LINEÁRNÍ ALGEBRA S MATLABEM

Tomáš Kozubek
Tomáš Brzobohatý
Marta Jarošová
Václav Hapla
Alexandros Markopoulos

Text byl vytvořen v rámci realizace projektu *Matematika pro inženýry 21. století* (reg. č. CZ.1.07/2.2.00/07.0332), na kterém se společně podílela Vysoká škola báňská – Technická univerzita Ostrava a Západočeská univerzita v Plzni

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Tomáš Kozubek, Tomáš Brzobohatý, Marta Jarošová, Václav Hapla, Alexandros Markopoulos
LINEÁRNÍ ALGEBRA S MATLABEM

ISBN

Předmluva

Stěží bychom v průmyslu - a nejen tam - hledali odvětví, které v nějaké míře neprofituje z lineární algebry. Náramkové hodinky, mobilní telefony, automobily, letadla nebo obrovské mostové konstrukce, to jsou jen některé příklady výrobků, které musí projít patřičným procesem návrhu a testů. Pro designery a konstruktéry mají velký význam reálné experimenty. Ty během simulovaných podmínek ukáží, jak se výrobek skutečně zachová, zda-li dostatečně odolá všem vlivům, nebo naopak není-li zbytečně předimenzován a nedochází tak k plýtvání materiálem. Mnohdy je však experiment a jeho opakování finančně příliš nákladné. Mohou nastat i případy, kdy experiment nelze provést vůbec. Numerické simulace, za nimiž je lineární algebra schována, do jisté míry experiment nahrazují, nebo blíže specifikují podmínky, za kterých by se měl provádět, čímž v obou případech snižují s tím spojené náklady. Pro představu vezměme např. mostovou konstrukci, u které hledáme maximální možný průhyb. Bavíme-li se o konstrukci s charakteristickými rozměry blízkými sta a více metrům, není těžké si uvědomit, že provádět nákladný experiment pro různá zatížení v různých místech, a podle toho upravovat rozměry, je nesmysl. Numerické výpočty v porovnání s předešlým způsobem již tak problematické nejsou. Provedou se některým z komerčních programů, v nichž je lineární algebra přítomna. Tak například složité parciální diferenciální rovnice popisující deformace mostu jsou řešeny *metodou konečných prvků* (kapitola 18) nebo také *metodou sítí* (kapitola 17), čímž je původní problém převeden na *soustavu lineárních rovnic* (kapitola 5)

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

zde $\mathbf{A}$ je *matice* soustavy, $\mathbf{b}$ *vektor* pravé strany a $\mathbf{x}$ je (taktéž *vektor*) hledané řešení - například pole posunutí. Matice $\mathbf{A}$ je tzv. *řídká* (kapitola 6), tzn. převážná část prvků je nulová. Charakter *rozložení nenulových prvků* (kapitola 10) rozhoduje o náročnosti výpočtu, neboť má vliv na *Gaussovu eliminaci*, popřípadě, jsou-li použity, také na maticové rozklady (*Choleského* a LDL pro symetrickou matici, LU pro obecnou matici (podrobnosti v části II)).

Tyto a další nástroje lineární algebry jsou popsány v textu rozděleného do tří tématických bloků. V první části se hovoří o základních programovacích technikách v programu Matlab, práci s daty a tvorbě uživatelských aplikací. Druhá část je věnována soustavám lineárních rovnic a metodám, které se používají k jejich řešení. Poslední část je zaměřena na diferenciální rovnice a numerické metody sloužící k jejich přibližnému výpočtu. Projitím všech kapitol získá čtenář potřebné znalosti z li-

neární algebry a navíc bude schopen psát své vlastní aplikace v prostředí programu Matlab.

1

Tento i ostatní v rámci projektu *Matematika pro inženýry 21. století* připravované výukové materiály lze najít na stránkách http://mi21.vsb.cz/.

[1] Všechny připomínky (výhrady, komentáře a doporučení) zasílejte na e-mailovou adresu: tomas.kozubek@vsb.cz

Obsah

Část I

Matlab

Kapitola 1

Úvod do Matlabu

1.1 Co je Matlab?

Matlab společnosti The MathWorks^{TM} je systém skládající se z několika ingrediencí:

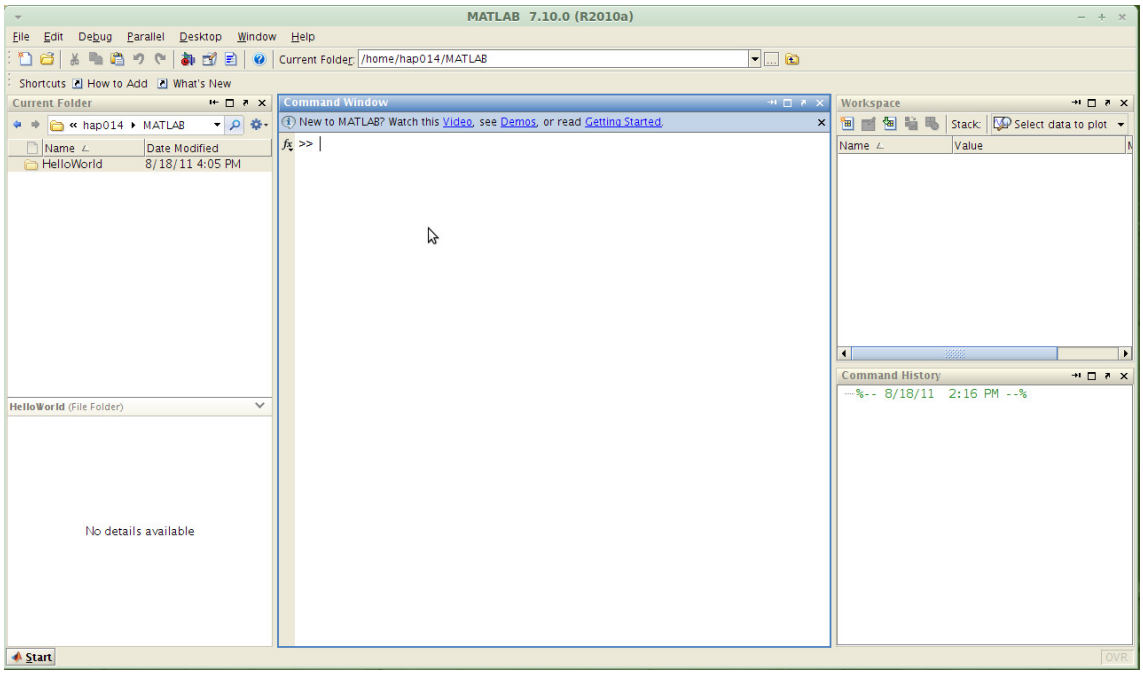
- jazyk uzpůsobený zejména pro numerické výpočty, vývoj numerických algoritmů, analýzu a vizualizaci dat;
- grafické prostředí sloužící zejména pro interaktivní zadávání příkazů v jazyce Matlab;
- rozsáhlá knihovna funkcí v jazyce Matlab pro základní i pokročilou matematiku, import a export dat, datovou analýzu, grafickou vizualizaci dat, tvorbu uživatelského rozhraní a rozhraní k externím aplikacím a jazykům.
- Možnosti Matlabu lze dále rozšiřovat pomocí tzv. toolboxů, což jsou specializované sady funkcí rozšiřující základní sadu, zaměřené často na specifickou aplikační oblast (např. Image Processing Toolbox, Statistics Toolbox).

1.2 Matlab jako program

První tvář Matlabu, se kterou se pravděpodobně setkáte, je *aplikace* Matlab – interaktivní prostředí s grafickým uživatelským rozhraním (obr. 1.1). Je-li nainstalován, spustíme jej jedním z těchto způsobů:

- dvojklikem na jeho ikonu na ploše nebo v nabídce Start (spíše uživatelé Windows)
- zadáním příkazu *matlab* do příkazového řádku (spíše uživatelé Linux)

Prostředí Matlab se skládá z hlavní nabídky, kterou se nyní nemusíme zaobírat, a několika panelů, které si dále představíme. *V prvé řadě si však zapamatujme, že Matlab má propracovanou nápovědu, kterou lze vyvolat z nabídky Help / Product Help.*



Obrázek 1.1: Prostředí Matlab

1.2.1 Command Window a zadávání příkazů

Základním panelem je *Command Window* (příkazové okno). Tento panel obsahuje příkazovou řádku pro interaktivní zadávání příkazů v jazyce Matlab. Můžeme ji přirovnat k velmi chytré kalkulačce.

Zkuste kliknout do panelu Command Window, zadat 2 + 2 a potvrdit klávesou `Enter`. Matlab vrátí výsledek, měli byste vidět toto:

```
>> 2 + 2

ans =

    4
```

To, co Matlab vrátil, je výpis proměnné `ans`. Tu Matlab automaticky použije, pokud sami nespecifikujeme proměnnou, do které chceme výsledek uložit. Takto bychom výsledek uložili do proměnné `var`:

```
>> var = 2 * 3

var =

     6
```

Lze také zadávat více příkazu na jednom řádku. Specialitou Matlabu je, že příkaz ukončený zalomením řádku (`Enter`) nebo čárkou vypisuje vracené hodnoty. Pokud chceme toto chování potlačit, napíšeme za příkaz středník.

```
>> var = 3 + 4;
>> var1 = 1; var2 = 7 / 2, var3 = 8 - 5

var2 =

    3.5000

var3 =

     3
```

Hodnotu proměnné tedy můžeme vypsat prostým zadáním jejího názvu bez středníku:

```
>> var

var =

     7
```

Poznamenejne, že názvy proměnných musí začínat písmenem následovaným libovolnou kombinací písmen, číslic a znaků _ (podtržítko). Písmena mohou být malá i velká. Zkontrolovat, zda lze daný řetězec použít jako název proměnné, můžeme příkazem `isvarname`:

```
isvarname 1clovek

ans =

     0
```

1.2.2 Workspace

V panelu *Workspace* (pracovní prostor) jsou pod sebou vypsány všechny proměnné viditelné v daném kontextu s jejich názvy i hodnotami. Dvojklikem na název proměnné se otevře panel *Variable Editor* (editor proměnných), který slouží k interaktivní editaci a poněkud připomíná Excel. Můžeme kliknout na hodnotu, zadat třeba 55 a potvrdit Enterem; změna se ihned projeví, o čemž se můžete přesvědčit jak pohledem na Variable Editor, tak vypsáním proměnné v *Command Window*.

1.2.3 Command History

V panelu *Command History* (historie příkazů) jsou pod sebou vypsány všechny příkazy, které jsme v minulosti zadali do *Command Window*. Dvojklikem se příkaz znovu provede. Standardním způsobem za pomoci kláves `Ctrl` a `Shift` lze vybrat více příkazů, lze je kopírovat do schránky standardními klávesovými zkratkami nebo přes kontextové menu vyvolané pravým tlačítkem myši. V historii příkazů lze listovat

také přímo v *Command Window* klávesami „šipka nahoru“ a „šipka dolů“. Hodí se to hlavně tehdy, když chceme vícekrát opakovat stejný příkaz.

1.2.4 Current Folder

Podobně lze pracovat i s panelem *Current Folder* (aktuální adresář), kde vidíme obsah aktuálního adresáře, nad kterým pracujeme. Pomocí pravého tlačítka myši navíc můžeme provádět základní souborové operace. Za zmínku stojí též možnost porovnání dvou souborů (*Compare Selected Files* nebo *Compare Against...*) a možnost přidávání nebo odstraňování adresářů z „Path“, tedy seznamu adresářů, jejichž soubory jsou pro Matlab viditelné, i když pracujeme nad jiným adresářem.

1.3 Matlab jako jazyk

Matlab používá jednotný jazyk pro interaktivní použití i zdrojové soubory programů. Syntaxí se snaží být co nejbližší matematické notaci. Základním datovým typem je na rozdíl od jiných jazyků matice. Jednoduchý příklad příkazů v jazyce Matlab jsme již viděli v sekci 1.2.1. Více o jazyce Matlab se dozvíte v kapitole 2.

1.4 Matlab jako knihovna

Systém Matlab zahrnuje také rozsáhlou sadu funkcí v jazyce Matlab:

- základní i pokročilá matematika – např. generování a práce s hustými a řídkými maticemi, lineární algebra, polynomy, numerické a optimalizační metody
- import a export dat
- datová analýza – např. konvoluce, interpolace, regrese, Fourierova transformace
- grafická vizualizace dat – např. 2D a 3D grafy funkcí, zobrazení obrázků
- tvorba uživatelského rozhraní
- rozhraní k externím aplikacím a jazykům – např. MS Excel, C/C++, Java.

Možnosti Matlabu lze dále rozšiřovat pomocí tzv. toolboxů, což jsou specializované sady funkcí rozšiřující základní sadu, zaměřené často na specifickou aplikační oblast (např. Image Processing Toolbox, Statistics Toolbox). Zjistit, které toolboxy máme k dispozici, lze např. zobrazením nápovědy (nabídka Help / Product Help) – každý toolbox má vlastní sadu manuálových stránek reprezentovanou ikonkou knížky.

1.5 Volně dostupné alternativy Matlabu

Pro úplnost dodejme, že existují také volně dostupné alternativy Matlabu. Z těch, které jsou Matlabu velmi podobné a jsou s ním do značné míry kompatibilní, jmenujme trojici GNU Octave, Freemat a Scilab. Nebudeme je zde podrobněji popisovat, zájemce můžeme odkázat např. na srovnání http://userpages.umbc.edu/~gobbert/papers/SharmaGobbertTR2010.pdf.

K podobným účalům jako Matlab je také používán skriptovací jazyk Python s rozšířeními jako NumPy, SciPy a Matplotlib. Tento jazyk vyniká možnostmi propojení s jinými jazyky. Je stále častěji používán zejména v rámci HPC (High Performance Computing). Srovnání s Matlabem najdete např. v http://web.bryant.edu/~bblais/bryant/numerical_computing/python_matlab.pdf.

Kapitola 2

Základy programování v jazyce Matlab

V této kapitole se seznámíte se základy jazyka Matlab. Doporučujeme zkoušet si vše rovnou v prostředí Matlabu. Matlab používá jednotný jazyk pro interaktivní zadávání příkazů v Command Window i pro psaní programů, jehož syntaxe připomíná matematickou notaci. Jedná se o jazyk *slabě typový*, to znamená, že se nedeklaruje typ proměnné. Ve skutečnosti proměnnou není třeba deklarovat vůbec, inicializuje se při prvním přiřazení hodnoty.

2.1 Matice, vektory, skaláry

Základním datovým typem je na rozdíl od jiných jazyků matice; vektor a skalár jsou pouze speciální případy. Matice jsou standardně komplexní s plovoucí desetinnou čárkou s dvojitou přesností.

2.1.1 Zadávání matic

Zadávání matic v jazyce Matlab je snadné. Jak je obvyklé ve všech jazycích, k oddělení desetinných míst slouží tečka a záporná čísla se označují pomlčkou před číslem. Prvky matice zadáváme po řádcích, jednotlivé prvky se oddělují mezerou nebo čárkou, řádky pak středníkem nebo znakem zalomení řádku (`Enter`). Tyto způsoby lze libovolně kombinovat. Např. matici

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

můžeme zadat

```
>> A = [1 2; 3.1 -4; 5 6]; % ale take
>> A = [1,2;3.1,-4;5,6]; % nebo
>> A = [1 2; 3.1 -4
5 6]; % nebo treba
>> A = [1 2
3.1, -4
5 6]; % atd. atd.
```

Komentáře Znak % značí komentář, text za tímto znakem až do konce řádku je Matlabem ignorován:

```
>> % pisu si, co chci, a nikoho to nezajima
>>
```

Bílé znaky Mezera mezi dvěma hodnotami odděluje sloupce matice. Zalomení řádku odděluje řádky matice. Všude jinde jsou tyto „bílé znaky“ ignorovány a můžeme je používat třeba k přehlednějšímu zápisu.

```
>> [ 1 + 2*3   12
    -8          2  ]

ans =

     7     12
    -8      2
```

Jak jsme si řekli, skaláry a vektory jsou pouze zvláštní případy matic. Skalár je tedy matice rozměrů 1×1; můžeme jej, ale nemusíme, uzavírat do hranatých závorek:

```
>> skalar = 1; % je zcela to same jako
>> skalar = [1]

skalar =

     1
```

Vektor se zadává prostě jako matice s jediným řádkem nebo sloupcem:

```
>> sloupcovyVektor = [1;2;3]

sloupcovyVektor =

     1
     2
     3

>> radkovyVektor = [1 2 3]
```

```
radkovyVektor =

     1     2     3
```

Matlab zná také matice 0×0, které mají význam prázdné hodnoty[1].

```
>> A = []

A =

     []
```

2.1.2 Základní operace

Jak jsme již viděli, základní aritmetické operace se zadávájí podobně jako v jiných jazycích operátory +, -, *, /. Mocnina se značí symbolem ^ (to je ten znak na stejné klávese jako šestka), např. 2^3 zadáme jako `2^3`. Tyto operátory však pracují nejen s čísly, ale také s maticemi podle pravidel lineární algebry:

```
>> [1 2; 3 4] * [-1 -2; -3 -4] % nasobeni matice-matice

ans =

    -7   -10
   -15   -22

>> [-1 -2; -3 -4]^2 % mocnina matice

ans =

     7    10
    15    22
```

Chceme-li vynutit operaci po prvcích, vložíme před operátor tečku:

```
>> [1 2; 3 4] .* [-1 -2; -3 -4] % nasobeni po prvcich

ans =

    -1    -4
    -9   -16
```

Pro transpozici matice použijeme apostrof:

```
>> A = [1 2; 3 4], A'

A =

     1     2
```

[1] odpovídá `NULL` v jazyce C++

```
     3     4

ans =

     1     3
     2     4
```

Pozor, u komplexních matic se rozlišuje „obyčejná“ transpozice $\mathbf{A}^T$ a *hermitovská transpozice* $\mathbf{A}^H$ (někdy nazývána *matice konjugovaná k* $\mathbf{A}$), kdy dochází nejen k „překlopení“ podél diagonály, ale každý prvek matice je navíc nahrazen komplexně sdruženým číslem. Pro transpozici $\mathbf{A}^T$ je nutné apostrof opatřit tečkou:

```
>> C = [2i;1+i]; C', C.' % C^H, C^T

ans =

        0 - 2.0000i   1.0000 - 1.0000i

ans =

        0 + 2.0000i   1.0000 + 1.0000i
```

Operátory porovnání `== ~= < <= >= >` a logické operátory `| & ~` (disjunkce, konjunkce, negace) mohou operovat nad maticemi po prvcích; levý a pravý operand tedy musí mít stejnou velikost a výsledkem je logická matice (s hodnotami 0 a 1) stejné velikosti. Lze také kombinovat skalár a matici, v tom případě je skalár aplikován na každý prvek matice.

```
>> A = [3 1; 1 2] > 1

A =

     1     0
     0     1

>> B = [5 6; 7 8] <= [7 8; 5 6]

B =

     1     1
     0     0

>> A | B, A & B, ~A

ans =

     1     1
     0     1


ans =
```

```
     1     0
     0     0


ans =

     0     1
     1     0
```

Operátory `||`, resp. `&&`, (disjunkce, konjunkce) pracují pouze se skaláry, přičemž pravý operand se vůbec nevyhodnocuje, pokud je levý operand vyhodnocen jako 1, resp. 0.

2.1.3 Lomítkové operátory

Operátor '/' pro maticové operandy slouží k řešení maticové rovnice

$$\mathbf{XA} = \mathbf{B},$$

přičemž pro výpočet neznámé matice $\mathbf{X}$ se používá notace

$$\mathbf{X} = \mathbf{B}/\mathbf{A}.$$

Pro skaláry $\mathbf{A}, \mathbf{B}, \mathbf{X}$ jde tedy o běžné dělení skaláru skalárem. Jazyk Matlab přináší také operátor '\' . Zápis

$$\mathbf{X} = \mathbf{A}\backslash\mathbf{B}.$$

znamená řešení maticové rovnice

$$\mathbf{AX} = \mathbf{B}.$$

Mezi oběma lomítky tedy platí vztah $(\mathbf{B}/\mathbf{A})^{\mathrm{T}} = (\mathbf{A}^{\mathrm{T}}\backslash\mathbf{B}^{\mathrm{T}})$; v Matlabu se mnohem častěji používá zpětné lomítko '\'. Uvedené skutečnosti ilustruje následující jednoduchý příklad:

```
>> A = [1 2; 3 4]; x = [1;2]; b = A*x;
>> A\b, (b'/A')'  % v obou pripadech by melo vyjit x

ans =

     1
     2

ans =

     1
     2
```

Všimněte si, že $\mathbf{X} = \mathbf{A}\backslash\mathbf{B}$ matematicky odpovídá $\mathbf{X} = \mathbf{B}^{-1}\mathbf{A}$[1] a rovněž $\mathbf{B}/\mathbf{A} = \mathbf{B}\mathbf{A}^{-1}$. Rozdíl je v tom, že v případě lomítek při výpočtu není explicitně vyčíslena inverze a příkaz je obvykle mnohem rychlejší a robustnější. Matlab používá sofistikovaný algoritmus [2], který vnitřně volá různé lineární řešiče podle toho, jaké vlastnosti má matice $\mathbf{B}$. Pro symetrické matice je to Choleského rozklad, pro obecné čvercové LU rozklad a speciálně jsou ošetřeny diagonální, pásové, trojúhelníkové a řídké matice. Pro obdélníkovou matici $\mathbf{B}$ je $\mathbf{X} = \mathbf{A}\backslash\mathbf{B}$ řešení ve smyslu nejmenších čtverců, tedy řešení tzv. normální rovnice

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{X} = \mathbf{A}^{\mathrm{T}}\mathbf{B}.$$

2.1.4 Vestavěné funkce pro generování matic

Naším prvním seznámením s používáním funkcí v jazyce Matlab bude několik vestavěných funkcí pro generování matic. U následujících funkcí jedním argumentem specifikujeme velikost čtvercové matice, dvěma argumenty velikost obdélníkové matice:

```
>> M = 3; N = 2;
>> O1 = zeros(M); O2 = zeros(M,N);      % same nuly
>> Ones1 = ones(M); Ones2 = ones(M,N); % same jednicky
>> R1 = rand(M); R2 = rand(M,N);        % nahodna cisla
>> I1 = eye(M), I2 = eye(M,N)           % jednicky na
                                        % hlavni diagonale
                                        % (jednotkova
                                        %  pro M = N)

I1 =

     1     0     0
     0     1     0
     0     0     1

I2 =

     1     0
     0     1
     0     0
```

Jako testovací matice nám dále poslouží „magický čtverec“ generovaný funkcí `magic`:

```
>> A = magic(4)

A =

    16     2     3    13
     5    11    10     8
```

[1] tj. násobení inverzní maticí zleva – v Matlabu `X = inv(B)*A`

[2] Pro podrobný popis algoritmu viz nápovědu Matlabu, příkaz `doc mldivide`, oddíl Algorithm

```
     9     7     6    12
     4    14    15     1
```

Matice je „magická“ tím, že má stejné součty všech řádků, sloupců i obou diagonál.

Nápověda k funkci Více informací o volání funkce vypíšete zadáním příkazu `help nazevFunkce`.

```
>> help magic
MAGIC  Magic square.
  MAGIC(N) is an N-by-N matrix constructed from the integers
  1 through N^2 with equal row, column, and diagonal sums.
  Produces valid magic squares for all N > 0 except N = 2.

  Reference page in Help browser
      doc magic
```

Jak vidíme dole ve výpisu, podrobnou grafickou dokumentaci v samostatném okně vyvoláte příkazem `doc nazevFunkce`.

2.1.5 Dvojtečkový operátor

V Matlabu se velmi často používá dvojtečkový operátor ':' *(colon operator)* který slouží k vygenerování vektoru po sobě jdoucích čísel. Syntaxe je `start:konec` pro posloupnost s krokem 1, příp. `start:krok:konec`, pokud chceme jiný krok než 1. Krok může být i číslo záporné nebo s desetinnou tečkou.

```
>> 1:4, 1:2:10, 10:2:1, 10:-2:1, 1:0.1:1.3

ans =

     1     2     3     4

ans =

     1     3     5     7     9

ans =

   Empty matrix: 1-by-0

ans =

    10     8     6     4     2

ans =

    1.0000    1.1000    1.2000    1.3000
```

2.1.6 Zjištění velikosti matice

Velikost matice můžeme zjistit pomocí funkce `size`:

```
>> M = zeros(4,5);
>> size(M) % vraci dvouslozkovy vektor

ans =

     4     5

>> [m,n] = size(M) % vraci dve cisla

m =

     4

n =

     5
```

Délku vektoru pohodlněji získáme funkcí `length`:

```
>> length([1 2 3])

ans =

     3
```

2.1.7 Práce s prvky matice

Nyní si ukážeme některé funkce pracující s jednotlivými prvky matice. Přesvědčte se, že máte ve workspace magický čtverec **A** z oddílu 2.1.4

```
>> A

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Prvek $[\mathbf{A}]_{ij}$ se vybírá pomocí notace `A(i,j)`:

```
>> A(3,2)

ans =

     7
```

Můžeme vybrat také více prvků najednou – submatici. Specifikujeme přitom vektor indexů řádků a vektor indexů sloupců, které chceme vybrat. Pro výběr určitého rozmezí se používá operátor ':', který byl představen v kapitole 2.1.5. Speciálně, pokud zadáme pouze dvojtečku bez uvedení rozsahu, vybereme celý řádek nebo sloupec. Klíčové slovo `end` pak reprezentuje maximální index bez toho, abychom museli zjišťovat velikost matice.

```
>> A(2,[1 3]), A(2:3, 1:2), A(2,:), A(2:end, 3), A(end-1,end)

ans =

     5    10

ans =

     5    11
     9     7

ans =

     5    11    10     8

ans =

    10
     6
    15

ans =

    12
```

Pomocí funkce `diag` můžeme vybírat diagonály jako sloupcové vektory. Hlavní diagonála má číslo 0, směrem nahoru mají diagonály čísla 1, 2, ..., směrem dolů -1, -2, ... Není-li číslo zadáno, bere se 0-tá diagonála.

```
>> diag(A)', diag(A,1)', diag(A,-2)

ans =

    16    11     6     1

ans =

     2    10    12

ans =

     9
    14
```

Funkce `sum` vrací řádkový vektor, který obsahuje součty prvků přes každý sloupec. Speciálně `sum(v)`, kde `v` je vektor, vrací sumu prvků vektoru `v`. Konečně přišel čas, abychom se přesvědčili, že naše matice je skutečně magickým čtvercem:

```
>> sum(A), sum(A'), sum(diag(A)), sum(diag(rot90(A)))

ans =

    34    34    34    34

ans =

    34    34    34    34

ans =

    34

ans =

    34
```

Najděte si sami pomocí `help` nebo `doc`, k čemu slouží funkce `rot90`!

2.1.8 Mazání řádků a sloupců, konkatanace

Z již sestavené matice lze mazat řádky nebo sloupce přiřazením prázdné matice místo zadaných řádků nebo sloupců:

```
>> B = A; % zkopiruje A do B
>> B(:,1) = []; % smaze prvni sloupec B
>> B([1 3],:) = [] % smaze 1. a 3. radek B

B =

    11    10     8
    14    15     1
```

Matlab umožňuje také blokové zadávání, kdy místo jednotlivých prvků zadáváme submatice. Jinak řečeno slučujeme dvě nebo více matic do jedné. Říká se tomu konkatanace:

```
>> C = [B B+10; B+10 B]

C =

    11    10     8    21    20    18
    14    15     1    24    25    11
    21    20    18    11    10     8
    24    25    11    14    15     1
```

2.1.9 Elementární funkce a matematické konstanty

Matlab má zabudovánu celou řadu elementárních matematických funkcí jako absolutní hodnota (`abs`), odmocnina (`sqrt`), exponenciální funkce (`exp`), sinus (`sin`) aj. Užitečný seznam těchto funkcí, odkazující na jednotlivé stránky nápovědy, lze vypsat zadáním help elfun (nápověda přímo v příkazové řádce) nebo doc elfun (manuálová stránka)[1].

```
>> help elfun
```

Několik speciálních funkcí vrací důležité konstanty:

```
pi      % Ludolfovo cislo 3.14159265...
i, j    % imaginarni jednotka
eps     % relativni presnost realnych cisel
realmin % nejmensi podporovane kladne realne cislo
realmax % nejvetsi podporovane realne cislo
Inf     % nekonecno, napr. 1/0
NaN     % nedefinovana hodnota, napr. 0/0
```

2.2 Řetězce

Řetězce jsou speciální vektory, jejichž prvky nejsou interpretovány jako čísla, ale jako znaky. Každý znak má totiž svůj celočíselný kód a řetězec je ve skutečnosti vektor těchto kódů. Funkce jako `size` nebo přístup k prvkům fungují stejně jako u vektorů.

2.2.1 Konstrukce řetězců

Hodnota řetězce se zapisuje jako text ohraničený apostrofy:

```
>> str = 'ja jsem retezec'

str =

ja jsem retezec
```

Více řetězců lze spojit (konkatanovat) do jednoho dvěma způsoby:

```
>> str = 'Dr. John Doe ';
>> str1 = strcat(str,', ','1970') % Ignoruje mezery

str1 =

Dr. John Doe,1970
```

[1] Jak bylo zmíněno již v 2.1.4, help elfun vypíše čistě textovou nápovědu přímo v příkazové řádce, doc elfun zobrazí grafickou manuálovou stránku v samostatném okně.

```
>> str2 = [str,', ','1970'] % Neignoruje mezery

str2 =

Dr. John Doe, 1970
```

2.2.2 Formátované řetězce

Funkce `sprintf` slouží k zápisu formátovaných dat do řetězce. První argument je formátovací řetězec, do nějž jsou dosazovány další argumenty na místa tzv. *konverzních specifikací (conversion specifications)*. Konverzní specifikace je speciální podřetězec začínající znakem %. Není v našich možnostech zde uvést všechny varianty, k tomu nechť čtenáři poslouží nápověda k funkci `sprintf`; nejčastěji ale budete potřebovat `'%d'` pro celá čísla a `'%8.4f'` pro reálná čísla, kde 8, resp. 4, je max. počet cifer před, resp. za, desetinnou tečkou. Pro vložení znaku tabulátoru, resp. nového řádku, vepište `'\t'`, resp.`'\n'`.

```
>> T=1323.56; str = sprintf('Teplota\nT=%10.4fK', T)

str =

Teplota
T= 1323.5600K
```

2.2.3 Porovnávání řetězců

Následující funkce srovnávají dva řetězce a vracejí logickou hodnotu, 0 = neshoda, 1 = shoda.

```
strcmp('hello','Hello')
strncmp('hello','hell',4)  % prvnich n znaku retezcu
strcmpi('hello','Hello')   % nerozlisuje mala a velka
strncmpi('HELLO','hell',4) % prvnich n znaku retezcu
                           % (nerozlisuje mala a velka)
```

2.2.4 Prohledávání řetězců

Funkce `strrep` slouží k vyhledání všech výskytů daného podřetězce a jejich nahrazení jiným podřetězcem. Funkce `findstr` vrací index prvního výskytu daného podřetězce. Pokud podřetězec není nalezen, vrátí `[]`. Funkce `strtok` oddělí první slovo od zbytku řetězce.

```
>> str = 'hura juchu hura';
>> % Najde vsechny vyskyty 'hu' a prepise na 'hoo'
str = strrep(str,'hu','hoo')
```

```
str =

hoora juchoo hoora

>> findstr('cho',str)

ans =

     9

>> findstr('hu',str)

ans =

     []

>> % V a budou vsechny znaky od zacatku az po prvni mezeru;
   % v b bude zbytek vety.
>> [a,b] = strtok(str)

a =

hoora


b =

 juchoo hoora

>> % lze pouzit i jiny oddelovac nez mezeru
[a,b] = strtok(str,'r')

a =

hoo


b =

ra juchoo hoora
```

2.2.5 Konverze řetězců

Často potřebujeme převést řetězce převést na jiný datový typ, převést číslo v desítkové soustavě na jinou soustavu, či převést malá písmena v řetězci na velká. A to vše také v opačném směru. Následuje výčet některých funkcí Matlabu, které k tomu slouží.

```
double('012')      % retezec na vektor ciselnych kodu
char([72 105])         % ciselne kody na retezec 'Hi'
```

```
int2str([72 105])      % cela cisla na retezec '72 105'
num2str(pi, '%10.5e') % cisla na retezce
                       % podle zadaneho formatu
                       % '3.14159e+000'
mat2str([3.85 2.91; 7.74 8.99]) % matice na retezec
                                % '[3.85 2.91; 7.74 8.99]'

str2num('72 105')      % [72 105]
str2double('72 105')   % NaN
str2num('1 -2i')       % [1.0000 -2.0000i]
str2double('1 +2i')    % 1.0000 - 2.0000i,

dec2hex(255)           % decimalni cisla na hexadecimalni jako
  retezec
dec2bin(12)           % decimalni cisla na binarni jako retezec
dec2base(100,5)        % decimalni cisla na cisla s lib. zakladem

% a naopak
hex2dec('FF')
bin2dec('1100')
base2dec('400',5)

% viz take num2hex, hex2num

lower(str)             % Velka pismena v retezci zmeni na mala
upper(str)             % Mala pismena v retezci zmeni na velka
```

2.3 Pole buněk

Pole buněk (cell array) je kolekce „paměťových buněk“, ve kterých mohou být uloženy různé typy dat – různě velkých matic, řetězců aj., přičemž buňky jsou na sobě nezávislé a každá může obsahovat data jiného typu a velikosti. Pole buněk mohou být jedno-, dvou- i vícerozměrná.

2.3.1 Konstrukce pole buněk

Pole buněk můžeme vytvořit postupným přidáváním buněk, celé najednou nebo kombinací obého.

Konstrukce celého pole najednou Pole buněk se konstruuje podobně jako matice, jen místo hranatých závorek použijeme složené a místo číselných prvků matice můžeme zadávat libovolný typ dat. Zde je příklad pole, které má velikost 2x2 a obsahuje smíšená řetězcová a vektorová data:

```
>> woman={'Jonah Doe',[90 60 91]; 'Anne Black',[89 57 86]}
```

```
woman =

    'Jonah Doe'      [1x3 double]
    'Anne Black'     [1x3 double]
```

Takto vytvoříme prázdné pole:

```
>> {}  % prazdne pole

ans =

     {}
```

Pomocí funkce `cell` můžeme vytvořit pole s prázdnými buňkami:

```
>> C = cell(2,3)

C =

    []    []    []
    []    []    []
```

Postupné vkládání buněk Stejné pole také můžeme sestavit po jednotlivých buňkách; můžeme též vkládat více buněk najednou jako vidíme na druhém řádku. Pole je dynamicky realokováno, takže při nastavení obsahu neexistující buňky je tato buňka automaticky vytvořena. Buňky se indexují kulatými závorkami a konstruují složenými závorkami:

```
>> clear woman
>> woman(1,1)={'Jonah Doe'};woman(1,2)={[90 60 91]};
>> woman(2,1:2)={'Anne Black', [89 57 86]}

woman =

    'Jonah Doe'      [1x3 double]
    'Anne Black'     [1x3 double]
```

Postupné vkládání obsahů buněk Pole je dynamicky realokováno, takže při nastavení hodnoty neexistující položky je tato položka automaticky vytvořena. Můžeme také vkládat přímo obsahy buněk, indexují se složenými závorkami:

```
>> clear woman
>> woman{1,1}='Jonah Doe'; woman{1,2}=[90 60 91];
>> woman{2,1}='Anne Black'; woman{2,2}=[89 57 86]

woman =

    'Jonah Doe'      [1x3 double]
    'Anne Black'     [1x3 double]
```

Např. příkaz `woman{1,1}='Jonah Doe'` je třeba chápat takto: vytvoř buňku na pozici (1,1) v poli `woman` a vlož do ní řetězcovou hodnotu `'Jonah Doe'`.

2.3.2 Přístup k prvkům pole

Následující komentovaný kód ukazuje několik způsobů, jak lze přistupovat k jednotlivým buňkám.

```
>> w=woman(1,1) % Vrati bunku, jejimz obsahem je jmeno zeny

w =

    'Jonah Doe'

>> w{1}         % Vrati obsah bunky w - jmeno zeny (string)

ans =

Jonah Doe

>> w=woman{1,1} % Vrati obsah bunky se jmenem zeny (string)

w =

Jonah Doe

>> woman{1,2}(2) % Vrati 2. prvek matice v bunce (1,2)
% (obvod pasu 1. zeny)

ans =

    60
```

2.3.3 Vnořování polí

Buňka může obsahovat libovolná data včetně pole buněk. Pole buněk tedy do sebe můžeme vnořovat. Matlab podporuje také vnořené indexování, a to může kombinovat `()` i `{}` přístup.

```
>> clear woman
>> woman={ 'Jonah Doe', {'60kg',[90 60 91]}
'Anne Black',{'56kg',[89 57 86]} }
% bunka (1,2) obsahuje pole bunek

woman =

    'Jonah Doe'      {1x2 cell}
    'Anne Black'     {1x2 cell}
```

```
>> woman{2,2}{1} % (vaha zeny 2)

ans =

56kg

>> woman{2,2}{2}(2) % obvod pasu 2. zeny

ans =

    57
```

2.3.4 Výpis obsahů buněk

Zavoláním funkce `celldisp` můžeme vypsat obsahy všech buněk pole najednou.

```
>> celldisp(woman)

woman{1,1} =

Jonah Doe

woman{2,1} =

Anne Black

woman{1,2} =

    90    60    91

woman{2,2} =

    89    57    86
```

2.4 Struktury

Struktura se podobá poli buněk v tom, že může obsahovat libovolná smíšená data. Avšak k jejím položkám se nepřistupuje přes číselné indexy, ale prostřednictvím názvů položek.

2.4.1 Konstrukce struktury

Strukturu můžeme vytvořit postupným přidáváním položek, celou najednou nebo kombinací obého.

Postupné přidávání položek Podobně jako pole buněk, je i struktura dynamicky realokována, takže při nastavení hodnoty neexistující položky je tato položka automaticky vytvořena.

```
>> %zajisti, aby odstraneni studenta, byl-li jiz v pameti
>> clear student;

>> student.jmeno = 'John Doe';
>> student.predmet = 'LAM';
>> student.znamky = [1 2 1];
>> student

student =

      jmeno: 'John Doe'
    predmet: 'LAM'
     znamky: [1 2 1]
```

Vytvoření celé struktury najednou Pomocí funkce `struct` můžeme vytvořit celou strukturu najednou. Střídavě zadáváme názvy položek a jejich hodnoty.

```
>> student = struct('jmeno','John Doe', ...
'predmet','LAM','znamky',[1 2 1])

student =

      jmeno: 'John Doe'
    predmet: 'LAM'
     znamky: [1 2 1]
```

2.4.2 Přístup k položkám struktury

Pro získání hodnoty položky struktury lze použít tečkovou syntaxi nebo funkci `getfield`. Ta umožňuje proměnný název položky. Pro vložení hodnoty můžeme užít tečkovou syntaxi nebo funkci `setfield`.

```
>> sjmeno = student.jmeno      % Vrati jmeno studenta, 1. zp.

sjmeno =

John Doe

>> sjmeno = getfield(student,'jmeno')  % 2. zp.

sjmeno =

John Doe
```

```
>> % 2. zp. umoznuje pouzit nazev polozky ulozeny v promenne
field = 'jmeno'; sjmeno = getfield(student,field)

sjmeno =

John Doe

>> sznamka = student.znamky(2) % Vrati 2. znamku studenta

sznamka =

     2

>> % Nastavi jmeno studenta, 1.zp.
student.jmeno = 'Jimmy White';
>> % 2.zp., umoznuje pouzit nazev polozky ulozeny v promenne
student = setfield(student,'jmeno','Jimmy White')

student =

      jmeno: 'Jimmy White'
    predmet: 'LAM'
     znamky: [1 2 1]
```

2.4.3 Vnořování struktur

Podobně jako buňka, může též položka struktury obsahovat libovolná data včetně pole nebo struktury. Struktury můžeme vnořovat do sebe. Přístup do vnořené struktury je jednoduchý, můžeme použít zřetězenou tečkovou syntaxi.

```
>> student=struct('jmeno','John Doe','predmet','LAM',...
  'znamky',struct('test1',1,'test2',2,'zkouska',1));
>> student.znamky % Vypis obsahu unorene struktury

ans =

      test1: 1
      test2: 2
    zkouska: 1

>> student.znamky.test1 % Pristup k polozce unorene struktury

ans =

     1
```

2.4.4 Test existence položky

Jak zjistit, zda daná struktura obsahuje danou položku?

