ 1 \\ 2 \end{bmatrix}. \tag{1.1}$$

2 Teorie a matematická metodika

2.1 Třídiagonální matice

Třídiagonální matice jsou takové čtvercové matice, jejichž prvky mimo hlavní a dvě první vedlejší diagonály nabývají nulové hodnoty. Tyto matice se v řešení numerických úloh vyskytují velmi často (např. při interpolaci pomocí splinových funkcí). Literatura běžně definuje tyto matice mají na zmíněných diagonálách nenulové prvky (Míka, 1985), pro obecnější implementaci v jazyce MATLAB však budeme pro účely této práce chápat třídiagonální matice z definice předchozí věty.

$$\boldsymbol{A}\vec{x} = \vec{b} \tag{2.1}$$

2.2 Thomasův algoritmus

Pro řešení systémů rovnic ve tvaru z rovnice 2.1 byl vybrán **Thomasův algoritmus** založený na separaci $\boldsymbol{L}$ a $\boldsymbol{U}$ matice coby složky matice $\boldsymbol{A}$. Protože vybírání hlavního prvku (dále nazývaného „pivot“) by změnilo třídiagonální charakter matice $\boldsymbol{A}$, může dojít k dělení nulou i když je $\boldsymbol{A}$ regulární, nebo že výsledek výpočtu bude ovlivněn zaokrouhlováním (Rektorys, 1995). Některé z těchto problémů jsou řešeny v rámci implementace algoritmu v jazyce MALTAB.

Účelem Thomasova algoritmu je úpravami získat systém ve tvaru z rovnice 2.3 (pro rovnici 1.1, kde $n = 5$, viz 2.2). Matice v tomto tvaru je někdy označována jako matice $\boldsymbol{U}$ a Thomasův algoritmus pak zprostředkovává tzv. LU faktorizaci (Rektorys, 1995).

K takovémuto tvaru lze dospět násobením koeficientem $m_i = a_{i-1}/b_{i-1}$, jež vychází z prvků matice z rovnice 2.2. Dále jsou prvky matice a vektoru pravé strany $\vec{d}$ upravovány podle vztahů popsaných v rovnicích 2.4, čímž je dosaženo kýženého stavu.

Z tohoto tvaru systému je pak výpočetně snadné vyjádřit řešení zpětnými substitucemi sestupně (nejdříve je vyjádřeno $x_5 = d_5'/b'5$ atd.).

Během návrhu algoritmu bylo zváženo, zda nulovat prvky hlavní diagonály, což by zjednodušilo některé výpočty především při výpočtu člověkem. Protože je však algoritmus implementován do počítače, byla tato varianta zavržena pro snížení počtu matematických operací, jež zvyšují zaokrouhlovací chybu.

$$
\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 \\ a_5 & b_2 & c_2' & 0 & 0 \\ 0 & a_5 & b_3 & c_3' & 0 \\ 0 & 0 & a_5 & b_4 & c_4 \\ 0 & 0 & 0 & a_5 & b_5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix} \tag{2.2}
$$

$$
\begin{bmatrix} b_1' & c_1' & 0 & 0 & 0 \\ 0 & b_2' & c_2' & 0 & 0 \\ 0 & 0 & b_3' & c_3' & 0 \\ 0 & 0 & 0 & b_4' & c_4' \\ 0 & 0 & 0 & 0 & b_5' \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} d_1' \\ d_2' \\ d_3' \\ d_4' \\ d_5' \end{bmatrix} \tag{2.3}
$$

$$
\begin{cases} b_i' = b_i m \cdot c_{i1} \\ d_i' = d_i m \cdot d_{i1} \end{cases} \tag{2.4}
$$

3 Implementace v jazyce MATLAB

Algoritmus popsaný v kapitole 2 byl následně dle zadání implementován v jazyce MATLAB. Tato implementace je obsažena v jednom zdrojovém souboru s názvem ***llev_projekt.m***.

