ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta stavební
Katedra mechaniky

Analýza implementace tradičních příkladů rozměrové optimalizace

Implementation analysis of sizing optimization benchmarks

Soutěžní práce

Studijní program: Stavební inženýrství
Studijní obor: Konstrukce pozemních staveb

Vedoucí práce: Ing. Matěj Lepš, Ph.D.

Adéla Pospíšilová

Praha 2011

Obsah

Abstrakt

Tato práce se zabývá studiem klasických příkladů rozměrové diskrétní optimalizace. Chceme-li najít globální optimum, je nutno použít např. metody větví a mezí. Použijeme-li dolní odhad získaný spojitou rozměrovou optimalizací a horní odhad získaný z publikovaných článků, přesto bude nutno spouštět výpočty mnohokrát. Proto se vyplatí udělat časovou optimalizaci jednotlivých kódů.

Z nepřeberného množství byly vybrány čtyři nejčastěji používané konstrukce dostupné v zahraniční i tuzemské literatuře. Na těchto konstrukcích budeme testovat jednotlivé výpočetní metody. Pro spočítání potřebných veličin je zvolena metoda konečných prvků, neboť je vhodná pro její následnou algoritmizaci. Je třeba najít optimální skript, který bude mít nejrychlejší čas svého výpočtu. Doba výpočtu závisí na užitých matematických metodách, proto byly vybrány tři nejznámější přímé metody a jedna metoda iterační. Závisí však i na jazyce, ve kterém je výpočetní metoda implementována.

Prostředí MATLAB je dnes velmi populární. Je pro něj vytvářeno a optimalizováno mnoho funkcí, k dispozici je i široká škála toolboxů. Ke všem funkcím je k dispozici obsáhlá nápověda. Jedná se však o interpretovaný jazyk, který bývá zpravidla pomalejší než jazyk kompilovaný. Nicméně i MATLAB nabízí zkompilování kódu například do samostatně spustitelné aplikace, což může výrazně ovlivnit čas výpočtu.

Naproti tomu přímo kompilované jazyky jako například C++ by měly dosahovat větší rychlosti výpočtu. Navíc jsou k nim (většinou zdarma) dostupné knihovny nabízející vykonávání různých matematických operací. Není pak třeba tyto funkce programovat a následně optimalizovat. V této práci proto budou tyto různé způsoby výpočtu porovnány.

Abstract

This contribution focuses on studies of classical sizing discrete optimization benchmarks. Our vision is to use these scripts for the next optimization with branch-and bound methods. Although equipped with upper bounds from optima found in literature and lower bounds given by continuous solutions, still there will be a need for a dozen of evaluations. Therefore, the following time optimization of computing codes is on demand.

We have chosen the four most often used structures which can be found in the available literature. Required deflections and stresses are computed with the finite element method, which is not only efficient but also easy to implement. For a practical usage of benchmarks, it is necessary to find an optimal script, which is the least computationally demanding. Time of one computation dominantly depends on the used mathematical methods for solving systems of linear equations. Three well-known direct methods and one iterative method were chosen for the study. However, performance also depends on the implementation details, i.e. mainly on the selection of an appropriate programming language.

Nowadays, the MATLAB environment is very popular. A lot of methods and procedures are created and optimized for it and many scientific toolboxes are available. Moreover, the development of a new code is supported by comprehensive help. However, MATLAB is an interpreted language, which should be slower than a compiled language. To solve this deficiency, MATLAB offers a compilation of scripts that can improve performance.

Oppositely, compiled languages like C++ should be faster. Free libraries providing routines for mathematical methods are available and therefore, it is not necessary to code and

optimize these procedures. All these various computational methods and implementations are investigated in this contribution.