```
>> % Test, zda polozka 'jmeno' existuje ve strukture student
>> isfield(student,'jmeno')

ans =

     1

>> % Test, zda polozky 'jmeno','znameni'
   % existuji ve strukture student
>> isfield(student,{'jmeno','znameni'})

ans =

     1     0
```

2.5 Skripty

Skripty slouží k tomu, abychom jedním, námi definovaným příkazem spustili celou posloupnost příkazů místo toho, abychom je pokaždé ručně spouštěli jeden za druhým. *Funkce* navíc mohou přijímat *vstupní argumenty* ovlivňující průběh výpočtu a výsledky výpočtů mohou být vráceny jako *výstupní argumenty*. Funkce probereme v oddíle 2.8.

Psaní skriptů je velmi jednoduché. Příkazem `edit` se otevře nový panel *Editor*. Lze to provést také jinými způsoby[1]:

- kliknutí na ikonku `New Script` v panelu nástrojů
- klik pravým tlačítkem do panelu *Current Folder* a výběr New File > Script.
- výběr File > New > Script. v hlavním menu
- stisknutí klávesové zkratky `Ctrl+N`

Do editoru nyní můžeme vepsat libovolnou posloupnost příkazů v jazyce Matlab úplně stejně jako do *Command Window*. Zkusme např. vepsat sérii příkazů z oddílu 2.1.8[2], která vede k vytvoření matice `C`:

```
A = magic(4);
B = A; % zkopiruje A do B
B(:,1) = []; % smaze prvni sloupec B
B([1 3],:) = [] % smaze prvni a treti radek B
C = [B B+10; B+10 B]
```

[1] Podobně jako v jiných programech s graf. uživ. rozhraním tedy existuje někdy mnoho možností jak provést tu samou akci, záleží jen na vašich preferencích. Další příkazy již takto podrobně rozebírat nebudeme.

[2] nejjednodušeji copy&paste z panelu *Command History*

Pak skript uložíme do souboru kliknutím na ikonku `Save`[1]. Pojmenujte jej např. `generateC.m`. Přípona `.m` je důležitá; soubory s touto příponou jsou Matlabem indexovány a lze je spustit. Říkame jim M-soubory *(M-files)*. Pokud je skript uložen v aktuálním adresáři[2], je od této chvíle pro Matlab viditelný a rozšiřuje sadu příkazů, které lze spustit. Spustíme jej tedy zadáním jeho názvu v *Command Window*:

```
>> generateC

B =

    11    10     8
    14    15     1

C =

    11    10     8    21    20    18
    14    15     1    24    25    11
    21    20    18    11    10     8
    24    25    11    14    15     1
```

Není překvapením, že se vypsaly vrácené hodnoty `B` a `C`, protože příslušné příkazy nejsou ukončeny středníkem.

Dále si všimněte, že proměnné zůstaly v pracovním prostoru[3]:

```
>> who

Your variables are:

A  B  C
```

To je pro skripty typické: *všechny proměnné vytvořené za běhu skriptu zůstávají v pracovním prostoru, dokud je nesmažeme.*

2.6 Textový výstup do příkazové řádky

Již jsme viděli, že Matlab vypisuje vrácené hodnoty u příkazů, jež nejsou zakončeny středníkem. Nyní se podívejme na funkce Matlabu, které umožňují sofistikovanější textový výstup do příkazové řádky. Funkce `disp` vypíše pouze hodnotu proměnné a nic víc:

```
>> str = ['MiG-' num2str(21)];
>> str

str =

MiG-21
```

[1] nebo `File / Save` nebo klávesovou zkratkou `Ctrl+S`.
[2] adresář, který vidíme v řádku *Current Folder* v panelu nástrojů a v panelu *Current Folder*
[3] vizte také obsah panelu *Workspace*

```
>> disp(str)
MiG-21

>> disp([1 22; 33 4]);
     1    22
    33     4
```

Funkce `fprintf` (kapitola 3) umožňuje formátovaný výstup. Je podobná funkci `sprintf` (viz 2.2.2), ale výstup jde do příkazové řádky a ne do řetězce:

```
>> T=1323.56; fprintf('Teplota\nT =%10.4fK\n', T)
Teplota
T =  1323.5600K
```

2.7 Řízení toku programu

Nyní se seznámíme s příkazy, které nám umožní psát skutečné algoritmy. Reálné kódy musí při svém běhu reagovat na data, větvit se na základě logických podmínek a iterovat, tj. opakovat stejnou sadu příkazů, dokud není splněna určitá podmínka. Byť lze toto činit i v příkazové řádce, význam to má zejména pro skripty a funkce, o kterých bude nyní řeč.

2.7.1 Podmínkový blok (`if-elseif-else`)

Základním způsobem větvení programu je konstrukce `if-elseif-else-end`. Obecně vypadá takto:

```
if podminka1
  % prikazy, jez se provedou pri splneni podminka1
elseif podminka2
  % prikazy, jez se provedou, pokud neni splnena
  % podminka1 a zaroven je splnena podminka2
else
  % prikazy, ktere se provedou v ostatnich pripadech
end % konec if
```

Jak vidíme, ukončuje se klíčovým slovem `end`. A zde máme konkrétní příklad:

```
if n <= 0
    % Je-li n zaporne nebo nulove, zobraz zpravu.
    disp('Vstupni argument n musi byt kladny!');
    a = NaN;
elseif rem(n,2) == 0
    % Je-li n>0 a sude, podel ho 2.
    a = n/2;
else
    % Je-li n>0 a liche, zvetsi ho o 1 a podel 2.
    a = (n+1)/2;
```

```
end
```

Tento kousek kódu můžeme uložit jako `ex1.m`. Podívejme se, jak skript `ex1` reaguje na různé hodnoty `n`:

```
>> n = -10; ex1; a
Vstupni argument n musi byt kladny!

a =

   NaN

>> n = 10; ex1; a

a =

     5

>> n = 7; ex1; a

a =

     4
```

Matlab podporuje i maticové podmínky:

```
if A>=0
  % pokud jsou vsechny prvky A>=0 ruzne od nuly
  disp(sqrt(A));
else
  disp('A neni nezaporne')
end
```

Po uložení do souboru `ex2.m` měníme `A`, monitorujeme logickou matici `A>=0` a spouštíme skript `ex2`:

```
>> A=[1 4; -9 1]; A>=0, ex2

ans =

     1     1
     0     1

A neni nezaporne
>> A=[1 4; 9 1]; A>=0, ex2

ans =

     1     1
     1     1

     1     2
     3     1
```

2.7.2 Výhybkový blok (`switch-case-otherwise`)

Někdy je vhodnější použít poněkud speciálnější konstrukci `switch-case-otherwise-end`, kdy běh programu pokračuje za návěstím `case` odpovídajícím hodnotě řídící proměnné a před následujícím návěstím `case` vyskočí ven z bloku `switch` až za `end`. Kód za návěstím `otherwise` je použit tehdy, když hodnotě řídící proměnné neodpovídá žádné návěstí `case`.

```
switch var
  case 1
    disp('var is 1')
  case {2,3,4}
    disp('var is 2 or 3 or 4')
  case 5
    disp('var is 5')
  otherwise
    disp('var is something else')
end
```

Po uložení do souboru `ex3.m` měníme `x` a spouštíme skript `ex2`:

```
>> x=1; ex3
x is 1
>> x=3; ex3
x is 2 or 3 or 4
>> x=5; ex3
x is 5
>> x=9; ex3
x is something else
```

Tento příklad by se dal ekvivalentně, ale méně čitelně napsat pomocí `if-elseif-else-end`:

```
if x==1
    disp('x is 1')
elseif x==2 || x==3 || x==4
    disp('x is 2 or 3 or 4')
elseif x==5
    disp('x is 5')
else
    disp('x is something else')
end
```

2.7.3 Cyklus s podmínkou (`while`)

Když běh programu dorazí do bloku `while-end`, vyhodnotí podmínku. Pokud není splněna, skočí za `end`, jinak pokračuje dalším řádkem až po příslušný `end`, načež skočí zpět na `while` a příběh se opakuje, a tak kód iteruje.

```
n = 1;
while prod(1:n) < 1e10
```

```
    n = n + 1;
end
disp(n-1);
```

Uvedený kód vypíše 13. Proč?

2.7.4 Cyklus se známým počtem iterací (`for`)

Oproti `while` se cyklus `for` používá tehdy, když je již před vstupem do smyčky znám počet iterací. Zpravidla se používá pro iterování nad polem. Sleduje se hodnota iterační proměnné, která se při každém průchodu zvyšuje nebo snižuje o konstantu (zpravidla 1). Colon operator zde nevytváří nový vektor, ale určuje, v jakém rozmezí půjde iterační proměnná.

```
x=zeros(1,6);
x(1)=1;
for i=2:length(x)
  x(i)=2*x(i-1);
end
x
```

Uvedený kód vypíše hodnoty `1 2 4 8 16 32`.

Poznamejme, že cykly lze libovolně vnořovat jako zde:

```
A=[];
m=3; n=2;
A=zeros(3,2);
for i=1:m
  for j=1:n
    A(i,j)=i^j;
  end
end
```

Také např. v těle cyklu `for` může být cyklus `while` a naopak, může zde být libovolné větvení, zkrátka libovolný kód...

2.7.5 Přerušení cyklu (`break`, `continue`)

Příkazy `break` a `continue` slouží k řízení cyklu zevnitř. Příkaz `break` úplně přerušuje smyčku a přesměruje běh programu až za `end` ukončující cyklus. Příkaz `continue` ukončuje aktuální iteraci a přesměruje běh programu zpět na `while` nebo `for`.

```
A=[0 1 2 3 2 1 2 3 4 5 1];

% Pokud je soucet prvku j-teho sloupce mensi nez 3,
% pak se tento soucet nevytiskne
disp('smycka 1');
for j=A
  s=sum(j);
```

```
  % Je-li splneno, preskoc nasledujici prikazy
  % a jdi na dalsi sloupec
  if s<3; continue; end;

  disp(s);
end
```

Tento kód vypíše:

```
smycka 1
     3

     3

     4

     5

smycka 2
     0

     1

     2
```

2.8 Funkce

Funkce se odlišují od skriptů tím, že akceptují vstupní argumenty (žádný, jeden nebo více) a vracejí výstupní argumenty (žádný, jeden nebo více).

2.8.1 Základní struktura funkce

Otevřeme okno editoru stejně jako v případě psaní skriptu (viz 2.5) a můžeme do něj vepsat tuto základní strukturu funkce:

```
function [vystup1, vystup2] = nazevFunkce(vstup1, vstup2)
%NAZEVFUNKCE strucny popis (H1)
%   Podrobna napoveda k funkci ...
%   ...
%
%   See also jinaFunkce1, jinaFunkce2, ...

end
```

Lze také použít nabídku File > New > Function, v tomto případě dostaneme předpřipravené tělo funkce. Můžeme ji uložit do souboru `nazevFunkce.m`.

Řádek 1 obsahuje *signaturu funkce*. Tento řádek je nejdůležitější, protože zde stanovujeme název funkce a její vstupní a výstupní argumenty. Název funkce musí

být stejný jako je název daného M-souboru. Do kulatých závorek za název funkce uvádíme vstupní argumenty oddělené čárkami. Výstupní argumenty uvádíme do hranatých závorek před znak rovnítka a název funkce. Pro jeden nebo žádný argument můžeme použít zjednodušený zápis deklarace:

```
function funkce1           % zadne argumenty
function funkce2(vstup)    % pouze jeden vstupni argument
function vystup = funkce3 % pouze jeden vystupni argument
```

Vraťme se k prvnímu výpisu. Řádky 2 až 6 jsou komentáře, které ale Matlab interpretuje speciálně. Řádek 2, takzvaný H1, obsahuje název funkce (podle konvencí velkými písmeny) a jednořádkový popis funkce. Tento řádek je zobrazen, pokud voláme `help` nad celým adresářem [1] nebo při použití funkce `lookfor`, která prohledává všechny M-soubory v Matlabem sledovaných adresářích[2]:

```
>> help(pwd)  % pwd vraci aktuani adresar
Contents of hap014:

nazevFunkce                    - strucny popis (H1)

>> lookfor nazevFunkce
nazevFunkce                    - strucny popis (H1)
```

Řádky 3 až 6 obsahují podrobný popis funkce, zpravidla se uvádí různé možnosti volání funkce, příklady použití apod. Za výrazem "See also" se uvádějí názvy souvisejících funkcí, Matlab je automaticky zobrazuje jako hypertextové odkazy, pokud jsou to pro něj viditelné funkce. Dobrým příkladem podrobně napsané nápovědy je např. `help rand`. V případě naší první funkce uvidíme toto:

```
>> help nazevFunkce
 NAZEVFUNKCE strucny popis (H1)
    Podrobna napoveda k funkci ...
    ...

    See also jinaFunkce1, jinaFunkce2, ...
```

Řádky pod úvodním komentářem (7 a dále) tvoří *tělo funkce*, v tuto chvíli je prázdné, takže naše funkce nic nedělá:

```
>> nazevFunkce(1,2)
>>
```

Kíčové slovo `end` na konci funkce není povinné, pokud v daném M-souboru uvádíme pouze jednu funkci.

[1] Místo `pwd` můžeme použít libovolnou adresářovu cestu, vypíší se funkce v daném adresáři
[2] tzn. aktuální adresář `pwd` a adresáře, které vypisuje `path`, viz `help path`

2.8.2 Volání funkce

Pojďme si nyní napsat trochu rozumnější funkci, která spočítá základní statistiky souboru dat x uloženého jako vektor. Můžeme použít předchozí funkci jako šablonu.

```
function [mean_x,min_x,max_x,mode_x,med_x] = statistics(x)
% STATISTICS zakladni statistiky vstupniho vektoru x
%    [mean_x,min_x,max_x,mode_x,med_x] = STATISTICS(x) vraci
%    zakladni statistiky spoctene z ciselnych dat ulozenych
%    ve vstupnim vektoru x:
%    mean_x    - stredni hodnota
%    min_x     - minimum
%    max_x     - maximum
%    mode_x    - modus
%    med_x     - median
%
% See also mean, std, var, min, max, mode, median

% Spocti zakladni statistiky
mean_x=mean(x);
min_x=min(x);
max_x=max(x);
mode_x=mode(x);
med_x=median(x);
```

Nyní zkusme funkci zavolat:

```
>> compute_statistics
??? Input argument "x" is undefined.

Error in ==> compute_statistics at 19
mean_x=mean(x); std_x=std(x); var_x=var(x);
```

Matlab hlásí chybu, protože jsme nespecifikovali vstupní argument `x`. Tak jej zadejme:

```
>> x = [1 2 2 3 3 3];
>> compute_statistics(x)

ans =

    2.3333
```

Nespecifikovali jsme výstupní argumenty, Matlab automaticky vrátil hodnotu prvního, tedy `meanx`, uloženou ve speciální proměnné `ans`. Pokud chceme získat hodnoty všech výstupních argumentů, musíme si o to explicitně říci. Stačí vykopírovat deklaraci bez klíčového slova `function`:

```
>> [mean_x,min_x,max_x,mode_x,med_x] = statistics(x)

mean_x =

    2.3333
```

```
min_x =

     1

max_x =

     3

mode_x =

     3

med_x =

    2.5000
```

Všimněme si, že hranaté závorky v tomto případě neznamenají vektor, ale seznam samostatných proměnných.

Kapitola 3

Načítaní a ukládání dat v Matlabu

Matlab umí načítat řadu formátů. Ať se už jedná o soubory textové, binární, excelovské, grafické, či zvukové. Nejčastěji se načítají textové a binární soubory. Nejjednodušší cestou, jak do Matlabu načíst externí data z disku, je použití průvodce pro import. Průvodce umožní uživateli najít potřebná data a definovat proměnné pro použití v pracovním prostoru Matlabu.

Průvodce importem spustíme následujícím postupem:

- přes hlavní menu File>Import Data
- dvojklikem na soubor v aktuální složce v adresářovém stromu Matlabu
- voláním funkce `uiimport`

Pokud nechceme využít průvodce pro import dat, můžeme použít funkci `importdata`.

3.1 Import/Export souboru MAT

Mat soubor je binárním souborem, který obsahuje data z pracovního prostředí Matlabu. Načíst soubor můžeme pomocí příkazu `load('filename.mat')`. Pokud soubor obsahuje více dat, můžeme načíst pouze několik proměnných. Pokud tedy soubor obsahuje více proměnných, a uživatel chce načíst například pouze proměnnou X, příkaz load se použije takto `load('filename.mat','X')`. Pro výpis proměnných, které daný soubor obsahuje bez jejich načtení, můžeme použít příkaz `whos -file filename.mat` . Výstup tohoto příkazu je potom následující:

```
whos -file data.mat
  Name         Size                Bytes  Class     Attributes

  X          720x540x3          1166400  uint8
  a            1x1                    8  double
  b            1x4                   32  double
  info         1x1                 1890  struct
```

Pro export dat z pracovního prostředí můžeme využít příkazu `save`. Jeho použití je obdobné jako u příkazu `load`. Všechny položky vytvořené v Matlabu se uloží do souboru filename.mat. `save('filename.mat')` Pokud chceme uložit pouze několik položek, je opět nutné příkaz modifikovat zapsáním jmen daných veličin. `save('filename.mat','X')`

```
A = [1 2 3 4; 6 7 8 9];
v = [9 12 -6 3.4 -65.21];
I = eye(4);
O = ones(2);
Z = zeros(3);
z1 = 'matlab';
z2 = 'program';
z3 = 'je';
z4 = 'Dnes je ctvrtek a spatne pocasi!';

%ulozeni a nacteni objektu v pracovnim prostoru
save moje.mat;        % Ulozi vsechny objekty do souboru
save A.txt A -ASCII; % Textove
save A.mat A;         % Binarne
load moje.mat;        % Ulozi vsechny objekty do souboru
load A.txt;           % Textove
load A.mat;           % Binarne
```

3.2 Import textových souborů

Pro otevření binárního souboru nám slouží funkce `fopen`, která má jako parametr jméno souboru, který chceme otevřít a vrací identifikátor, na který se dále odkazujeme například při čtení ze souboru nebo při uzavírání souboru. (`fid = fopen(filename)`) Pokud se nepodaří soubor otevřít, identifikátor je roven -1. Pro načtení binárních dat ze souboru se používá funkce `fread`. Této funkci dáváme jako argument identifikátor, který nám vrátila funkce `fopen`. Načtená data se ukládájí do matice (`A = fread(fid)`). Tento příklad by byl pro načtení celého souboru do matice `A`. V některých případech potřebujeme načíst jenom určitou část a k tomu použijeme další parametr `SIZE` (`A = fread(fid,size)`), kde `size` může být číslo `N`. Pro načtení `N` položek ze sloupcového vektoru, nebo `inf` pro čtení dat až do konce souboru, nebo `[M,N]` pro načtení matice položek o rozměrech M x N. Pro zápis do binárního souboru se nejčasteji používá funkce `fwrite`, která má dva parametry. První je `fid` neboli identifikátor na konkrétní soubor, který jsme si otevřeli a druhý parametr je to, co chceme do souboru zapsat. Tato funkce vrací počet elementů, které byly úspěšne zapsány do souboru. Funkce `fprintf` zapisuje data do souboru ve zvoleném formátu. (`COUNT = fprintf(fid,format,A,...)`), kde `fid` je opět identifikátor otevřeného souboru, `format` je zvolený formát, jakým se má do souboru zapisovat, `A` je proměnná (matice), která se má zapsat. Tři tečky říkají, že můžeme současně zapsat více proměnných. `count` je počet úspěšně zapsaných baitů.

Když se zapíší formátované data do souboru, musíme je také zpět načíst. K tomu slouží funkce `fscanf`. (`[A,COUNT] = fscan(fid,format,size)`), `fid` je identifikátor souboru, `format` je formát v jakém se má načítat, `size` je stejně jako u funkce `fread` parametr, který říká kolik toho chceme načíst. Výstupní parametr `A` je matice, do které načítáme a `COUNT` je počet elementů, které jsme načetli. Funkce `fgets` čte řádek ze souboru a nechává v něm znak nového řádku narozdíl od funkce `fgetl`. Pokud nastala chyba při práci se souborem, můžeme si ji zjistit pomocí funkce `ferror`. (`MESSAGE = ferror(fid)`), kde `fid` je identifikátor souboru.

Pro práci s textovými soubory nejvíce používáme funkce `dlmread`, `dlmwrite`, `textscan`. Funkce `dlmread` čte numerická data ze ASCII souboru. Parametrem je přímo jméno souboru. (např. `A = dlmread('filename.txt',",range)`) Načte obsah souboru `'filename.txt'` s oddělovačem *nt* (tabulátor) do matice **A** v zadaném rozsahu, kde v range zadáváme levý horní a pravý dolní roh v datech. (např. `range = [1 0 12 10]`) Funkce `dlmwrite` zapisuje textový soubor s definovaným oddělovačem. (např. `A = dlmwrite('filename.txt',matice,',')`) Zapíše proměnnou matice do sourboru `filename.txt`, ve kterém jsou jednotlivé prvky odděleny čárkou. Funkce `textscan` čte formátované data nebo řetězec. `C = textscan(fid,'format')`), kde `fid` je identifikátor otevřeného souboru, `format` je formát načítaného souboru a `C` je pole buněk. Pokud chceme načítat a zapisovat ASCII data oddělených čárkou, mužeme použít funkce `csvread` a `csvwrite`.

```
fid1 = fopen('soubor1.dat','w');
if fid1 == -1
  message = ferror(fid1);
  disp(message);
else
  count = fwrite(fid1, A);
  count = fwrite(fid1, v);
  count = fwrite(fid1, z4);
  fclose(fid1);
end
%nacteni binarniho souboru
fid2 = fopen('soubor1.dat', 'r');
if fid2 == -1
  message = ferror(fid2);
  disp(message)
else
  AA = fread(fid2,[2,4]);
  fclose(fid2);
end
%vytvoreni textoveho souboru
fid3 = fopen('soubor2.txt','wt');
if fid3 == -1
  message = ferror(fid3);
  disp(message);
else
  x = 0:.1:1;
  y = [x;exp(x)]';
```

```
  fprintf(fid3,'Exponential Function\n\n');
  fprintf(fid3,'%10.3f %10.5f\n', y);
  status = fclose(fid3);
end
%nacteni textoveho souboru
fid4 = fopen('soubor2.txt','rt');
title = fgetl(fid4);
[table, count] = fscanf(fid4,'%f',[11 2]);
status = fclose(fid4);
```

Pro načítání xls souboru z MS Excel se používá funkce `xlsread`. (např. `A = xlsread('filename.xls','list2','A5:E5')`). Data se načtou v rozsahu buněk A2 až E5 z listu2 ze souboru `filename.xls`.

3.3 Import obrázků

Při načítání obrázku si můžeme zvolit z mnoha formátu. Matlab zpracovává tři typy rastrových obrazů:

- indexované (vztah mezi údaji v matici obrazu - indexy a barevnou škálou). Pokud je matice obrazu třídy *double,* odpovídá první bod této matice prvnímu řádku barevné škály. Jsou-li data třídy *uint8*, nebo *uint16,* odpovídá první bod matice obrazu druhého řádku škály, druhý bod třetímu, atd. Je zde posunutí o 1.
- ve škále šedi (intensity images) každý prvek datové matice odpovídá jednomu pixelu v obraze, data mohou být ve tříde *double, uint8* a *uint16*.
- RGB (truecolor), barva 1 pixelu se skládá ze tří barev červené, zelené a modré. Nejvyšší hodnota, jakou můžeme každé barvě nastavit, je 1 a nejnižší 0 při reprezentaci *double* a 0 až 255 při reprezentaci *uint8*. Podle toho, jaké kombinace hodnot u jednotlivých barev použijeme, dostaneme výslednou barvu.

Pro čtení rastrových obrázku většinou používáme tyto funkce: `imread`, `image`, `imfinfo`. Jednou z možností načtení obrázku je funkce `imread`, která má parametry *FILENAME* a *FMT*, kde *FILENAME* je jméno souboru v apostrofech a FMT je text, který specifikuje formát načítaného souboru. Tato funkce nám vrací matici, která má rozměry podle obrázku a zvoleného formátu, například

```
rgb = imread('filename.jpg','jpg');
```

Funkce `imread` podporuje tyto formáty obrázku: JPEG(Joint Photographic Experts Group), TIFF(Tagged Image File Format), GIF(Graphics Interchange Format), BMP(Windows Bitmap), PNG(Portable Network Graphics), CUR(Cursor File), HDF(Hierarchical Data Format) atd. viz. help Matlabu.

Pro získání informací o grafickém souboru lze použít příkaz `imfinfo`. Tento příkaz vrací strukturu obsahující informace o obrázku, které jsou uloženy v souboru na

disku. Například pro obrázek *pejsek.jpg* získáme informace o tomto souboru zadáním `info = imfinfo('pejsek.jpg')`. Potom struktura `info` obsahuje následující informace:

```
info =
           Filename: 'pejsek.jpg'
         FileModDate: '16-8-2011 09:29:09'
            FileSize: 130123
              Format: 'jpg'
       FormatVersion: ''
               Width: 540
              Height: 720
            BitDepth: 24
           ColorType: 'truecolor'
     FormatSignature: ''
     NumberOfSamples: 3
        CodingMethod: 'Huffman'
       CodingProcess: 'Sequential'
             Comment:
```

Pro načtení tohoto obrázku můžeme použít příkaz

```
pejsek =  imread('pejsek.jpg','jpg')
```

V pracovním prostoru Matlabu se nám objeví trojrozměrné pole `pejsek`. Pro zobrazení načteného obrázku použijeme příkaz `image(pejsek)` (obr. (3.1)). Správné zobrazení poměrů stran zajistíme pomocí příkazu `axis equal`.

3.4 Import zvukových a video souborů

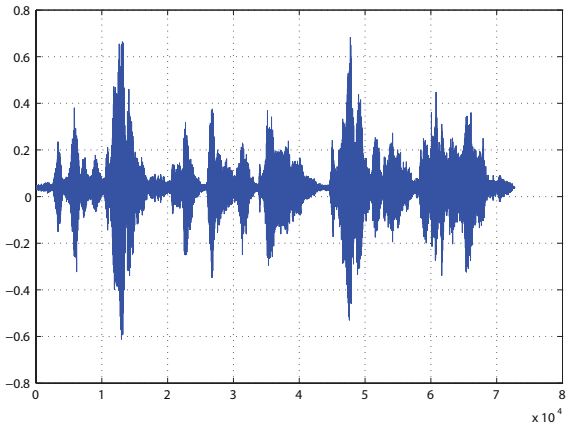
Matlab umí pracovat i s multimediálními soubory jako je video a audio signál. Audio signál je v souboru reprezentován jako série vzorků zachycující amplitudu zvuku v čase. Vzorkovací frekvence je počet samostatných vzorků za vteřinu v Hertzech. Přesnost vzorků je závislá na bitové hloubce, to jest na počtu bitů na vzorek, což je dáno dostupným hardware. Funkce v Matlabu používají pro čtení a ukládání mono signálu jeden sloupcový vektor a pro stereo signál dva vektory uložené v jedné matici. Pro stereo signál obsahuje první sloupec levý kanál a druhý sloupec pravý kanál. Audio soubor můžeme do Matlabu načíst přes menu File>Data Import, nebo například pro načtení souboru s koncovkou WAV můžeme použít příkaz `wavread`. Tento příkaz má jeden vstup *FILENAME* a dva výstupy *Y, Fs*, kde *Y* je signál a *Fs* je vzorkovací frekvence. Příklad použití příkazu:

Obrázek 3.1: Výstup příkazu `image(pejsek)`

```
[shortWave Fs] =  wavread('shortWave.wav')
```

Pro přehrání zvukové stopy je potřeba vytvořit objekt audioplayeru Matlabu a následně přehrát tento objekt příkazem `play`.

```
ShortWave_Object = audioplayer(ShortWave, Fs);
play(ShortWave_Object)
```



Obrázek 3.2: Zobrazení signálu ShortWave

Zvukovou stopu si můžete poslechnout zde:

Stejně jako u obrázků, je i u audia a videa možné zobrazit informace od daném multimediálním souboru.

```
info = mmfileinfo('shortWave.wav');

info =
    Filename: 'shortWave.wav'
        Path: 'd:\LA\Audio\Radio'
    Duration: 6.5976
       Audio: [1x1 struct]
       Video: [1x1 struct]

info.Audio =
              Format: 'PCM'
    NumberOfChannels: 1
```

Pro import videa ze souboru můžeme opět využít načítání přes menu File>Data Import, nebo přes příkazovou řádku, kde je nutné vytvořit multimediální objekt pomocí příkazu `mmreader`, a dále načtení videa proběhne pomocí příkazu `read`. Informace o videu lze vyvolat pomocí příkazu `get` s argumentem multimediálního objektu.

```
vidObject = mmreader('video.avi');
vidFrames = read(vidObject);
info = get(vidObject)

info =
          Duration: 0.6250
              Name: 'spring.avi'
              Path: 'd:\LA\Video'
               Tag: ''
              Type: 'mmreader'
          UserData: []
      BitsPerPixel: 24
         FrameRate: 24.0000
            Height: 688
    NumberOfFrames: 15
       VideoFormat: 'RGB24'
             Width: 804
```

3.5 Cvičení

Textový soubor `mpdata.dat` obsahuje dva materiálové modely. První obsahuje veličiny *EX*, *PRXY*, *DENS* a druhý obsahuje *KXX*,*C*. První dvě konstanty prvního

materiálového modelu jsou definovány pro dvě odlišné teploty. Navrhněte strukturu `MaterialModels` a naplňte ji z textového souboru.

`Výpis Souboru mpdata.dat :`

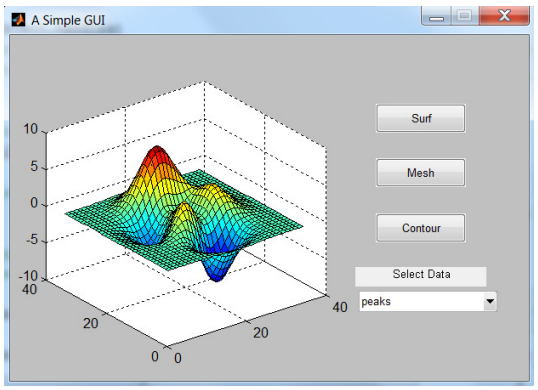
```
MPTEMP
MPTEMP,     1, 0.2000000E+02, 0.5000000E+03,
MPDATA,EX  ,       1,   1, 0.2100000E+06, 0.1900000E+06,
MPTEMP
MPTEMP,     1, 0.0000000E+00,
MPDATA,DENS,       1,   1, 0.7850000E-08,
MPTEMP
MPTEMP,     1, 0.2000000E+02, 0.5000000E+03,
MPDATA,PRXY,       1,   1, 0.3000000E+00, 0.3200000E+00,
MPTEMP
MPTEMP,     1, 0.0000000E+00,
MPDATA,KXX ,       2,   1, 0.2000000E+02,
MPTEMP MPTEMP,     1, 0.0000000E+00,
MPDATA,C   ,       2,   1, 0.3000000E+03,
```

Kapitola 4

GUI v Matlabu

GUI (graphical user interface) je grafické zobrazení v jednom či více oknech obsahující komponenty pro interaktivní obsluhu a řízení programu. GUI může obsahovat komponenty různých typů, jako jsou menu, ikony, tlačítka, list boxy, editovatelné texty nebo mohou umožňovat zobrazování dat v podobě tabulek a obrázků. Následující obrázek ukazuje jednoduché GUI, které obsahuje komponenty:

- axes pro zobrazení grafu
- pop-up menu, které umožňuje měnit příkazy matlabu
- statický text pro popis pop-up menu
- tři tlačítka, která umožňují změnu zobrazení grafu



Obrázek 4.1: Jednoduché GUI

Toto GUI funguje tak, že uživatel vybere příkaz z pop-up menu.

Každá komponenta, i GUI samotné, má jednu nebo více rutin (spustitelných kódů Matlabu), kterým se říká tzv. *callback* funkce, které žádají Matlab o provedení příslušné akce. Vyvolání definované callback funkce proběhne vždy po určité akci provedené uživatelem, například stiskem tlačítka na obrazovce, zadáním řetězce, nebo výběrem položky z nabídky. GUI poté reaguje na uživatelské akce.

4.1 Návrh GUI

Při vytváření grafického uživatelského rozhraní je nejdůležitější jeho návrh. Než tedy začneme vytvářet GUI, je nutné rozhodnout

- kdo bude uživatel GUI
- co od GUI očekáváme
- jak bude uživatel s GUI pracovat
- jaké kompotenty bude GUI obsahovat pro splnění funkčnosti

Z tohoto výčtu je patrné, že při návrhu software musíme nejdříve pochopit účel nového GUI a dále pochopit požadavky pro uživatelské ovládání aplikace.

Je dobré si nejdříve GUI rozvrhnout například na papír, kde si můžeme řádně rozmyslet celkový koncept a rozmístění jednotlivých komponent. Tento postup velice urychluje postup při následném vytváření GUI.

Po zprovoznění uživatelského rozhraní je nutné jej otestovat pro ujištění, že se chová v reálných podmínkách tak jak bylo zamýšleno. Testování je dobré svěřit jiné osobě než je jeho tvůrce, vzhledem ke snadnějšímu odhalení chyb (volí například jiné cesty ovládání než byly zamýšleny tvůrcem).

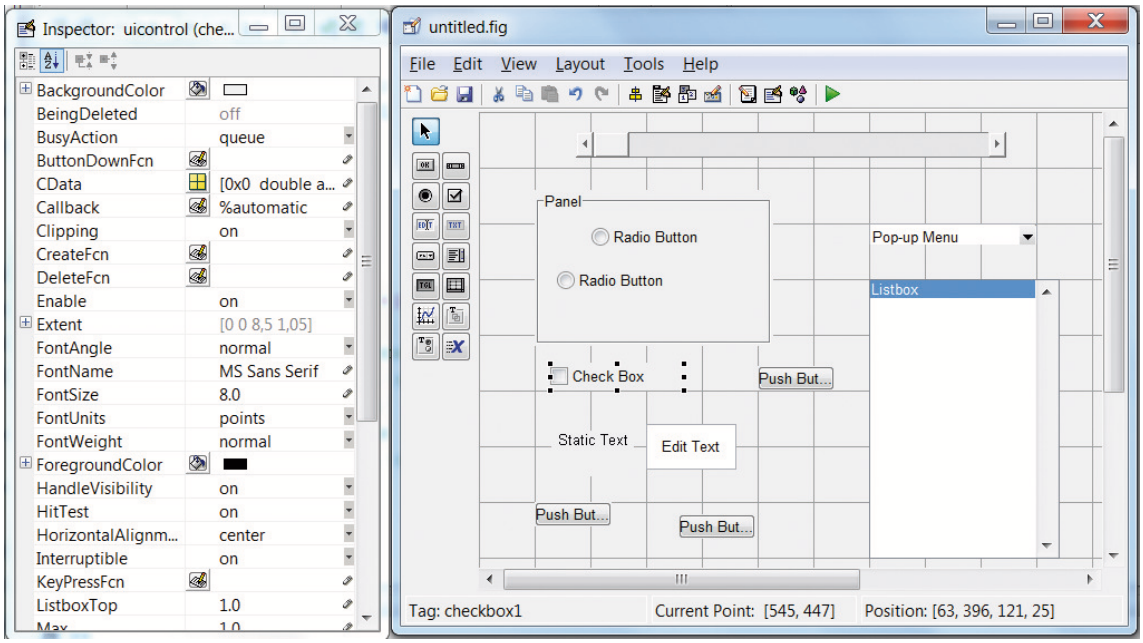
4.2 Tvorba GUI

Matlab GUI je okno, které obsahuje ovládací prvky, jejichž rozmístění si můžeme zvolit. Pomocí callback funkcí můžete ovládat jednotlivé komponenty a řídit tak chování úživatele při používání software.

GUI v Matlabu můžeme vytvořit dvěma způsoby:

- s použitím GUIDE (GUI Development Environment)
- vytvořením souboru obsahující kód tvořící GUI

První přístup s použitím GUI Development Environment umožňuje uživateli Matlabu vytvořit grafický návrh GUI pomocí nástroje implementovaného v Matlabu.

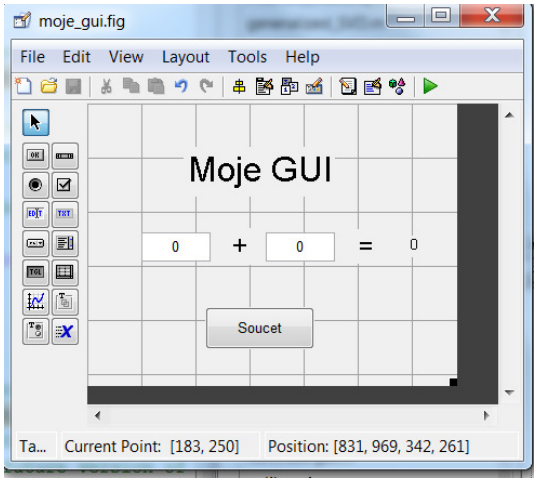


Obrázek 4.2: GUI Development Environment

GUIDE při spuštění vytvoří okno NameGUI.fig, ve kterém lze velice snadno pomocí dostupných ikon sestavit potřebný návrh GUI. Lze zde vkládat jednotlivé komponenty a nastavovat jim pomocí okna Inspector jejich atributy. Lze jednoduše seřazovat, zarovnávat jednotlivá tlačítka, vytvářet oddělené panely nebo vytvářet menu. Po uložení návrhu GUI se vytvoří dva soubory NameGUI.fig a NameGUI.m. První obsahuje okno s rozvržením polohy jednotlivých komponent, druhý je řídícím kódem pro GUI. S použitím GUIDE lze poté snadno GUI oživit přidáním potřebných callback funkcí.

GUI lze ovšem vytvořit i přímo pomocí přímého naprogramování. Lze tedy napsat pouze řídící kód GUI NameGUI.m, který bude obsahovat i vytvoření jednotlivých komponent a případnou modifikaci jejich atributů.

Příklad Vytvoření jednoduchého GUI pro sčítání dvou čísel s využitím GUIDE.

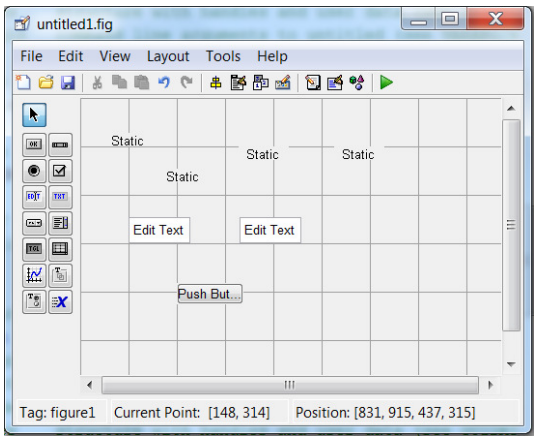


Obrázek 4.3: GUI pro součet dvou čísel

Pro vytvoření GUI podle obrázku potřebujeme tři základní typy komponent a to:

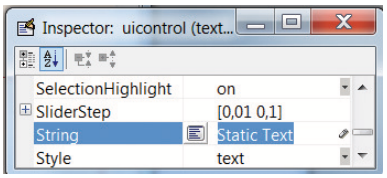
- dvě editovatelná textová okna
- čtyři statická textová okna
- jedno tlačítko

Zadáním příkazu guide se spustí GUIDE quick start okno, kde zvolíme Blank GUI (Default). Poté se objeví nové GUI s možností přidání jednotlivých komponent.

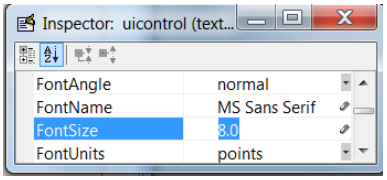


Jednoduše tedy přidáme do okna GUI všechny potřebné komponenty, jak je vidět na obrázku.

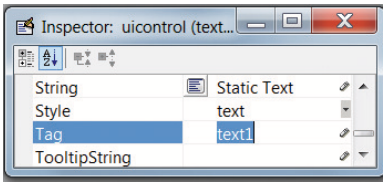
Dále můžeme editovat jednotlivé vlastnosti komponent. Poklikáním na jednotlivé komponenty se otevře *Property Inspector*, kde můžeme změnit jednotlivé atributy příslušející dané komponentě. Například pro naše GUI potřebujeme změnit parametr *String* ze `Static Text` na `+`.



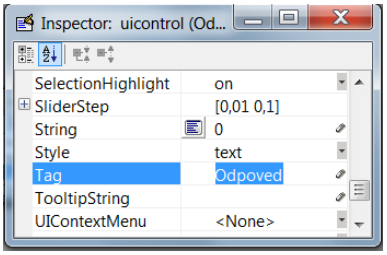
Dále můžeme změnit například velikost písma.



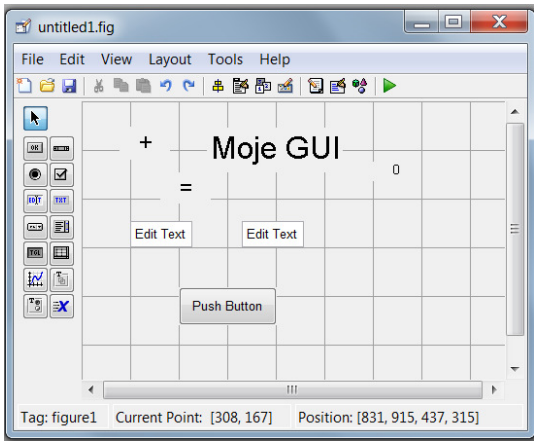
Stejným postupem změníme jednotlivé atributy u následujících statických textů a to změnou na `=` a na nadpis `Moje GUI`. Poslední statický text bude vracet hodnotu součtu, proto jeho atribut *String* nastavíme na `0`. Dále nastavíme jeho *Tag*, což je vlastně název komponenty. Nastavíme tedy jméno komponenty na `Odpoved`.



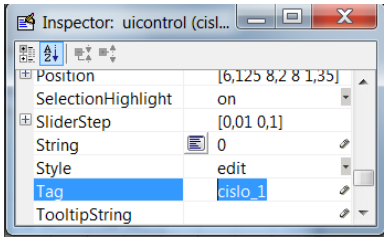
Nastavíme tedy jméno komponenty na `Odpoved`.



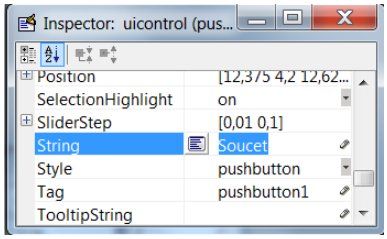
Nyní by náš návrh gui měl vypadat následovně



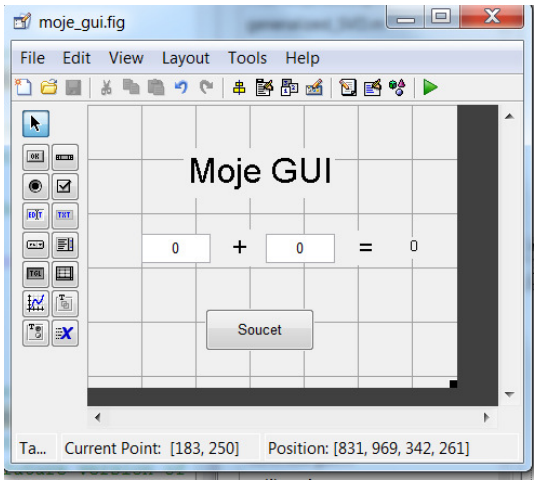
Dalším krokem bude nastavení atributů *String* a *Tag* pro editovatelné textové pole. Jako jméno komponenty pro první cifru v součtu zvolíme např. `cislo_1` a *String* nastavíme na `0`. Obdobně nastavíme *Tag* pro druhé editovatelné pole na `cislo_2`.



Poslední komponentou pro editaci atributů je komponenta klikacího tlačítka, kde změnímě *String* např. na `Soucet`.



Po vhodném uspořádání jednotlivých komponent by okno gui mělo vypadat jako na následujícím obrázku.



Nyní můžeme GUI uložit. Při ukládání se vytvoří dva soubory: moje_gui.fig a moje_gui.m. První ze souborů obsahuje okno právě vytvořeného GUI a druhý soubor obsahuje řídící kód GUI, který je uveden níže.