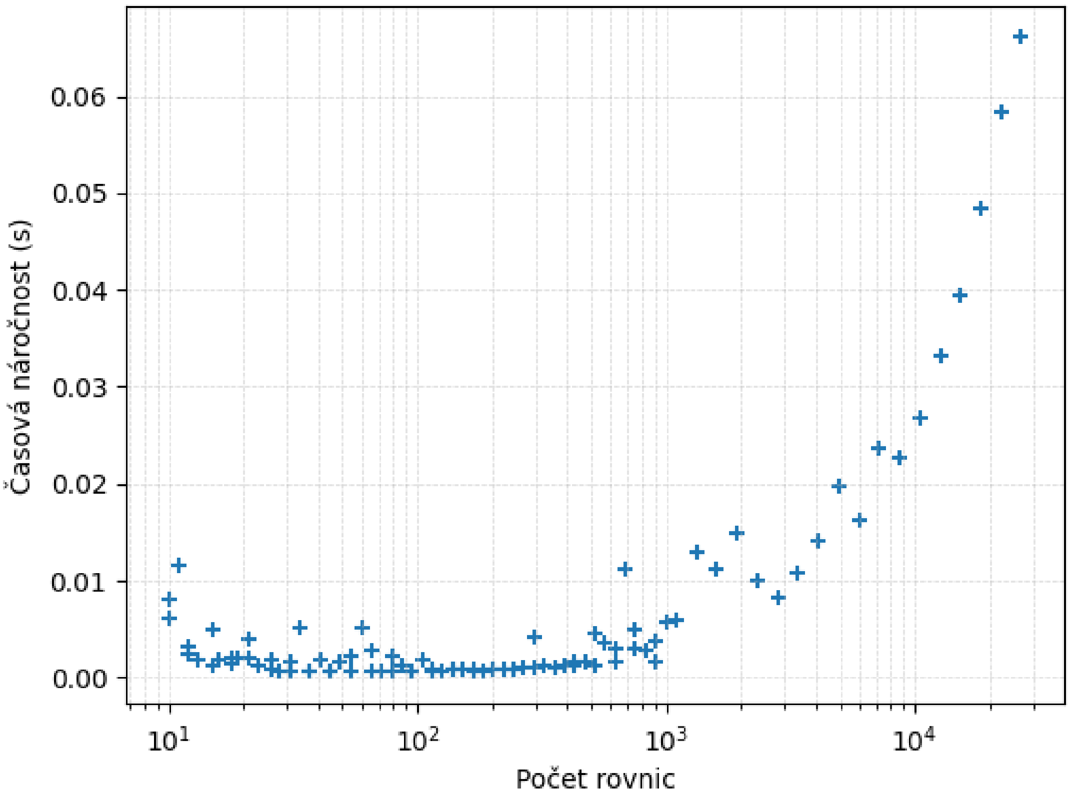
Součástí tohoto souboru je hlavička, deklarace vstupních proměnných, základní funkce a uživatelská část kódu s voláním implementovaných funkcí. Implementace hlavní funkce pro řešení soustavy s třídiagonální maticí je součástí úryvku 2 (tento úryvek ukazuje zjednodušenou formu implementace, v souboru ***llev_projekt.m*** jsou jejím obsahem i další funkcionality).

Dalšími dvěma použitými funkcemi jsou funkce pouze pomocného charakteru pro naplnění matice $\boldsymbol{A}$ a vektoru $\vec{b}$ (viz rovnici 2.1).

Z úryvku 2 je patrné, že se s diagonálami matice zachází jako s vlastními vektory, což sice znamená vytvoření separátních proměnných (může být pozitivní z pohledu modularity kódu), zároveň to však umožňuje snazší manipulaci s jejími prvky. Zároveň v této implementaci nejsou nulovány prvky a_i matice $\boldsymbol{A}$ (viz rovnici 2.2), tak jako v kapitole 2. Je to z důvodu snížení výpočetního času, jelikož se dále ve funkci pro výpočet tyto prvky nepoužívají.