Klíčová slova

rozměrová diskrétní optimalizace, příklady konstrukcí pro optimalizaci, přímé a iterační řešiče, plná a řídká matice tuhosti, implementace

Keywords

sizing discrete optimization, benchmarks, direct and iterative methods, dense and sparse stiffness matrix, implementation

1 Úvod

V dnešní době se počítačový výkon dá využít na ledacos. Jednou z možných úloh pro využití tohoto výkonu jsou optimalizační metody. Tyto metody se dají využít ve stavebnictví k nalezení vhodných skladeb profilů pro jednotlivé konstrukční prvky. Jednotlivé profily může dodat přímo výrobce, proto mají tyto úlohy reálné využití a investorovi mohou ušetřit mnoho peněz.

Naše pozornost je tedy zaměřena na tzv. rozměrovou optimalizaci konstrukcí. Cílem je najít příčné řezy příhradové konstrukce tak, aby byla minimalizována hmotnost konstrukce, ale aby byla zároveň splněna jistá omezení, nejčastěji limitní napětí a posuny. Speciální podtřídou jsou příklady tzv. diskrétní, kde příčné řezy nabývají jen diskrétních hodnot, obvykle z předem daného seznamu.

Pro spuštění optimalizační úlohy je třeba nalézt vhodné výpočetní metody. Jednak metody, které řeší vnitřní síly respektive napětí na konstrukci (např. deformační metoda, metoda konečných prvků apod.) a též použité matematické metody pro řešení soustav lineárních algebraických rovnic (např. Gaussova metoda, Choleského dekompozice, LDL^T rozklad, metoda největšího spádu, metoda sdružených gradientů, Jakobiho metoda atd.). Tyto výpočetní metody se pak použijí v sestavovaných algoritmech. U každého algoritmu je potřeba ověřit jeho správnost a zajistit jeho možnou aplikaci na ostatní konstrukce. Pro implementaci je vhodné použít dostatečně malé, ale zároveň reprezentativní konstrukce, tzv. benchmarky.

Cílem této práce je zoptimalizovat jednotlivé implementované kódy a prozkoumat jejich efektivnost, tedy dobu výpočtu. Krátký výpočetní čas je nezbytný ke spuštění optimalizační úlohy. Vhodné úpravy kódů však nejsou důležité jen pro tuto úlohu, ale i pro všechny úlohy ostatní. Správný kód by měl vždy trvat co nejkratší dobu a zabírat pokud možno co nejméně paměti.

Pro implementaci se dají použít různé programovací jazyky. V této práci bude zvoleno prostředí MATLAB, které nabízí kromě interpretovaného jazyka i množství knihoven funkcí, tzv. toolboxů, nástrojů pro pomoc s optimalizací kódu, tzv. profilerů, a též je možné využít nástroj pro kompilaci kódu do samostatně spustitelné aplikace, což by mohlo znamenat značné urychlení výpočtu. Pro porovnání bude zvolen ještě další programovací jazyk, tentokrát kompilovaný, a to C/C++.

2 Optimalizace stavebních konstrukcí

Dle prof. Stevena [Steven, 2003] existuje několik druhů optimalizace konstrukcí:

Topologická optimalizace (Topology optimization) se zabývá hledáním tvaru konstrukce, když předem neznáme její přesný tvar. Známe ale prostředí (jako např. umístění podpor), optimalizační kritéria (např. hmotnost konstrukce) a omezení [Sigmund and Bendsœ, 2003].

U *optimalizace tvaru* (Shape optimization) známe dopředu topologii konstrukce, ovšem problémy může dělat část konstrukce nebo její detail. Snahou je zde najít optimální tvar, aby byla zajištěna nejlepší distribuce napětí v inkriminovaném místě [Lepš, 2004].

U *rozměrové optimalizace* (Size optimization) máme předem známou sadu ploch příčných průřezů, tvar konstrukce a prostředí. Cílem je zkombinovat průřezy tak, aby byly dodrženy určité předem dané omezující podmínky (maximální deformace konstrukce, maximální napětí apod.) za minimalizace váhy, a tudíž i celkové ceny spotřebovaného materiálu. Rozměrová optimalizace může být spojitá a diskrétní.

Optimalizace skladby (Layout optimization) je speciálním typem rozměrové optimalizace. Pokud se blíží průřezová plocha prvku nulové hodnotě, pak tento prvek nemusí být v konstrukci vůbec použit [Kirsch, 1995].