```
function varargout = moje_gui(varargin)
% MOJE_GUI M-file for moje_gui.fig
%      MOJE_GUI, by itself, creates a new MOJE_GUI or raises the existing
%      singleton*.
%
%      H = MOJE_GUI returns the handle to a new MOJE_GUI or the handle to
%      the existing singleton*.
%
%      MOJE_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MOJE_GUI.M with the given input arguments.
%
%      MOJE_GUI('Property','Value',...) creates a new MOJE_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before moje_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to moje_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help moje_gui
% Last Modified by GUIDE v2.5 09-Sep-2011 00:50:36
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1; gui_State = struct('gui_Name',mfilename,...
                   'gui_Singleton',  gui_Singleton,...
                   'gui_OpeningFcn', @moje_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @moje_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% -- Executes just before moje_gui is made visible.
function moje_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to moje_gui (see VARARGIN)
% Choose default command line output for moje_gui
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes moje_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% -- Outputs from this function are returned to the command line.
function varargout = moje_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
function cislo_1_Callback(hObject, eventdata, handles)
% hObject    handle to cislo_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of cislo_1 as text
%        str2double(get(hObject,'String')) returns contents of cislo_1 as a double
% -- Executes during object creation, after setting all properties.
function cislo_1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cislo_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function cislo_2_Callback(hObject, eventdata, handles)
% hObject    handle to cislo_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of cislo_2 as text
%        str2double(get(hObject,'String')) returns contents of cislo_2 as a double
% -- Executes during object creation, after setting all properties.
```

```
function cislo_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cislo_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor')
 get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% -- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Nyní lze snadno editací jednotlivých callback funkcí vytvořit funkční GUI. Jako první přidáme funkčnost editovatelným textovím polím. Takže do funkce:

```
function cislo_1_Callback(hObject, eventdata, handles)
% hObject    handle to cislo_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of cislo_1 as text
%        str2double(get(hObject,'String')) returns contents of cislo_1 as a double
```

přidáme následující řádky

```
vstup = str2num(get(hObject,'String'));

if (isempty(vstup))
     set(hObject,'String','0')
end
guidata(hObject, handles);
```

Tato část kódu zajišťuje správnost zadaného vstupu. Je možné zadávat pouze číselné hodnoty. Poslední řádek aktualizuje ukazatele v GUI, neboli zachovává aktuální ukazatele vždy po volání *callback* funkce. Stejnou část kódu můžeme vložit i do funkce `cislo_2_Callback`.

Nyní budeme editovat callback funkci pro tlačítko Soucet

```
% -- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

přidáním kódu

```
a = get(handles.cislo_1,'String');
b = get(handles.cislo_2,'String');
SOUCET = str2num(a) + str2num(b);
c = num2str(SOUCET);
set(handles.Odpoved,'String',c);
guidata(hObject, handles);
```

První dva řádky po stisku tlačítka *Soucet* přiřadí *string* v editovatelných polích do proměnných *a, b*, dále se provede součet těchto dvou čísel a nastaví se výsledný součet do statického textu jako výstup pomocí příkazu *set*. Po uložení těchto změn, lze GUI spustit příkazem `moje_GUI`.

Část II

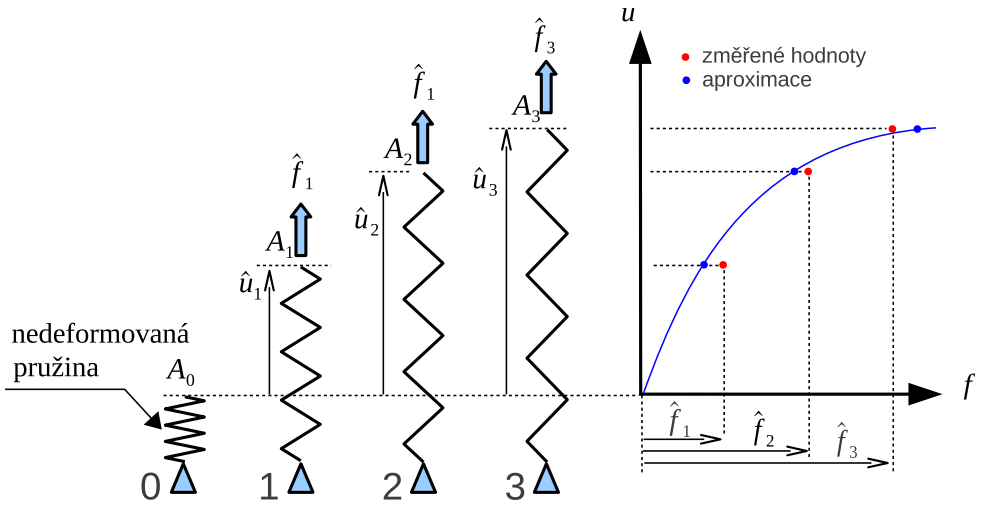
Soustavy rovnic

Kapitola 5

Obecné řešení soustav lineárních rovnic

5.1 Přeurčená soustava rovnic

Začněme motivačním příkladem.



Obrázek 5.1: Pružina.

Na obr. 5.1 je pružina, u které chceme definovat závislost mezi deformací u a silou f. Pokud roste tuhost s deformací, obvykle se uvažuje kubická závislost

$$f(u) = k_1 u + k_2 u^2. \tag{5.1}$$

Na obr. 5.1 je zakresleno několik případů zatížení. Pružina je nejprve nedoformována, v dalších krocích se postupně zatěžuje známými silami a vzniklé posuvy jsou měřeny. Vektor změřených posunutí $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_n)^T$ je spárován s vektorem známých sil

$\hat{\mathbf{f}} = (\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_n)^T$. Podle rovnice (5.1) můžeme pro neznáme tuhostní paremetry k_1 a k_2 psát

$$\begin{aligned} \hat{f}_1 &= k_1\hat{u}_1 + k_2\hat{u}_1^2 \\ \hat{f}_2 &= k_1\hat{u}_2 + k_2\hat{u}_2^2 \\ &\vdots \\ \hat{f}_n &= k_1\hat{u}_n + k_2\hat{u}_n^2, \end{aligned}$$

nebo maticovou symbolikou

$$\hat{\mathbf{f}} = \mathbf{U}\mathbf{k}, \tag{5.2}$$

kde

$$\mathbf{U} = \begin{pmatrix} \hat{u}_1 & \hat{u}_2 & \cdots & \hat{u}_n \\ \hat{u}_1^2 & \hat{u}_2^2 & \cdots & \hat{u}_n^2 \end{pmatrix}^T.$$

Obecně nemusí existovat žádný vektor $\mathbf{k} = (k_1,\ k_2)^T$, pro který by rovnost v (5.2) platila, což lze jednoduše ukázat. Experimentátor provede měření pro sílu 30 N, nasledně pro 60 N a opět pro 30 N. Zjistí, že aby docílil v posledním měření stejné deformace jako v měření prvním, musí sílu navýšit o 0.05 N na 30.05 N. Rozdíl sil může být způsoben např. chybným odečítáním z použitého měřidla, nebo změnou teploty, na která tuhost pružiny také závisí. Nezávisle na příčině jsme obdrželi tři rovnice:

$$\begin{aligned} \text{měření I.} \quad & k_1 2.9 + k_2 2.9^2 = 30 \\ \text{měření II.} \quad & k_1 5.6 + k_2 5.6^2 = 60 \\ \text{měření III.} \quad & k_1 2.9 + k_2 2.9^2 = 30.05, \end{aligned}$$

z nichž I. a III. se liší pouze pravou stranou. Proto pro vzniklou soustavu neexistuje žádná dvojce k_1, k_2 (resp. vektor $\mathbf{k}$), pro kterou by byly rovnice I. a III. splněny současně ($\hat{\mathbf{f}} \notin \mathrm{Im}\mathbf{U}$). Zaveďme tzv. reziduum ve tvaru

$$\mathbf{r} = \hat{\mathbf{f}} - \mathbf{U}\mathbf{k}$$

a vektor $\mathbf{k}$ hledejme takový, pro který bude $||\mathbf{r}||$ minimální (konvexní funkce mající právě jedno minimum).

5.1.1 Řešení přeurčených soustav

Metoda nejmenších čtverců

Se znalostí vlastností $||\mathbf{r}||$ zaveďme

$$||\mathbf{r}||^2 = (\mathbf{r},\ \mathbf{r}) = \left(\hat{\mathbf{f}},\ \hat{\mathbf{f}}\right) + (\mathbf{U}\mathbf{k},\ \mathbf{U}\mathbf{k}) - 2\left(\hat{\mathbf{f}},\ \mathbf{U}\mathbf{k}\right), \tag{5.3}$$

což je hladká konvexní funkce mající jediné minimum (shodné s $\min(||\mathbf{r}||)$). Minimalizací $||\mathbf{r}||^2$ obdržíme

$$\mathbf{k} = \left(\mathbf{U}^\mathbf{T}\mathbf{U}\right)^{-1}\mathbf{U}^T\hat{\mathbf{f}}. \tag{5.4}$$

Mooreova-Penroseova inverze

Stejné řešení dostaneme aplikací $\mathbf{U}^+$ na vektor pravých stran, což je tzv. Mooreova-Penroseova inverze a můžeme ji spočítat pomocí SVD rozkladu (kapitola 13). Koeficienty tuhosti obdržíme z

$$\mathbf{k} = \mathbf{U}^+\hat{\mathbf{f}}. \tag{5.5}$$

QR rozklad

V Matlabu lze úlohu vyřešit dle $\mathbf{k} = \mathbf{U}\backslash\hat{\mathbf{f}}$. Pro obdélníkovou matici se vždy použije QR rozklad. Chceme-li pracovat s maticemi přímo, můžeme zapsat příkaz `[Q,R]=qr(U,0)`. Jelikož $\mathbf{U}$ je obdélníková, druhý argument „0“ zajistí, že matice $\mathbf{R}$ bude čtvercová ($\mathbf{R}$ má obecně shodný rozměr jako matice na vstupu) a hledaný vektor spočteme dle

$$\mathbf{k} = \mathbf{R}^{-1}\mathbf{Q}^T\hat{\mathbf{f}}. \tag{5.6}$$

Příklad 5.1.

Pružina dole pevně uchycená je na druhém konci postupně zatěžována hodnotami sil 30, 60, 30.05 a 90 Newtonů a výchylky k tomu změřené jsou 2.9, 5.6, 2.9 a 8.2 milimetrů. Vypočítejte konstanty k_1 a k_2 v rovnici (5.1). Nejprve sestavíme potřebnou matici $\mathbf{U}$ a vektor $\hat{\mathbf{f}}$:

$$\mathbf{U} = \begin{pmatrix} 2.9 & 8.41 \\ 5.6 & 31.36 \\ 2.9 & 8.41 \\ 8.2 & 67.24 \end{pmatrix}, \quad \hat{\mathbf{f}} = \begin{pmatrix} 30 \\ 60 \\ 30.05 \\ 90 \end{pmatrix}$$

Nyní stačí vybrat jeden z postupů (5.4), (5.5), (5.6) a vyřešením obdržíme hledané konstanty

$$k_1 = 10.0489 \text{ a } k_2 = 0.1138.$$

Zpětným dosazením známých posuvů do (5.1) obdržíme aproximaci sil

$$\mathbf{f} = \mathbf{U}\mathbf{k} = \begin{pmatrix} 30.0992 \\ 59.8434 \\ 30.0992 \\ 90.0545 \end{pmatrix}.$$

Nakolik je navržená kubická závislost (5.1) mezi posuvem a silou platná, můžeme měřit velikostí normy rezidua vztaženého k velikosti normy $||\hat{\mathbf{f}}||$

$$100\frac{||\mathbf{r}||}{||\hat{\mathbf{f}}||} = 0.1715\%$$

tzv. relativní chybou.

5.2 Nedourčená soustava rovnic

5.2.1 Čtvercová matice

Uvažujme pružinu na obr. (5.2) vlevo s lineární charakteristikou

$$f = k\Delta u, \tag{5.7}$$

kde k je tuhost pružiny (nezáporná), $\Delta u = u_b - u_a$ je změna délky pružiny (rozdíl posunutí) a f síla přímo uměrná Δu . Oproti předešlému příkladu $k = k_1$ a $k_2 = 0$. V bodě a je pružina uchycena ($u_a = 0$) a v bodě b tažena silou f.

Obrázek 5.2: Řešení.

Rovnice rovnováhy pro jednu pružinu v bodech a a b jsou

$$\begin{aligned} k\,(u_a - u_b) &= f_a = -f \\ k\,(-u_a + u_b) &= f_b = f \end{aligned}$$

a maticově

$$\mathbf{K}\mathbf{u} = \mathbf{f}. \tag{5.8}$$

Je-li pružina tažena na pravém konci silou f, na levém bude působit stejně velká síla s opačným znaménkem. Úkolem je spočíst a vyjádřit vektor posuvů konců pružiny. Pokud bychom chtěli soustavu vyřešit pomocí inverze matice soustavy, zjistili bychom, že to není možné, neboť její inverze neexistuje. Neexistující inverzi nahradíme Mooreovou-Penroseovou inverzí.

Součtem jeho partikulárního a homogenního řešení rovnice (5.8) dostáváme

$$\mathbf{u} = \mathbf{u}_p + \mathbf{u}_h. \tag{5.9}$$

Partikulární část spočítáme z rovnice

$$\mathbf{u}_p = \mathbf{K}^+\mathbf{f}, \tag{5.10}$$

a homogenní z

$$\mathbf{u}_h = \mathbf{N}\mathbf{g}. \tag{5.11}$$

Lineární obal sloupců matice $\mathbf{N}$ tvoří celé jádro matice $\mathbf{K}$ (zde $\mathbf{N} = (1,1)^T$) a platí, že

$$\mathbf{K}\mathbf{N} = \mathbf{O}. \tag{5.12}$$

Zpětným dosazením řešení $\mathbf{u}$ do rovnice (5.8) obdržíme

$$\mathbf{K}\left(\mathbf{K}^+\mathbf{f} + \mathbf{N}\mathbf{g}\right) = \mathbf{K}\mathbf{K}^+\mathbf{f}.$$

Zde součin $\mathbf{K}\mathbf{K}^+$ je projektor na Im$\mathbf{K}$ (viz [1]) a jelikož $\mathbf{f} \in$ Im$\mathbf{K}$, pak platí

$$\mathbf{K}\mathbf{K}^+\mathbf{f} = \mathbf{f}.$$

Rovnice rovnováhy má v závislosti na $\mathbf{g}$ nekonečně mnoho řešení. Jsou-li zadány okrajové podmínky (v našem případě je pružina pevně uchycena v bodě a, tj. $u_a = 0$), potom se spočítá $\mathbf{g}$ tak, aby jim bylo vyhověno a ze všech možných řešení splňujících rovnici (5.8) vybíráme pouze jediné, pro které je posuv levého konce pružiny skutečně nulový.

Příklad 5.2.

Spočtěte deformaci pružiny na obr. 5.2 s tuhostí $k = 1$, která je na levém konci držena a na pravém konci zatížena silou $f = 1$. Rovnice rovnováhy je

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} u_a \\ u_b \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

K výpočtu partikulárního řešení dle (5.10) vyčíslíme Mooreovou-Penroseovou inverzi matice $\mathbf{K}$ a to

$$\mathbf{K}^+ = \frac{1}{4}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

a obdržíme

$$\mathbf{u}_p = \begin{pmatrix} -0.5 \\ 0.5 \end{pmatrix}.$$

Homogenní řešení je

$$\mathbf{u}_h = \mathbf{N}\mathbf{g}, \ \mathbf{N} = (1\ 1)^T$$

s neznámou $\mathbf{g} \in \mathbb{R}^n$ (jelikož $n = 1$, potom $\mathbf{g} = g$). Dosazením do (5.9) ($u_a = 0$) píšeme

$$0 = -0.5 + g$$
$$u_b = 0.5 + g.$$

Z první rovnice spočítáme, že $g = 0.5$ a ze druhé hledanou deformaci $u_b = 1$.

5.2.2 Obdélníková matice

V kapitole 16 je odvozena rovnice rovnováhy struny.

Příklad 5.3.

Pro strunu délky 1 m pevně uchycenou v krajních bodech, s pěti uzly (krok sítě $h = 0.25$), zatíženou jednotkovou hustotou sil použitím metody sítí (kapitola 17) obdržíme soustavu rovnic 5.8 rozepsanou

$$\begin{pmatrix} -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{pmatrix} = 0.25^2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},$$

což jsou tři rovnice rovnováhy ve vnitřních uzlech struny (všechny uzly vyjma krajních). Matice soustavy je obdélníková a k výpočtu partikulárního řešení dojdeme rovnicí (5.10)

$$\mathbf{u}_p = \begin{pmatrix} -0.0625 & 0.03125 & 0.0625 & 0.03125 & -0.0625 \end{pmatrix}^T.$$

Homogenní řešení

$$\mathbf{u}_h = \begin{pmatrix} 0 & 0.25 & 0.5 & 0.75 & 1 \\ 1 & 0.75 & 0.5 & 0.25 & 0 \end{pmatrix}^T \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}$$

s libovolým vektorem $\mathbf{g} = (g_1, g_2)^T$ v součtu s řešením partikulárním splňuje podmínku rovnováhy (5.10). Abychom vyhověli okrajovým podmínkám ($u_1 = u_5 = 0$), volíme $g_1 = g_2 = 0.0625$. Deformace struny ve všech uzlech sítě bude

$$\mathbf{u} = \begin{pmatrix} 0, & 0.09325, & 0.125, & 0.09325, & 0 \end{pmatrix}^T.$$

5.3 Příklady soustav se čtvercovou maticí

V řadě technických problému se vyskytují matice, které jsou čtvercové.

5.3.1 Regulární matice soustavy - jediné řešení

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{pmatrix} \text{ a } \mathbf{b} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}.$$

V výpočtu v Matlabu použijeme zpětné lomítko a potvrdíme klávesou Enter. V příkazové řádce by se nám mělo objevit

```
>> x=A\b

x =

     5
    -1
     0
     0
```

K výpočtu byla použita Gaussova eliminace, která je např. oproti způsobu

```
>> x=inv(A)*b
```

ve kterém se inverze skutečně spočítala, rychlejší, má nižší šíření chyb, menší nároky na paměť. Použijeme-li zpětné lomítko, Matlab na pozadí vyhodnocuje vlastnosti konkrétní matice soustavy a k řešení použije optimální metodu (Choleského, LU, QR rozklad apod.).

5.3.2 Singulární matice soustavy - případ s nekonečně mnoho řešeními

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 4 & 4 \end{pmatrix} \text{ a } \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}.$$

Je-li matice soustavy singulární (v této soustavě rovnic je třetí rovna součtu předešlých dvou), zpětné lomítko vrátí hodnotu NaN. Soustavu lze vyřešit pomocí Mooreovy-Penroseovy inverze zápisem

```
x=pinv(A)*b

x =

    0.1667
    0.1667
    0.1667
```

Uvedený způsob minimalizje normy vektorů $\mathbf{Ax} - \mathbf{b}$ a $\mathbf{x}$.

5.3.3 Singulární matice soustavy - případ, kdy žádné přesné řešení neexistuje

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 4 & 4 \end{pmatrix} \text{ a } \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}.$$

Matice soustavy $\mathbf{A}$ je singulární (shodná s maticí předešlého příkladu), ale pro prvky vektoru $\mathbf{b}$ neplatí $b_3 = b_1 + b_2$, tzn. poslední rovnice není lineární kombinací předešlých dvou. K výpočtu opět použijeme Mooreovu-Penroseovu inverzi

```
x =

    0.2222
    0.2222
    0.2222
```

Pro další možný způsob výpočtu rozšíříme soustavu o nulový řádek

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 4 & 4 \\ 0 & 0 & 0 \end{pmatrix} \text{ a } \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 0 \end{pmatrix},$$

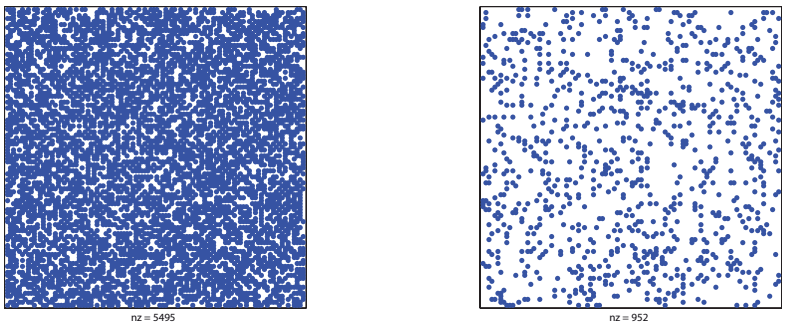
čímž „přinutíme“ Matlab použít QR rozklad, který minimalizuje normu $\mathbf{Ax} - \mathbf{b}$.

Kapitola 6

Ukládání a práce s řídkými maticemi

V mnoha případech použití numerické matematiky se při sestavení matic obsahujících různorodý typ dat stává, že matice obsahuje mnohem menší počet nenulových prvků než je celkový počet prvků dané matice. Tyto matice mají tzv. malou hustotu zaplnění a mluvíme o nich jako o řídkých maticích. Při výpočtu s použitím řídkých matic je efektivnější ukládat pouze nenulové prvky dané matice. Existuje celá řada základních předpisů pro ukládání řídkých matic, ale většina z nich je založena na stejném principu. Základním principem je uložení všech nenulových prvků matice do jednorozměrného pole, jež má přístup k pomocným polím, které popisují polohu umístění nenulových prvků v řídké matici. Způsob vytvoření pomocných polí pak odlišuje různé přístupy ukládání řídkých matic.

Ukládání nenulových prvků řídké matice do jednorozměrného pole se provádí průchodem celé matice po jednotlivých řádcích (řádková komprese) nebo po jednotlivých sloupcích (sloupcová komprese) a postupném ukládání nenulových prvků do jednorozměrného pole v pořadí daném průchodem řídké matice. Při ukládání symetrických matic je nutné ukládat pouze horní trojúhelníkovou nebo dolní trojúhelníkovou část matice. Tři základní typy komprese řídkých matic jsou uvedeny níže.



Obrázek 6.1: Hustá vs. řídká matice

6.1 CSR Formát

V tomto odstavci je detailně popsána struktura ukládání řídké matice založená na řádkové kompresi *CSR* (compressed sparse row format). *CSR* formát je jeden ze základních typů komprese řídkých matic. Struktura formátu *CSR* je dána třemi nebo čtyřmi poli:

1. $[hodnoty, sloupce, indexyRadku]$
2. $[hodnoty, sloupce, ukazatelA, ukazatelB]$

Následující tabulka popisuje tato pole v závislosti na prvcích řídké matice a jejich poloze vzhledem k sloupcům a řádkům matice.

Varianta 1

hodnoty Reálné nebo komplexní pole obsahující nenulové prvky matice **A**. Jednotlivé prvky matice **A** jsou mapovány do pole *hodnoty* s použitím řádkové komprese.

sloupce I-tý prvek pole čísel typu integer *sloupce* obsahuje číslo sloupce v matici **A** příslušející I-tému prvku v poli *hodnoty*

indexyRadku J-tý prvek pole typu integer *indexyRadku* vrací index prvku v poli *hodnoty*, který je první nenulový prvek v J-tém řádku matice **A**

Délka pole *hodnoty* a *sloupce* je dána počtem nenulových prvků matice **A**. Vzhledem k tomu, že pole *indexyRadku* obsahuje polohu prvního nenulového prvku v řádku, a nenulové prvky jsou uloženy za sebou, je tedy počet nenulových prvků v I-tém řádku roven rozdílu $indexyRadku\,(I+1) - indexyRadku\,(I)$. Aby tento vztah platil i pro poslední řádek matice, je nutné na konec pole *indexyRadku* přidat integer jehož hodnota je rovna celkovému počtu nenulových prvků +1. Proto je délka pole *indexyRadku* o jedničku vyšší než je počet řádků v matici.

Varianta 2 První dvě pole *hodnoty*, *sloupce*, jsou shodné s variantou 1. Pole *ukazatelA* je shodné s polem *indexyRadku*, s tím rozdílem, že neobsahuje poslední hodnotu tohoto pole.

ukazatelA J-tý prvek pole čísel typu integer *ukazatelA* vrací index prvku v poli *hodnoty* který je první nenulový prvek v J-tém řádku matice **A**.

ukazatelB pole čísel typu integer obsahující indexy řádků, a to tak, že *ukazatelB(J)--ukazatelA(1)* je index prvku v poli *hodnoty*, který je posledním nenulovým prvkem v J-tém řádku matice **A**.

Příklad 6.1. Mějme čtvercovou matici **A**:

$$\mathbf{A}=\begin{bmatrix} 2 & -5 & * & -2 & * \\ -5 & 3 & * & * & * \\ * & * & 2 & 4 & * \\ -7 & * & * & 8 & * \\ * & * & 7 & * & -13 \end{bmatrix}$$

Převedením této matice do *CSR* formátu dostaneme jednorozměrná pole jednoznačně popisující řídkou matici. Pole pro variantu 1 je uvedeno v tabulce 6.1, pole pro variantu 2 v tabulce 6.2.

číslování od jedničky												
hodnoty	=	[	2	-5	-2	-5	3	2	4	-7	8	7	-13	]
sloupce	=	[	1	2	4	1	2	3	4	1	4	3	5	]
indexyRadku	=	[	1	4	6	8	10	12	]					
číslování od nuly														
hodnoty	=	[	2	-5	-2	-5	3	2	4	-7	8	7	-13	]
sloupce	=	[	0	1	3	0	1	2	3	0	3	2	4	]
indexyRadku	=	[	0	3	5	7	9	11	]					

Tabulka 6.1: Pole pro uložení řídké matice ve formátu *CSR Varianta1*

číslování od jedničky													
hodnoty	=	[	2	-5	-2	-5	3	2	4	-7	8	7	-13	]
sloupce	=	[	1	2	4	1	2	3	4	1	4	3	5	]
ukazatelA	=	[	1	4	6	8	10	]						
ukazatelB	=	[	4	6	8	10	12	]						
číslování od nuly														
hodnoty	=	[	2	-5	-2	-5	3	2	4	-7	8	7	-13	]
sloupce	=	[	0	1	3	0	1	2	3	0	3	2	4	]
ukazatelA	=	[	0	3	5	7	9	]						
ukazatelB	=	[	3	5	7	9	11	]						

Tabulka 6.2: Pole pro uložení řídké matice ve formátu *CSR Varianta2*

Použití tohoto způsobu ukládání matic můžeme demonstrovat na příkladu výpočtu násobení matice a vektoru. Násobení matice-vektor $\mathbf{x} = \mathbf{Ab}$ lze pomocí *CRS* formátu vyjádřit jako:

$$x_i = \sum_j a_{i,j} b_j,$$

Potom součin matice $\mathbf{A}_{m\times m}$ a vektoru b je dán kódem:

```
for i = 1:m
    x(i) = 0;
    for J = indexyRadku(i):indexyRadku(i+1)-1
        x(i)  = x(i) + hodnoty(J)*b(sloupce(J));
    end
end
```

6.2 CSC Formát

CSC (compressed sparse column format) je obdobou předchozí řádkové komprese řídkých matic, s tím rozdílem, že pro kompresi se nepoužívají řádky ale sloupce. Pokud bychom tedy použili formát *CSR* na transponovanou matici $\mathbf{A}$, obdržíme sloupcovou kompresi matice $\mathbf{A}$. Tedy

$$CSC(\mathbf{A}) = CSR(\mathbf{A}^T).$$

Tato komprese je tedy obdobně jako v předchozím případě dána třemi nebo čtyřmi poli

1. $[hodnoty, radky, indexySloupcu]$
2. $[hodnoty, radky, ukazatelA, ukazatelB]$

Následující tabulka popisuje tyto pole v závislosti na prvcích řídké matice a jejich poloze vzhledem k sloupcům a řádkům matice.

Varianta 1

hodnoty Reálné nebo komplexní pole obsahující nenulové prvky matice $\mathbf{A}$. Jednotlivé prvky matice $\mathbf{A}$ jsou mapovány do pole *hodnoty* s použitím sloupcové komprese.

radky I-tý prvek pole typu integer *radky* obsahuje číslo řádku v matici $\mathbf{A}$ příslušející I-tému prvku v poli *hodnoty*.

indexySloupcu J-tý prvek pole typu integer *indexySloupcu* vrací index prvku v poli *hodnoty*, který je první nenulový prvek v J-tém sloupci matice $\mathbf{A}$.

Délka pole *hodnoty* a *sloupce* je opět dána počtem nenulových prvků matice $\mathbf{A}$. Jako u *CSR* formátu, je nutné poslední pole čísel typu integer upravit podle stejného klíče. Pole *indexySloupcu* obsahuje polohu prvního nenulového prvku ve sloupci, nenulové prvky jsou uloženy za sebou, počet nenulových prvků v I-tém sloupci je tedy roven rozdílu $indexySloupcu\,(I+1) - indexySloupcu\,(I)$. Aby tento vztah platil i pro poslední sloupce matice, je nutné na konec pole *indexySloupcu* přidat integer jehož hodnota je rovna počtu nenulových prvků +1. Proto je délka pole *indexySloupcu* o jedničku vyšší než je počet sloupců v matici.

Varianta 2 První dvě pole *hodnoty*, *sloupce*, jsou shodné s variantou 1. Pole *ukazatelA* je shodné s polem *indexySloupcu*, s tím rozdílem, že neobsahuje poslední hodnotu tohoto pole.

ukazatelA *J*-tý prvek pole typu integer *ukazatelA* vrací index prvku v poli *hodnoty* který je první nenulový prvek v *J*-tém sloupci matice **A**.

ukazatelB Pole typu integer obsahující indexy sloupců, a to tak, že *ukazatelB(J)-ukazatelA(1)* je index prvku v poli *hodnoty*, který je posledním nenulovým prvkem v *J*-tém sloupci matice **A**.

Příklad 6.2. Pro čtvercovou matici **A** z předchozího příkladu vytvořte pole popisující sloupcovou kompresi *CSC*. Převedením řídké matice do *CSC* formátu dostaneme jednorozměrná pole popsaná výše. Pole pro variantu 1 je uvedeno v tabulce 6.3, pole pro variantu 2 v tabulce 6.4.

číslování od jedničky												
hodnoty	=	[2	-5	-7	-5	3	2	7	-2	4	8	-13]
radky	=	[1	2	4	1	2	3	5	1	3	4	5]
indexySloupcu	=	[1	4	6	8	11	12]					
číslování od nuly												
hodnoty	=	[2	-5	-7	-5	3	2	7	-2	4	8	-13]
radky	=	[0	1	3	0	1	2	4	0	2	3	4]
indexySloupcu	=	[0	3	5	7	10	11]					

Tabulka 6.3: Pole pro uložení řídké matice ve formátu *CSC Varianta1*

číslování od jedničky												
hodnoty	=	[2	-5	-7	-5	3	2	7	-2	4	8	-13]
radky	=	[1	2	4	1	2	3	5	1	3	4	5]
ukazatelA	=	[1	4	6	8	11]						
ukazatelB	=	[4	6	8	11	12]						
číslování od nuly												
hodnoty	=	[2	-5	-7	-5	3	2	7	-2	4	8	-13]
radky	=	[0	1	3	0	1	2	3	0	3	2	4]
ukazatelA	=	[0	3	5	7	10]						
ukazatelB	=	[3	5	7	10	11]						

Tabulka 6.4: Pole pro uložení řídké matice ve formátu *CSC Varianta2*

6.3 Souřadnicová komprese

Souřadnicový typ ukládání řídkých matic je jedním z nejjednodušších postupů komprese. Tento typ ukládání řídkých matic je také použit v Matlabu při zadávání řídké

matice. Systém ukládání matice je dán třemi poli, kde jsou uloženy pouze nenulové prvky matice a souřadnice jednotlivých prvků. Mějme tedy tři pole *hodnoty, sloupce, radky*, které obsahují:

hodnoty Reálné nebo komplexní pole obsahující nenulové prvky matice **A**. Jednotlivé prvky matice **A** jsou mapovány do pole *hodnoty* s použitím sloupcové komprese.

radky I-tý prvek pole typu integer *radky* obsahuje číslo řádku v matici **A** příslušející I-tému prvku v poli *hodnoty*

sloupce I-tý prvek pole typu integer *sloupce* obsahuje číslo sloupce v matici **A** příslušející I-tému prvku v poli *hodnoty*

Příklad 6.3. Pro čtvercovou matici **A** z příkladu 1 vytvořte pole popisující sloupcově orientovanou souřadnicovou kompresi (pole *hodnoty* vytvářeno postupně po sloupcích, pole *sloupce* je neklesající).

číslování od jedničky													
hodnoty	=	[	2	-5	-7	-5	3	2	7	-2	4	8	-13	]
radky	=	[	1	2	4	1	2	3	5	1	3	4	5	]
sloupce	=	[	1	1	1	2	2	3	3	4	4	4	5	]
číslování od nuly													
hodnoty	=	[	2	-5	-7	-5	3	2	7	-2	4	8	-13	]
radky	=	[	0	1	3	0	1	2	4	0	2	3	4	]
sloupce	=	[	0	0	0	1	1	2	2	3	3	3	4	]

Tabulka 6.5: Pole pro uložení řídké matice v souřadnicovém formátu (*CSC* komprese prvků)

6.4 Práce s řídkými maticemi v Matlabu

Matlab nikdy nevytváří řídké matice automaticky. Uživatel musí vždy nejdříve rozhodnout, zda se vyplatí použít řídkou matici namísto matice plné. Zda použít plnou nebo řídkou matici můžeme rozhodnout pomocí faktoru hustoty zaplnění matice. Matice s malou hustotou zaplnění[1] je dobrým kandidátem pro použití zápisu řídkého formátu. Hustota zaplnění matice je dána jako podíl počtu nenulových prvků s celkovým počtem prvků v matici. V Matlabu se tedy hustota zaplnění matice určí jako:

[1] každý prvek matice je reprezentován ve třech polích, proto za „malou" hustotu zaplnění může být považována hodnota $< 0,3$

```
hustota = nnz(A)/prod(size(A));
```

Pro sestavení řídké matice v Matlabu je možné použít několik základních příkazů. Například jednotkovou matici vytvoříme příkazem

```
I = speye(m,n);
```

kde `m,n,` je rozměr jednotkové matice. Diagonální matici vytvoříme příkazem

```
D = spdiags(B,d,m,n);
```

Příkaz umístí j-tý sloupec z matice `B(:,j)` na j-tou diagonálu `d` (za nultou je považována hlavní diagonála) v matici o rozměrech `m,n`. Mějme matici

$$\mathbf{A}=\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}.$$

Diagonální řídkou matici vyskládáme pomocí následujícího kódu:

```
n = 3;
b = ones(n,1);
B = [-b 2*b -b];
D = spdiags(B,-1:1,n,n);
```

Řídkou matici s náhodnými prvky s rovnoměrným rozložením vygenerujeme příkazem `sprand(m,n,dens)`, kde `m,n` je rozměr matice a `dens` je hustota zaplnění matice.

Existující plnou matici můžeme převést na řídkou příkazem `sparse`.

```
S = sparse(A);
```

Řídkou matici můžeme převést na plnou příkazem `full`.

```
A = full(S);
```

Řídkou matici však můžeme sestavit přímo s použitím souřadnicového zápisu řídké matice. Mějme matici

$$\mathbf{A}=\begin{bmatrix} 2 & -5 & 0 & -2 & 0 \\ -5 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 4 & 0 \\ -7 & 0 & 0 & 8 & 0 \\ 0 & 0 & 7 & 0 & -13 \end{bmatrix}.$$

Použitím příkazu `sparse` s 5 argumenty `sparse(i,j,v,m,n)`, kde `i,j,v` jsou vektory reprezentující matici v souřadnicovém zápisu (odstavec 6.3) v pořadí *radky, sloupce, hodnoty* a `m,n` je počet řádku a sloupců matice dostaneme řídkou matici `S`.

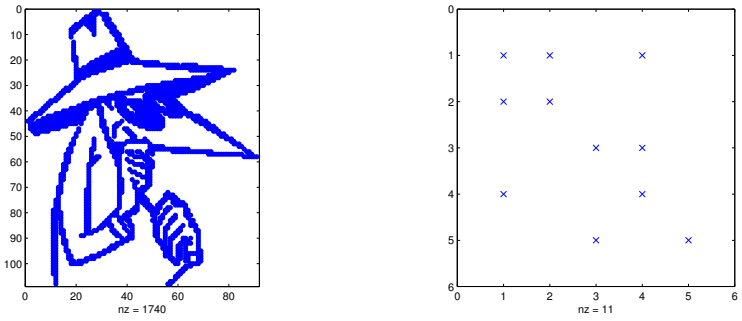
```
i = [1 2 4 1 2 3 5 1 3 4 5];
j = [1 1 1 2 2 3 3 4 4 4 5];
v = [2 -5 -7 -5 3 2 7 -2 4 8 -13];
m = 5; n = 5;
S = sparse(i,j,v,m,n);
```

Pro zjištění polí `i,j,v` existující matice lze použít příkaz `find`.