Uživatel by pak implementované funkce volal způsobem uvedeným v úryvku 1. Pokud by byl takový kód spuštěn, jeho výstupem by byla definice matice $\boldsymbol{A}$, jež by byla naplněna prvky o hodnotě -1 na prvních vedlejších diagonálách a prvky s hodnotou 3 nebo 4, symetricky se střídající. Vektor pravých stran $\vec{b}$ by byl zase symetricky naplněn čísly 2 a 1. Takovéto volání odpovídá příkladu ze zadání (kapitola 1), ovšem pro $n = 10$.

Celý kód je přiložen na straně 9.



Obrázek 3.1: Závislost časové složitosti algoritmu na počtu rovnic

```
%% uzivatelska cast kodu
% soustava ze zadani
A = fill_diagonal(A,-1,[3;4],-1);
b = fill_rhsvector(b,[2;1],true);

% % reseni vlastni soustavy - test
% A = fill_diagonal(A,2,1,2);
% b = fill_rhsvector(b,[3;2],true);

disp(A);  % kontrola vstupu
disp(b);  % kontrola vstupu

x = solve_tridiagonal(A,b);  % volani funkce pro reseni

disp(x);  % kontrola vystupu

```

Listing 1: Příklad uživatelského volání funkcí implementovaných v souboru ***llev_projekt.m***

```
function x = solve_tridiagonal(A, b)
    % ARGUMENTY:
    %   A ... tridiagonalni matice
    %   b ... vektor pravych stran

    n = length(b);  % pocet rovnic
    x = zeros(n,1);  % vystupni vektor

    % rozpoznani diagonal
    diag_mid = zeros(n,1);
    diag_low = zeros(n-1,1);
    diag_high = zeros(n-1,1);
    for i = 1:n
        diag_mid(i) = A(i,i);
        if i > 1
            diag_low(i) = A(i,i-1);
        end
        if i < n
            diag_high(i) = A(i,i+1);
        end
    end

    % eliminace prvku
    for i = 2:n
        if diag_mid(i-1) == 0  % zpetna kontrola pivotu
            error('System is not solvable: zero pivot at row %d', i-1);
        end
        m = diag_low(i-1) / diag_mid(i-1);  % multiplikator
        diag_mid(i) = diag_mid(i) - m * diag_high(i-1);  % eliminace
        b(i) = b(i) - m * b(i-1);  % aktualizace vektoru p. strany
    end

    % kontrola pivotu pro resitelnost
    if diag_mid(n) == 0
        error('System is not solvable: zero pivot at last row.');
    end

    % zpetna substituce
    x(n) = b(n) / diag_mid(n);
    for i = n-1:-1:1  % dekrementace i od poctu radku po 1
        if diag_mid(i) == 0  % overeni resitelnosti
            error('System is not solvable: zero pivot at row %d', i);
        end

        % zapis vysledku
        x(i) = (b(i) - diag_high(i) * x(i+1)) / diag_mid(i);
    end
end

```

Listing 2: Implementace funkce pro řešení třídiagonální matice v jazyce MATLAB

4 Zpracování

Podle zadání byla implementace testována pro počet rovnic $n = 100$, kde se v porovnání s analytickým řešením projevila jako spolehlivá (ačkoliv vykazuje asymetrie mezi oběma konci vektoru neznámých v řádu desetin pro hodnoty ze zadání).

Zároveň byla testována časová složitost algoritmu, jež by bylo možno vyhodnotit jako $\mathcal{O}(n)$ podle grafu z obrázku 3.1. Je také třeba brát v potaz, že během měření časové složitosti není separován čas pro naplnění matice a vektoru pravé strany od času pro vlastní řešení systému, což může způsobit zkreslení.

Zpracování této závislosti probíhá za pomocí vlastního kódu, který není součástí souboru v ***llev_projekt.m*** (viz GitHub repozitář).