V této práci se budeme zabývat pouze diskrétní rozměrovou optimalizací, kdy ze sady ploch příčných průřezů budeme počítat deformace konstrukce a vnitřní osové síly, resp. napětí na jednotlivých prutech. Jelikož pro aplikace optimalizačních metod bude nutno spočítat mnoho variant kombinací příčných řezů, je vhodné najít pro podobné typy konstrukcí nejrychlejší výpočetní postupy, aby bylo vůbec reálné tyto optimalizační metody použít. Pro hledání vhodných postupů bude použito programu MATLAB a programovacího jazyka C++.

3 Implementace v programu MATLAB

3.1 Možnosti uložení matice

Jelikož hledáme nejrychlejší výpočetní postup, i způsob uložení matice by nám mohl ovlivnit čas výpočtu. Matice se dá uložit jako plná, řídká, diagonální, trojúhelníková a podobně.

Matice tuhosti konstrukce je pásová [Bittnar, 1983], obsahuje tedy mnoho nulových prvků. MATLAB v plné matici ukládá nulu jako ostatní čísla, kdy toto uložení pak zbytečně zabírá mnoho paměti navíc. Plná matice má tedy na každé její pozici zachované číslo, kdežto řídká matice ukládá pouze nenulové prvky a jejich pozice [The MathWorks, 2010a].

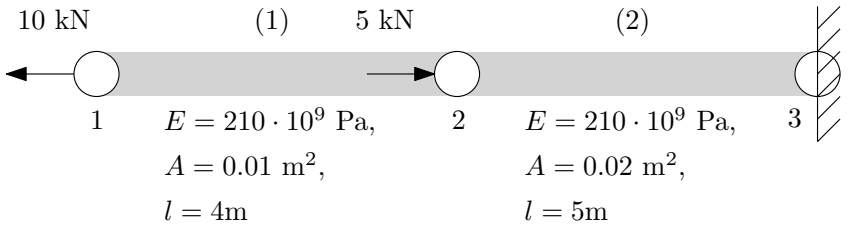
Dále víme, že matice tuhosti konstrukce je symetrická [Patzák, 2009], a proto by další alternativou mohlo být uložení matice jako trojúhelníkové.

3.2 Možnosti sestavení globální matice tuhosti

Program MATLAB však nabízí různé možnosti sestavení globální matice tuhosti. Na obrázku 1 je nakreslena dvouprutová konstrukce, ke které jsou definovány charakteristiky materiálu, topologie i okrajové podmínky. K této úloze budou následně předvedeny kód pro jednotlivé způsoby sestavení globálních matic tuhosti.

3.2.1 Lokalizace přes kódová čísla s adresováním řádků a sloupců

Jako první si ukážeme lokalizaci přes kódová čísla za pomoci adresování řádků a sloupců jednotlivých elementů. Tím nám vypadnou dva for cykly při sestavování. Na začátku definu-



Obrázek 1: Dvouprutová příhradová konstrukce se zadanými okrajovými podmínkami

jeme Youngův modul pružnosti `Emod`, průřezové plochy `A`, začáteční `inode` a koncové `jnode` uzly prutů a x-ové souřadnice uzlů `x`. Jelikož máme konstrukci orientovanou pouze v ose x, není třeba definovat souřadnice y-ové a z-ové. Globální matici tuhosti sestavíme tak, že si vygenerujeme matici nul o rozměru `numnod` × `numnod`, kde `numnod` je délka vektoru `x`. Jelikož MATLAB umí adresovat řádky a sloupce, stačí nám vybrat jednotlivé řádky a sloupce globální matice tuhosti za pomoci `Stiff(n,n)` a na tyto pozice pak přičíst lokální matici tuhosti `k` o stejném rozměru. Nakonec můžeme celou matici pro názornost vypsat pomocí `Stiff`.