```
[i,j,v] = find(S);
```

Příkazy pro práci s řídkými maticemi

`issparse(S)`	určí, zda je vstupní argument řídká matice.
`nnz(s)`	vrátí počet nenulových prvků matice
`nonzeros(S)`	vrátí nenulové prvky matice
`nzmax(S)`	vrátí množství paměti alokované pro nenulové prvky
`spones(S)`	nahradí nenulové prvky matice jedničkami
`spfun(@sin,S)`	aplikuje zadanou funkci na nenulové prvky matice
`spy(S)`	zobrazení struktury řídké matice (poloha nenulových prvků)



Obrázek 6.2: Výstup příkazu `spy, spy(S,'x',8)`

Pokud používáme řídké matice, musíme mít na paměti, že ne všechny příkazy Matlabu jsou použitelné pro počítání s řídkými maticemi. Například pro výpočet vlastních čísel plné matice se použije příkaz `eig`, ale pro výpočet vlastních čísel řídké matice musíme použít ekvivalentní příkaz pro řídké matice, tedy `eigs`.

6.5 Cvičení

1. Vyskládejte efektivně řídkou matici

$$\mathbf{A}=\begin{bmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 2 & 0 \\ 3 & 1 & 2 & 0 & 0 \end{bmatrix}.$$

2. Vyskládejte efektivně řídkou matici $\mathbf{B}$ tvořenou 6×8 bloky $\mathbf{A}$.

3. Vyskládejte efektivně řídkou matici $\mathbf{C}$ tvořenou 8 bloky $\mathbf{A}$ na diagonále.

4. Napište funkci, která sestaví řídkou 5 diagonální matici $\mathbf{D}$ rozměru 10×10, s prvky $-1, -2, 3, 2, 1$ na diagonálách $-2 : 2$.

Kapitola 7

Gaussova eliminace a LU rozklad

V této kapitole se budeme věnovat Gaussově eliminační metodě, která se používá k řešení soustav lineárních rovnic. V první části si metodu stručně popíšeme a vysvětlíme její princip na jednoduchém příkladě. Implementaci si pak přestavíme v kapitole společně s **LU** rozkladem, který můžeme chápat jako maticový zápis Gaussovy eliminace.

Budeme se zabývat řešením soustavy

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{7.1}$$

kde $\mathbf{A} \in \mathbb{R}^{m,n}, \mathbf{x} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$. Jedná se o soustavu m lineárních rovnic o n neznámých $x_1, \ldots, x_n$, kterou můžeme zapsat ve tvaru

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad\qquad\qquad\quad &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned} \tag{7.2}$$

kde čísla a_{ij} nazýváme koeficienty soustavy, b_i pravé strany.

Chceme-li tuto soustavu řešit, budeme postupovat následovně. Nejprve pomocí *ekvivalentních úprav* převedeme soustavu $\mathbf{A}\mathbf{x} = \mathbf{b}$ na soustavu *ekvivalentí* $\mathbf{U}\mathbf{x} = \mathbf{y}$ s horní trojúhelníkovou maticí $\mathbf{U}$. Této fázi říkáme *dopředná redukce*. V druhé části tuto soustavu vyřešíme. Protože budeme postupně napočítávat jednotlivé neznámé od konce, říkáme této fázi *zpětná substituce*.

Definice 7.1. Ekvivalentními úpravami soustavy lineárních rovnic nazýváme následující úpravy:

(E1) Vzájemná výměna libovolných dvou rovnic soustavy.

(E2) Násobení obou stran některé rovnice soustavy nenulovým číslem.

(E3) Přičtení nenulového násobku některé rovnice soustavy k jiné rovnici.

Lemma 7.2. *Jestliže soustava* $\tilde{\mathbf{A}}\mathbf{x} = \tilde{\mathbf{b}}$ *vznikla ze soustavy* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *pomocí ekvivalentních úprav, pak jsou soustavy* $\tilde{\mathbf{A}}\mathbf{x} = \tilde{\mathbf{b}}$ *a* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *ekvivalentní.*

Běžně se setkáme také se zápisem soustavy pomocí *rozšířené matice soustavy*

$$(\mathbf{A}|\mathbf{b}) = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ & \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{array}\right). \tag{7.3}$$

Každá rovnice je tedy reprezentovaná jedním řádkem v matici $(\mathbf{A}|\mathbf{b})$. Ekvivalentním úpravám soustavy rovnic pak odpovídají operace s řádky rozšířené matice soustavy, které nazýváme elementární (řádkové) operace.

Definice 7.3. Elementární řádkové operace rozšířené matice soustavy nazýváme následující operace:

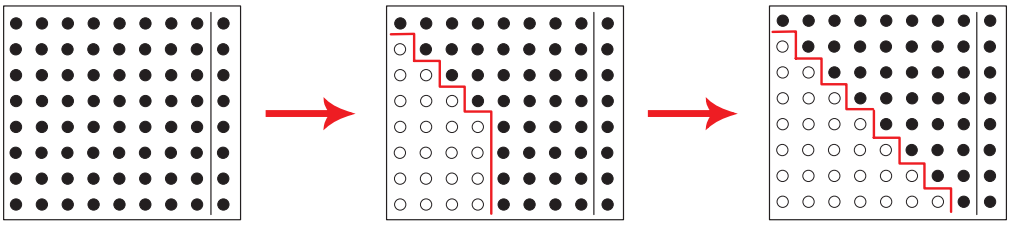
(e1) Vzájemná výměna libovolných dvou řádků.

(e2) Násobení některého řádku nenulovým číslem.

(e3) Přičtení nenulového násobku některého řádku k jinému řádku.

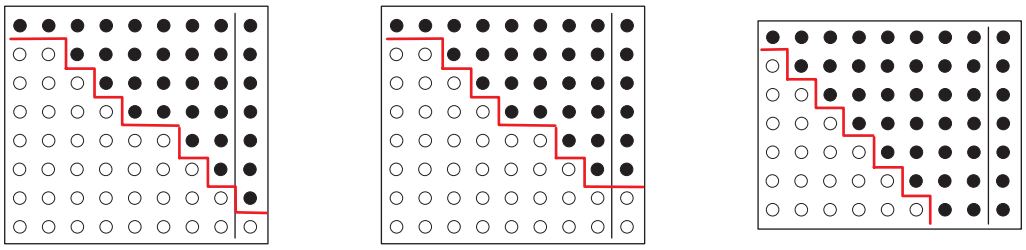
7.1 Gaussova eliminace bez pivotizace

Sestavíme rozšířenou matici soustavy, kterou budeme pomocí *elementárních úprav* převádět na *schodový tvar*. Matice je ve schodovém tvaru, jestliže první nenulové prvky řádků (*vedoucí prvky*) jsou uspořádány jako schody, klesající zleva doprava. Navíc požadujeme, aby vedoucí prvky nebyly nad sebou a případné nulové řádky byly dole. Schéma tohoto procesu je znázorněno na obrázku 7.1. Je-li matice $\mathbf{A}$ singulární, a nebo obdélníková, může schodový tvar vypadat jako na obrázku 7.2.



Obrázek 7.1: Dopředná redukce.

Matici $\mathbf{A}$ budeme během procesu eliminace zapisovat jako $\mathbf{A}_j$, kde index j značí, který sloupec byl naposledy nulován. Po $(k-1)$-tém kroku eliminace můžeme matici



Obrázek 7.2: Schodový tvar může vypadakt třeba takto, je-li $\mathbf{A}$ singulární nebo obdélníková.

soustavy zapsat jako

$$\mathbf{A}_{k-1} = \begin{pmatrix} a_{11}^{k-1} & \dots & a_{1,k-1}^{k-1} & a_{1k}^{k-1} & \dots & a_{1n}^{k-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & a_{k-1,k-1}^{k-1} & a_{k-1,k}^{k-1} & \dots & a_{k-1,n}^{k-1} \\ 0 & \dots & 0 & a_{kk}^{k-1} & \dots & a_{kn}^{k-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{nk}^{k-1} & \dots & a_{nn}^{k-1} \end{pmatrix}.$$

Toto odpovídá matici na obrázku 7.1 uprostřed.

Příklad 7.4. Řešte soustavu lineárních rovnic

$$\begin{array}{rcrcrcr} 2x_1 & - & x_2 & + & 2x_3 & = & -1 \\ x_1 & & & + & 3x_3 & = & 4 \\ -3x_1 & + & 3x_2 & + & 6x_3 & = & 21 \end{array}. \tag{7.4}$$

Řešení.

$$(\mathbf{A}|\mathbf{b}) = \left(\begin{array}{ccc|c} 2 & -1 & 2 & -1 \\ 1 & 0 & 3 & 4 \\ -3 & 3 & 6 & 21 \end{array}\right) \begin{array}{l} \longrightarrow \\ r_2 - \frac{1}{2}r_1 \\ r_3 + \frac{3}{2}r_1 \end{array} \left(\begin{array}{ccc|c} 2 & -1 & 2 & -1 \\ 0 & 1/2 & 2 & 9/2 \\ 0 & 3/2 & 9 & 39/2 \end{array}\right) \begin{array}{l} \longrightarrow \\ \\ r_3 - 3r_2 \end{array}$$

$$\longrightarrow \left(\begin{array}{ccc|c} 2 & -1 & 2 & -1 \\ 0 & 1/2 & 2 & 9/2 \\ 0 & 0 & 3 & 6 \end{array}\right) = (\mathbf{U}|\mathbf{y}).$$

Přepsali jsme si zadanou soustavu do rozšířené matice. V prvním kroku eliminujeme neznámé v prvním sloupci. První řádek (r_1) zůstává nezměněn. Do r_2 zapíšeme $r_2 - \frac{1}{2}r_1$, aby na první pozici byla nula. Do r_3 zapíšeme součet $r_3 + \frac{3}{2}r_1$, opět aby na první pozici byla nula. V dalším kroku eliminujeme v druhém sloupci. Achychom na pozici $\frac{3}{2}$ získali nulu, musíme do r_3 zapsat výsledek operace $r_3 - 3r_2$. Získáváme matici ve schodovém tvaru. ▲

Matici si můžeme přepsat opět jako soustavu

$$\begin{array}{rcrcrcl} 2x_1 & - & x_2 & + & 2x_3 & = & -1 \\ & & \frac{1}{2}x_2 & + & 2x_3 & = & \frac{9}{2} \\ & & & & 3x_3 & = & 6 \end{array} . \tag{7.5}$$

V další fázi postupně vyčíslíme jednotlivé neznámé. Začínáme od konce. Z poslední rovnice vidíme, že $x_3 = 2$. Výsledek dosadíme do rovnice předešlé, spočítáme x_2. Již známé výsledky zase dosadíme do předešlé rovnice, atd. Po dosazení do první rovnice získáme řešení

$$\mathbf{x} = (-2, 1, 2)^T.$$

V následující kapitole si ukážeme implementaci Gaussovy eliminace. Využijeme k tomu tzv. **LU** rozklad, který můžeme chápat jako maticový zápis Gaussovy eliminace, přičemž $\mathbf{L}^{-1}$ je matice transformace, která převede $\mathbf{A}$ na $\mathbf{U}$, tj. $\mathbf{L}^{-1}\mathbf{A} = \mathbf{U}$.

7.2 LU rozklad

Uvažujeme regulární čtvercovou matici $\mathbf{A} \in \mathbb{R}^{n,n}$, kterou chceme rozložit na horní trojúhelníkovou matici $\mathbf{U}$ a dolní trojúhelníkovou matici $\mathbf{L}$ s jedničkami na diagonále, aby platilo

$$\mathbf{A} = \mathbf{LU}.$$

V předchozí kapitole jsme si ukázali, jak pomocí elementárních řádkových úprav převést matici $\mathbf{A}$ na horní trojúhelníkovou matici $\mathbf{U}$. Zbývá nám ukázat, jak získáme matici $\mathbf{L}$.

Operace prováděné s řádky matice si budeme zapisovat do matic $\mathbf{L}_k$. Do k-tého sloupce jednotkové matice zapíšeme l_i^k násobky řádků, kterými vynulujeme odpovídající pozice v k-tém sloupci matice $\mathbf{A}_{k-1}$. Těmto prvkům říkáme *multiplikátory* a spočítáme je podle předpisu (7.7). Matici $\mathbf{L}_k$ [2] můžeme zapsat ve tvaru

$$\mathbf{L}_k = \mathbf{I} + \boldsymbol{\ell}^k \mathbf{e}_k^T = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l_{k+1}^k & 1 & \\ & & \vdots & & \ddots \\ & & l_n^k & & 1 \end{pmatrix},$$

kde $\boldsymbol{\ell}^k = (0, \ldots, 0, l_{k+1}^k, \ldots, l_n^k)^T$ a $\mathbf{e}_k$ je k-tý vektor standardní báze $\mathbb{R}^n$. V k-tém kroku tedy získáváme

$$\mathbf{A}_k = \mathbf{L}_k \mathbf{A}_{k-1}, \quad \mathbf{A}_0 = \mathbf{A}. \tag{7.6}$$

Pro odvození multiplikátorů [2], uvažujme eliminaci prvků v k-tém sloupci matice $\mathbf{A}_{k-1}$

$$\mathbf{s}_k^{\mathbf{A}_{k-1}} = \begin{pmatrix} a_{1k}^{k-1} \\ \vdots \\ a_{k-1,k}^{k-1} \\ a_{kk}^{k-1} \\ \vdots \\ a_{nk}^{k-1} \end{pmatrix} \quad \longrightarrow \quad \begin{pmatrix} a_{1k}^{k} \\ \vdots \\ a_{k-1,k}^{k} \\ a_{kk}^{k} \\ 0 \\ 0 \end{pmatrix} = \mathbf{s}_k^{\mathbf{A}_k}.$$

Ze vztahu (7.6) plyne, že tuto eliminaci můžeme zapsat také ve tvaru

$$\mathbf{L}_k \mathbf{s}_k^{\mathbf{A}_{k-1}} = \mathbf{s}_k^{\mathbf{A}_k}.$$

Z této rovnosti získáváme

$$l_j^k a_{kk}^{k-1} + a_{jk}^{k-1} = 0, \quad \text{pro } j = k+1, \ldots, n,$$

takže multiplikátory

$$l_j^k = -\frac{a_{jk}^{k-1}}{a_{kk}^{k-1}}, \quad \text{pro } j = k+1, \ldots, n, \quad a_{kk} \neq 0. \tag{7.7}$$

Postupným dosazováním do (7.6) eliminujeme prvky v jednotlivých sloupcích a můžeme psát

$$\mathbf{L}_{n-1} \ldots \mathbf{L}_2 \mathbf{L}_1 \mathbf{A} = \mathbf{U},$$

kde $\mathbf{L}_1$ nuluje prvky v prvním sloupci, $\mathbf{L}_2$ ve druhém, atd. Přenásobením obou stran rovnosti maticemi $\mathbf{L}_k^{-1}$ dostaneme

$$\mathbf{A} = \mathbf{L}_1^{-1} \mathbf{L}_2^{-1} \ldots \mathbf{L}_{n-1}^{-1} \mathbf{U} \quad \Rightarrow \quad \mathbf{A} = \mathbf{L}\mathbf{U},$$

kde

$$\mathbf{L} = \mathbf{L}_1^{-1} \mathbf{L}_2^{-1} \ldots \mathbf{L}_{n-1}^{-1}. \tag{7.8}$$

Snadno ověříme, že chceme-li získat matici $\mathbf{L}_k^{-1}$ stačí mimodiagonální prvky v matici $\mathbf{L}_k$ přenásobit -1. Tedy

$$\mathbf{L}_k^{-1} = \mathbf{I} - \boldsymbol{\ell}^k \mathbf{e}_k^T = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1}^k & 1 & \\ & & \vdots & & \ddots \\ & & -l_n^k & & 1 \end{pmatrix}.$$

Matici $\mathbf{L}$ můžeme vyjádřit po jednoduchých úpravách taky jako

$$\begin{aligned} \mathbf{L} &= \mathbf{L}_1^{-1} \ldots \mathbf{L}_{n-1}^{-1} \\ &= (\mathbf{I} - \boldsymbol{\ell}^1 \mathbf{e}_1^T) \ldots (\mathbf{I} - \boldsymbol{\ell}^{n-1} \mathbf{e}_{n-1}^T) \\ &= \mathbf{I} - \sum_{k=1}^{n-1} \boldsymbol{\ell}^k \mathbf{e}_k^T. \end{aligned} \tag{7.9}$$

Příklad 7.5. Naším úkolem je spočítat **LU** rozklad matice **A** z příkladu 7.4. Budeme postupovat stejně jako dříve, jenom zvolíme jiný způsob zápisu.

Řešení. Matice soustavy je

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 2 \\ 1 & 0 & 3 \\ -3 & 3 & 6 \end{pmatrix}.$$

Do matice $\mathbf{L}_1$ si zapíšeme multiplikátory řádkových operací, které nám vynulují prvky v prvním sloupci matice **A**, takže

$$\mathbf{L}_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 3/2 & 0 & 1 \end{pmatrix}, \quad \mathbf{A}_1 = \mathbf{L}_1\mathbf{A} = \begin{pmatrix} 2 & -1 & 2 \\ 0 & 1/2 & 2 \\ 0 & 3/2 & 9 \end{pmatrix}.$$

Dále sestavíme stejným způsobem matici $\mathbf{L}_2$, která nám po přenásobení vynuluje druhý sloupec, získáváme tedy

$$\mathbf{L}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}, \quad \mathbf{A}_2 = \mathbf{L}_2\mathbf{A}_1 = \begin{pmatrix} 2 & -1 & 2 \\ 0 & 1/2 & 2 \\ 0 & 0 & 3 \end{pmatrix} = \mathbf{U}.$$

Zbývá nám dopočítat matici **L**. Podle vztahu (7.9) můžeme psát

$$\mathbf{L} = \mathbf{L}_1^{-1}\mathbf{L}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -3/2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -3/2 & 3 & 1 \end{pmatrix}.$$

▲

7.3 Základní algoritmus

V algoritmech 7.1 a 7.2 ukážeme základní verzi implementace, ve které sestavujeme každou matici zvlášť. V prvním případě postupně odečítáme řádky, ve druhém sloupce. V algoritmu 7.3 pak ukážeme, že můžeme všechny operace provádět přímo na matici **A**. Výsledná matice má pod hlavní diagonálou prvky matice **L** a na hlavní diagonále a nad ní prvky matice **U**. Formát výsledné matice můžeme zapsat jako

$$\mathbf{A} = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ l_{21} & \ddots & & \vdots \\ & \ddots & \ddots & \\ l_{n1} & \dots & l_{n,n-1} & u_{nn} \end{pmatrix}.$$

Pro sestavení matice **L**, resp. **U**, použijeme příkazy `tril`, resp. `triu`. Na hlavní diagonálu matice **L** nesmíme zapomenout přičíst jednotkovou matici.

Poznámka 7.6. Symbolem **I** v algoritmech značíme jednotkovou matici, kterou vytvoříme v Matlabu následovně:

$$\texttt{n = size(A,1);} \quad \texttt{I = eye(n);}$$

Algoritmus 7.1: **LU** rozklad: řádková verze

```
L = I; U = A;
for k=1:n-1
  L(k+1:n,k) = U(k+1:n,k)/U(k,k); %multiplikatory
  for j = k+1:n
    U(j,k:n) = U(j,k:n)-L(j,k)*U(k,k:n); %radky
  end
end
```

Algoritmus 7.2: **LU** rozklad: sloupcová verze

```
L = I; U = A;
for k = 1:n-1
  L(k+1:n,k) = U(k+1:n,k)/U(k,k);  %multiplikatory
  for j = k:n
    U(k+1:n,j) = U(k+1:n,j)-L(k+1:n,k)*U(k,j); %sloupce
  end
end
```

Algoritmus 7.3: **LU** rozklad: řádková verze: přepisujeme **A**

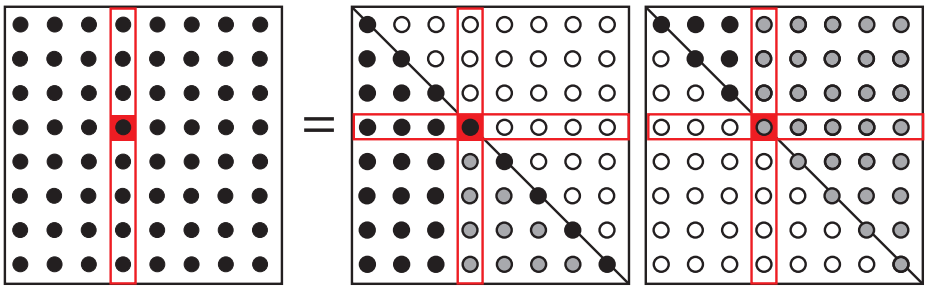
```
for k = 1:n-1
  A(k+1:n,k) = A(k+1:n,k)/A(k,k); %multiplikatory
  for j = k+1:n
    A(j,k+1:n) = A(j,k+1:n)-A(j,k)*A(k,k+1:n); %radky
  end
end
L = tril(A,-1)+I;
U = triu(A);
```

7.4 Další možnost implementace

Další možností implementace je postupně dopočítávat prvky obou matic současně. Ukážeme si postup [3], jak dopočítat k-té sloupce matic **L** a **U**, známe-li $k-1$ sloupců těchto matic. Tato situace je zachycena na obrázku 7.3.

Uvažujme rozklad $\mathbf{A} = \mathbf{LU}$ zapsaný blokově ve tvaru

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{1k} & \mathbf{A}_{13} \\ \mathbf{A}_{k1} & \mathbf{A}_{kk} & \mathbf{A}_{k3} \\ \mathbf{A}_{31} & \mathbf{A}_{3k} & \mathbf{A}_{33} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \mathbf{O} & \mathbf{O} \\ \mathbf{L}_{k1} & \mathbf{L}_{kk} & \mathbf{O} \\ \mathbf{L}_{31} & \mathbf{L}_{3k} & \mathbf{L}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{1k} & \mathbf{U}_{13} \\ \mathbf{O} & \mathbf{U}_{kk} & \mathbf{U}_{k3} \\ \mathbf{O} & \mathbf{O} & \mathbf{U}_{33} \end{pmatrix}, \qquad (7.10)$$



Obrázek 7.3: **LU** rozklad: výpočet k-tých sloupců matic $\mathbf{L}$ a $\mathbf{U}$.

kde index k označuje prvky v k-tém řádku či sloupci, takže např. blok $\mathbf{A}_{k1}$ má jeden řádek a $k-1$ sloupců a blok $\mathbf{A}_{33}$ má $n-k$ řádků i sloupců. Vyjádřeme si nyní k-tý sloupec matice $\mathbf{A}$ jako soustavu

$$\begin{aligned} \mathbf{A}_{1k} &= \mathbf{L}_{11}\underline{\mathbf{U}_{1k}} \\ \mathbf{A}_{kk} &= \mathbf{L}_{k1}\underline{\mathbf{U}_{1k}} + \mathbf{L}_{kk}\underline{\mathbf{U}_{kk}} \\ \mathbf{A}_{3k} &= \mathbf{L}_{31}\underline{\mathbf{U}_{1k}} + \underline{\mathbf{L}_{3k}\mathbf{U}_{kk}}, \end{aligned} \tag{7.11}$$

ve které jsme podtržením zvýraznili neznámé. Poznamenejme, že $\mathbf{L}_{kk} = 1$. Řešením první rovnice získáme neznámé $\mathbf{U}_{1k}$, které můžeme dosadit do dalších rovnic. Z druhé rovnice dopočítáme $\mathbf{U}_{kk}$ a dosadíme do třetí rovnice, ze které pak dopočteme $\mathbf{L}_{3k}$.

Ukažme si způsob implementace tohoto výpočtu podrobněji. Soustavu (7.11) si můžeme přepsat jako

$$\mathbf{A}_{1k} = \mathbf{L}_{11}\mathbf{U}_{1k} \quad \text{a} \quad \begin{pmatrix} \mathbf{A}_{kk} \\ \mathbf{A}_{3k} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{k1} & \mathbf{L}_{kk} \\ \mathbf{L}_{31} & \mathbf{L}_{3k} \end{pmatrix} \begin{pmatrix} \mathbf{U}_{1k} \\ \mathbf{U}_{kk} \end{pmatrix}.$$

Z první rovnice dostáváme $\mathbf{U}_{1k}$. Z druhé si vyjádříme

$$\begin{pmatrix} \mathbf{v}_k \\ \mathbf{v}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{kk} \\ \mathbf{A}_{3k} \end{pmatrix} - \begin{pmatrix} \mathbf{L}_{k1} \\ \mathbf{L}_{31} \end{pmatrix} \mathbf{U}_{1k} = \begin{pmatrix} \mathbf{L}_{kk}\mathbf{U}_{kk} \\ \mathbf{L}_{3k}\mathbf{U}_{kk} \end{pmatrix}.$$

Jelikož $\mathbf{L}_{kk} = 1$ stačí dopočítat

$$\mathbf{L}_{3k} = \mathbf{v}_3/\mathbf{v}_k \quad \text{a} \quad \mathbf{U}_{kk} = \mathbf{v}_k.$$

Tento postup výpočtu **LU** rozkladu je zapsán v algoritmu 7.4.

Algoritmus 7.4: **LU** rozklad

```
L = I; U = 0;
for k = 1:n
  if k == 1,
    v(k:n) = A(k:n,k);
  else
    z = L(1:k-1,1:k-1)\A(1:k-1,k);
    U(1:k-1,k) = z;
    v(k:n) = A(k:n,k)-L(k:n,1:k-1)*z;
  end
  if k < n, L(k+1:n,k) = v(k+1:n)/v(k); end
  U(k,k) = v(k);
end
```

7.5 Řešení soustav pomocí LU rozkladu

Uvažujme soustavu $\mathbf{Ax} = \mathbf{b}$. Můžeme-li zapsat $\mathbf{A} = \mathbf{LU}$, pak

$$\mathbf{L}(\mathbf{Ux}) = \mathbf{b} \quad \Rightarrow \quad \mathbf{Lz} = \mathbf{b}, \quad \text{a} \quad \mathbf{Ux} = \mathbf{z}.$$

7.6 Příklady

1. Nahraďte řádek 4 v algoritmu 7.3 cyklem

   ```
   for i=k+1:n
     A(j,i)=A(j,i)-A(j,k)*A(k,i);
   end
   ```

   Na dostatečně velké matici porovnejte rychlost původního a upraveného algoritmu. Získané výsledky vysvětlete.

2. Upravte algoritmus 7.2, tak aby se matice **L** a **U** zapisovaly přímo do matice **A**.

3. Upravte výše uvedené algoritmy pro případ, že **A** je singulární nebo obdélníková matice.

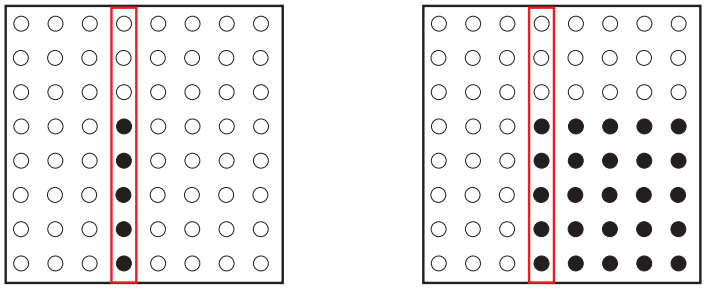
Kapitola 8

Pivotizace

Během odvozování předchozích algoritmů jsme v (7.7) předpokládali, že v k-tém kroku je $a_{kk} \neq 0$. Nebude-li tento předpoklad splněn, může dojít k dělení nulou a algoritmus selže. Bez tohoto předpokladu se obejdeme, budeme-li mít možnost nahradit prvek na pozici (k, k) v matici $\mathbf{A}_k$ jiným vhodným prvkem, tzv. *pivotem*. Vybírat můžeme podle různých kritérií, záleží na tom, čeho potřebujeme dosáhnout. Pro zlepšení numerické stability budeme vybírat pivot podle jeho velikosti. Množina, ze které pivot vybíráme, je

- sloupec $\mathbf{A}_k(k:n, k)$ u částečné pivotizace,
- submatice $\mathbf{A}_k(k:n, k:n)$ u úplné pivotizace.

Obě varianty jsou zakresleny na obrázku 8.1.



Obrázek 8.1: Množina potenciálních pivotů pro částečnou a úplnou pivotizaci.

8.1 Částečná pivotizace

LU *rozklad s částečnou pivotizací* můžeme chápat jako rozklad bez pivotizace přepermutované matice, tedy

$$\mathbf{PA} = \mathbf{LU}, \tag{8.1}$$

kde $\mathbf{P}$ je permutační matice. Podívejme se, jak tento rozklad vznikne.

Chceme-li v matici $\mathbf{A}_k$ nahradit prvek na pozici (k,k) prvkem na pozici (p,k), prohodíme řádky k a p v dané matici. Tuto operaci budeme značit

$$\mathbf{A}_k(k,:) \leftrightarrow \mathbf{A}_k(p,:).$$

Index p určíme tak, aby

$$|\mathbf{A}(p,k)| = \max_{i=k,\ldots,n} \{|a_{i,k}|\}.$$

Pivot je tedy prvek, který je v absolutní hodnotě největší z prvků $\mathbf{A}_k(k:n,k)$.

Prohazování řádků by mohlo být časově náročné, proto si prováděné operace budeme pouze zaznamenávat do permutační matice $\mathbf{P}$, takže

$$\Big\{\mathbf{A}_k(k,:) \leftrightarrow \mathbf{A}_k(p,:)\Big\} \quad \Rightarrow \quad \mathbf{P}_{k+1}\mathbf{A}_k. \tag{8.2}$$

Následně provedeme eliminaci v k-tém sloupci, takže

$$\mathbf{L}_{k+1}\mathbf{P}_{k+1}\mathbf{A}_k = \mathbf{A}_{k+1}. \tag{8.3}$$

Postupným eliminováním podle (8.3) získáváme

$$\begin{aligned} \mathbf{L}_1\mathbf{P}_1\mathbf{A} &= \mathbf{A}_1 \\ \mathbf{L}_2\mathbf{P}_2\mathbf{A}_1 &= \mathbf{A}_2 \\ &\vdots \\ \mathbf{L}_{n-1}\mathbf{P}_{n-1}\cdots\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1\mathbf{A} &= \mathbf{U} \end{aligned}$$

Zaveďme značení [2]

$$\mathbf{L}'_3 = \mathbf{L}_3, \quad \mathbf{L}'_2 = \mathbf{P}_3\mathbf{L}_2\mathbf{P}_3^{-1}, \quad \mathbf{L}'_1 = \mathbf{P}_3\mathbf{P}_2\mathbf{L}_1\mathbf{P}_2^{-1}\mathbf{P}_3^{-1}$$

pro $n = 4$. Není těžké ověřit, že platí

$$\mathbf{L}_3\mathbf{P}_3\mathbf{L}_2\mathbf{P}_2\mathbf{L}_1\mathbf{P}_1 = \mathbf{L}'_3\mathbf{L}'_2\mathbf{L}'_1\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1.$$

Obecně tedy můžeme psát

$$\mathbf{U} = \big(\mathbf{L}'_{n-1}\cdots\mathbf{L}'_2\mathbf{L}'_1\big)\big(\mathbf{P}_{n-1}\cdots\mathbf{P}_2\mathbf{P}_1\big)\mathbf{A},$$

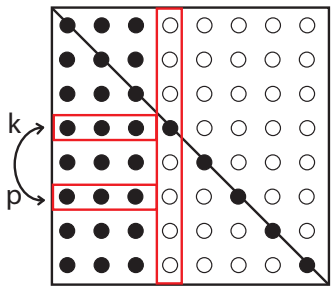
kde

$$\mathbf{L}'_k = \mathbf{P}_{n-1}\cdots\mathbf{P}_{k+1}\mathbf{L}_k\mathbf{P}_{k+1}^{-1}\cdots\mathbf{P}_{n-1}^{-1}$$

a tedy

$$\big(\mathbf{L}'_{n-1}\cdots\mathbf{L}'_2\mathbf{L}'_1\big)^{-1}\mathbf{U} = \big(\mathbf{P}_{n-1}\cdots\mathbf{P}_2\mathbf{P}_1\big)\mathbf{A} \quad \Leftrightarrow \quad \mathbf{LU} = \mathbf{PA}.$$

Začlenění pivotizace do algoritmu není obtížné. Podívejme se podrobněji jak vložit pivotizaci např. do algorimu 7.4. Na pozici (k,k) zde umisťujeme prvek $v(k)$.



Obrázek 8.2: Prohazování částí řádků v matici **L** při částečné pivotizaci.

V případě částečné pivotizace, chceme na tuto pozici vybrat vhodný prvek z $v(k:n)$. Řekněme, že vhodný prvek, který chceme přesunout, je na pozici p. Budeme tedy prohazovat řádky k a p, jak je ukázáno na obrázku 8.2 [3]. Vidíme taky, že se změna netýká pouze vektrou v. Zaměňovat tedy budeme

$$\begin{aligned} \mathbf{v}(k) &\leftrightarrow \mathbf{v}(p) \\ \mathbf{P}(k,:) &\leftrightarrow \mathbf{P}(p,:) \\ \mathbf{A}(k,:) &\leftrightarrow \mathbf{A}(p,:). \end{aligned}$$

Implementace **LU** rozkladu s částečnou pivotizací je ukázána v algoritmu 8.1.

Algoritmus 8.1: **LU** rozklad s částečnou pivotizací

```
function [L,U,P] = my_lu_piv(A)
n = size(A,1); I = eye(n); O = zeros(n); L = I; U = O; P = I;
    function change_rows(k,p)
        x = P(k,:);  P(k,:) = P(p,:); P(p,:) = x;
        x = A(k,:);  A(k,:) = A(p,:); A(p,:) = x;
        x = v(k); v(k) = v(p); v(p) = x;
    end
    function change_L(k,p)
        x = L(k,1:k-1); L(k,1:k-1) = L(p,1:k-1); L(p,1:k-1) = x;
    end
for k = 1:n
    if k == 1, v(k:n) = A(k:n,k);
    else
        z = L(1:k-1,1:k -1)\ A(1:k-1,k); U(1:k-1,k) = z;
        v(k:n) = A(k:n,k)-L(k:n,1:k-1)*z;
    end
    if k<n
        x = v(k:n); p = (k-1)+find(abs(x) == max(abs(x)));
        change_rows(k,p);
        L(k+1:n,k) = v(k+1:n)/v(k);
        if k > 1, change_L(k,p); end
    end
    U(k,k) = v(k);
end
end
```

Příklad 8.1. Pomocí **LU** rozkladu s částečnou pivotizací vyřešte soustavu rovnic $\mathbf{Ax} = \mathbf{b}$, kde

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ -4 & 1 & 3 \\ 2 & 3 & 2 \end{pmatrix} \quad , \quad \mathbf{b} = \begin{pmatrix} 1 \\ 8 \\ 3 \end{pmatrix}.$$

Řešení.

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 \\ -4 & 1 & 3 \\ 2 & 3 & 2 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_1 \leftrightarrow r_2 \end{array} \begin{pmatrix} -4 & 1 & 3 \\ 1 & 1 & 1 \\ 2 & 3 & 2 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_2 + \frac{1}{4} r_1 \\ r_3 + \frac{1}{2} r_1 \end{array}$$

$$\longrightarrow \begin{pmatrix} -4 & 1 & 3 \\ 0 & \frac{5}{4} & \frac{7}{4} \\ 0 & \frac{7}{2} & \frac{7}{2} \end{pmatrix} \begin{array}{l} \longrightarrow \\ \\ r_2 \leftrightarrow r_3 \end{array} \begin{pmatrix} -4 & 1 & 3 \\ 0 & \frac{7}{2} & \frac{7}{2} \\ 0 & \frac{5}{4} & \frac{7}{4} \end{pmatrix} \begin{array}{l} \longrightarrow \\ \\ r_3 - \frac{5}{14} r_2 \end{array} \begin{pmatrix} -4 & 1 & 3 \\ 0 & \frac{7}{2} & \frac{7}{2} \\ 0 & 0 & \frac{1}{2} \end{pmatrix} = \mathbf{U}$$

Při prohození řádků v matici **L** musíme prohodit také odpovídající sloupce.

$$\tilde{\mathbf{L}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_1 \leftrightarrow r_2 \\ s_1 \leftrightarrow s_2 \end{array} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_2 - \frac{1}{4} r_1 \\ r_3 - \frac{1}{2} r_1 \end{array}$$

$$\longrightarrow \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{4} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_2 \leftrightarrow r_3 \\ s_2 \leftrightarrow s_3 \end{array} \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & 0 & 1 \end{pmatrix} \begin{array}{l} \longrightarrow \\ \\ r_3 + \frac{5}{14} r_2 \end{array} \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & \frac{5}{14} & 1 \end{pmatrix} = \mathbf{L}$$

V matici **P** zaznamenáme všechny záměny řádků.

$$(\tilde{\mathbf{P}}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_1 \leftrightarrow r_2 \end{array} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{array}{l} \longrightarrow \\ r_2 \leftrightarrow r_3 \end{array} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \mathbf{P}$$

Zbývá nám vyřešit samotnou soustavu. Z $\mathbf{PA} = \mathbf{LU}$ si vyjádříme $\mathbf{A} = \mathbf{P^T LU}$ a dostadíme do $\mathbf{Ax} = \mathbf{b}$. Získáváme $\mathbf{P^T LUx} = \mathbf{b}$, po úpravě pak $\mathbf{L}(\mathbf{Ux}) = \mathbf{Pb}$. Budeme tedy řešit 2 soustavy s trojúhelníkovými maticemi, $\mathbf{Lz} = \mathbf{Pb}$ a následně $\mathbf{Ux} = \mathbf{z}$. Příklad již zvládnete sami dopočítat. ▲

8.2 Úplná pivotizace

LU *rozklad s úplnou pivotizací* můžeme chápat jako rozklad bez pivotizace matice s přepermutovanými řádky i sloupci, tedy

$$\mathbf{PAQ} = \mathbf{LU}, \tag{8.4}$$

kde $\mathbf{P}$ a $\mathbf{Q}$ jsou permutační matice. Podívejme se, jak tento rozklad vznikne.

Chceme-li v matici $\mathbf{A}_k$ nahradit prvek na pozici (k,k) prvkem na pozici (p,q) prohodíme řádky k a p a sloupce k a q v dané matici. Tuto operaci budeme značit

$$\mathbf{A}_k(k,:) \leftrightarrow \mathbf{A}_k(p,:),$$

$$\mathbf{A}_k(:,k) \leftrightarrow \mathbf{A}_k(:,q).$$

Určíme indexy p a q tak, aby

$$|\mathbf{A}(p,q)| = \max_{\substack{i=k,\dots,n \\ j=k,\dots,n}} \{|a_{i,j}|\}.$$

Pivot je tedy prvek, který je v absolutní hodnotě největší z prvků $\mathbf{A}_k(k:n,k:n)$.

Prováděné operace si budeme zaznamenávat do permutačních matic, do $\mathbf{P}$ přehazování řádků, do $\mathbf{Q}$ prohazování sloupců. Můžeme psát

$$\left\{ \begin{array}{l} \mathbf{A}_k(k,:) \leftrightarrow \mathbf{A}_k(p,:) \\ \mathbf{A}_k(:,k) \leftrightarrow \mathbf{A}_k(:,q) \end{array} \right\} \quad \Rightarrow \quad \mathbf{P}_{k+1}\mathbf{A}_k\mathbf{Q}_{k+1}. \tag{8.5}$$

Následně provedeme eliminaci v k-tém sloupci, takže

$$\mathbf{L}_{k+1}\mathbf{P}_{k+1}\mathbf{A}_k\mathbf{Q}_{k+1} = \mathbf{A}_{k+1}. \tag{8.6}$$

Podobným způsobem jako v předchozí kapitole odvodíme vztah (8.4).

Poznámka 8.2. Vhodnou permutací můžeme ovlivnit zaplněnost matice. Více se dozvíme v kapitole 10.

Poznámka 8.3. Místo permutačních matic $\mathbf{P}$ a $\mathbf{Q}$ můžeme využít permutační vektory $\mathbf{p}$ a $\mathbf{q}$ a nepřímé adresování.

8.3 Funkce Matlabu

- `[L,U] = lu(sparse(A),0)` ... bez pivotizace ... $\mathbf{LU} = \mathbf{A}$
- `[L,U,P] = lu(A)` ... s částečnou pivotizací ... $\mathbf{LU} = \mathbf{PA}$
- `Y = lu(A)` ... alternativa předešlého, kde $\mathbf{Y} = \mathbf{U} + \mathbf{L} - \mathbf{I}$
- `[L,U,P,Q] = lu(sparse(A))` ... s pivotizací a řídkým přeuspořádáním ... $\mathbf{LU} = \mathbf{PAQ}$

Více podrobností najdete v [4].

Kapitola 9

LDL^T a Choleského rozklad

V numerických metodách se snažíme využít speciální strukturu matic, kdykoliv je to možné. Je zřejmé, že při práci se symetrickými maticemi stačí v paměti uchovávat pouze polovinu prvků. Je tedy přirozené se ptát, zda můžeme řešit soustavu $\mathbf{Ax} = \mathbf{b}$ s využitím symetrie matice $\mathbf{A}$.

Uvažujme nejprve rozklad (obecné) čtvercové matice $\mathbf{A} \in \mathbb{R}^{n,n}$ ve tvaru

$$\mathbf{A} = \mathbf{LDM}^T, \tag{9.1}$$

kde $\mathbf{U} = \mathbf{DM}^T$, $\mathbb{R}^{n,n} \ni \mathbf{D} = \mathrm{diag}(\mathbf{d})$ je diagonální matice s prvky vektoru $\mathbf{d}$ na diagonále, $\mathbf{L} \in \mathbb{R}^{n,n}$ a $\mathbf{M} \in \mathbb{R}^{n,n}$ jsou dolní trojúhelníkové matice s jedničkami na diagonále.

Odvoďme nyní algoritmus pro výpočet tohoto rozkladu. Budeme postupovat podobně jako v kapitole 7.4. Předpokládejme tedy, že již známe $k-1$ sloupců matic $\mathbf{L}, \mathbf{D}$ a $\mathbf{M}^T$. Vyjádřeme si k-tý sloupec matice $\mathbf{A}$ z (7.10) ve tvaru

$$\begin{pmatrix} \mathbf{A}_{1k} \\ \mathbf{A}_{kk} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \mathbf{O} \\ \mathbf{L}_{k1} & \mathbf{L}_{kk} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_k \end{pmatrix} \quad \text{a} \quad \mathbf{A}_{3k} = \begin{pmatrix} \mathbf{L}_{31} & \mathbf{L}_{3k} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_k \end{pmatrix}, \tag{9.2}$$

kde $\mathbf{v} = \mathbf{DM}^T(k,:)$. Z první soustavy vyčíslíme $(\mathbf{v}_1, \mathbf{v}_k)^T$ a můžeme dopočítat $\mathbf{M}_{k1}^T = \mathbf{v}_1 \cdot / \mathbf{d}_1$ z $\mathbf{v}_1 = \mathbf{D}_{11}\mathbf{M}_{k1}^T$, a jelikož $\mathbf{M}_{kk}^T = 1$ dostáváme $\mathbf{d}_k = \mathbf{v}_k$ z $\mathbf{v}_k = \mathbf{d}_k \mathbf{M}_{kk}^T$. Dále dosadíme $(\mathbf{v}_1, \mathbf{v}_k)^T$ do druhé soustavy a dopočítáme $\mathbf{L}_{3k} = (\mathbf{A}_{3k} - \mathbf{L}_{31}\mathbf{v}_1)/\mathbf{v}_k$.

9.1 LDL^T rozklad

Bude-li $\mathbf{A} \in \mathbb{R}^{n,n}$ matice symetrická $\mathbf{A}^T = \mathbf{A}$, pak bude $\mathbf{M} = \mathbf{L} \in \mathbb{R}^{n,n}$ a obdržíme rozklad ve tvaru

$$\mathbf{A} = \mathbf{LDL}^T.$$

Výpočet je analogický z předchozím postupem. Z první soustavy (9.2) získáme $(\mathbf{v}_1, \mathbf{v}_k)^T$ a určíme $\mathbf{d}_k = \mathbf{v}_k$. Z druhé soustavy pak dopočítáme $\mathbf{L}_{3k}$. Implementace je uvedena jako algoritmus 9.1.

Poznámka 9.1. Matici **L** můžeme efektivně ukládat do dolní trojúhelníkové části matice **A** a složky matice **D** na diagonále matice **A**.

Algoritmus 9.1: $\mathbf{LDL}^T$ rozklad