5 Závěr

V rámci semestrální práce byla navržena metoda obdobná Gaussově eliminační metodě v kapitole 2. Tato metoda je založena na separaci matice $\boldsymbol{L}$ a $\boldsymbol{U}$, někdy je také nazývána Tomasovým algoritmem. Některé nedostatky tohoto algoritmu byly ošetřeny v implementaci v jazyce MATLAB.

Samotná implementace pak byla obsahem jednoho souboru, jež je přiložen k řešení. Nejdůležitější částí tohoto souboru je implementace funkce „solve_tridiagonal“, která je blíže zdokumentována v úryvku 2.

Nakonec byla implementována metoda srovnána s analytickými výsledky a byla vyjádřena její časová složitost. Empiricky vyhodnocená složitost $\mathcal{O}(n)$ je překvapivá v kontextu dvoudimenzionálních matic (předpokládala by se složitost $\Omega(n^2)$). Tato vlastnost je však důsledkem pásového charakteru matice.

Seznam použité literatury

MÍKA, Stanislav, 1985. *Numerické metody algebry.* 2., nezm. vyd. Praha: SNTL - Nakladatelství technické literatury. ISBN (Brož.):

REKTORYS, Karel, 1995. *Přehled užité matematiky II.* 6., přeprac. vyd. Praha: Prometheus. ISBN 80-85849-62-3.

```matlab
%% ===== 2NU, semestralni projekt =====
% Zadani:
% Navrhnete a implementujte obdobu Gaussovy eliminacni metody, ktera
% resi soustavz s tridiagonalnimi maticemi.
%
% Autor:
% Lukas Lev, 2566660

%% Deklarace
n = 100;  % pocet rovnic
A = zeros(n);  % vstupni matice
b = zeros(n,1);


%% Funkce pro plneni diagonaly
% temp: naplneni diagonaly jen jednim cislem
function A = fill_diagonal(A,fill_low,fill_mid,fill_high,symmetric)
    % ARGUMENTY:
    %   A          ... vstupni matice
    %   fill_low   ... cislo / vektor pro vyplneni prvni vedlejsi diagonaly
    %                  pod hlavni
    %   fill_mid   ... cislo / vektor pro vyplneni hlavni diagonaly
    %   fill_high ... cislo / vektor pro vyplneni prvni vedlejsi diagonaly
    %                  nad hlavni
    %   symmetric ... nepovinne, pro naplneni symetricky

    % kontrola vstupu
    if ~ismatrix(A) && not(size(A,1) == size(1,A))
        error('Error: Input must be a square matrix.\n');
    end

    % pokud je vstupem cislo, napln nim vektor pro modularni naplneni
    % matice
    if ~isvector(fill_mid) & isnumeric(fill_mid)
        fill_mid = fill_mid(:);
    end
    if ~isvector(fill_low) & isnumeric(fill_low)
        fill_low = fill_low(:);
    end
    if ~isvector(fill_high) & isnumeric(fill_high)
        fill_high = fill_high(:);
    end
    if not(isvector(fill_high) || isvector(fill_low) || isvector(fill_mid))
        error('Error: Input must be numeric.\n');
    end

    % kontrola symetrie
    if nargin < 5
```

```
        symmetric = false;
    end

    % naplneni matice
    A = zeros(size(A));
    for i = 1:size(A)
        if symmetric
            idx = min(i, size(A) - i + 1);  % zaruceni symetrie
        else
            idx = i;
        end
        A(i, i) = fill_mid(mod(idx-1, length(fill_mid)) + 1);
        if i > 1
            A(i, i-1) = fill_low(mod(idx-2, length(fill_low)) + 1);
        end
        if i < size(A)
            A(i, i+1) = fill_high(mod(idx-1, length(fill_high)) + 1);
        end
    end
end


%% Funkce pro naplneni vektoru prave strany
function b = fill_rhsvector(b,fill,symmetric)
    % ARGUMENTY:
    %   b         ... vstupni vektor