```
L = I; d = o;
for j = 1:n
  if j = 1,
    d(j) = A(j,j);
    L(j+1:n,j) = A(j+1:n,j)/d(j);
  else
    v = d(1:j-1).*L(j,1:j-1)';
    d(j) = A(j,j)-L(j,1:j-1)*v;
    L(j+1:n,j) = (A(j+1:n,j)-L(j+1:n,1:j-1)*v)/d(j);
  end
end
```

9.2 Choleského rozklad

Uvažujme rozklad symetrické a positivně definitní matice $\mathbf{A} \in \mathbb{R}^{n,n}$ ve tvaru

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T.$$

Dosazením $\mathbf{U} = \mathbf{L}$ do (7.10) dostáváme relace (9.2) s $\mathbf{v}_1 = \mathbf{L}_{k1}^T$ a $\mathbf{v}_k = \mathbf{L}_{kk}$.

Z první soustavy (9.2) pak dostáváme $\mathbf{L}_{kk} = \sqrt{\mathbf{A}_{kk} - \mathbf{L}_{k1}\mathbf{v}_1}$. Z druhé soustavy pak dopočítáme $\mathbf{L}_{3k}$. Postup výpočtu je zapsán jako algoritmus 9.2.

Poznámka 9.2. Choleského faktorizaci můžeme použít také ve spojení s pivotizací. Zde mluvíme *o symetrické pivotizaci* ve tvaru

$$\mathbf{P}\mathbf{A}\mathbf{P}^T = \mathbf{L}\mathbf{L}^T.$$

Do **P** můžeme zahrnout také optimalizaci zaplnění.

Příklad 9.3. Řešte soustavu $\mathbf{Ax} = \mathbf{b}$, kde

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} -1 \\ \\ \vdots \\ \\ -1 \end{pmatrix},$$

Choleského rozkladem.

Řešení. Vytvoříme m-funkci s názvem `chol1.m` s hlavičkou

`function L = chol1(A)`

a tělem podle algoritmu 9.2.

Sestavení matice soustavy a vektoru pravých stran:

```
n=5;
A=speye(ones(n,1)*[-1,2,1],[-1,0,-1],n,n);
b=-ones(n,1);
```

Vlastní rešení:

```
L=chol1(A);
y=L\b; x=L'\y;
```

▲

Algoritmus 9.2: Choleského rozklad

```
L = 0;
for j = 1:n
  if j = 1
    L(j,j) = sqrt(A(j,j));
    L(j+1:n,j) = A(j+1:n,j)/L(j,j);
  else
    v = L(j,1:j-1)';
    L(j,j) = sqrt(A(j,j)-L(j,1:j-1)*v);
    L(j+1:n,j) = (A(j+1:n,j)-L(j+1:n,1:j-1)*v)/L(j,j);
  end
end
```

9.3 Funkce Matlabu

- `[L,D] = ldl(full(A))` ... bez pivotizace ... $\mathbf{LDL}^T = \mathbf{A}$
- `[L,D,P] = ldl(A)` ... s pivotizací ... $\mathbf{LDL}^T = \mathbf{P}^T\mathbf{AP}$
- `R = chol(A)` ... pro pozitivně definitní $\mathbf{A}$ vrací horní trojúhelníkovou matici $\mathbf{R}$, že platí $\mathbf{R}^T\mathbf{R} = \mathbf{A}$, pracuje pouze s horní trojúhelníkovou částí matice $\mathbf{A}$, předpokládá, že $\mathbf{A} = \mathbf{A}^T$
- `L = chol(A,'lower')` ... pro pozitivně definitní $\mathbf{A}$ vrací dolní trojúhelníkovou matici $\mathbf{L}$, že platí $\mathbf{LL}^T = \mathbf{A}$, pracuje pouze s dolní trojúhelníkovou částí matice $\mathbf{A}$
- `[L,p,s] = chol(A,'lower','vector')` ... pro $p = 0$ vrací dolní trojúhelníkovou matici $\mathbf{L}$ a permutační vektor $\mathbf{s}$ takový, že $\mathbf{A}(\mathbf{s},\mathbf{s}) = \mathbf{LL}^T$

Více podrobností najdete v [4].

9.4 Příklady

1. Implementujte algoritmus popsaný v poznámce 9.1.
2. Naimplementujte algoritmus se symetrickou pivotizací popsaný v poznámce 9.2.
3. Využitím Choleského rozkladu spočítejte

$$\mathbf{y} = \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{x},$$

kde $\mathbf{G} \in \mathbb{R}^{m,n}$ je libovolná matice taková, že $m > n$ a vektor $\mathbf{x}$ je libovolný vektor vhodné dimenze.

Kapitola 10

Přeuspořádávací algoritmy

Informace této kapitoly jsou převzaty z [4].

10.1 Základní přeuspořádání

Jak už jsme se zmínili v předchozí kapitole, vhodné přeuspořádání řádků/sloupců matice **A** pomocí permutačních matic **P** a **Q** (případně permutačních vektorů **p** a **q**) může ovlivnit řídkost jejích **LU** a **QR** faktorů či Choleského faktoru. Nejjednodušší takové přeuspořádání je setřídit sloupce podle počtu nenulových prvků. Tento postup může dobře fungovat pro matice s velmi nepravidelnou strukturou, zvláště pokud jsou velké rozdíly v počtu nenulových prvků v řádcích či sloupcích.

Funkce `p = colperm(A)` počítá toto přeuspořádání. Soubor `colperm.m` má pouze jeden řádek:

$$[\sim, \mathtt{p}] = \mathtt{sort}(\mathtt{sum}(\mathtt{spones}(\mathtt{A})));$$

Postupně se provádí tyto kroky:

1. Funkce `spones` vytvoří řídkou matici s jedničkami na pozicích všech nenulových prvků v matici **A**.

2. Funkce `sum` sečne sloupce této matice, takže vrátí vektor s početem nenulových prvků v každém sloupci.

3. Funkce `sort` seřadí hodnoty ve vzestupném pořadí. Druhý výstupní argument je požadovaný permutační vektor.

10.2 Přeuspořádání s redukcí šířky pásu (RCM)

Reverzní Cuthill-McKee algoritmus slouží k redukci šířky pásu matice. Není zaručeno, že algoritmus najde nejmenší šířku pásu, ale obvykle tomu tak bývá. Funkce `symrcm(A)` pracuje nad nenulovou strukturou symetrické matice $\mathbf{A} + \mathbf{A}^T$.

10.3 Přeuspořádání pomocí aproximace minimálního stupně (AMD)

Stupeň uzlu v grafu je dán počtem hran, které do tohoto uzlu zasahují. Je to stejný počet, jako počet nediagonálních nenulových prvků v příslušném řádku matice sousednosti. Algoritmus AMD sleduje, jak se v průběhu Gaussovy eliminace nebo Choleského faktorizace mění tyto stupně. Na základě toho pak generuje přeuspořádání s minimálním stupněm.

Jedná se o složitý a výkonný algoritmus, který obvykle vede k řidším faktorům než většina jiných přeuspořádávacích algoritmů. Vzhledem k tomu, že sledování stupně každého uzlu je velmi časově náročné, využívá se v algoritmu pouze přibližná hodnota. V `MATLABu` tento algoritmus provádějí tyto funkce:

- `symamd` - pro symetrické matice,
- `colamd` - pro nesymetrické matice a symetrické matice tvaru $\mathbf{A}\mathbf{A}^T$ nebo $\mathbf{A}^T\mathbf{A}$.

Různé parametry algoritmu se dají nastavovat pomocí funkce `spparms`.

Více podrobností najdete v [4].

Poznámka 10.1. Přeuspořádávací algoritmy jsou zabudovany do algoritmů `lu`, `chol` apod., ale aktivují se pouze při některých způsobech volání.

Příklad 10.2. Spusťte následující zdrojový kód [4] v Matlabu a prostudujte vygenerované obrázky.

Algoritmus 10.1: Ukázka algoritmů `symrcm` a `symamd`

```
B = bucky+4*speye(60);
r = symrcm(B);
s = symamd(B);
R = B(r,r);
S = B(s,s);
figure
subplot(2,2,1), spy(R,4), title('B(r,r)')
subplot(2,2,2), spy(S,4), title('B(s,s)')
subplot(2,2,3), spy(chol(R),4), title('chol(B(r,r))')
subplot(2,2,4), spy(chol(S),4), title('chol(B(s,s))')
```

Kapitola 11

QR rozklad

V této kapitole si ukážeme, jak prakticky ortogonalizovat množinu lineárně nezávislých aritmetických vektorů tvořících sloupce matice $\mathbf{A}$, představíme si $\mathbf{QR}$ rozklad a popíšeme si několik způsobů, jak tento rozklad implementovat. Začneme Gramovým-Schmidtovým algoritmem, pak představíme matici rotace a s ní související Givensovu transformaci a vše zakončíme maticí zrcadlení a Householderovou transformací.

S ortogonálními maticemi se často setkáváme v numerických metodách, neboť jejich inverzní matici lze získat transponováním a násobení ortogonálními maticemi nezesiluje zaokrouhlovací chyby. Jak uvidíme v kapitole 12, $\mathbf{QR}$ rozklad můžeme využít také například k výpočtu spektrálního rozkladu.

Definice 11.1. Čtvercová matice $\mathbf{Q}$, která splňuje

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$$

se nazývá *ortogonální matice*. Ortogonální matice má tedy ortonormální sloupce a splňuje

$$\mathbf{Q}^{-1} = \mathbf{Q}^T.$$

Sloupce matice $\mathbf{Q}$ tvoří ortonormální množinu vektorů, t.j.

$$(\mathbf{q}_i, \mathbf{q}_j) = \mathbf{q}_i^T\mathbf{q}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases},$$

kde $(\cdot,\cdot)$ značí Euklidovský skalární součin a $\mathbf{q}_i$ označuje sloupec matice $\mathbf{Q}$.

Nechť $\mathbf{A} \in \mathbb{R}^{m,n}$ je libovolná matice, pak existuje ortogonální matice $\mathbf{Q} \in \mathbb{R}^{m,m}$ a horní trojúhelníková matice $\mathbf{R} \in \mathbb{R}^{m,n}$ takové, že platí

$$\mathbf{A} = \mathbf{QR}.$$

Uvažujme, nejprve že $m = n$. Rozepišme si rozklad $\mathbf{A} = \mathbf{QR}$, jako v [2], ve tvaru

$$\left(\begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_n \end{array}\right) = \left(\begin{array}{c|c|c|c} \mathbf{q}_1 & \mathbf{q}_2 & \dots & \mathbf{q}_n \end{array}\right) \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix}.$$

Jednotlivé sloupce matice $\mathbf{A}$ můžeme zapsat jako lineární kombinaci sloupců matice $\mathbf{Q}$, takže získáváme [2]

$$\begin{aligned} \mathbf{a}_1 &= r_{11}\mathbf{q}_1, \\ \mathbf{a}_2 &= r_{12}\mathbf{q}_1 + r_{22}\mathbf{q}_2, \\ &\vdots \\ \mathbf{a}_n &= r_{1n}\mathbf{q}_1 + r_{2n}\mathbf{q}_2 + \cdots + r_{nn}\mathbf{q}_n. \end{aligned} \tag{11.1}$$

11.1 Gramův-Schmidtův proces

Naším úkolem je sestavit $\mathbf{QR}$ faktorizaci matice $\mathbf{A}$, tedy ke známým sloupcům $\mathbf{a}_1, \dots, \mathbf{a}_n$ matice $\mathbf{A}$ chceme dopočítat sloupce $\mathbf{q}_1, \dots, \mathbf{q}_n$ matice $\mathbf{Q}$ a prvky horní trojúhelníkové matice $\mathbf{R}$.

Proces ortonormalizace začínáme volbou

$$\mathbf{v}_1 = \mathbf{a}_1, \quad \mathbf{q}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}, \quad r_{11} = \|\mathbf{v}_1\|.$$

Předpokládejme nyní, že již známe vektory

$$\mathbf{q}_1, \dots, \mathbf{q}_{j-1}.$$

Vektor $\mathbf{a}_j$ si můžeme vyjádřit jako lineární kombinaci vektorů $\mathbf{q}_1, \dots, \mathbf{q}_j$ ve tvaru

$$\mathbf{a}_j = r_{1j}\mathbf{q}_1 + \cdots + r_{ij}\mathbf{q}_i + \cdots + r_{j-1,j}\mathbf{q}_{j-1} + r_{jj}\mathbf{q}_j. \tag{11.2}$$

Označme si nyní

$$\mathbf{v}_j = r_{jj}\mathbf{q}_j. \tag{11.3}$$

Pro pomocný vektor $\mathbf{v}_j$ můžeme z (11.2) psát

$$\mathbf{v}_j = \mathbf{a}_j - r_{1j}\mathbf{q}_1 - \ldots - r_{ij}\mathbf{q}_i - \ldots - r_{j-1,j}\mathbf{q}_{j-1}.$$

Požadujeme, aby vektor $\mathbf{v}_j$ byl ortogonální k již známým vektorům $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$. Proto pro $k = 1, \dots, j-1$ musí platit

$$0 = (\mathbf{q}_k, \mathbf{v}_j) = (\mathbf{q}_k, \mathbf{a}_j - r_{1j}\mathbf{q}_1 - \ldots - r_{ij}\mathbf{q}_i - \ldots - r_{j-1,j}\mathbf{q}_{j-1}).$$

Jelikož $(\mathbf{q}_k, \mathbf{q}_i) = 0$ pro $k \neq i$ dostáváme

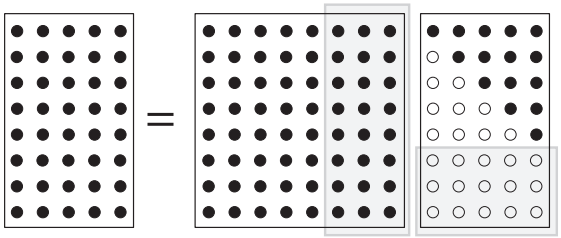
$$(\mathbf{q}_i, \mathbf{a}_j - r_{ij}\mathbf{q}_i) = (\mathbf{q}_i, \mathbf{a}_j) - r_{ij}\underbrace{(\mathbf{q}_i, \mathbf{q}_i)}_{=1} = 0 \quad \Rightarrow \quad r_{ij} = (\mathbf{q}_i, \mathbf{a}_j).$$

Normováním získáváme hledaný ortonormální vektor $\mathbf{q}_j$, tedy

$$\mathbf{q}_j = \frac{\mathbf{v}_j}{\|\mathbf{v}_j\|} \overset{\text{podle (11.3)}}{=} \frac{\mathbf{v}_j}{r_{jj}} \quad \Rightarrow \quad r_{jj} = \|\mathbf{v}_j\|.$$

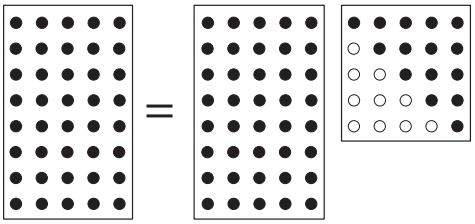
Je-li $m > n$ můžeme **QR** rozklad uvažovat ve dvou verzích

- v plné $\mathbf{A} = \mathbf{QR}$



Obrázek 11.1: Úplný **QR** rozklad.

- v redukované $\mathbf{A} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}}$



Obrázek 11.2: Redukovaný **QR** rozklad.

Implementace pro $m = n$ je uvedena jako algoritmus 11.1. Implementaci pro $m > n$ ponecháváme na samostatné procvičení.

Algoritmus 11.1: Gramův-Schmidtův proces

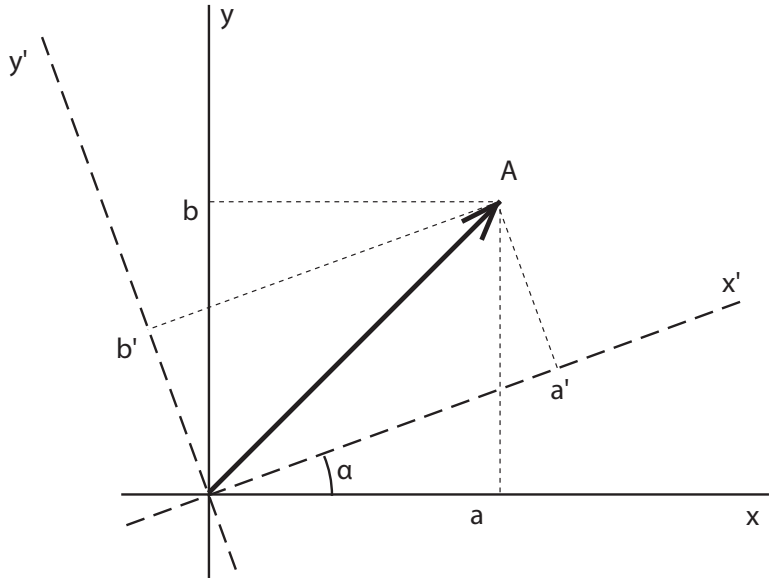
```
Q=zeros(n,n); R=zeros(n,n);
for j=1:n
  v=A(:,j);
  for i=1:j-1
    R(i,j)=Q(:,i)'*A(:,j);
    v=v-R(i,j)*Q(:,i);
  end
  R(j,j)=norm(v);
  Q(:,j)=v/R(j,j);
end
```

11.2 Givensova transformace

Givensova transformace využívá v procesu ortogonalizace *matici rotace*, nazývanou také *Givensova matice*. Začněme tedy odvozením této matice.

11.2.1 Odvození matice rotace

Uvažujme dva souřadné systémy xy a $x'y'$, které svírají úhel α, jak je znázorněno na obrázku 11.3. Bod $\mathbf{A}$ má rouřadnice $\mathbf{A} = (a, b)$, resp. $\mathbf{A} = (a', b')$. Naším úkolem bude vyjádřit jeho souřadnice v čárkovaných souřadnicích využitím souřadnic nečárkovaných.



Obrázek 11.3: Odvození matice rotace.

K odvození budeme potřebovat definiční vztahy goniometrických funkcí sin a cos. Uvažujme pravoúhlý trojúhelník jako na obrázku 11.4 s odvěsnami u, v a přeponou w. Označme úhel svíraný stranami v, w jako φ. Můžeme tedy psát

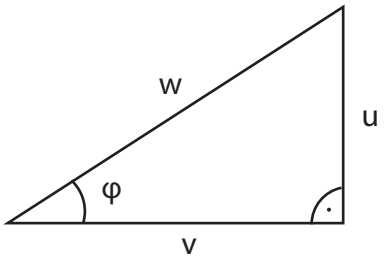
$$\sin\varphi = \frac{u}{w} \quad \text{a} \quad \cos\varphi = \frac{v}{w}.$$

Užitím těchto vztahů z obrázku 11.5 odvodíme, že

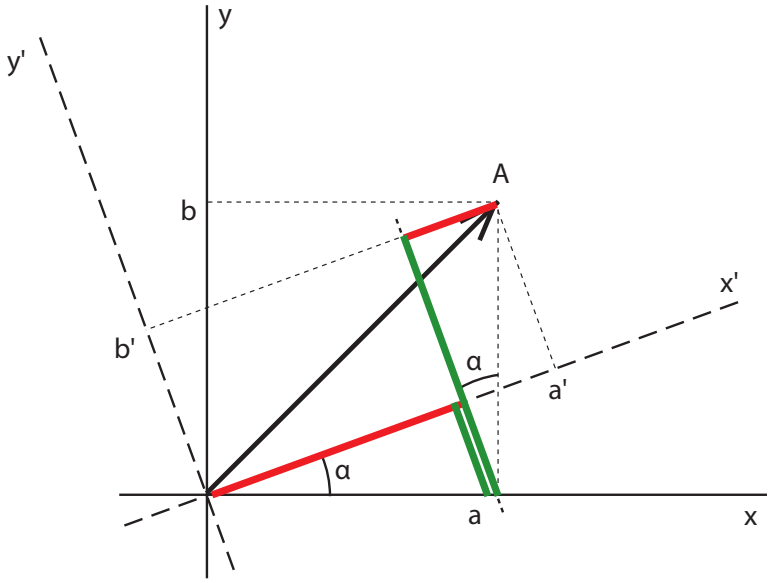
$$\begin{aligned} a' &= a\cos\alpha + b\sin\alpha \\ b' &= -a\sin\alpha + b\cos\alpha. \end{aligned} \tag{11.4}$$

První rovnice vznikne součtem červeně znázorněných úseček, druhá rovnice pak rozdílem zelených úseček z obrázku 11.5. Tuto soustavu pak můžeme zapsat maticově

$$\begin{pmatrix} a' \\ b' \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$



Obrázek 11.4: Definice finkcí sin a cos.



Obrázek 11.5: Odvození matice rovinné rotace.

Matici

$$\mathbf{G} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \tag{11.5}$$

pak nazýváme *maticí rovinné rotace*, která vektor $(a, b)^T$ otočí o úhel $-\alpha$. V dalším textu používáme značení $c = \cos\alpha$ a $s = \sin\alpha$.

11.2.2 Nulování prvků

Matici rotace můžeme využít k nulovaní prvků ve vektoru. Uvažujme [3], že

$$\mathbf{G}^T\mathbf{x} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sqrt{a^2+b^2} \\ 0 \end{pmatrix}.$$

Není těžké dopočítat, že

$$c = \frac{a}{\sqrt{a^2+b^2}} \quad \text{a} \quad s = \frac{-b}{\sqrt{a^2+b^2}}.$$

Tento postup můžeme zobecnit také pro vektory délky n. Zaveďme matici rotace v rovině ij [3] ve tvaru

$$\mathbf{G}_{ij} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & & s & & i \\ & & & \ddots & & & \\ & & -s & & c & & j \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}.$$

Jedná se tedy o jednotkovou matici, do které umístíme na odpovídající pozice v i-tém a j-tém řádku a i-tém a j-tém sloupci matici rovinné rotace (11.5).

Podívejme se na operaci

$$\mathbf{y} = \mathbf{G}_{ij}^T \mathbf{x}.$$

Jednotlivé složky vektoru $\mathbf{y}$ budou definovány jako [3]

$$y_k = \begin{cases} cx_i - sx_j, & k = i, \\ sx_i + cx_j, & k = j, \\ x_k, & k \neq i, j. \end{cases}$$

Pro vynulování prvku v j-tém řádku vektoru $\mathbf{x}$ pak

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad s = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}.$$

Poznámka 11.2. Matice $\mathbf{G}_{ij}$ je ortogonální a součin ortogonálních matic je opět ortogonální matice.

11.2.3 Givensova QR metoda

Pro vynulování prvku v i-tém řádku sestavíme matici rotace ve tvaru

$$\mathbf{G}(i-1, i), \quad \text{kde} \quad c = \frac{x_{i-1}}{\sqrt{x_{i-1}^2 + x_i^2}}, \quad s = \frac{-x_i}{\sqrt{x_{i-1}^2 + x_i^2}}.$$

Budeme tedy upravovat matici $\mathbf{A}$. Postup výpočtu vypadá takto

$$\mathbf{A} = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} \overset{\mathbf{G}(2,3)^T}{\longrightarrow} \begin{pmatrix} * & * & * \\ * & * & * \\ 0 & * & * \end{pmatrix} \overset{\mathbf{G}(1,2)^T}{\longrightarrow}$$

$$\begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix} \overset{\mathbf{G}(2,3)^T}{\longrightarrow} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix} = \mathbf{R}.$$

Postupnou aplikací jednotlivých rotačních matic jsme získali matici $\mathbf{R}$. Můžeme tedy psát

$$\left\{\mathbf{G}_p^T \dots \mathbf{G}_1^T \mathbf{A} = \mathbf{R} \quad \Leftrightarrow \quad \mathbf{Q}^T\mathbf{A} = \mathbf{R}\right\}$$
$$\Downarrow$$
$$\mathbf{A} = \mathbf{QR}, \quad \mathbf{Q} = \mathbf{G}_1 \dots \mathbf{G}_p.$$

Implementace je znázorněna v algoritmu 11.2.

Příklad 11.3. Pomocí Givensovy **QR** metody spočítejte **QR** rozklad matice

$$\mathbf{A} = \begin{pmatrix} -1 & 4 & -1 \\ -2 & -1 & -11 \\ 2 & 10 & 2 \end{pmatrix}.$$

Řešení. Pro vynulování prvku $\mathbf{A}(3,1)$ potřebujeme sestavit matici rotace $\mathbf{G}_1(2,3)^T$. Určíme hodnoty

$$c = \frac{-2}{\sqrt{(-2)^2+2^2}} = \frac{-\sqrt{2}}{2}, \quad s = \frac{-2}{\sqrt{(-2)^2+2^2}} = \frac{-\sqrt{2}}{2}$$

a získáváme

$$\mathbf{G}_1(2,3) = \begin{pmatrix} 1 & & \\ & -\sqrt{2}/2 & -\sqrt{2}/2 \\ & \sqrt{2}/2 & -\sqrt{2}/2 \end{pmatrix},$$

dále pak

$$\mathbf{A}_1 = \mathbf{G}_1(2,3)^T\mathbf{A} = \begin{pmatrix} -1 & 4 & -1 \\ 2\sqrt{2} & 11\sqrt{2}/2 & 13\sqrt{2}/2 \\ 0 & -9\sqrt{2}/2 & 9\sqrt{2}/2 \end{pmatrix}.$$

Dále budeme nulovat prvek $\mathbf{A}_1(2,1)$ pomocí $\mathbf{G}_2(1,2)$. Získáváme $c = -1/3$, $s = -2\sqrt{2}/3$

$$\mathbf{G}_2(1,2) = \begin{pmatrix} -1/3 & -2\sqrt{2}/3 & \\ 2\sqrt{2}/3 & -1/3 & \\ & & 1 \end{pmatrix}$$

a

$$\mathbf{A}_2 = \mathbf{G}_2(1,2)^T\mathbf{A}_1 = \begin{pmatrix} 3 & 6 & 9 \\ 0 & -9\sqrt{2}/2 & -3\sqrt{2}/2 \\ 0 & -9\sqrt{2}/2 & 9\sqrt{2}/2 \end{pmatrix}.$$

Další prvek, který budeme nulovat je $\mathbf{A}_2(3,2)$. Sestavovat budeme $\mathbf{G}_3(2,3)$, tedy $c = -\sqrt{2}/2$, $s = -\sqrt{2}/2$ a pak

$$\mathbf{G}_3(2,3) = \begin{pmatrix} 1 & & \\ & -\sqrt{2}/2 & \sqrt{2}/2 \\ & -\sqrt{2}/2 & -\sqrt{2}/2 \end{pmatrix}.$$

Následně můžeme psát

$$\mathbf{A}_3 = \mathbf{G}_3(2,3)^T\mathbf{A}_2 = \begin{pmatrix} 3 & 6 & 9 \\ 0 & 9 & -3 \\ 0 & 0 & -6 \end{pmatrix} = \mathbf{R}.$$

Matici $\mathbf{Q}$ získáme ve tvaru

$$\mathbf{Q} = \mathbf{G}_1(2,3)\mathbf{G}_2(1,2)\mathbf{G}_3(2,3) = \begin{pmatrix} -1/3 & 2/3 & -2/3 \\ -2/3 & 1/3 & 2/3 \\ 2/3 & 2/3 & 1/3 \end{pmatrix}.$$

Snadno ověříme, že opravdu platí $\mathbf{QR} = \mathbf{A}$. ▲

Algoritmus 11.2: Givensova **QR** metoda

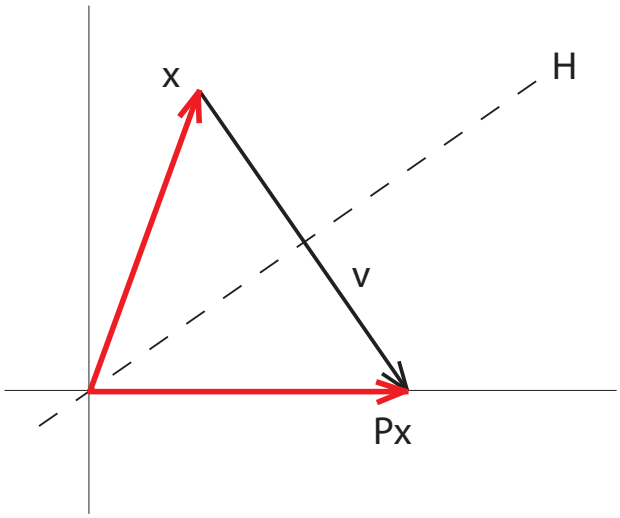
```
Q=eye(m); R=A;
for j=1:n
  for i=m:(-1):j+1
    x=R(:,j);
    if norm([x(i-1),x(i)])>0
      c=x(i-1)/norm([x(i-1),x(i)]);
      s=-x(i)/norm([x(i-1),x(i)]);
      G=eye(m);
      G([i-1,i],[i-1,i])=[c,s;-s,c];
      R=G'*R;
      Q=Q*G;
    end
  end
end
```

11.3 Householderova transformace

Začněmě popisem obrázku 11.6. K zadanému vektoru $\mathbf{x}$ umíme sestrojit jeho projekci $\mathbf{Px}$ do osy x stejné délky [2]. Můžeme tedy psát

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \quad \xrightarrow{\mathbf{P}} \quad \mathbf{Px} = \begin{pmatrix} \|\mathbf{x}\| \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \|\mathbf{x}\|\mathbf{e}_1,$$

kde $\mathbf{e}_1 = (1,0,0,\ldots,0)^T$ je první vektor standardní báze. Vektor $\mathbf{Px}$ má tedy nenulovou pouze první složku, ostatní se díky projekci vynulují. Chceme, aby $\mathbf{Px}$ byl stejné délky jako $\mathbf{x}$, tedy $\|\mathbf{Px}\| = [\mathbf{Px}]_1 = \|\mathbf{x}\|$.



Obrázek 11.6: Zrcadlení.

Na vektor $\mathbf{Px}$ můžeme pohlížet také jako na zrcadlový obraz vektoru $\mathbf{x}$ s osou souměrnosti H. Tato osa prochází počátkem a je kolmá k vektoru

$$\mathbf{v} = \mathbf{Px} - \mathbf{x} = \|\mathbf{x}\|\mathbf{e}_1 - \mathbf{x}. \tag{11.6}$$

V následujícím odstavci si ukážeme, jak vypadá *matice zrcadlení* $\mathbf{P}$.

11.3.1 Odvození matice zrcadlení

Z obrázku 11.7 je zřejmé, že můžeme $\mathbf{Px}$ vyjádřit ve tvaru

$$\mathbf{Px} = \mathbf{x} - 2\mathbf{x}_\mathbf{v}, \tag{11.7}$$

kde $\mathbf{x}_\mathbf{v}$ je složka vektoru $-\mathbf{x}$ zobrazeného do směru vektoru $\mathbf{v}$. Tuto složku můžeme vyjádřit ve tvaru

$$\mathbf{x}_\mathbf{v} = \mathbf{v}_n\|\mathbf{x}\|\cos\alpha, \tag{11.8}$$

kde

$$\mathbf{v}_n = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

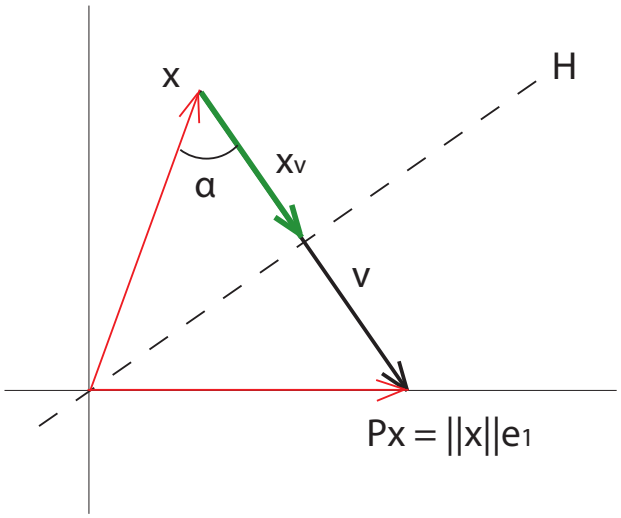
udává směr a $\|\mathbf{x}\|\cos\alpha$ určuje velikost.

V dalším kroku dosadíme do (11.8) za $\mathbf{v}_n$, rozšíříme $\|\mathbf{v}\|$ a využijeme definici skalárního součinu

$$\mathbf{v}^T\mathbf{x} = \|\mathbf{v}\|\|\mathbf{x}\|\cos\alpha.$$

Můžeme tedy psát

$$\mathbf{x}_\mathbf{v} = \mathbf{v}_n\|\mathbf{x}\|\cos\alpha = \frac{\mathbf{v}}{\|\mathbf{v}\|}\|\mathbf{x}\|\cos\alpha = \frac{\mathbf{v}}{\|\mathbf{v}\|^2}\|\mathbf{v}\|\|\mathbf{x}\|\cos\alpha = \frac{\mathbf{v}}{\|\mathbf{v}\|^2}\mathbf{v}^T\mathbf{x}.$$



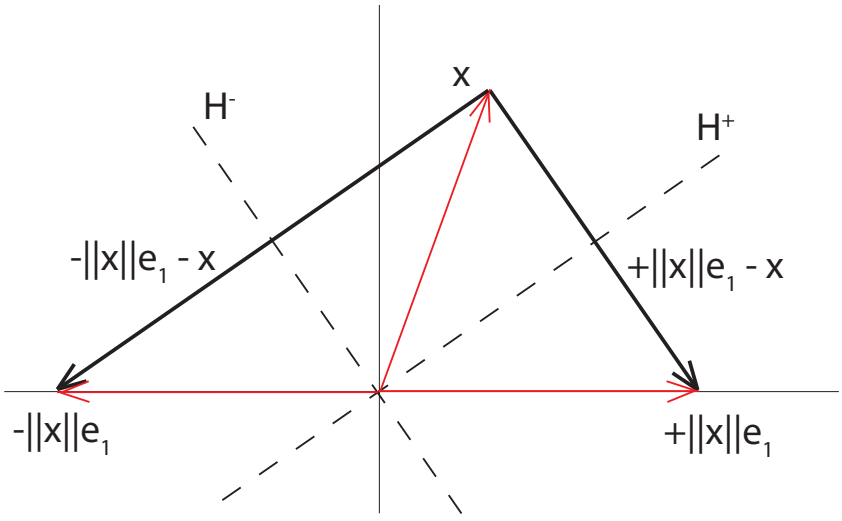
Obrázek 11.7: Zrcadlení.

Dosazením do (11.7) získáváme

$$\mathbf{Px} = \mathbf{x} - 2\mathbf{x_v} = \mathbf{x} - 2\frac{\mathbf{vv}^T}{\|\mathbf{v}\|^2}\mathbf{x} = \left(\mathbf{I} - 2\frac{\mathbf{vv}^T}{\|\mathbf{v}\|^2}\right)\mathbf{x}.$$

Matice zrcadlení (nebo také *Householderova matice*) má tedy tvar

$$\mathbf{P} = \mathbf{I} - 2\frac{\mathbf{vv}^T}{\|\mathbf{v}\|^2} = \mathbf{I} - 2\frac{\mathbf{vv}^T}{\mathbf{v}^T\mathbf{v}}.$$



Obrázek 11.8: Zrcadlení.

Všimněme si, že obraz vektoru $\mathbf{x}$ není definovaný jednoznačně [2]. Může být zobrazen do libovolného vektoru $z\|\mathbf{x}\|\mathbf{e}_1$, kde $|z| = 1$. V oboru reálných čísel máme tedy dvě možnosti, jak je vidět z obrázku 11.8.

Aby tato metoda nebyla citlivá na zaokrouhlovací chyby, budeme požadovat, aby projekce $z\|\mathbf{x}\|\mathbf{e}_1$ nebyla příliš blízko $\mathbf{x}$. Budeme tedy volit $z = -\text{sign}(x_1)$, kde x_1 je první souřadnice vektoru $\mathbf{x}$. Definujme $\text{sign}(0) = 1$. Vektor $\mathbf{v}$ pak bude mít tvar

$$\mathbf{v} = -\text{sign}(x_1)\|\mathbf{x}\|\mathbf{e}_1 - \mathbf{x}. \tag{11.9}$$

Poznámka 11.4. Pokud by osa zrcadlení H^+ svírala s osou x velmi malý úhel [2], pak by vektor $\mathbf{x}$ ležel blízko svému obrazu. Vektor $\mathbf{v}$ by byl mnohem kratší než $\mathbf{x}$ a v důsledku odčítání dvou velmi blízkých hodnot ve vektoru $\mathbf{v}$, by pak vznikaly zaokrouhlovací chyby. Tomuto předejdeme, budeme-li vektor $\mathbf{v}$ počítat podle předpisu (11.9).

Poznámka 11.5. Snadno ověříme, že projekce $\mathbf{P}$ je symetrická ortogonální matice. A připomeňme, že součin ortogonálních matic je opět ortogonální matice.

11.3.2 Householderova QR metoda

Postup popsaný v předchozí kapitole můžeme využít k výpočtu **QR** rozkladu [2]. Budeme sestavovat projekce $\mathbf{P}_i$, které nám postupně vynulují prvky v jednotlivých sloupcích. Schématicky tento proces můžeme znázornit takto:

$$\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} \xrightarrow{\mathbf{P}_1} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix} \xrightarrow{\mathbf{P}_2} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \end{pmatrix} \xrightarrow{\mathbf{P}_3} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{pmatrix}$$

Postupným nulováním nám vznikla trojúhelníková matice $\mathbf{R}$. Maticově pak můžeme psát

$$\mathbf{P}_n \cdots \mathbf{P}_2\mathbf{P}_1\mathbf{A} = \mathbf{Q}^T\mathbf{A} = \mathbf{R} \quad \Rightarrow \quad \mathbf{A} = \mathbf{QR}, \quad \mathbf{Q} = \mathbf{P}_1 \dots \mathbf{P}_n, \tag{11.10}$$

kde n je počet sloupců matice $\mathbf{A}$.

Implementaci tohoto postupu uvádíme v algoritmu 11.3.

Příklad 11.6. Pomocí Householderovy **QR** metody spočítejte **QR** rozklad matice

$$\mathbf{A} = \begin{pmatrix} -1 & 4 & -1 \\ -2 & -1 & -11 \\ 2 & 10 & 2 \end{pmatrix}.$$

Řešení. V prvním kroku budeme pracovat s prvním sloupcem a tedy

$$\mathbf{x} = (-1, -2, 2)^T, \quad \|\mathbf{x}\| = \sqrt{1+4+4} = 3, \quad \text{sign}(x_1) = -1.$$

Pro vektor $\mathbf{v}$ můžeme podle vztahu (11.9) psát

$$\mathbf{v} = -\text{sign}(x_1)\|\mathbf{x}\|\mathbf{e}_1 - \mathbf{x} = \|\mathbf{x}\|\mathbf{e}_1 - \mathbf{x} = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} -1 \\ -2 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ -2 \end{pmatrix}.$$

První projekční matice pak má tvar

$$\mathbf{P}_1 = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} = \begin{pmatrix} -1/3 & -2/3 & 2/3 \\ -2/3 & 2/3 & 1/3 \\ 2/3 & 1/3 & 2/3 \end{pmatrix}$$

a její aplikací pak získáváme

$$\mathbf{R}_1 = \mathbf{P}_1\mathbf{A} = \begin{pmatrix} 3 & 6 & 9 \\ 0 & 0 & -6 \\ 0 & 9 & -3 \end{pmatrix}.$$

V další kroku budeme pracovat s druhým sloupcem této matice a tedy

$$\mathbf{x} = (0, 9)^T, \qquad \|\mathbf{x}\| = 9, \qquad \text{sign}(x_1) = 1.$$

Z vektoru

$$\mathbf{v} = -\|\mathbf{x}\|\mathbf{e}_1 - \mathbf{x} = \begin{pmatrix} -9 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 9 \end{pmatrix} = \begin{pmatrix} -9 \\ -9 \end{pmatrix}$$

pak získáváme

$$\mathbf{P'}_2 = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}, \quad \Rightarrow \quad \mathbf{P}_2 = \left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & 0 & -1 \\ 0 & -1 & 0 \end{array}\right)$$

a tedy

$$\mathbf{R}_2 = \mathbf{P}_2\mathbf{P}_1\mathbf{A} = \begin{pmatrix} 3 & 6 & 9 \\ 0 & -9 & 3 \\ 0 & 0 & 6 \end{pmatrix}.$$

Matici $\mathbf{Q}$ pak podle (11.10) získáváme ve tvaru

$$\mathbf{Q} = \mathbf{P}_1\mathbf{P}_2 = \begin{pmatrix} -1/3 & -2/3 & 2/3 \\ -2/3 & -1/3 & -2/3 \\ 2/3 & -2/3 & -1/3 \end{pmatrix}.$$

▲

Algoritmus 11.3: Householderova **QR** metoda

```
Q=eye(m); R=A;
for j=1:n
  x=R(j:m,j);
  v=-sign(x(1))*norm(x)*eye(m-j+1,1)-x;
  if norm(v)>0,
    v=v/norm(v);
    P=I; P(j:m,j:m)=P(j:m,j:m)-2*v*v';
    R=P*R;
    Q=Q*P;
  end
end
```

11.4 Funkce Matlabu

- `[Q,R] = qr(A)` ... plná verze
- `[Q,R] = qr(A,0)` ... redukovaná verze. Pokud $m \leqq n$, funkce funguje stejně jako `[Q,R] = QR(A)`.

Další možnosti a více podrobností najdete v [4].

11.5 Příklady

1. Implementujte Gramův-Schmidtův proces pro $m > n$. Plnou i redukovanou variantu.

Kapitola 12

Vlastní čísla a spektrální rozklad

Problém vlastních čísel a vektorů se vyskytuje v mnoha fyzikálních a inženýrských aplikacích. Jako příklad můžeme zmínit analýzu stability systémů nebo třeba fyziku rotujících těles.

V této kapitole si představíme vlastní čísla matice, s nimi související spektrální rozklad a několik algoritmů pro výpočet tohoto rozkladu. Jak uvidíme, využijeme k tomu s výhodou nám již známý **QR** rozklad.

Uvažujme čtvercovou matici $\mathbf{A}$ řádu n. Nenulový vektor $\mathbf{v}$ nazýváme *vlastním vektorem* matice $\mathbf{A}$ a λ příslušné *vlastní číslo* matice $\mathbf{A}$, platí-li

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}. \tag{12.1}$$

Tento vztah můžeme přepsat ve tvaru

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{o}.$$

Jelikož $\mathbf{v} \neq \mathbf{o}$, musí být matice $\mathbf{A} - \lambda\mathbf{I}$ singulární a tedy

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0. \tag{12.2}$$

Vlasní čísla matice $\mathbf{A}$ jsou pak kořeny rovnice (12.2), která se nazývá *charakteristická rovnice*. Jelikož hledání kořenů takovéto rovnice patří mezi tzv. *np* těžké problémy (jejich náročnost roste s p-tou mocninou n), hledáme jiné cesty k určení vlastních čísel.

12.1 Spektrální rozklad

Rovnici (12.1) můžeme zapsat také maticově ve tvaru

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Lambda}, \tag{12.3}$$

kde

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \quad \text{a} \quad \mathbf{V} = \left(\begin{array}{c|c|c} & & \\ \mathbf{v}_1 & \dots & \mathbf{v}_n \\ & & \end{array} \right).$$

Vidíme tedy, že jestliže $\mathbf{v}_j$ je j-tý sloupec matice $\mathbf{V}$ a λ_j je j-tý diagonální prvek matice $\mathbf{\Lambda}$, pak $\mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j$.

Nechť je dána realná **symetrická** matice $\mathbf{A}$ řádu n. Pak existují ortogonální matice $\mathbf{Q}$ a diagonální matice $\mathbf{D}$ takové, že

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T. \tag{12.4}$$

Diagonální prvky matice $\mathbf{D}$ jsou vlastní čísla matice $\mathbf{A}$ a sloupce matice $\mathbf{Q}$ jsou ortonormální vlastní vektory matice $\mathbf{A}$. Množinu vlastních čísel $\mathbf{A}$ nazýváme *spektrem* matice $\mathbf{A}$.

12.2 Výpočet spektrálního rozkladu pomocí QR algoritmu

Pro nalezení spektrálního rozkladu $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T$ se přímo nabízí využít **QR** rozkladu a to tak [2], že po každém vynulování části sloupce pod diagonálním prvkem bychom se pokusili vynulovat stejnou transformací i část řádku za diagonálním prvkem, tedy

$$\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1 \Rightarrow \mathbf{Q}_1^T\mathbf{A} = \mathbf{R}_1 = \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}, \tag{12.5}$$

avšak

$$\mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1 = \mathbf{R}_1\mathbf{Q}_1 = \begin{pmatrix} * & 0 & 0 & 0 \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}. \tag{12.6}$$

Z předchozího je tedy patrné, že aplikací stejné ortogonální transformace na sloupce se prvky vynulované ve sloupci mohou opět změnit na nenulové [2]. Nicméně lze dokázat, že postup

$$\begin{aligned} \mathbf{A}_0 &= \mathbf{A} \\ \mathbf{Q}_k\mathbf{R}_k &= \mathbf{A}_{k-1} \\ \mathbf{A}_k &= \mathbf{R}_k\mathbf{Q}_k \end{aligned}$$

konverguje ke spektrálnímu rozkladu. Postupně získáváme

$$\begin{aligned} \mathbf{A}_1 &= \mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1, \\ \mathbf{A}_2 &= \mathbf{Q}_2^T\mathbf{A}_1\mathbf{Q}_2 = \mathbf{Q}_2^T\mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1\mathbf{Q}_2, \\ &\vdots \\ \mathbf{A}_k &= \mathbf{Q}_k\mathbf{A}_{k-1}\mathbf{Q}_k = \underbrace{\mathbf{Q}_k^T \ldots \mathbf{Q}_2^T\mathbf{Q}_1^T}_{\tilde{\mathbf{Q}}_k^T}\,\mathbf{A}\,\underbrace{\mathbf{Q}_1\mathbf{Q}_2 \ldots \mathbf{Q}_k}_{\tilde{\mathbf{Q}}_k}, \end{aligned}$$

a tedy

$$\mathbf{A} = \tilde{\mathbf{Q}}_k \mathbf{A}_k \tilde{\mathbf{Q}}_k^T.$$

Pro $k \longrightarrow \infty$ můžeme psát

$$\tilde{\mathbf{Q}}_k \longrightarrow \mathbf{Q} \quad \text{a} \quad \mathbf{A}_k \longrightarrow \mathbf{D},$$

takže

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T.$$

Tento postup je zapsán v algoritmu 12.1.

Algoritmus 12.1: $\mathbf{QDQ}^T$ rozklad pomocí **QR** metody

```
D = A; Q = I;
while norm(D-diag(diag(D))) >= eps
  [Qk,Rk] = qr(D);
  D = Rk*Qk;
  Q = Q*Qk;
end
```

Poznámka 12.1. V podmínce cyklu while lze použít jakoukoli normu.

12.3 Modifikovaná QR metoda

Možné modifikace [2] v algoritmu 12.1:

1. Před započetím iterací je možno redukovat obecnou symetrickou matici na třidiagonální tvar pomocí konečného počtu symetrických **QR** transformací.
2. Místo matice $\mathbf{A}_{k-1}$ rozkládáme matici s posunutým spektrem

   $$\mathbf{A}_{k-1} - \mu_{k-1}\mathbf{I},$$

   kde μ_{k-1} je odhad některého z vlastních čísel, např. $\mu_{k-1} = \mathbf{A}_{k-1}(n, n)$.

Redukce na třídiagonální tvar

Na následujícím příkladě si ukážeme postup redukce libovolné symetrické matice pomocí Householderových transformací na třídiagonální tvar [2]. Postupnými úpravami získáváme

$$\mathbf{Q}_1^T\mathbf{A} = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ \mathbf{0} & * & * & * \\ \mathbf{0} & * & * & * \end{pmatrix}, \qquad \mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1 = \begin{pmatrix} * & * & \mathbf{0} & \mathbf{0} \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix},$$

$$\mathbf{Q}_2^T\mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1 = \begin{pmatrix} * & * & 0 & 0 \\ * & * & * & * \\ 0 & * & * & * \\ 0 & \mathbf{0} & * & * \end{pmatrix}, \quad \mathbf{Q}_2^T\mathbf{Q}_1^T\mathbf{A}\mathbf{Q}_1\mathbf{Q}_2 = \begin{pmatrix} * & * & 0 & 0 \\ * & * & * & \mathbf{0} \\ 0 & * & * & * \\ 0 & 0 & * & * \end{pmatrix}.$$

Zvýrazněny jsou ty prvky, které jsou v daném kroku modifikovány. Je tedy patrné, že jednou vynulované prvky se již znovu nemohou stát nenulovými. Možný způsob implementace tohoto výpočtu ukazujeme v algoritmu 12.2.

Algoritmus 12.2: Redukce na třídiagonální tvar

```
Q = I;
for j = 1:n-2
  x = A(j+1:n,j);
  v = sign(x(1))*norm(x)*eye(n-j,1)-x;
  if norm(v)>0,
    v = v/norm(v);
    P = I; P(j+1:n,j+1:n) = P(j+1:n,j+1:n)-2*v*v';
    A = P*A*P;
    Q = Q*P;
  end
end
```

V algoritmu (12.3) ukazujeme modifikovaný algoritmus.

Algoritmus 12.3: $\mathbf{QDQ}^T$ rozklad: modifikovaná **QR** metoda

```
D = A; Q = I;
for j = 1:n-2
  x = D(j+1:n,j);
  v = sign(x(1))*norm(x)*eye(n-j,1)-x;
  if norm(v) > 0,
    v = v/norm(v);
    P = I; P(j+1:n,j+1:n) = P(j+1:n,j+1:n)-2*v*v';
    D = P*D*P;
    Q = Q*P;
  end
end

while norm(D-diag(diag(D))) >= eps
  mu = D(n,n);
  [Qk,Rk] = qr(D-mu*eye(n));
  D = Rk*Qk+mu*eye(n);
  Q = Q*Qk;
end
```

12.4 Funkce Matlabu

- pro plné matice
  - `lambda = eig(A)` ... vrací vektor vlasních čísel
  - `[Q,D] = eig(A)` ... vrací matice spektrálního rozkladu, platí $\mathbf{AQ} = \mathbf{QD}$.

- pro řídké matice
  - `D = eigs(A)` ... vrací vektor šesti největších vlasních čísel
  - `[V,D] = eigs(A)` ... vrací na diagonále matice **D** šest největších vlasních čísel, ve sloupcích matice **V** odpovídající vlasní vektory.

Matice, jejíž vlastní čísla chceme spočítat, nemusí být sestavena explicitně. Funkce `eigs` nabízí možnost volání pomocí odkazu na funkci implementující akci matice na vektor. Více podrobností najdete v [4].

Kapitola 13

Singulární rozklad

Nechť je dána matice $\mathbf{A} \in \mathbb{R}^{m,n}$. Pak existují ortogonální matice $\mathbf{U} \in \mathbb{R}^{\mathbf{m},\mathbf{m}}$, $\mathbf{V} \in \mathbb{R}^{\mathbf{n},\mathbf{n}}$ a diagonální matice $\mathbf{S} \in \mathbb{R}^{m,n}$ tak, že platí

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T. \tag{13.1}$$

Diagonální prvky matice $\mathbf{S}$ jsou nezáporné, uspořádané sestupně a nazývají se *singulární čísla* matice $\mathbf{A}$.

13.1 Využití spektrálního rozkladu $\mathbf{A}^T\mathbf{A}$

Hledáme rozklad $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, kde $\mathbf{A} \in \mathbb{R}^{m,n}$. Pak

$$\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{S}^T\mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{V}\mathbf{S}^T\mathbf{S}\mathbf{V}^T.$$

Pokud $m \geqq n$ a

$$\mathbf{A}^T\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T \Rightarrow \operatorname{diag}(\mathbf{S}) = \sqrt{\operatorname{diag}(\mathbf{D})},$$

$\mathbf{V} = \mathbf{Q}$ a $\mathbf{U}$ je taková ortogonální matice, která řeší soustavu $\mathbf{U}\mathbf{S} = \mathbf{A}\mathbf{V}$.

Poznámka 13.1. Zde využíváme toho, že matice $\mathbf{A}^T\mathbf{A}$ je symetrická pozitivně semidefinitní.

Singulární rozklad tedy můžeme získat např. následovně:

1. Sestavíme matici $\mathbf{A}^T\mathbf{A}$.
2. Nalezneme spektrální rozklad této matice $\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$ se současným sestupným uspořádáním prvků na diagonále matice $\mathbf{D}$, viz poznámka 13.2.
3. Z odmocnin diagonálních prvků matice $\mathbf{D}$ sestavíme matici $\mathbf{S}$ typu m, n.
4. Řešíme soustavu $\mathbf{U}\mathbf{S} = \mathbf{A}\mathbf{V}$ pro neznámou $\mathbf{U}$ (lze využít např. **QR** rozklad).

Podrobně se podívejme na řešení $\mathbf{U}\mathbf{S} = \mathbf{A}\mathbf{V}$.

1. Je-li $m \geqq n$ a má-li matice $\mathbf{A}$ všechna singulární čísla nenulová, jsou všechny sloupce matice $\mathbf{AV}$ ortogonální neboť musí být rovna matici $\mathbf{US}$. Sloupce matice $\mathbf{US}$ jsou totiž sloupce matice $\mathbf{U}$ vynásobené diagonálními prvky matice $\mathbf{S}$ a tudíž jsou ortogonální. Pokud tedy provedeme $\mathbf{QR}$ rozklad matice $\mathbf{AV}$, obdržíme ortogonální matici $\mathbf{Q}$ a matici $\mathbf{R}$, která bude diagonální. Pak $\mathbf{U} = \mathbf{QR}^S$, kde
$$\mathbf{R}^S = \mathrm{diag}(\mathbf{R}_{11}/\mathbf{S}_{11}, \ldots, \mathbf{R}_{nn}/\mathbf{S}_{nn}, 1, ..., 1)$$
je diagonální čtvercová matice řádu m.

2. Je-li $m \geqq n$ a má-li matice $\mathbf{A}$ alespoň jedno singulární čislo nulové, pak i některé sloupce matice $\mathbf{AV}$ budou nulové, neboť musí být rovna matici $\mathbf{US}$. Pokud tedy budeme provádět $\mathbf{QR}$ rozklad matice $\mathbf{AV}$, je vhodné provádět tento rozklad pouze na nenulových sloupcích. U takto získané matice $\mathbf{Q}$ je pak potřeba zajistit případnou permutaci sloupců, ke které mohlo dojít díky vynechání řádků.

3. Je-li $m < n$, pak můžeme doplnit matici $\mathbf{A}$ na matici čtvercovou přidáním nulových řádků nebo ji nejdříve transponovat. Tímto převedeme úlohu na typ uvedený v bodě (1) nebo (2). Na závěr pak v prvním případě odebereme z matice $\mathbf{S}$ příslušné přidané nulové řádky a obdobně matici $\mathbf{U}$ redukujeme o přidané řádky a jim příslušné sloupce.

13.2 Další možnost využití spektrálního rozkladu

V této kapitolce si ukážeme alternativní možnost, jak využít spektrální rozklad pro výpočet rozkladu singulárního.

V dalším odvozování se nám bude hodit singulární rozklad matice transponované. Podle (13.1) můžeme psát

$$\mathbf{A}^T = (\mathbf{USV}^T)^T = \mathbf{VS}^T\mathbf{U}^T = \mathbf{VSU}^T. \tag{13.2}$$

Uvažujme nyní matici $\mathbf{H}$ definovanou vztahem

$$\mathbf{H} = \begin{pmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{pmatrix}. \tag{13.3}$$

Spektrální rozklad této matice pak můžeme zapsat ve tvaru

$$\mathbf{H} = \mathbf{QDQ}^T,$$

kde

$$\mathbf{Q} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{V} & \mathbf{V} \\ \mathbf{U} & -\mathbf{U} \end{pmatrix} \quad \text{a} \quad \mathbf{D} = \begin{pmatrix} \mathbf{S} & \mathbf{O} \\ \mathbf{O} & -\mathbf{S} \end{pmatrix}.$$

Můžeme tedy psát

$$\mathbf{H} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T = \frac{1}{2}\begin{pmatrix} \mathbf{V} & \mathbf{V} \\ \mathbf{U} & -\mathbf{U} \end{pmatrix}\begin{pmatrix} \mathbf{S} & \mathbf{O} \\ \mathbf{O} & -\mathbf{S} \end{pmatrix}\begin{pmatrix} \mathbf{V}^T & \mathbf{U}^T \\ \mathbf{V}^T & -\mathbf{U}^T \end{pmatrix}.$$

Roznásobením získáváme matici

$$\mathbf{H} = \frac{1}{2}\begin{pmatrix} \mathbf{O} & 2\,\mathbf{V}\mathbf{S}\mathbf{U}^T \\ 2\,\mathbf{U}\mathbf{S}\mathbf{V}^T & \mathbf{O} \end{pmatrix} = \begin{pmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{pmatrix}.$$

Využitím rovnic (13.1) a (13.2) jsme dostali předpis pro matici $\mathbf{H}$ definovanou ve vztahu (13.3).

Vidíme tedy, že singulární rozklad čtvercové matice $\mathbf{A}$ je možné získat pomocí spektrálního rozkladu matice $\mathbf{H}$. Příklad výpočtu uvádíme v algoritmu 13.1, kde funkce

$$[\mathtt{Q}, \mathtt{D}] = \mathtt{qreigm}(\mathtt{H})$$

je modifikovaná **QR** metoda pro výpočet spektrálního rozkladu, uvedená v algoritmu 12.3.

Singulární rozklad tedy můžeme získat např. následovně:

1. Sestavíme matici $\mathbf{H}$.
2. Nalezneme spektrální rozklad této matice $\mathbf{H} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T$ se současným sestupným uspořádáním prvků matice $\mathbf{D}$, viz poznámka 13.2.
3. Z poloviny diagonálních prvků matice $\mathbf{D}$ sestavíme matici $\mathbf{S}$.
4. Z prvních n sloupců matice $\mathbf{Q}$ vybereme matice $\mathbf{V}$ a $\mathbf{U}$.

Poznámka 13.2. Problémy může způsobit symetrická permutace řádků a sloupců matice $\mathbf{D}$ reprezentovaná permutační maticí $\mathbf{P}$. Matici $\mathbf{P}$, která má tvar

$$\mathbf{H}\tilde{\mathbf{Q}} = \tilde{\mathbf{Q}}\tilde{\mathbf{D}}, \quad \tilde{\mathbf{D}} = \mathbf{P}\mathbf{D}\mathbf{P}^T$$

využíváme pro sestupné uspořádání singulárních čísel. Odtud pak

$$\mathbf{H}\tilde{\mathbf{Q}}\mathbf{P} = \tilde{\mathbf{Q}}\tilde{\mathbf{D}}\mathbf{P} = \tilde{\mathbf{Q}}\underbrace{\mathbf{P}\mathbf{P}^T}_{\mathbf{I}}\tilde{\mathbf{D}}\mathbf{P} = \tilde{\mathbf{Q}}\mathbf{P}\mathbf{D} \quad \Rightarrow \quad \mathbf{Q} = \tilde{\mathbf{Q}}\mathbf{P}.$$

Algoritmus 13.1: Singulární rozklad

```
A= rand(n);
O= zeros(n);
H=[O,A;A,O];
[Q,D]= qreigm (H);
[ds,p]= sort(diag(D),1,'descend');
Dp=D(p,p);
Qp= sqrt(2)*Q(:,p);
U=Qp(n +1:2*n,1:n)
V=Qp (1:n,1:n)
S=Dp (1:n,1:n)
norm (A-U*S*V')
```

13.3 Mooreova-Penroseova inverze

Lze ukázat, že ke každé matici $\mathbf{A} \in \mathbb{R}^{m\times n}$ existuje tzv. *zobecněná inverze* $\mathbf{A}^{\#}$, pro níž platí rovnost

$$\mathbf{A}\mathbf{A}^{\#}\mathbf{A} = \mathbf{A}. \tag{13.4}$$

Pomocí SVD rozkladu (13.1) matice $\mathbf{A}$ sestavíme speciální případ zobecněné inverze, tzv. *Mooreovu-Penroseovu inverzi*

$$\mathbf{A}^{+} = \mathbf{V}\mathbf{S}^{+}\mathbf{U}^{T},$$

kde matice $\mathbf{S}^{+}$ je diagonální a její prvky jsou dány předpisem

$$\begin{aligned} s^{+}_{i,i} &= \tfrac{1}{s_{i,i}} \quad \text{pro } s_{i,j} > 0 \\ s^{+}_{i,i} &= 0 \quad\;\; \text{pro } s_{i,j} = 0. \end{aligned}$$

Skutečně lze ukázat, že po dosazení $\mathbf{A}^{+}$ za $\mathbf{A}^{\#}$ bude vztah (13.4) stále platit, což je mimo jiné první ze čtyř tzv. Mooreových-Penroseových podmínek. Tato a zbylé tři Mooreovy-Penroseovy podmínky jsou

$$\begin{array}{lrcl} 1. & \mathbf{A}\mathbf{A}^{+}\mathbf{A} & = & \mathbf{A} \\ 2. & \mathbf{A}^{+}\mathbf{A}\mathbf{A}^{+} & = & \mathbf{A}^{+} \\ 3. & \mathbf{A}^{+}\mathbf{A} & = & (\mathbf{A}^{+}\mathbf{A})^{T} \\ 4. & \mathbf{A}\mathbf{A}^{+} & = & (\mathbf{A}\mathbf{A}^{+})^{T}. \end{array}$$

13.4 Příklady

1. Pomocí singulárního rozkladu spočítejte inverzní matici matice

$$\mathbf{A} = \begin{pmatrix} 0 & 2 & 1 \\ 3 & 1 & 2 \\ 1 & 5 & 2 \end{pmatrix}.$$

Poznámka: Inverzní matici matice diagonání snadno spočteme jako převrácenou hodnotu prvků na diagonále.

2. Pomocí singulárního rozkladu spočítejte tzv. pseudoinverzi $\mathbf{A}^+$ (Mooreova-Penroseova inverze)) matice

$$\mathbf{A} = \begin{pmatrix} 0 & 2 & 1 \\ 3 & 1 & 2 \\ 3 & -1 & 1 \end{pmatrix} .$$

Ověřte, že platí vztah $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$.
Poznámka: Řešení je pouhou modifikací předchozího příkladu.

13.5 Funkce Matlabu

- pro plné matice
  - `[U,S,V] = svd(A)` ... vrací matice **SVD** rozkladu, kde **S** je stejné velikosti jako **A**
  - `S = svd(A)` ... vrací vektor singulárních čísel
  - `[U,S,V] = svd(A,0)` ... redukovaná verze rozkladu

- pro řídké matice
  - `S = svds(A)` ... vrací 6 největších singulárních čísel
  - `S = svds(A,k)` ... vrací k největších singulárních čísel

Další možnosti a více podrobností najdete v [4].

Kapitola 14

Lanczosova metoda

14.1 Motivace

Výpočet vlastních čísel je spojený s mnoha inženýrskými problémy. Lanczosova metoda je vhodná pro situace, kdy je předmětem zájmu výpočet dominantních vlastních čísel. Prakticky to znamená, že pro aproximaci se původní matice transformuje do třidiagonálního tvaru a navíc je možné redukovat její řád. Proto Lanczosova metoda není nástrojem k výpočtu vlastních čísel, poskytuje však podklady pro algoritmy (např. mocinná metoda), které by pro původní matici byly nepoužitelné (problémy s velkými řády). Podrobnější popis metody naleznete např. v [5].

14.2 Definice

Mějme symetrickou pozitivně definitní matici $\mathbf{A} \in \mathbb{R}^{n,n}$. Pomocí podobnostní transformace převedeme matici $\mathbf{A}$ na třidiagonální tvar

$$\mathbf{T} = \begin{pmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \\ 0 & \cdots & 0 & 0 & b_{n-1} & a_n \end{pmatrix} \tag{14.1}$$

podle vztahu

$$\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q}, \tag{14.2}$$

kde $\mathbf{Q} \in \mathbb{R}^{n,n}$ je matice ortonormální.

Při odvození uvažujme, že matice $\mathbf{Q}$ je známa. Zapišme hledanou matici $\mathbf{T}$ v obecném tvaru a přepišme rovnost (14.2) do tvaru

$$\mathbf{Q}\mathbf{T} = \mathbf{A}\mathbf{Q}.$$

Za předpokladu, že $\mathbf{Q} = (\mathbf{q}^1, \mathbf{q}^2, \cdots, \mathbf{q}^n)$, lze předešlou rovnost napsat

$$\left(\mathbf{q}^1, \mathbf{q}^2, \cdots, \mathbf{q}^n\right) \begin{pmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & b_{n-2} & a_{n-1} & b_{n-1} \\ 0 & \cdots & 0 & 0 & b_{n-1} & a_n \end{pmatrix} = \mathbf{A}\left(\mathbf{q}^1, \mathbf{q}^2, \cdots, \mathbf{q}^n\right)$$

a vyjádřit k-tou rovnici

$$b_{k-1}\mathbf{q}^{k-1} + a_k\mathbf{q}^k + b_k\mathbf{q}^{k+1} = \mathbf{A}\mathbf{q}^k, \tag{14.3}$$

pro jejíž obecnost se s ohledem na další úpravy definujme $b_0 = 0,\ \mathbf{q}^0 = 0$.

Výpočet koeficientů

Přenásobme rovnici (14.3) vektorem $\mathbf{q}^k$ a díky ortonormalitě vektorů obdržíme

$$a_k = \left(\mathbf{A}\mathbf{q}^k, \mathbf{q}^k\right).$$

Stejnou rovnici nyní přenásobme $\mathbf{q}^{k+1}$ a obdržíme

$$b_k = \left(\mathbf{A}\mathbf{q}^k, \mathbf{q}^{k+1}\right).$$

Upravme rovnici (14.3) do tvaru

$$\mathbf{q}^{k+1} = \frac{\left(\mathbf{A} - a_k\mathbf{I}\right)\mathbf{q}^k - b_{k-1}\mathbf{q}^{k-1}}{b_k}, \tag{14.4}$$

což lze považovat za předpis ortonormalizačního procesu. Označme

$$\mathbf{r}^k = \left(\mathbf{A} - a_k\mathbf{I}\right)\mathbf{q}^k - b_{k-1}\mathbf{q}^{k-1}$$

a koeficient

$$b_k = \|\mathbf{r}^k\|,$$

potom vektor následující z matice $\mathbf{Q}$ je dán předpisem

$$\mathbf{q}^{k+1} = \frac{\mathbf{r}^k}{\|\mathbf{r}^k\|}.$$

14.3 Algoritmus Lanczosovy metody

Všechny potřebné vztahy pro výpočet jsou odvozeny a výpočet matice $\mathbf{T}$ se sestaví v n krocích. V průběhu výpočtu se generují vektory $\mathbf{q}^i$ a můžeme ověřit rovnost (14.2).

Algoritmus 14.1 Lanczosova metoda.

$\mathbf{A} \in \mathbb{R}^{n,n}$ vstupní matice
$\mathbf{T} \in \mathbb{R}^{n,n}$ nulová matice, prvek matice $t_{i,j} = 0$
$\mathbf{q}^0 \in \mathbb{R}^n$ nulový vektor
$\mathbf{r}^1 \in \mathbb{R}^n$ libovolný jednotkový vektor
$b_1 = 1$

for $j = 1, ..., n$
$\mathbf{q}^j = \mathbf{r}^j / b_j$
$a_j = (\mathbf{A}\mathbf{q}^j, \mathbf{q}^j)$
$\mathbf{r}^{j+1} = (\mathbf{A} - a_j\mathbf{I})\,\mathbf{q}^j - b_j\mathbf{q}^{j-1}$
$b_{j+1} = \|\mathbf{r}^{j+1}\|$
$t_{j,j} = a_j$
if $j < n$ **then**
$t_{j+1,j} = b_j$
$t_{j,j+1} = b_j$
end if
end for

Je-li Lanzsosova metoda použita pro výpočet vlastních čísel λ, často se algoritmus ukončuje dříve, než jsou spočteny kompletní matice. Z vlastností matice $\mathbf{T}$ se může výpočet λ s ohledem na žádanou přesnost kontrolovat dvěma parametry. Stanoví se počet hledaných vlastních čísel n_λ a počet potřebných iterací k v relaci $n_\lambda < k$. Také platí omezení $k < n$. Prakticky to znamená, že odhad dominantních vlastních čísel je počítán z matice, která může mít výrazně menší řád, než původní.

Algoritmus 14.2 Použití Lanczosovy metody k výpočtu vlastních čísel.

$\mathbf{A} \in \mathbb{R}^{n,n}$ vstupní matice
$\tilde{\mathbf{T}} \in \mathbb{R}^{k,k}$ nulová matice, prvek matice $t_{i,j} = 0$
$\mathbf{q}^0 \in \mathbb{R}^n$ nulový vektor
$\mathbf{r}^1 \in \mathbb{R}^n$ libovolný jednotkový vektor
$b_1 = 1$

for $j = 1, ..., k$

$\mathbf{q}^j = \mathbf{r}^j / b_j$

$a_j = (\mathbf{A}\mathbf{q}^j, \mathbf{q}^j)$

$\mathbf{r}^{j+1} = (\mathbf{A} - a_j \mathbf{I})\, \mathbf{q}^j - b_j \mathbf{q}^{j-1}$

$b_{j+1} = \|\mathbf{r}^{j+1}\|$

$t_{j,j} = a_j$

if $j < k$ **then**

$\tilde{t}_{j+1,j} = b_j$

$\tilde{t}_{j,j+1} = b_j$

end if

end for

Výpočet n_λ vlastních čísel tridiagonální matice $\tilde{\mathbf{T}}$

.

Kód matlabu

Algoritmus 14.1: Lanzosova metoda

```
function [lambda,T,Q]=lanczos(A,n_lambda,n_it)
% A          ... vstupni matice
% n_it       ... max. pocet iteraci
% n_lambda   ... pocet pocitanych vlastnich cisel
n=size(A,1);
if n_it>=n || n_lambda >= n_it || nargin~=3
  error('Chyba v zadani!')
end
Q=zeros(n,n_it);
s=zeros(n,1);
r=rand(n,1);r=r/norm(r);b=1;
Alpha=zeros(n_it,1);Beta =zeros(n_it-1,1);

for j=1:n_it
  s_bef    = s;
  s        = r/b;
  Q(:,j)   = s;
```

```
  a         = dot(s,A*s);
  Alpha(j)  = a;
  r         = (A-a*eye(n))*s-b*s_bef;
  b         = norm(r);
  if j<n_it,Beta(j)=b;end
end