    %   fill      ... cislo / vektor pro vyplneni vektoru prave strany
    %   symmetric ... nepovinne, pro naplneni symetricky

    % kontrola vstupu
    if ~isvector(b)
        error('Error: Input must be a vector.\n');
    end
    if ~isvector(fill) & isnumeric(fill)
        fill = fill(:);
    elseif ~isvector(fill)
        error('Error: Input must be numeric.\n');
    end

    % kontrola symetrie
    if nargin < 3
        symmetric = false;
    end

    % naplneni vektoru
    for i = 1:length(b)
        if symmetric
                idx = min(i, length(b)-i+1);  % zaruceni symetri
        else
            idx = i;
        end
```

```
        b(i) = fill(mod(idx-1, length(fill)) + 1);
    end
end


% Funkce pro reseni systemu
function x = solve_tridiagonal(A, b)
    % ARGUMENTY:
    %   A          ... tridiagonalni matice
    %   b          ... vektor pravych stran

    % kontrola vstupu
    if ~isvector(b)
        error('Error: Input must be a vector.\n');
    end
    if ~ismatrix(A)
        error('Error: Input must be a matrix.\n')
    end

    n = length(b);
    x = zeros(n,1);

    % rozpoznani diagonal
    diag_mid = zeros(n,1);
    diag_low = zeros(n-1,1);
    diag_high = zeros(n-1,1);
    for i = 1:n
        diag_mid(i) = A(i,i);
        if i > 1
            diag_low(i) = A(i,i-1);
        end
        if i < n
            diag_high(i) = A(i,i+1);
        end
    end

    % eliminace prvku
    for i = 2:n
        if diag_mid(i-1) == 0  % zpetna kontrola pivotu
            error('System is not solvable: zero pivot at row %d', i-1);
        end
        m = diag_low(i-1) / diag_mid(i-1);  % multiplikator pro aktualni radek
        diag_mid(i) = diag_mid(i) - m * diag_high(i-1);  % eliminace pod pivotem
        b(i) = b(i) - m * b(i-1);  % aktualizace vektoru p. strany
    end

    % kontrola pivotu pro resitelnost
    if diag_mid(n) == 0
        error('System is not solvable: zero pivot at last row.');
```

```
    end

    % zpetna substituce
    x(n) = b(n) / diag_mid(n);
    for i = n-1:-1:1  % dekrementace i od poctu radku po 1
        if diag_mid(i) == 0  % overeni resitelnosti
            error('System is not solvable: zero pivot at row %d', i);
        end

        % zapis vysledku
        x(i) = (b(i) - diag_high(i) * x(i+1)) / diag_mid(i);
    end
end


% %% Vyhodnoceni casove narocnosti
% filename = 'casova_narocnost.csv';
%
% % hlavicka csv
% if exist(filename, 'file') ~= 2
%     writetable(table("n", "elapsedTime"), filename, 'WriteVariableNames', false);
% end
%
% for n = logspace(1, 3)
%     n = round(n);  % n je cele cislo
%
%     A = zeros(n);  % reset promennych
%     b = zeros(n,1);
%
%     tic  % mereni casu
%         A = fill_diagonal(A, -1, [3;4], -1);
%         b = fill_rhsvector(b, [2;1], true);
%         x = solve_tridiagonal(A, b);
%     elapsedTime = toc;
%
%     % zapis do csv
%     T = table(n, elapsedTime);
%     writetable(T, filename, 'WriteMode', 'append');
% end



%% Uzivatelska cast kodu
% soustava ze zadani
A = fill_diagonal(A,-1,[3;4],-1);
b = fill_rhsvector(b,[2;1],true);

% % reseni vlastni soustavy - test
% A = fill_diagonal(A,2,1,2);
% b = fill_rhsvector(b,[3;2],true);
```

```

disp(A);  % kontrola vstupu
disp(b);  % kontrola vstupu

x = solve_tridiagonal(A,b);  % volani funkce pro reseni

disp(x);  % kontrola vystupu
```