T=spdiags([[Beta;0],Alpha,[0;Beta]],[-1 0 1],n_it,n_it);
lambda = eigs(T,n_lambda);
```

Příklad 14.1. Spočtěte přibližně největší vlastní číslo $\tilde{\lambda}_{max}$ třídiagonální matice $\mathbf{A} \in \mathbb{R}^{n\times n}$, $n = 50$, pro kterou platí

$$\begin{aligned} a_{i,j} &= 2 \quad \text{pro} \quad i = j \\ a_{i,j} &= 1 \quad \text{pro} \quad |i-j| = 1 \\ a_{i,j} &= 0 \quad \text{pro} \quad |i-j| > 1 \end{aligned}$$

redukcí na matici řádu 5, 10 a 15 použitím Lanczosovy metody.

V Matlabu sestrojíme matici příkazem `A=spdiags([-I, 2*I,-I],[-1 0 1],n,n)`, kde je $n = 50$ a I je vektor jedniček řádu n. Vypsáním příkazu `lambda=lanczos(A,1,5)` se nejprve sestaví redukovaná třídiagonální matice $\mathbf{A}_r$ rozměru 5×5, spočtou se vlastní čísla, z nichž největší je $\tilde{\lambda}_{max} = 3.893$ (Matlabem spočtená hodnota největšího vlastního čísla matice $\mathbf{A}$ je $\lambda_{max} = 3.996$). Relativní chyba je 2.595%. Tato a další dvě varianty jsou vypsány v tabulce pod textem.

varianta	1	2	3
řád matice $\mathbf{A}_r$	5	10	15
$\tilde{\lambda}_{max}(\mathbf{A}_r)$	3.893	3.979	3.995
$\lambda_{max}(\mathbf{A})$		3.996	
ε [%]	2.595	0.437	0.040

Kapitola 15

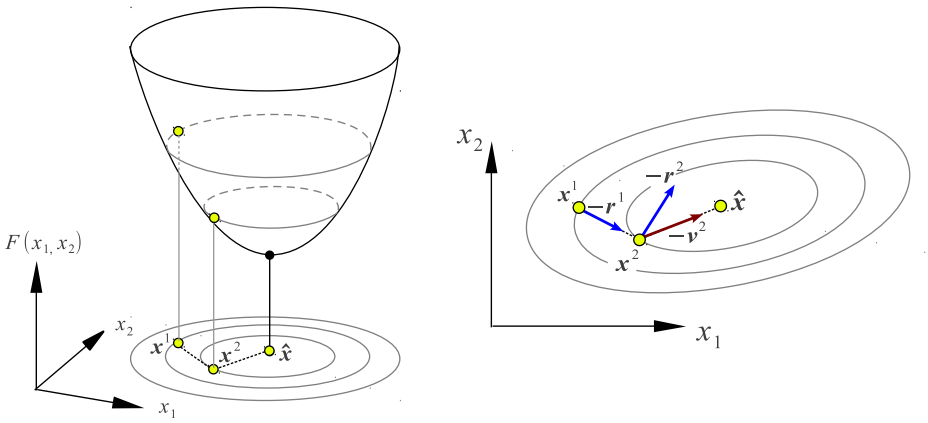
Metoda sdružených gradientů

15.1 Motivace

Inženýrské problémy řešené metodou konečných prvků či metodou sítí převádí původně formulovaný fyzikální problém na soustavu lineárních rovnic tvaru

$$\mathbf{A}\hat{\mathbf{x}} = \mathbf{b}, \tag{15.1}$$

kde $\hat{\mathbf{x}} \in \mathbb{R}^n$ je hledané řešení a $\mathbf{A} \in \mathbb{R}^{n,n}$, $\mathbf{b} \in \mathbb{R}^n$. Rovnici lze řešit např. dříve popsanou Gaussovou eliminací, pro kterou platí pracnost $\mathcal{O}(n^3)$. Metoda sdružených gradientů má v nejnepříznivějším případě (plná a špatně podmíněná matice, ...) také pracnost $\mathcal{O}(n^3)$, ale pro matice generované metodou konečných prvků (resp. sítí), které jsou řídké, je pracnost výrazně menší. Hlubší teoretický rozbor naleznete v [1, 5].



Obrázek 15.1: Funkcionál

15.2 Definice

Řešení rovnice (15.1) je ekvivalentní úloze nalezení minima funkcionálu

$$F(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, \mathbf{A}\mathbf{x}) - (\mathbf{x}, \mathbf{b}). \tag{15.2}$$

Pro dvě proměnné si funkcionál můžeme přestavit jako graf paraboloidu nad rovinou x_1x_2, který přiřazuje každému vektoru $\mathbf{x} = (x_1, x_2)$ jedinečné číslo odpovídající souřadnici kolmé na x_1x_2. Existuje $\mathbf{x} := \hat{\mathbf{x}}$, pro které je hodnota fukcionálu minimální a které je řešením rovnice (15.1). Pro případ na obr. 15.1 obdržíme řešení po dvou iteracích. Prvně se spočte minimum v záporném směru gradientu $\mathbf{r}^1 = \mathbf{A}\mathbf{x}^1 - \mathbf{b}$, druhým vektorem $\mathbf{v}^2$ již dojdeme k výsledku. Předpokládejme, že přesné řešení $\hat{\mathbf{x}}$ můžeme napsat ve tvaru

$$\hat{\mathbf{x}} = \mathbf{x}^1 + \sum_{i=1}^{n} \alpha_i \mathbf{v}^i, \tag{15.3}$$

nebo také

$$\hat{\mathbf{x}} = \mathbf{x}^1 + \mathbf{V}\boldsymbol{\alpha}, \tag{15.4}$$

kde $\mathbf{V} = (\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^n)$ je $\mathbf{A}$-ortogonální báze , pro kterou platí

$$(\mathbf{v}^i, \mathbf{A}\mathbf{v}^j) = 0 \text{ pro } i \neq j \ , \tag{15.5}$$

$$(\mathbf{v}^i, \mathbf{A}\mathbf{v}^j) \neq 0 \text{ pro } i = j \tag{15.6}$$

a $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_n)^T$ je vektor koeficientů. Dosadíme-li (15.3) do energetického funkcionálu (15.2), obdržíme

$$F(\hat{\mathbf{x}}) = \frac{1}{2}\left(\mathbf{x}^1 + \mathbf{V}\boldsymbol{\alpha}, \mathbf{A}\left(\mathbf{x}^1 + \mathbf{V}\boldsymbol{\alpha}\right)\right) - \left(\mathbf{x}^1 + \mathbf{V}\boldsymbol{\alpha}, \mathbf{b}\right), \tag{15.7}$$

a po úpravě

$$F(\hat{\mathbf{x}}) = F(\boldsymbol{\alpha}) = \frac{1}{2}\left(\boldsymbol{\alpha}, \mathbf{V}^T\mathbf{A}\mathbf{V}\boldsymbol{\alpha}\right) - \left(\boldsymbol{\alpha}, \mathbf{V}^T\left(\mathbf{b} - \mathbf{A}\mathbf{x}^1\right)\right) + C, \tag{15.8}$$

kde

$$C = \frac{1}{2}\left(\mathbf{x}^1, \mathbf{A}\mathbf{x}^1\right) - \left(\mathbf{x}^1, \mathbf{b}\right)$$

je konstanta bez vlivu na minimum funkcionálu. Nyní s argumentem $\boldsymbol{\alpha}$ lze díky speciálním vlastnostem báze (součin $\mathbf{V}^T\mathbf{A}\mathbf{V}$ vede na diagonální matici) zapsat ve tvaru

$$\begin{aligned} F(\boldsymbol{\alpha}) = \tfrac{1}{2}\alpha_1^2\left(\mathbf{v}^1, \mathbf{A}\mathbf{v}^1\right) + \tfrac{1}{2}\alpha_2^2\left(\mathbf{v}^2, \mathbf{A}\mathbf{v}^2\right) + ... + \tfrac{1}{2}\alpha_n^2\left(\mathbf{v}^n, \mathbf{A}\mathbf{v}^n\right) - \\ -\alpha_1\left(\mathbf{v}^1, -\mathbf{r}^1\right) - \alpha_2\left(\mathbf{v}^2, -\mathbf{r}^1\right) - ... - \alpha_n\left(\mathbf{v}^n, -\mathbf{r}^1\right) + C. \end{aligned}$$

Nyní stačí funkcionál derivovat postupně podle všech α_k

$$\frac{\partial F(\boldsymbol{\alpha})}{\partial \alpha_k} = 0 = \alpha_k\left(\mathbf{v}^k, \mathbf{A}\mathbf{v}^k\right) - \left(\mathbf{v}^k, -\mathbf{r}^1\right)$$

a hledané koeficienty spočítat z

$$\alpha_k = -\frac{\left(\mathbf{v}^k, \mathbf{r}^1\right)}{\left(\mathbf{v}^k, \mathbf{A}\mathbf{v}^k\right)}. \tag{15.9}$$

Pokud by byla báze $\mathbf{V}$ k dispozici, úloha je vyřešena, neboť koeficienty α_k se dopočítají velice jednoduše. V následující části budou podrobně rozebrány základní skalární součiny potřebné k odvození dílčích částí metody sdružených gradientů.

15.3 Analýza

Zaveďme předpoklad, že se k řešení blížíme (obr. 15.1) pomocí vztahu

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{v}^k, \tag{15.10}$$

každé následující přiblížení hledaného řešení lze vyjádřit kombinací předešlého $\mathbf{x}^k$ s tzv. sdruženým směrem $\mathbf{v}^k$. Zobecněním rovnice (15.10) můžeme psát dříve zavedenou (15.3). Je-li mez pro sčítání $k < n$, pro $\mathbf{x}^k$ můžeme také psát

$$\mathbf{x}^k = \mathbf{x}^1 + \sum_{i=1}^{k-1} \alpha_i \mathbf{v}^i. \tag{15.11}$$

V k-tém kroce je reziduum dáno vztahem

$$\mathbf{r}^k = \mathbf{A}\mathbf{x}^k - \mathbf{b} \tag{15.12}$$

a v kroce následujícím pomocí

$$\mathbf{r}^{k+1} = \mathbf{A}\mathbf{x}^{k+1} - \mathbf{b} = \mathbf{r}^k + \alpha_k \mathbf{A}\mathbf{v}^k. \tag{15.13}$$

Kombinace (15.11) a (15.12) dává

$$\mathbf{r}^k = \mathbf{r}^1 + \sum_{i=1}^{k-1} \alpha_i \mathbf{A}\mathbf{v}^i \tag{15.14}$$

Předpokládejme, že k-tý vektor $\mathbf{A}$-ortonormální báze je spočten Gramm-Schmidtovým ortogonalizačním procesem z báze reziduí dle

$$\mathbf{v}^k = \mathbf{r}^k + \sum_{i=1}^{k-1} \beta_{k,j} \mathbf{v}^j. \tag{15.15}$$

15.4 Definování skalárních součinů

V další části textu budou odvozeny všechny potřebné skalární součiny, které se v popisu dílčích částí metody objevují.

Součin $\left(\mathbf{r}^k, \mathbf{v}^j\right)$

Použijme obecný vektor $\mathbf{v}^j$ a relaci (15.14).

- Pro $j \geqq k$

$$\left(\mathbf{r}^k, \mathbf{v}^j\right) = \left(\mathbf{r}^1, \mathbf{v}^j\right) + \sum_{i=1}^{k-1} \alpha_i \left(\mathbf{A}\mathbf{v}^i, \mathbf{v}^j\right) = \left(\mathbf{r}^1, \mathbf{v}^j\right), \tag{15.16}$$

  nebo také

$$\left(\mathbf{r}^j, \mathbf{v}^j\right) = \left(\mathbf{r}^{j-1}, \mathbf{v}^j\right) = \left(\mathbf{r}^{j-2}, \mathbf{v}^j\right) = \left(\mathbf{r}^1, \mathbf{v}^j\right). \tag{15.17}$$

- Pro $j < k$ s využitím (15.9) a předešlého vztahu (15.16)

$$\begin{aligned} \left(\mathbf{r}^k, \mathbf{v}^j\right) &= \left(\mathbf{r}^1, \mathbf{v}^j\right) + \alpha_j \left(\mathbf{A}\mathbf{v}^j, \mathbf{v}^j\right) \\ &= \left(\mathbf{r}^1, \mathbf{v}^j\right) - \frac{\left(\mathbf{r}^j, \mathbf{v}^j\right)}{\left(\mathbf{v}^j, \mathbf{A}\mathbf{v}^j\right)} \left(\mathbf{A}\mathbf{v}^j, \mathbf{v}^j\right) = 0. \end{aligned} \tag{15.18}$$

  Využitím obou závislostí odvodíme další užitečnou rovnost

$$\left(\mathbf{r}^k, \mathbf{v}^k\right) = \left(\mathbf{r}^k, \mathbf{r}^k\right) + \sum_{i=1}^{k-1} \beta_{k,i} \left(\mathbf{v}^i, \mathbf{r}^k\right) = \left(\mathbf{r}^k, \mathbf{r}^k\right) = \|\mathbf{r}^k\|^2. \tag{15.19}$$

Součin $\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^j\right)$

Upravme rovnici (15.15) do tvaru

$$\mathbf{r}^k = \mathbf{v}^k - \sum_{i=1}^{k-1} \beta_{k,i} \mathbf{v}^i.$$

- Pro $j > k$

$$\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^j\right) = \left(\mathbf{v}^k - \sum_{i=1}^{k-1} \beta_{k,i} \mathbf{v}^i, \mathbf{A}\mathbf{v}^j\right) = 0. \tag{15.20}$$

- Pro $j = k$

$$\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^k\right) = \left(\mathbf{v}^k, \mathbf{A}\mathbf{v}^k\right). \tag{15.21}$$

- Pro $j < k$

$$\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^j\right) = -\beta_{k,j} \left(\mathbf{v}^j, \mathbf{A}\mathbf{v}^j\right), \tag{15.22}$$

  z čehož odvodíme

$$\beta_{k,j} = -\frac{\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^j\right)}{\left(\mathbf{v}^j, \mathbf{A}\mathbf{v}^j\right)}. \tag{15.23}$$

Součin $\left(\mathbf{r}^k, \mathbf{r}^j\right)$

- Pro $j = 1$ a $k = \{2, 3, ..., n\}$ platí

$$\begin{aligned}\left(\mathbf{r}^k, \mathbf{r}^1\right) &= \left(\mathbf{r}^1 + \sum_{i=1}^{k-1} \alpha_i \mathbf{A}\mathbf{v}^i, \mathbf{r}^1\right) \\ &= \left(\mathbf{r}^1, \mathbf{r}^1\right) + \sum_{i=1}^{k-1} \alpha_i \mathbf{A}\left(\mathbf{v}^i, \mathbf{r}^1\right) \\ &= \left(\mathbf{r}^1, \mathbf{r}^1\right) + \alpha_1 \left(\mathbf{A}\mathbf{v}^1, \mathbf{r}^1\right).\end{aligned} \tag{15.24}$$

  V poslední úpravě je suma součinů pro $i = \{2, ..., k-1\}$ dle (15.20) nulová, zůstává pouze člen pro $i = 1$. S platností $\mathbf{v}^1 = \mathbf{r}^1$ a dosazením za α_1 obdržíme

$$\left(\mathbf{r}^k, \mathbf{r}^1\right) = \left(\mathbf{r}^1, \mathbf{r}^1\right) - \frac{\left(\mathbf{r}^1, \mathbf{r}^1\right)}{\left(\mathbf{r}^1, \mathbf{A}\mathbf{r}^1\right)} \left(\mathbf{A}\mathbf{r}^1, \mathbf{r}^1\right) = 0$$

  a platí, že počáteční reziduum je kolmé ke všem následujícím.

- Pro $j = k - 1$

$$\begin{aligned}\left(\mathbf{r}^k, \mathbf{r}^{k-1}\right) &= \left(\mathbf{r}^{k-1} + \alpha_{k-1}\mathbf{A}\mathbf{v}^{k-1}, \mathbf{r}^{k-1}\right) \\ &= \left(\mathbf{r}^{k-1}, \mathbf{r}^{k-1}\right) + \alpha_{k-1}\left(\mathbf{A}\mathbf{v}^{k-1}, \mathbf{r}^{k-1}\right). \\ &= \left(\mathbf{r}^{k-1}, \mathbf{r}^{k-1}\right) - \frac{\left(\mathbf{v}^{k-1}, \mathbf{r}^{k-1}\right)}{\left(\mathbf{v}^{k-1}, \mathbf{A}\mathbf{v}^{k-1}\right)} \left(\mathbf{A}\mathbf{v}^{k-1}, \mathbf{v}^{k-1}\right).\end{aligned} \tag{15.25}$$

  Opět je využito dříve odvozených relací. S platností $\left(\mathbf{v}^{k-1}, \mathbf{r}^{k-1}\right) = \|\mathbf{r}^{k-1}\|^2$ jsou po dalších úpravách dvě po sobě jdoucí rezidua vzájemně ortogonální.

- Obecně pro $k > j$

$$\begin{aligned}\left(\mathbf{r}^k, \mathbf{r}^j\right) &= \left(\mathbf{r}^{k-1} + \alpha_{k-1}\mathbf{A}\mathbf{v}^{k-1}, \mathbf{r}^j\right) \\ &= \left(\mathbf{r}^{k-1}, \mathbf{r}^j\right) + \alpha_{k-1}\left(\mathbf{A}\mathbf{v}^{k-1}, \mathbf{r}^j\right).\end{aligned} \tag{15.26}$$

  Zde druhý člen je pro $j < k - 1$ nulový v důsledku (15.20) a zůstává

$$\left(\mathbf{r}^k, \mathbf{r}^j\right) = \left(\mathbf{r}^{k-1}, \mathbf{r}^j\right).$$

  Rovnost je možné zobecnit postupně

$$\left(\mathbf{r}^k, \mathbf{r}^j\right) = \left(\mathbf{r}^{k-1}, \mathbf{r}^j\right) = \left(\mathbf{r}^{k-2}, \mathbf{r}^j\right) = \cdots = \left(\mathbf{r}^{j+1}, \mathbf{r}^j\right)$$

  a protože jako poslední člen v řadě rovností jsou dvě po sobě jdoucí rezidua, jejichž skalární součin je nulový, součin reziduí s indexy $k > j$ je taktéž roven nule.

15.5 Algoritmus sdružených gradientů

Pro i-tý sdružený vektor patřící do postupně vytvářené $\mathbf{A}$-ortogonální báze platí rovnice (15.15). Koeficienty $\beta_{k,j}$ se spočítají ze vztahu (15.23), kde v čitateli za člen $\mathbf{A}\mathbf{v}^j$ dosadíme substituci odvozenou z (15.13)

$$\beta_{k,j} = -\frac{\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^j\right)}{\left(\mathbf{v}^k, \mathbf{A}\mathbf{v}^k\right)} = -\frac{\left(\mathbf{r}^k, \frac{1}{\alpha_j}\left(\mathbf{r}^{j+1} - \mathbf{r}^j\right)\right)}{\left(\mathbf{v}^k, \mathbf{A}\mathbf{v}^k\right)}.$$

Indexy j nabývají postupně hodnot $\{k-1, k-2, \ldots, 1\}$. S přihlédnutím k vlastnostem ortogonality rezidují vyjma $j = k-1$ budou všechny ostatní koeficienty rovny nule a předpis pro sdružený směr se zjednoduší na

$$\mathbf{v}^k = \mathbf{r}^k + \beta_{k-1}\mathbf{v}^{k-1}. \tag{15.27}$$

Všechny potřebné vztahy pro výpočet jsou odvozeny. Připomeňme ještě, že se spokojíme s přibližným řešením. Pro tyto účely je zavedena konstanta ε a výpočet ukončíme např. při $\left\|\mathbf{x}^{k+1} - \mathbf{x}^k\right\| < \varepsilon$.

Algoritmus 15.1 Algoritmus sdružených gradientů.

$\varepsilon > 0,\ \mathbf{x}^1,\ k = 1$
$\mathbf{r}^1 = \mathbf{A}\mathbf{x}^1 - \mathbf{b},\ \mathbf{v}^1 = \mathbf{r}^1,\ \alpha_1 = -\frac{\|\mathbf{r}^1\|^2}{(\mathbf{A}\mathbf{r}^1,\ \mathbf{r}^1)},\ \mathbf{x}^2 = \mathbf{x}^1 + \alpha_1\mathbf{v}^1$

while $||\mathbf{x}^{k+1} - \mathbf{x}^k|| \geqq \varepsilon$
$k = k + 1$
$\mathbf{r}^k = \mathbf{r}^{k-1} + \alpha_{k-1}\mathbf{A}\mathbf{v}^{k-1}$
$\beta_{k-1} = -\frac{\left(\mathbf{r}^k, \mathbf{A}\mathbf{v}^{k-1}\right)}{\left(\mathbf{A}\mathbf{v}^{k-1}, \mathbf{v}^{k-1}\right)}$
$\mathbf{v}^k = \mathbf{r}^k + \beta_{k-1}\mathbf{v}^{k-1}$
$\alpha_k = -\frac{\left(\mathbf{r}^k, \mathbf{v}^k\right)}{\left(\mathbf{A}\mathbf{v}^k, \mathbf{v}^k\right)}$
$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\mathbf{v}^k$
end while

Části kódu lze dále upravit.

Koeficient α

Platí-li ortogonalita, můžeme psát součin dvou po sobě následujících rezidují

$$\left(\mathbf{r}^{k+1}, \mathbf{r}^k\right) = \left(\mathbf{r}^k + \alpha_k\mathbf{A}\mathbf{v}^k, \mathbf{r}^k\right) = \left(\mathbf{r}^k, \mathbf{r}^k\right) + \alpha_k\left(\mathbf{A}\mathbf{v}^k, \mathbf{r}^k\right) = 0, \tag{15.28}$$

který je roven nule a nově pro α_k platí

$$\alpha_k = -\frac{\left(\mathbf{r}^k, \mathbf{r}^k\right)}{\left(\mathbf{A}\mathbf{v}^k, \mathbf{v}^k\right)}. \tag{15.29}$$

Koeficient β

Využijme ortogonality reziduí pro úpravu koeficientu β ze součinu

$$\left(\mathbf{v}^k, \mathbf{r}^{k-1}\right) = \left(\mathbf{r}^k + \beta_{k-1}\mathbf{v}^{k-1}, \mathbf{r}^{k-1}\right) = \left(\mathbf{r}^k, \mathbf{r}^{k-1}\right) + \beta_{k-1}\left(\mathbf{v}^{k-1}, \mathbf{r}^{k-1}\right). \tag{15.30}$$

Součin $\left(\mathbf{v}^k, \mathbf{r}^{k-1}\right)$ je roven $\left(\mathbf{v}^k, \mathbf{r}^j\right)$, kde $j = \{1, 2, ..., k\}$. Využitím další rovnosti (15.19) se vztah přepíše

$$\left(\mathbf{r}^k, \mathbf{r}^k\right) = \beta_{k-1}\left(\mathbf{r}^{k-1}, \mathbf{r}^{k-1}\right)$$

a konečný výraz pro koeficient se zjednoduší na

$$\beta_{k-1} = \frac{\left(\mathbf{r}^k, \mathbf{r}^k\right)}{\left(\mathbf{r}^{k-1}, \mathbf{r}^{k-1}\right)}. \tag{15.31}$$

V algoritmu je zavedena pomocná proměnná $\mathbf{w}^i = \mathbf{A}\mathbf{v}^i$ a za každou iteraci se v optimalizované verzi kódu provede pouze jedno násobení maticí $\mathbf{A}$. Jako ukončovací kritérium vězměme normu rezidua.

Algoritmus 15.2 Optimalizovaný algoritmus sdružených gradientů.

$\varepsilon > 0,\ \mathbf{x}^1,\ k = 1$
$\mathbf{r}^1 = \mathbf{A}\mathbf{x}^1 - \mathbf{b},\ \mathbf{v}^1 = \mathbf{r}^1,\ \mathbf{w}^1 = \mathbf{A}\mathbf{v}^1,\ \alpha_1 = -\frac{\left\|\mathbf{r}^1\right\|^2}{\left(\mathbf{w}^1, \mathbf{v}^1\right)},\ \mathbf{x}^2 = \mathbf{x}^1 + \alpha_1\mathbf{v}^1$
while $||\mathbf{r}^k|| \geqq \varepsilon$
    $k = k + 1$
    $\mathbf{r}^k = \mathbf{r}^{k-1} + \alpha_{k-1}\mathbf{w}^{k-1}$
    $\beta_{k-1} = \left\|\mathbf{r}^k\right\|^2 / \left\|\mathbf{r}^{k-1}\right\|^2$
    $\mathbf{v}^k = \mathbf{r}^k + \beta_{k-1}\mathbf{v}^{k-1}$
    $\mathbf{w}^k = \mathbf{A}\mathbf{v}^k$
    $\alpha_k = -\left\|\mathbf{r}^k\right\|^2 / \left(\mathbf{w}^k, \mathbf{v}^k\right)$
    $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\mathbf{v}^k$
end while

Algoritmus 15.1: Metoda sdružených gradientů

```
function x=cg(A,b,x)
% metoda sdruzenych gradientu
% A ... matice soustavy
% b ... vektor prave strany
% x ... pocatecni odhad
n=length(b);
if nargin<3,x=1*ones(n,1);end
eps0=1e-4*norm(b);
k=1;
r=A*x-b;norm_r=norm(r);
v=r;
w=A*v;
alph=-norm_r^2/dot(w,v);
x=x+alph*v;
while norm_r>eps0  k<=n
  r=r+alph*w;
  norm_r_next=norm(r);
  if norm_r_next<eps0;break,end
  beta=(norm_r_next/norm_r)^2;
  v=r+beta*v;  w=A*v;
  alph=-norm_r_next^2/dot(w,v);
  x=x+alph*v;
  norm_r=norm_r_next;
  k=k+1;
end
fprintf('||r||=%3.3e, iter= %i\n',norm_r_next,k);
```

Část III

Numerické řešení diferenciálních rovnic

Kapitola 16

Diferenciální rovnice - motivační příklady

16.1 Struna

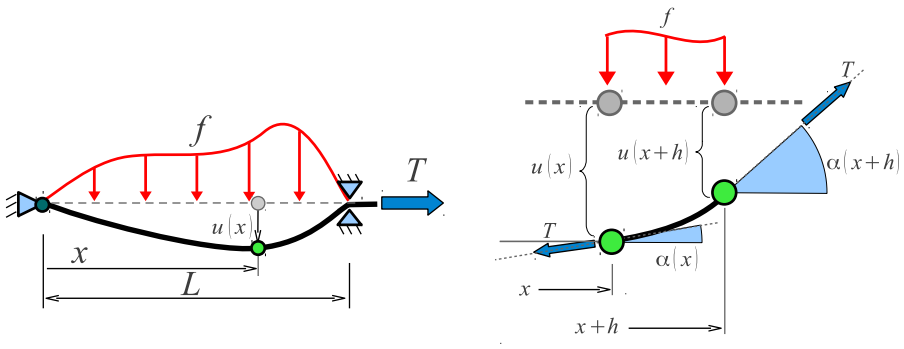
Struna je vlákno napnuté mezi dvěma body sloužící jako zdroj zvuku u strunných nástrojů jako je kytara, housle (viz obr. 16.1), loutna, buzuki, basa, ale také například klavír.

Obrázek 16.1: Strunné nástroje.

Předpokládejme, že struna je objekt, u kterého lze vyjádřit charakteristiky (průřezové, materiálové, geometrické apod.) podélně pomocí jedné nezávisle proměnné. Na obr. 16.2 vlevo je struna na levém konci uchycena, na pravém tažena tzv. předepínací silou T (= konst.) a po celé své délce je vertikálně zatížena hustotou sil f. Ačkoliv je ohybová tuhost struny téměř nulová, proti příčným výchylkám u působí předepínací síla

$$T = EA\frac{\Delta L}{L},$$

kde E tzv. je Youngův modul pružnosti ($E_{ocel} = 2.1 \cdot 10^5$ MPa) charakterizující



Obrázek 16.2: Struna

materiálové vlastnosti, A je plocha průřezu, L je původní délky struny a ΔL je změna délky v důsledku síly T. Funkce u tedy popisuje průhyb struny. Jsou-li dodrženy předpoklady malých deformací, lze považovat změny velikosti předepínací síly T za zanedbatelné. Například pro kytarovou strunu E_1 délky 628 mm je při příčném vychýlení bodu o 5 mm v polovině délky změna původní předepínací síly menší než 0.7 %.

Diferenciální rovnici odvodíme z rovnice rovnováhy na části struny, kterou dostaneme dvěma řezy ve vzdálenosti x a $x+h$ (obr. 16.2 vpravo), účinky odejmutých částí jsou nahrazeny silami T. Předpokládá se, že předepínací síla je neměnná po celé délce struny. V y-ovém směru pak platí rovnice rovnováhy

$$\sum F_{iy} = 0 = -T\,\sin\left(\alpha(x)\right) + T\,\sin\left(\alpha(x+h)\right) + \int_x^{x+h} f(\xi)d\xi. \tag{16.1}$$

Pro spojitou primitivní funkci $f(\xi)$ na intervalu $\xi \in (x,\ x+h)$ z *věty o střední hodnotě integrálního počtu* platí rovnost

$$\lim_{h\to 0}\frac{1}{h}\int_x^{x+h} f(\xi)d\xi = f(x).$$

Dále využijeme přibližné vztahy pro malé úhly. Je-li α dostatečně malé, pak $\sin(\alpha) \approx \alpha$ a také $\operatorname{tg}(\alpha) \approx \alpha$, resp. $\sin(\alpha) \approx \operatorname{tg}(\alpha)$. Jelikož $u'(x) = \frac{du(x)}{dx} = \operatorname{tg}(\alpha(x))$, kde $\operatorname{tg}(\alpha(x))$ je směrnice tečny funkce v bodě x, rovnice (16.1) přejde po úpravě do tvaru

$$-\left(Tu'(x+h) - Tu'(x)\right) = \int_x^{x+h} f(\xi)d\xi.$$

Rovnici rovnováhy v bodě x získáme limitním přechodem pro $h \to 0$

$$\lim_{h\to 0} -T\frac{u'(x+h)-u'(x)}{h} = \lim_{h\to 0}\frac{1}{h}\int_x^{x+h} f(\xi)d\xi,$$

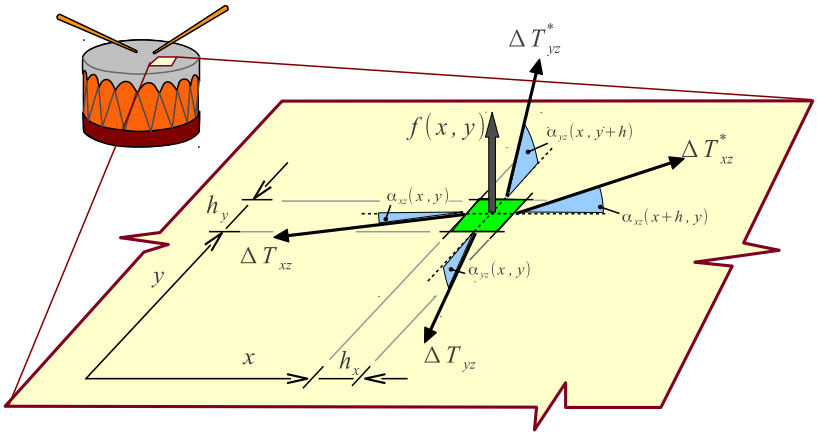
který vede na

$$-Tu'' = f. \tag{16.2}$$

Odvození diferenciální rovnice druhého řádu popisuje vztah hustoty sil f a předepínací síly T k průhybu struny $u(x)$. Chování v krajních bodech struny popisují tzv. okrajové podmínky. Je-li na obou koncích struna uchycena ($u(0) = u_0, u(L) = u_n$), jedná se o tzv. Dirichletovu podmínku. Je-li na některém konci zadaná síla, jedná se o tzv. Neumannovu podmínku (např. $-Tu'(L) = g$).

16.2 Membrána

Popišme úlohu průhybu membrány, u které rovnice rovnováhy v libovolném bodě tělesa vede na tzv. Poissonovu rovnici. Přes korpus bubnu je natažena membrána (umělá nebo kožená blána), která je výrazně předepnuta. Zaveďme souřadnicový systém, jehož osy x a y leží v rovině membrány. V obecném místě je vyjmut obdélníkový element o stranách h_x a h_y (obr. 16.3), na jehož hranách je účinek odejmutých částí nahrazen silami (ΔT).



Obrázek 16.3: Membrána.

Rovnice rovnováhy pro z-ový směr je

$$\begin{gathered}\sum F_{iz} = 0\\ \Delta T^*_{yz}\sin\left(\alpha_{yz}(x, y+h)\right) - \Delta T_{yz}\sin\left(\alpha_{yz}(x,y)\right) +\\ +\Delta T^*_{xz}\sin\left(\alpha_{xz}(x+h, y)\right) - \Delta T_{xz}\sin\left(\alpha_{xz}(x,y)\right) +\\ + \int_x^{x+h_x}\int_y^{y+h_y} f(\xi,\zeta)d\xi d\zeta.\end{gathered} \tag{16.3}$$

Z věty o střední hodnotě dostáváme

$$\lim_{h_x\to 0}\lim_{h_y\to 0}\frac{1}{h_x h_y}\int_x^{x+h_x}\int_y^{y+h_y} f(\xi,\zeta)d\xi d\zeta = f(\xi,\zeta) \tag{16.4}$$

Bez vnějšího zatížení je v membráně přítomno napětí

$$\sigma(x,y) = (\sigma_x(x,y),\ \sigma_y(x,y))^T = (\gamma,\ \gamma)^T,$$

které souvisí s počátečním předpětím a předpokládá se, že je konstantní. Hustotou sil f je membrána deformována ve vertikálním směru, přičemž vzniklé posuvy ve směru osy z neovlivní pole napětí v rovině xy (teorie malých deformací). Síla ΔT_{xz} a ΔT^*_{xz} je rovna integrálu

$$\Delta T_{xz} = \Delta T^*_{xz} = \gamma \int_y^{y+h_y} d\zeta = h_y \gamma. \tag{16.5}$$

Obdobně pro směr kolmý

$$\Delta T_{yz} = \Delta T^*_{yz} = \gamma \int_x^{x+h_x} d\xi = h_x \gamma. \tag{16.6}$$

Pro malé úhly je $\sin(\alpha_{xz}(x,y)) \approx \operatorname{tg}(\alpha_{xz}(x,y)) = \frac{\partial u(x,y)}{\partial x}$. S úpravou a substitucemi píšeme rovnici rovnováhy ve tvaru

$$\begin{aligned} &-\frac{\gamma}{h_x h_y}\left[h_y\left(\frac{\partial u(x+h_x,y)}{\partial x} - \frac{\partial u(x,y)}{\partial x}\right) + h_x\left(\frac{\partial u(x,y+h_y)}{\partial y} - \frac{\partial u(x,y)}{\partial y}\right)\right] \\ &\quad = \frac{1}{h_x h_y}\iint f(\xi,\zeta)d\xi d\zeta \end{aligned} \tag{16.7}$$

a limitním přechodem

$$\begin{aligned} &-\gamma\left(\lim_{h_x\to 0}\frac{1}{h_x}\left(\frac{\partial u(x+h_x,y)}{\partial x} - \frac{\partial u(x,y)}{\partial x}\right) + \lim_{h_y\to 0}\frac{1}{h_y}\left(\frac{\partial u(x,y+h_y)}{\partial y} - \frac{\partial u(x,y)}{\partial y}\right)\right) \\ &\quad = \lim_{h_x\to 0}\lim_{h_y\to 0}\frac{1}{h_x h_y}\iint f(\xi,\zeta)d\xi d\zeta \end{aligned} \tag{16.8}$$

dostáváme rovnici rovnováhy v bodě $[x,y]$ ve známém tvaru Poissonovy rovnice ve 2D

$$-\frac{\partial^2 u(x,y)}{\partial x^2} - \frac{\partial^2 u(x,y)}{\partial y^2} = \frac{1}{\gamma} f(x,y). \tag{16.9}$$

Chování na okraji membrány stejně jako pro strunu popisují tzv. okrajové podmínky. Je-li membrána na okrajích uchycena ($u = u_0,$ pro $x, y \in \Gamma$), jedná se o tzv. Dirichletovu podmínku . Je-li na části hranice zadaná síla, jedná se o tzv. Neumannovu podmínku (např. $-Tu'(L) = g$).

Kapitola 17

Metoda sítí

Metoda sítí (také metoda konečných diferencí) je numerický způsob řešení diferenciálních rovnic. Oblast pokryjeme pravidelnou (ve 2D obdélníkovou) sítí a objekty v diferenciální rovnici nahradíme jejich hodnotami v uzlech sítě. Původní problém tak převedeme na soustavu lineárních rovnic, jejíž řešení odpovídá hodnotám neznáme veličiny v uzlech sítě.

17.1 Metoda sítí v 1D

V mechanice se metoda sítí v 1D používá například k řešení nosníků či rovnice pro dříve odvozenou strunu. Formálně shodnou rovnicí je popsáno stacionární vedení tepla nebo šíření koncentrace.

Analýza problému

Uvažujme rovnici rovnováhy struny délky L v diferenciálním tvaru odvozenou v sekci 16 s jednotkovou předepínací silou

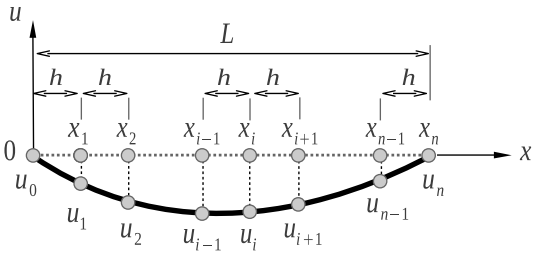
$$-u'' = f \tag{17.1}$$

a upevněnou na obou koncích $u(0) = u(L) = 0$. Přibližný vztah pro aproximaci druhé derivace řešení lze odvodit z Taylorova rozvoje funkce u v okolí bodu x. Hodnota funkce u pro bod $x + h$ je poté

$$u(x+h) = u(x) + u'(x)\frac{h}{1!} + u''(x)\frac{h^2}{2!} + C_1(h^3), \tag{17.2}$$

kde $C_1(h^3)$ je člen, který pro dostatečně malé h lze zanedbat. Obdobně pro $x - h$ je

$$u(x-h) = u(x) - u'(x)\frac{h}{1!} + u''(x)\frac{h^2}{2!} + C_2(h^3). \tag{17.3}$$



Obrázek 17.1: Rozložení uzlů sítě.

Odečtením obou výrazů a zanedbáním členů vyšších řádů získáme aproximaci první derivace funkce u ve tvaru

$$u'(x) \approx \frac{u(x+h) - u(x-h)}{2h} \tag{17.4}$$

a sečtením aproximací druhé derivace

$$u''(x) \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}. \tag{17.5}$$

Nyní strunu délky L rozdělíme pravidelně na n dílků (viz obr. 17.1). s krokem diskretizace h. Hodnoty řešení v uzlech x_i označme u_i a nechť $f_i = f(x_i)$. Užitím tohoto značení a dosazením (17.5) do (17.1) pro uzly $i = 1, 2, ..., n-1$ dostáváme soustavu lineárních rovnic

$$\begin{aligned} -(u_0 - 2u_1 + u_2) &= h^2 f_1 \\ -(u_1 - 2u_2 + u_3) &= h^2 f_2 \\ &\vdots \\ -(u_{i-1} - 2u_i + u_{i+1}) &= h^2 f_i \\ &\vdots \\ -(u_{n-2} - 2u_{n-1} + u_n) &= h^2 f_{n-1} \end{aligned} \tag{17.6}$$

s neznámými u_i, $i = 1, ..., n = 1$ a zadanými průhyby v krajních bodech $u(0) = u(L) = 0$. Převedením posunutí na pravou stranu dostáváme soustavu lineárních rovnic

$$\begin{pmatrix} 2 & -1 & 0 & & \cdots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & \\ & 0 & -1 & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} h^2 f_1 + u_0 \\ h^2 f_2 \\ h^2 f_3 \\ \vdots \\ h^2 f_{n-2} \\ h^2 f_{n-1} + u_n \end{pmatrix}.$$

Tuto soustavu označíme $\mathbf{Au} = \mathbf{f}$. K výpočtu lze použít z přímých řešičů např. Gaussovu eliminaci (viz kapitola 7), nebo některou z iteračních metod jako metoda sdružených gradientů (viz kapitola 15) nebo metoda největšího spádu apod.

Příklad 17.1. Určete prohnutí struny délky $L = 1$ $[m]$ uchycené na obou koncích a zatížené silou o hustotě $f(x) = \sin(x)$ $[N/m]$.

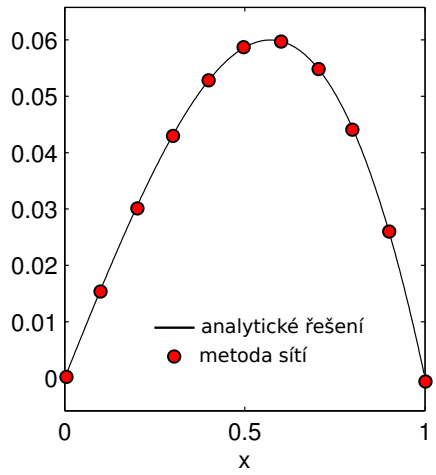
Příslušná okrajová úloha:

$$\begin{cases} -u'' = \sin(x), \ x \in (0, L). \\ u(0) = u(1) = 0 \end{cases}$$

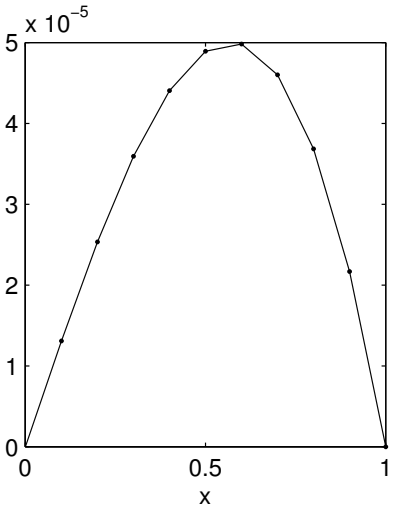
Algoritmus 17.1: Metoda sítí

```
 function struna1D_sit(L,N)
%% popis ulohy
%    L                              % delka struny
%    N                              % pocet prvku site
n  = N+1;                           % pocet uzlu
h  = L/N;                           % krok site
x  = (0:h:L)';                      % souradnice uzlu
%   -u''= f, u(0)=0, u(L)=0;        % rovnice struny
%    x z <0,L>                      % interval
%    f = sin(x)                     % zatizeni
%% Sestaveni ridke matice A
e = ones(n-2,1);
A = spdiags([-e 2*e -e], [-1 0 1], n-2, n-2);
%% vektor prave strany
b = eval('h^2*sin(x(2:end-1))');
%% numericke reseni
u=A \b;
u=[0;u;0];
%% analyticke reseni
u_analytic=@(x) (sin(x)-sin(L)*x/L);
%% vykresleni
figure(1),plot(x,u,'r.'),hold on,ezplot(u_analytic,[0,L]);hold
  off
figure(2),plot(x,abs(u-u_analytic(x)),'.-'),xlabel('x')
fprintf('Chyba reseni %3.2e\n',norm(u-u_analytic(x)));
```

Analytické a numerické řešení příkladu pro $N = 10$ je na obr. 17.2, chyba v normě 2 je na obr. 17.3.



Obrázek 17.2: Numerické a analytické řešení



Obrázek 17.3: Chyba řešení

17.2 Metoda sítí ve 2D

Metoda sítí ve 2D je vhodná např. pro řešení Poissonovy rovnice, která v mechanice popisuje rovnováhu sil v libovolném bodě membrány, nebo také vedení tepla v rovině atd. Ačkoliv poskytuje pouze přibližné numerické řešení, v mnoha případech jde pro složitost úlohy o jediné možné, které lze získat. Metoda sítí je silně omezena tvarem oblasti, na které se úloha řeší.

Analýza problému

Uvažujme úlohu napínání membrány na čtverci o straně délky L (viz obr. 17.4)

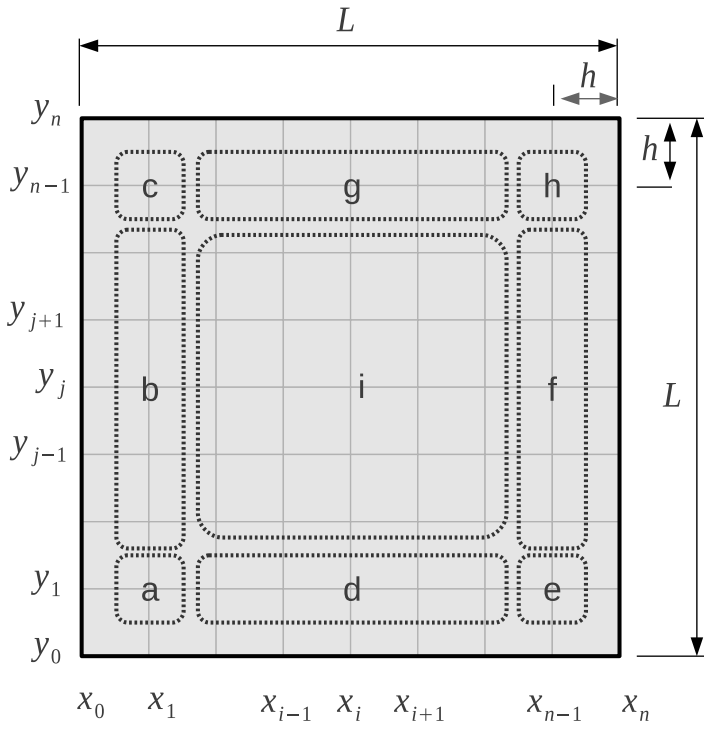
$$\begin{cases} -\triangle u = f \text{ pro } \forall (x,y) \in \Omega = (0,L) \times (0,L) \\ u(x,y) = 0 \text{ pro } \forall (x,y) \in \partial\Omega, \end{cases} \tag{17.7}$$

ve které podmínka $u(x,y) = 0$ pro $\forall (x,y) \in \partial\Omega$ zohledňuje uchycení na korpusu bubnu (hranici oblasti). Parciální derivaci u podle x a y odvodíme pomocí Taylorova rozvoje. Hodnota funkce u pro bod $(x+h,y)$ je

$$u(x+h,y) = u(x,y) + \frac{\partial}{\partial x}u(x,y)\frac{h}{1!} + \frac{\partial^2}{\partial x^2}u(x,y)\frac{h^2}{2!} + C_1(h^3),$$

a pro $(x-h,y)$

$$u(x-h,y) = u(x,y) - \frac{\partial}{\partial x}u(x,y)\frac{h}{1!} + \frac{\partial^2}{\partial x^2}u(x,y)\frac{h^2}{2!} + C_2(h^3).$$



Obrázek 17.4: Síť ve 2D.

Sečtením výrazů a zanedbáním členů vyšších řádů ($C_1(h^3)$, $C_2(h^3)$) obdržíme přibližný vztah druhé derivace u podle x ve tvaru

$$\frac{\partial^2}{\partial x^2}u(x,y) \approx \frac{u(x-h,y) - 2u(x,y) + u(x+h,y)}{h^2}.$$

Druhá derivace u podle y pak bude obdobně

$$\frac{\partial^2}{\partial y^2}u(x,y) \approx \frac{u(x,y-h) - 2u(x,y) + u(x,y+h)}{h^2}.$$

Nyní rozdělme čtverec o straně délky L na menší čtverce s krokem diskretizace h. Hodnoty řešení v uzlech $(x_i, y_j) = (x_0 + ih, y_0 + jh)$ označme $u_{i,j}$ a nechť $f_{i,j} = f(x_i, y_j)$. Přibližné vztahy pro druhé derivace u v okolí bodu (x_i, y_j) se přepíší do tvaru

$$\begin{aligned}\frac{\partial^2}{\partial x^2}u(x_i,y_j) &\approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}\\ \frac{\partial^2}{\partial y^2}u(x_i,y_j) &\approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}.\end{aligned} \tag{17.8}$$

Membrána je na krajích pevně uchycena a tedy $u_{k,0} = u_{0,k} = u_{k,n} = u_{n,k} = 0$ pro $k = 0, 1, ..., n$. Dříve odvozenou rovnici rovnováhy v libovolném uzlu sítě můžeme psát ve tvaru

$$-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{i,j}. \tag{17.9}$$

Podle obr. 17.4 zavádíme indexové množiny $\tilde{I} = \{2, ..., n-2\}$ a dostáváme tyto rovnice:

$$
\begin{array}{lcll}
a) & 4u_{1,1} - u_{2,1} - u_{1,2} = h^2 f_{1,1} + u_{0,1} + u_{1,0} & \\
b) & -u_{1,j-1} + 4u_{1,j} - u_{1,j+1} - u_{2,j} = h^2 f_{1,j} + u_{0,j} & j \in \tilde{I} \\
c) & -u_{1,n-2} + 4u_{1,n-1} - u_{2,n-1} = h^2 f_{1,n-1} + u_{0,n-1} + u_{1,n} & \\
d) & -u_{i-1,1} + 4u_{i,1} - u_{i+1,1} - u_{i,2} = h^2 f_{i,1} + u_{i,0} & i \in \tilde{I} \\
e) & -u_{n-2,1} - u_{n-1,2} + 4u_{n-1,1} = h^2 f_{n-1,1} + u_{n-1,0} + u_{n,1} & \\
f) & -u_{n-2,j} - u_{n-1,j-1} + 4u_{n-1,j} - u_{n-1,j+1} = h^2 f_{n-1,j} + u_{n,j} & j \in \tilde{I} \\
g) & -u_{i-1,n-1} - u_{i,n-2} + 4u_{i,n-1} - u_{i+1,n-1} = h^2 f_{i,n-1} + u_{i,n} & i \in \tilde{I} \\
h) & -u_{n-1,n-2} - u_{n-2,n-1} + 4u_{n-1,n-1} = h^2 f_{n-1,n-1} + u_{n-1,n} + u_{n,n-1} & \\
i) & -u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f_{i,j} & i,j \in \tilde{I}
\end{array}
$$

Jednotlivé rovnice v uzlech zapíšeme maticově a to tak, že uzly sítě jsou procházeny postupně po sloupcích tvořených uzly $\{(x_m, y_1), (x_m, y_2), ..., (x_m, y_{n-1})\}$. Hodnoty průhybů v uzlech jsou pak řazeny stejným způsobem

$$\mathbf{u} = \left((u_{1,1}, u_{2,1}, ..., u_{n-1,1}), (u_{1,2}, u_{2,2}, ..., u_{n-1,2}), ..., (u_{1,n-1}, u_{2,n-1}, ..., u_{n-1,n-1})\right)^T,$$

čemuž také odpovídá řazení prvků v matici $\mathbf{A}$ z předešlé soustavy rovnic a můžeme ji psát ve tvaru

$$
\mathbf{A} = \begin{pmatrix}
\overbrace{\begin{matrix} 4 & -1 & \cdots & 0 \\ -1 & 4 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 4 \end{matrix}}^{u_{1,1:n-1}} & \overbrace{\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & -1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & -1 & 0 \end{matrix}}^{u_{2,1:n-1}} & & \overbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & & \vdots \\ 0 & & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{matrix}}^{u_{n-1,1:n-1}} \\
\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & -1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & -1 & 4 \end{matrix} & \begin{matrix} 4 & -1 & \cdots & 0 \\ -1 & 4 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 4 \end{matrix} & \cdots & \begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & & \vdots \\ 0 & & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{matrix} \\
\vdots & \vdots & \ddots & \vdots \\
\begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & & \vdots \\ 0 & & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{matrix} & \begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & & \vdots \\ 0 & & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{matrix} & \cdots & \begin{matrix} 4 & -1 & \cdots & 0 \\ -1 & 4 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 4 \end{matrix}
\end{pmatrix}.
$$

Konečně vektor pravé strany téže soustavy vypadá následovně

$$\mathbf{f} = \begin{pmatrix} \begin{pmatrix} h^2 f_{1,1} + u_{0,1} + u_{1,0} \\ h^2 f_{1,2} + u_{0,2} \\ \vdots \\ h^2 f_{1,n-1} + u_{0,n-1} + u_{1,n} \end{pmatrix} \\ \begin{pmatrix} h^2 f_{2,1} + u_{2,0} \\ h^2 f_{2,2} \\ \vdots \\ h^2 f_{2,n-1} + u_{2,n} \end{pmatrix} \\ \vdots \\ \begin{pmatrix} h^2 f_{n-1,1} + u_{n-1,0} + u_{n,1} \\ h^2 f_{n-1,2} + u_{n,2} \\ \vdots \\ h^2 f_{n-1,n-1} + u_{n,n-1} + u_{n-1,n} \end{pmatrix} \end{pmatrix},$$

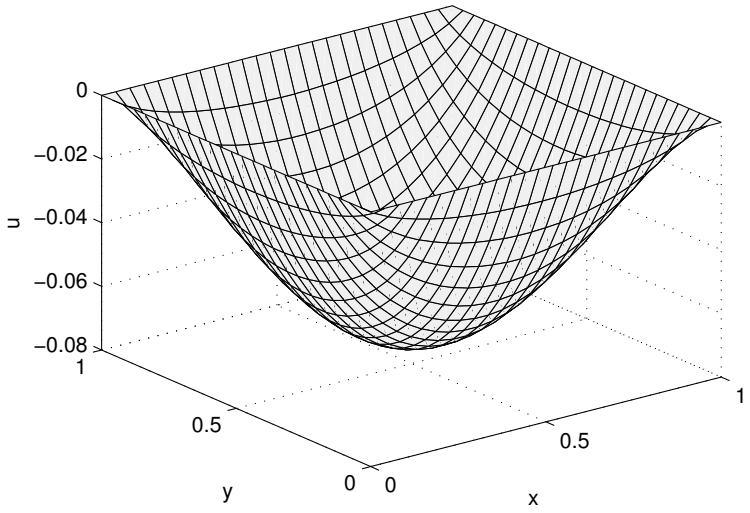
nebo po dosazení

$$\mathbf{f} = h^2 \begin{pmatrix} \begin{pmatrix} f_{1,1} \\ f_{1,2} \\ \vdots \\ f_{1,n-1} \end{pmatrix} \\ \begin{pmatrix} f_{2,1} \\ f_{2,2} \\ \vdots \\ f_{2,n-1} \end{pmatrix} \\ \vdots \\ \begin{pmatrix} f_{n-1,1} \\ f_{n-1,2} \\ \vdots \\ f_{n-1,n-1} \end{pmatrix} \end{pmatrix}.$$

Příklad 17.2. Určete průhyb mebrány uchycené na čtvercovém rámečku o straně délky $L = 1$ $[m]$ a zatížené silou o hustotě $f(x) = -1$ $[N/m^2]$.

Příslušná okrajová úloha:

$$\begin{cases} -\Delta u(x,y) = -1 \ \forall (x,y) \in \Omega = (0,1) \times (0,1). \\ u(x,y) = 0 \text{ pro } \forall (x,y) \in \partial\Omega \end{cases}$$



Obrázek 17.5: Řešení.

Algoritmus 17.2: Algoritmus metody sítí - membrána

```
function membrana2D_sit(N)
% N ... pocet elementu na hranu oblasti jednotkoveho ctverce
%% Rovnice
% -(d^2/dx^2+d^2/dy^2)u=f, f = -1;     Lx=Ly=1; u=0 na hranici
%% Pomocne konstanty
n=N-1;     % pocet vnitrnich uzlu na hranu (bez u_gamma)
np=n^2;    % pocet vsech vnitrnich uzlu
h=1/N;     % krok site
%% Efektivni sestaveni ridke matice A
e0  = 4*ones(np,1);
e2  = -ones(np,1);
e1h = -ones(np,1);e1h((n+1):n:end)=0;
e1d = -ones(np,1);e1d(n:n:end)=0;
A = spdiags([e2,e1d,e0,e1h,e2],[-n,-1,0,1,n],np,np);
%% Vektor prave strany
f = -ones(np,1)*h^2;
%% Vypocet u (vnitrni uzly)
u=A\f;
%% vizualizace
x=0:h:1; y=0:h:1;                  % souradnice
[X,Y]=meshgrid(x,y);
U=[zeros(1,N+1);...
   zeros(n,1),...
   reshape(u,n,[]),...
   zeros(n,1);zeros(1,N+1)];
surf(X,Y,U,'FaceColor','interp')
```

Výsledky příkladu pro $N = 25$ vidíme na obr. 17.5.

Kapitola 18

Metoda konečných prvků

Metoda konečných prvků (MKP) je nejrozšířenější numerickou metodou určenou k přibližnému řešení diferenciálních rovnic. Obdobně jako metoda sítí i MKP převádí původní eliptickou okrajovou úlohu na soustavu lineárních rovnic, jejímž řešením získáme přibližné řešení zadané úlohy. Existuje pouze málá skupina inženýrských problémů popsaných parciálními diferenciálními rovnicemi, u kterých je známo analytické řešení. Společným jmenovatelem v těchto případech je vhodný tvar oblasti, na které se řešení hledá (platí pro základní geometrické tvary jako čtverec, kruh, krychle, koule, válec apod.), ale také vhodná pravá strana a okrajové podmínky. Ve všech ostatních případech je nutné užít numerické řešení. Hlavní myšlenka MKP je rozdělit původní oblast na menší tzv. konečné prvky, kterými je možné pokrýt v podstatě libovolný tvar, což upřednostňuje tento přístup nad metodu sítí.

18.1 Metoda konečných prvků v 1D

Na rozdíl od metody sítí se v MKP nediskretizuje přímo původní okrajová úloha daná rovnicí rovnováhy, ale její tzv. slabá formulace.

Odvození slabé formulace

Řešme rovnici struny uchycené na obou koncích, tj. úlohu

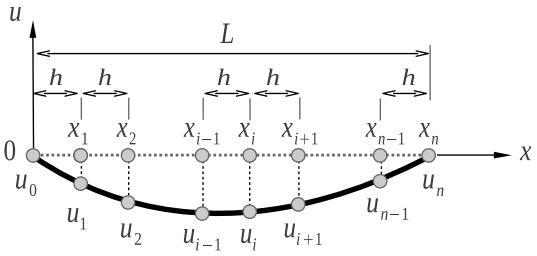
$$\begin{cases} -u''(x) = f(x) \text{ pro } \forall x \in (0,1) \\ u(0) = u(1) = 0, \end{cases}$$

kde pro jednoduchost uvažujeme $u \in C^2((0,1))$ a $f \in C((0,1))$ s jednotkovou předepínací silou $T = 1$. Zaveďme prostor testovacích funkcí

$$V = \left\{ v \in C^1((0,1)) \,|v(0) = v(1) = 0 \right\}.$$

Vynásobením direrenciální rovnice s testovací funkcí obdržíme rovnost

$$-u''(x)v(x) = f(x)v(x) \text{ pro } \forall x \in (0,1),\ \forall v \in V.$$



Obrázek 18.1: Diskretizace.

Výraz na levé i pravé straně dále integrujeme přes zadanou oblast a k integraci použijeme metodu per partes

$$-\int_0^1 u''(x)v(x)dx = \int_0^1 f(x)v(x)dx,\ \forall v \in V$$

a aplikujeme metodu per partes

$$-\left[u'(x)v(x)\right]_0^1 + \int_0^1 u'(x)v'(x)dx = \int_0^1 f(x)v(x)dx,\ \forall v \in V.$$

Z vlastností funkce v je člen $\left[u'(x)v(x)\right]_0^1$ pro obě meze nulový. Výsledná formulace vypadá následovně:

$$\begin{cases} \text{najdi } u \in V \text{ takové, že} \\ a(u,v) = b(v)\ \forall v \in V, \end{cases} \tag{18.1}$$

kde

$$a(u,v) = \int_0^1 u'(x)v'(x)dx \text{ a } b(v) = \int_0^1 f(x)v(x)dx,$$

jsou symetrická pozitivně definitní bilineární forma a lineární funkcionál.

Diskretizace

Označme posunutí v krajních bodech struny

$$u\left(x_0\right) = u_0,\ u\left(x_n\right) = u_n$$

a množinu uzlů sítě

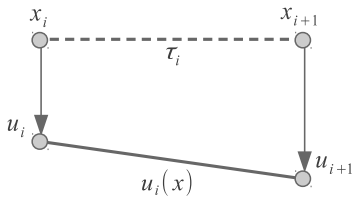
$$S_h = \left\{x_0, \ldots, x_n\right\},\ x_i = x_0 + ih,\ i = 0, \ldots, n. \tag{18.2}$$

Výpočetní oblast (v 1D interval) je pokryta $n_T = n$ konečnými prvky tvořící

$$T_h = \bigcup_{i=0}^{n_T-1} \tau_i,\ \tau_i = \langle x_i,\ x_{i+1} \rangle\ . \tag{18.3}$$

Při ekvidistantním kroku jsou délky všech konečných prvků rovny h. Aproximace průhybu $u(x)$ je realizována spojitou po částech lineární funkcí

$$u_i\left(x\right) = a_i x + b_i,\ \forall x \in \tau_i,\ u_i(x) = 0\ \forall x \notin \tau_i\ i = 0, 1, ..., n_T - 1. \tag{18.4}$$



Obrázek 18.2: Konečný prvek.

Pro i-tý prvek obdržíme posuvy v uzlech dosazením příslušných souřadnic (viz obr. 18.2)

$$u_i = u_i(x_i) = a_i x_i + b_i$$
$$u_{i+1} = u_i(x_{i+1}) = a_i x_{i+1} + b_i$$

a ze dvou rovnic o dvou neznámých spočteme konstanty

$$a_i = \tfrac{1}{h}(u_{i+1} - u_i), \; b_i = -\tfrac{1}{h}(x_i u_{i+1} - x_{i+1} u_i).$$

Zpětným dosazením dostáváme

$$u_i(x) = \tfrac{1}{h}(u_{i+1} - u_i) x_i - \tfrac{1}{h}(x_i u_{i+1} - x_{i+1} u_i).$$

Lokální matice tuhosti

Pro další popis využijeme ekvivalence slabé formulace s tzv. energetickou formulací (důkaz naleznete např. v [6]). Úkol zní následovně:

$$\text{najdi } u \in V: \; a(u,v) = b(v), \; \forall v \in V \iff \min_{u \in V} \frac{1}{2} a(u,u) - b(u). \tag{18.5}$$

Diskretizací bilineární formy dostáváme

$$a(u,u) = \int_0^1 (u'(x))^2 \, dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} (u'(x))^2 \, dx \approx$$
$$\sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} (u_i'(x))^2 \, dx = \int_0^1 (u_h'(x))^2 \, dx = a(u_h, u_h),$$

kde $u_h(x)$ $\forall x \in \tau_i$ je po částech lineární funkce aproximace řešení u. Derivace funkce posuvů u_h pro i-tý prvek je

$$u_i'(x) = (a_i x + b_i)' = a_i = \frac{1}{h}(-u_i + u_{i+1}) = \frac{1}{h}(-1, \; 1) \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix}$$

a pro krajní elementy

$$u_0'(x) = a_0 = \frac{u_1}{h},\ u_{n-1}'(x) = a_{n-1} = \frac{-u_{n-1}}{h}.$$

Nyní si vyjádříme člen $(u_i'(x))^2$ ve tvaru

$$(u_i'(x))^T (u_i'(x)) = \frac{1}{h^2} (u_i,\ u_{i+1}) \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix}$$

a integrujeme nad i-tým prvkem

$$\int_{x_i}^{x_{i+1}} (u_i'(x))^T (u_i'(x))\, dx = \frac{1}{h^2} (u_i,\ u_{i+1}) \begin{pmatrix} \int_{x_i}^{x_{i+1}} dx & -\int_{x_i}^{x_{i+1}} dx \\ -\int_{x_i}^{x_{i+1}} dx & \int_{x_i}^{x_{i+1}} dx \end{pmatrix} \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix}.$$

Prvky matice jsou rovny integrálu $\int_{x_i}^{x_{i+1}} dx$, mimodiagonální členy jsou záporné. Jelikož platí, že

$$\int_{x_i}^{x_{i+1}} dx = h,$$

po úpravě obdržíme

$$\int_{x_i}^{x_{i+1}} (u_i'(x))^2\, dx = (u_i,\ u_{i+1})\, \mathbf{A}_i \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix},$$

kde

$$\mathbf{A}_i = \frac{1}{h} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{18.6}$$

je tzv. lokální matice tuhosti.

Globální matice tuhosti

Globální matice tuhosti se sestavuje postupně z příspěvků všech prvků sítě, tj.

$$\mathbf{A} = \sum_{i=1}^{n_T} \tilde{\mathbf{A}}_i = \sum_{i=1}^{n_T} \frac{1}{h} \begin{pmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 1 & \cdots & -1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -1 & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{pmatrix}, \tag{18.7}$$

kde $\tilde{\mathbf{A}}$ je rozšíření $\mathbf{A}_i$ nulami na velikost $(n+1)\times(n+1)$ globální matice. Pro strunu s krokem sítě h má globální matice tvar

$$\mathbf{A} = \frac{1}{h}\begin{pmatrix} 1 & -1 & 0 & & & \\ -1 & 2 & -1 & & & \\ 0 & -1 & 2 & & & \\ & & & \ddots & & \\ & & & & 2 & -1 \\ & & & & -1 & 1 \end{pmatrix}. \tag{18.8}$$

Tato matice je pozitivně semidefinitní a lze ji regularizovat zohledněním okrajových podmínek.

Lokální vektor zatížení

Vektor zatížení obdržíme rozepsáním členu lineárního funkcionálu

$$b(u) = \int_0^1 u(x)f(x)dx = \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} u(x)f(x)dx \approx \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} u_i(x)f(x)dx = b(u_h).$$

Funkci $u_i(x)$ zapíšeme ve tvaru

$$u_i(x) = \tfrac{1}{h}\,(x_{i+1}-x,\ x-x_i)\begin{pmatrix} u_i \\ u_{i+1}\end{pmatrix}$$

a pronásobíme s $f(x)$. Intgrací přes τ_i dostaneme

$$\int_{x_i}^{x_{i+1}} u_i(x)^T f(x)dx = \frac{1}{h}\,(u_i,\ u_{i+1})\begin{pmatrix} \int_{x_i}^{x_{i+1}} f(x)\,(x_{i+1}-x)\,dx \\ \int_{x_i}^{x_{i+1}} f(x)\,(x-x_i)\,dx \end{pmatrix},$$

kde

$$\mathbf{f}_i = \frac{1}{h}\begin{pmatrix} \int_{x_i}^{x_{i+1}} f(x)\,(x_{i+1}-x)\,dx \\ \int_{x_i}^{x_{i+1}} f(x)\,(x-x_i)\,dx \end{pmatrix} \tag{18.9}$$

je tzv. lokální vektor zatížení i-tého prvku.

V řadě případů si vystačíme s přibližným výpočtem integrálu některou z numerických metod. Např. pro tzv. obdélníkové pravidlo je první člen lokálního vektoru zatížení

$$\int_{x_i}^{x_{i+1}} f(x)\,(x_{i+1}-x)\,dx \approx h\cdot f(x_s)\,(x_{i+1}-x_s) = \frac{h^2}{2}f(x_s),$$

kde $x_s = (x_i + x_{i+1})/2$ je souřadnice těžiště prvku.

Globální vektor zatížení

Globální vektor zatížení $\mathbf{f}$ se sestavuje obdobně jako globální matice tuhosti . Za i-tý prvek se na pozice i a $i+1$ přičte lokální příspěvek $\mathbf{f}_i = (f_1^i,\ f_2^i)^T$, tj.

$$\mathbf{f} = \sum_{i=0}^{n-1} \mathbf{f}_i = \frac{1}{h} \begin{pmatrix} \int_{x_0}^{x_1} f(x)\,(x_1 - x)\,dx \\ \vdots \\ \int_{x_{i-1}}^{x_i} f(x)\,(x - x_{i-1})\,dx + \int_{x_i}^{x_{i+1}} f(x)\,(x_{i+1} - x)\,dx \\ \vdots \\ \int_{x_{n-1}}^{x_n} f(x)\,(x - x_{n-1})\,dx \end{pmatrix}.$$

Poznámka 18.1. Zadané Dirichletovy podmínky $u(0) = u(1) = 0$ předepisují průhyb v koncových uzlech.

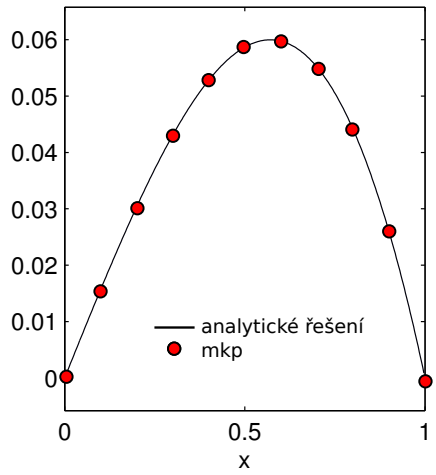
Příklad 18.2. Řešme příklad 17.1 pro $N = 10$ pomocí MKP.

Algoritmus 18.1: MKP - struna

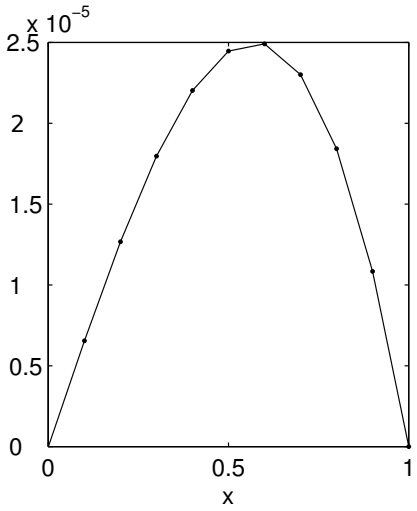
```
function struna1D_MKP(L,N)
%% popis ulohy
%    N                              % pocet prvku site
%    L                              % delka struny
n  = N+1;                           % pocet uzlu
h  = L/N;                           % krok site
x  = (0:h:L)';                      % souradnice uzlu
%   -u''= f, u(0)=0, u(L)=0;        % rovnice struny
%    x z <0,L>                      % interval
f = @(x) sin(x);                    % hustota sil
%% Sestaveni plne matice A
A = zeros(n);b=zeros(n,1);
A_local=[1 -1;-1 1]/h;              % lokalni matice tuhosti
% integral v b_local pocitan lichobeznikovym pravidlem
for i = 1 : N % cyklus pres prvky site
  A([i,i+1],[i,i+1])=A([i,i+1],[i,i+1])+A_local;
  x_s=0.5*(x(i)+x(i+1)); % souradnice teziste
  b_local = 0.5*h*f(x_s)*[1;1];
  b([i,i+1])=b([i,i+1])+b_local;
end
%% zohledneni okrajovych podminek v matici A a vektoru b
A([1,n],:)=0; A(:,[1,n])=0;A(1,1)=1;A(n,n)=1;
b(1)=0;b(n)=0;
%% numericke reseni
u=A\b;
%% analyticke reseni
u_analytic=@(x) sin(x)-sin(L)*x/L;
%% vykresleni
```

```
figure(1),plot(x,u,'r.'),hold on,ezplot(u_analytic,[0,L]);hold
  off
figure(2),plot(x,abs(u-u_analytic(x)),'.-'),xlabel('x')
fprintf('Chyba reseni %3.2e\n',norm(u-u_analytic(x)));
```

Numerické a analytické řešení je na obr. 18.3 a chyba MKP od přesného řešení v uzlech sítě v normě 2 je na obr. 18.4.



Obrázek 18.3: Numerické a analytické řešení



Obrázek 18.4: Chyba řešení

18.2 Metoda konečných prvků ve 2D

Stejně jako v 1D se diskretizuje slabá formulace úlohy.

Odvození slabé formulace

Řešme následující problém:

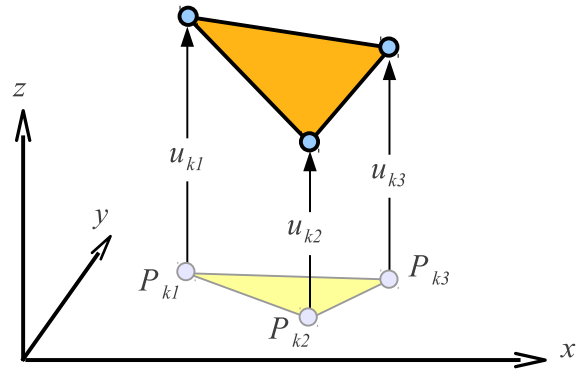
$$\begin{aligned} -\triangle u &= f \text{ pro } \forall (x,y) \in \Omega \\ u(x,y) &= 0 \text{ pro } \forall (x,y) \in \partial\Omega, \end{aligned} \tag{18.10}$$

kde

$$f \in C(\Omega),\ u \in C^2(\Omega).$$

Zaveďme prostor testovacích funkcí

$$V = \left\{ v \in C^1(\Omega) \,|\, v = 0 \text{ na } \partial\Omega \right\}$$



Obrázek 18.5: Konečný prvek.

a vynásobme rovnici rovnováhy sil membrány s funkcí v

$$-\frac{\partial^2 u}{\partial x^2}v - \frac{\partial^2 u}{\partial y^2}v = fv \text{ na } \Omega, \ \forall v \in V.$$

Pravou a levou stranu integrujeme na oblasti Ω

$$-\int_\Omega \left(\frac{\partial^2 u}{\partial x^2}v + \frac{\partial^2 u}{\partial y^2}v\right) \mathrm{d}\Omega = \int_\Omega fv \ \mathrm{d}\Omega, \ \forall v \in V.$$

Pomocí Greenovy formule upravíme rovnici

$$\int_\Omega \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y}\right) \mathrm{d}\Omega - \int_{\partial\Omega} \left(\frac{\partial u}{\partial x}n_x v + \frac{\partial u}{\partial y}n_y v\right) \mathrm{d}s = \int_\Omega fv \ \mathrm{d}\Omega, \ \forall v \in V.$$

$$a(u,v) = \int_\Omega \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y}\right) \mathrm{d}\Omega, \ b(v) = \int_\Omega fv \,\mathrm{d}\Omega + \int_{\partial\Omega} \left(\frac{\partial u}{\partial x}n_x v + \frac{\partial u}{\partial y}n_y v\right) \mathrm{d}s.$$

Trianglulace oblasti

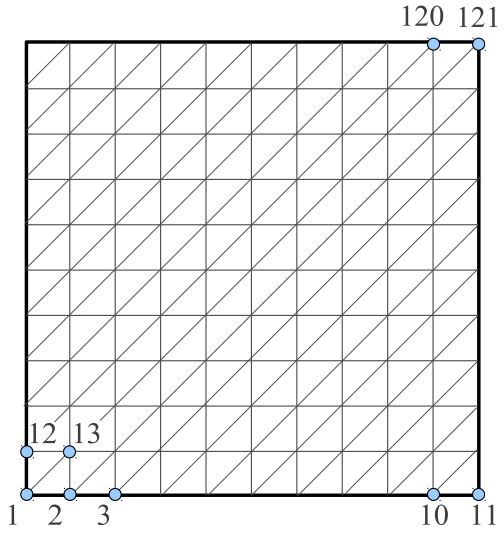
Uvažujme čtvercovou oblast (viz obr. 18.6) rozloženou na trojúhelníky. Označme uzly sítě

$$S_h = \{P_1, ..., P_n\}. \tag{18.11}$$

Triangulizací rozumíme rozdělení oblasti T_h na konečné prvky t_k o třech vrcholech. Jsou-li obě strany čtvercové oblasti rozděleny na n uzlů, pro konečné prvky platí

$$T_h = \bigcup_{k=1}^{n_T} t_k, \ t_k = \Delta_{P_{k1},P_{k2},P_{k3}}, \ n_T = 2(n-1)^2, \tag{18.12}$$

kde t_k je k-tý prvek s vrcholy P_{k1}, P_{k2}, P_{k3} a n_T je počet všech prvků. Součet plošek všech prvků přibližně odpovídá obsahu původní plochy



Obrázek 18.6: Triangularizace.

$$T_h \approx \bar{\Omega}. \tag{18.13}$$

Konkrétně pro zvolenou čtvercovou oblast platí v předešlé relaci rovnost. Průhyb k-tého uzlu je značen

$$u_k \approx u(P_k) = u(x_k, y_k) \text{ pro } k = 1, ..., n. \tag{18.14}$$

Interpolace řešení

Zavedmě aproximaci funkce posuvů na k-tém prvku (obr. 18.5) pomocí lineární funkce

$$\begin{aligned} u_k(x,y) &= a_k x + b_k y + c_k, \ \forall (x,y) \in t_k \\ u_k(x,y) &= 0, \ \forall (x,y) \notin t_k. \end{aligned} \tag{18.15}$$

Rovnici můžeme přepsat do tvaru

$$u_k(x,y) = \boldsymbol{\varphi}(x,y)\mathbf{a}_k = [x,y,1][a_k, b_k, c_k]^T, \tag{18.16}$$

kde $\boldsymbol{\varphi}$ je matice aproximačního polynomu a $\mathbf{a}_k$ je vektor konstant. Postupným dosazováním souřadnic pro vrcholy prvku pišme

$$\begin{aligned} a_k x_1 + b_k y_1 + c_k &= u_{k_1} \\ a_k x_2 + b_k y_2 + c_k &= u_{k_2} \\ a_k x_3 + b_k y_3 + c_k &= u_{k_3} \end{aligned} \tag{18.17}$$

a s označením $\mathbf{u}_k = \begin{pmatrix} u_{k_1} & u_{k_2} & u_{k_3} \end{pmatrix}^T$ relaci mezi vektorem konstant a vektorem zobecnělých posuvů

$$\mathbf{D}_k \mathbf{a}_k = \mathbf{u}_k. \tag{18.18}$$

Vyjádření konstant přes zobecnělé posuvy $\mathbf{u}_k$ má zásadní význam pro algoritmizaci. Zde matice

$$\mathbf{D}_k = \begin{pmatrix} x_{k_1} & y_{k_1} & 1 \\ x_{k_2} & y_{k_2} & 1 \\ x_{k_3} & y_{k_3} & 1 \end{pmatrix} \tag{18.19}$$

je regulární, a proto platí

$$\mathbf{a}_k = \mathbf{D}_k^{-1}\mathbf{u}_k. \tag{18.20}$$

Nyní stačí v rovnici (18.16) za $\mathbf{a}_k$ dosadit (18.20)

$$u_k(x,y) = \boldsymbol{\varphi}(x,y)\mathbf{D}_k^{-1}\mathbf{u} = \mathbf{N}_k\mathbf{u}. \tag{18.21}$$

Zde $\mathbf{N}_k = \boldsymbol{\varphi}(x,y)\mathbf{D}_k^{-1}$ je tzv. matice bázových funkcí (k-tého prvku).

Lokální matice tuhosti

Pro sestavení lokální matice tuhosti použijeme vztah

$$a(u_k,u_k) = \int_\Omega \nabla u_k \left(\nabla u_k\right)^T \mathrm{d}\Omega. \tag{18.22}$$

Gradient pole posuvů je zapsán ve tvaru

$$\left(\nabla u_k\right)^T = \begin{pmatrix} \partial u_k/\partial x \\ \partial u_k/\partial y \end{pmatrix} = \nabla\boldsymbol{\varphi}(x,y)\mathbf{D}_k^{-1}\mathbf{u}_k = \mathbf{G}_k\mathbf{u}_k, \tag{18.23}$$

přičemž parcilální derivace podle x a y se projeví pouze na matici aproximačního polynomu. Zde $\mathbf{G}_k = \nabla\left(\boldsymbol{\varphi}(x,y)\right)\mathbf{D}_k^{-1}$ je transformační matice (tzv. derivace bázových funkcí). Dosazením (18.23) do (18.22) dostáváme

$$\int_\Omega \nabla u_k \left(\nabla u_k\right)^T \mathrm{d}\Omega = \mathbf{u}_k^T \int_{t_k} \mathbf{G}_k^T\mathbf{G}_k \mathrm{d}\Omega\ \mathbf{u}_k = \mathbf{u}_k^T\mathbf{A}_k\mathbf{u}_k. \tag{18.24}$$

Jelikož je volený aproximační polynom $\boldsymbol{\varphi}$ prvního stupně, gradient pole posunutí je na konečném prvku konstantní a mezi jednotlivými prkvy obecně nespojitý. Ve výrazu $\int_{t_k} \mathbf{G}_k^T\mathbf{G}_k\mathrm{d}\Omega$ se integruje pouze přes konstanty, stačí provést součiny matic a výsledek přenásobit plochou konečného prvku. Platí, že plocha trojúhelníku $S_\triangle = \frac{1}{2}\left|\det\left(\mathbf{D}_k\right)\right|$, potom

$$\mathbf{A}_k = \mathbf{G}_k^T\mathbf{G}_k\frac{1}{2}\left|\det\left(\mathbf{D}_k\right)\right|. \tag{18.25}$$

je lokální maticí tuhosti .

Globální matice tuhosti

Funkcionál energie systému je popsán rovnicí

$$U = \frac{1}{2}a(u_h, u_h) - b(u_h) = \sum_{k=1}^{n_T} \left(\frac{1}{2}\mathbf{u}_k^T \mathbf{A}_k \mathbf{u}_k - \mathbf{u}_k^T \mathbf{f}_k \right) \tag{18.26}$$

a je výsledkem součtu potenciální energie vnitřních a vnějších sil $(U_I + U_E)$ všech prvků úlohy. Potenciální energie vnitřních sil je

$$U_I = \frac{1}{2}a(u_h, u_h) = \sum_{k=1}^{n_T} \frac{1}{2}\mathbf{u}_k^T \mathbf{A}_k \mathbf{u}_k. \tag{18.27}$$

Globální matici $\mathbf{A}$ lze sestavit s využitím předešlého vzorce

$$\mathbf{A} = \sum_{k=1}^{n_T} \mathbf{A}_k, \text{ kde } \mathbf{A}_k = \begin{pmatrix} a_{1,1}^k & a_{1,2}^k & a_{1,3}^k \\ a_{2,1}^k & a_{2,2}^k & a_{2,3}^k \\ a_{3,1}^k & a_{3,2}^k & a_{3,3}^k \end{pmatrix}. \tag{18.28}$$

Označme globální čísla uzlů k-tého prvku k_1, k_2, k_3 a příspěvek lokální matice $\mathbf{A}_k$ v matici $\mathbf{A}$ je $a_{k_i,k_j} = a_{k_i,k_j} + a_{i,j}^k$ pro $i, j = 1, 2, 3$. Obecně lze globální matici zapsat ve tvaru

$$\mathbf{A} = \sum_{k=1}^{n_T} \begin{pmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & a_{1,1}^k & \cdots & a_{1,2}^k & \cdots & a_{1,3}^k & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & a_{2,1}^k & \cdots & a_{2,2}^k & \cdots & a_{2,3}^k & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & a_{3,1}^k & \cdots & a_{3,2}^k & \cdots & a_{3,3}^k & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{pmatrix}.$$

Lokální vektor zatížení

Vektor uzlových sil je rozdělen na příspěvek od hustoty sil

$$\int_{t_k} u^T f \, \mathrm{d}\Omega = \mathbf{u}_k^T \int_{t_k} \mathbf{N}_k^T f \, \mathrm{d}\Omega = \mathbf{u}_k^T \mathbf{f}_k$$

a příspěvek liniového zatížení

$$\oint_{\Gamma_T^k} u^T T \, \mathrm{d}S = \mathbf{u}_k^T\big|_{\Gamma_T^k} \oint_{\Gamma_T^k} \mathbf{N}_k^T\big|_{\Gamma_T^k} T \, \mathrm{d}S = \mathbf{u}_k^T\big|_{\Gamma_T^k} \mathbf{T}_k, \ \Gamma_T^k := \overline{P_{k_I} P_{k_{II}}}, \ I, II \in \{1, 2, 3\}.$$

Globální vektor zatížení

Do globálního vektorů pravé strany se postupně nasčítávají příspěvky všech prvků

$$\int_\Omega u^T f \,\mathrm{d}\Omega + \oint_{\Gamma_T} u^T T \,\mathrm{d}S = \sum_{k=1}^{n_T} \int_{t_k} u^T f \,\mathrm{d}\Omega + \sum_{k=1}^{n_{\Gamma_T}} \oint_{\Gamma_T^k} u^T T \,\mathrm{d}S = \sum_{k=1}^{n_T} \mathbf{u}_k^T \mathbf{f}_k + \sum_{k=1}^{n_{\Gamma_T}} \mathbf{u}_k^T\big|_{\Gamma_T^k} \mathbf{T}_k$$

$$\mathbf{f}_{V,k} = (f_i^k)_{i=1,\dots,3},\ \mathbf{f}_{V,k} = \sum_{k=1}^{n_T} \mathbf{f}_k = \sum_{k=1}^{n_T} \begin{pmatrix} 0 & \cdots & f_1^k & \cdots & f_2^k & \cdots & f_3^k & \cdots & 0 \end{pmatrix}^T$$

$$\mathbf{T}_k = (t_i^k)_{i=1,..,2},\ \mathbf{f}_T = \sum_{k=1}^{n_{\Gamma_T}} \mathbf{T}_k = \sum_{k=1}^{n_{\Gamma_T}} \begin{pmatrix} 0 & \cdots & t_1^k & \cdots & t_2^k & \cdots & 0 \end{pmatrix}^T.$$

Předepsání okrajových podmínek

Předpokládejme, že na hranici je předepsáno posunutí $g(x, y)$

$$u(x,y) = g(x,y)\ \forall (x,y) \in \partial\Omega$$
$$\partial\Omega \approx B_h = \{P_i = (x_i, y_i) \in S_h \big| P_i \in \partial\Omega\}$$
$$I_B = \{i \in N \big| P_i \in B_h\}$$

$$i - \text{tá rovnice},\ i \in I_B : u_i = g_i = g(P_i)$$

$$i - \text{tá rovnice},\ i \notin I_B : \sum_{j=1}^{n} a_{i,j} u_j = f_i \implies \sum_{\substack{j=1 \\ j \notin I_B}}^{n} a_{i,j} u_j = f_i - \sum_{j \in I_B}^{n} a_{i,j} g_j$$

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & \cdots & 0 & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & 0 & \cdots & a_{n,n} \end{pmatrix},\ \mathbf{f} = \begin{pmatrix} f_1 - \sum_{j \in I_B} a_{1,j} g_j \\ \vdots \\ g_i \\ \vdots \\ f_n - \sum_{j \in I_B} a_{n,j} g_j \end{pmatrix}\ i,\ \forall i \in I_B$$

Algoritmus 18.2: MKP - membrána

```
function membrana2D_MKP(N)
% N ... pocet elementu na hranu x a y
%% Rovnice
% -(d^2/dx^2+d^2/dy^2)*u=f, f = 1;
%% Volitelne hodnoty (velikost, pocet prvku)
Lx = 1;          % delka hrany ve smeru osy x
Ly = 1;          % delka hrany ve smeru osy y
const = -1;      % fakor sily
```

```
%% sit
[elements,coordinates]=mesh_rectangle(N,N,Lx,Ly);
%% pomocne konstanty
Nx=N;Ny=N;nx=Nx+1; ny=Ny+1; n=nx*ny;
%% matice tuhosti a vektor prave strany
A = zeros(n,n);f=zeros(n,1);
Psi = [1 0 0;0 1 0];
for i=1:2*Nx*Ny
  x=coordinates(elements(i,:),1);
  y=coordinates(elements(i,:),2);
  D=[x y ones(3,1)]; detD=abs(det(D));  G=Psi/D;
  A_local = G'*G*detD*0.5;
  f_local = const*0.5*detD*[1;1;1]/3;
  A(elements(i,:),elements(i,:))=...
    A(elements(i,:),elements(i,:))+A_local;
  f(elements(i,:))=f(elements(i,:))+f_local;
end
%% indexy uzlu na hranici (u_{\Gamma}=0)
i_u0=[1:nx, (nx+1):nx:(nx*ny-2*nx+1),...
  2*nx:nx:(nx*(ny-1)), nx*ny-nx+1:nx*ny];
%% regularizace matice tuhosti a uprava vektoru prave strany
A(i_u0,:)=0;A(:,i_u0)=0;
A(i_u0,i_u0)=A(i_u0,i_u0)+eye(length(i_u0));f(i_u0)=0;
%% Reseni
u=A\f;
trisurf(elements,coordinates(:,1),coordinates(:,2),u,...
  'Facecolor','interp');
```

Rejstřík

Literatura

[1] Z. Dostál: Optimal quadratic programming algorithms: with applications to variational inequalities, Springer

[2] L. N. Trefethen a D. Bau, Numerical Linear Algebra, SIAM, 1997, ISBN:0-89871-361-7

[3] Stewart, G. W., Matrix algorithms, SIAM, 2001, ISBN:0-89871-503-2

[4] MATLAB Documentation, MathWorks, R2011a, http://www.mathworks.com/

[5] V. Vondrák a L. Pospíšil, Numerické metody I., Matematika pro inženýry 21. století.

[6] R. Blaheta, Numerické modelování a metoda konečných prvků., Matematika pro inženýry 21. století.