Ukázka indexování řádků a sloupců:

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> B=A([1 3],[1 3])
B =
    16     3
     9     6
```

Kód MATLABu: Lokalizace přes kódová čísla s adresováním řádků a sloupců na konstrukci z obrázku 1:

```
% Zadání parametrů konstrukce
   Emod  =[210*10^6 210*10^6];
   Area  =[0.01 0.02];
   inode =[1 2];
   jnode =[2 3];
   x     =[0 4 9];
% Sestavení kódových čísel
   ij=[inode',jnode'];                                    ij = 1   2
                                                               2   3

% Umístění lokální matice tuhosti prutu do globální matice tuhosti konstrukce
   numnod=length(x);                                   numnod = 3
   numelm=length(Area);                                numelm = 2
   Stiff=zeros(numnod);

   for m=1:numelm,
     % Sestavení lokální matice tuhosti jednotlivých prutů
     i=inode(m)
```

```
      j=jnode(m)
      L=abs(x(j)-x(i))
      k=Area(m)*Emod(m)/L*[1,-1;-1,1]

      n=ij(m,:)
      Stiff(n,n)=Stiff(n,n)+ k
   end

% Mezivýsledky for cyklu
 m = 1:                                m = 2:
    i = 1                                 i = 2
    j = 2                                 j = 3
    L = 4                                 L = 5
    k =      525 000  -525 000            k =      840 000  -840 000
            -525 000   525 000                    -840 000   840 000
    n = 1   2                             n = 2   3

    Stiff =  525 000  -525 000  0          Stiff =  525 000  -525 000        0
            -525 000   525 000  0                   -525 000  1 365 000 -840 000
                  0         0  0                         0  -840 000   840 000

% Vypsání matice tuhosti konstrukce
   Stiff
    Stiff =  525 000  -525 000        0
            -525 000  1 365 000 -840 000
                  0  -840 000   840 000
```

3.2.2 Lokalizace přes kódová čísla se třemi for cykly

Dalším způsobem sestavení globální matice tuhosti je lokalizací za pomoci tří do sebe vnořených `for` cyklů. Tento způsob je vhodný pro programovací jazyk C, jelikož C neumí adresovat jednotlivé složky vektoru a matice tak jako MATLAB. Pro porovnání s předchozím způsobem si nyní ukážeme, jak bude takový kód pod MATLABem vypadat. Opět si definujeme Youngův modul pružnosti `Emod`, průřezové plochy `A`, začáteční `inode` a koncové `jnode` uzly prutů a x-ové souřadnice uzlů `x`. Sestavíme si matici kódových čísel `ij` a vygenerujeme matici nul o rozměru `numnod` × `numnod`, do které budeme postupně načítat přes řádky (`for` cyklus s proměnnou `m`) a sloupce (`for` cyklus s proměnnou `j`) jednotlivé prvky lokálních matic tuhosti `k` (`for` cyklus s proměnnou `i`) na pozice určené z matice kódových čísel `ij`. Tímto získáme globální matici tuhosti, kterou si opět můžeme vypsat za pomoci `Stiff`.

```
% Zadání parametrů konstrukce
   Emod  =[210*10^6 210*10^6];
   Area  =[0.01 0.02];
   inode =[1 2];
   jnode =[2 3];
   x     =[0 4 9];

% Sestavení kódových čísel
   ij=[inode',jnode'];                                    ij = 1   2
                                                               2   3

% Umístění lokální matice tuhosti prutu do globální matice tuhosti konstrukce
   numnod=length(x);                                   numnod = 3
   numelm=length(Area);                                numelm = 2
   numdis=2; [1]
```

```
    Stiff=zeros(numnod);
    Length=abs(x(jnode)-x(inode));

    for i=1:numelm
       k=Area(i)*Emod(i)/Length(i)*[1,-1;-1,1]
       for j=1:numdis
          for m=1:numdis
             Stiff(ij(i,j),ij(i,m))=Stiff(ij(i,j),ij(i,m))+k(j,m)
          end
       end
    end

% Mezivýsledky for cyklů
 i = 1:                                        i = 2:
    k =      525 000  -525 000                  k =      840 000  -840 000
            -525 000   525 000                           -840 000   840 000
    j = 1:                                      j = 1:
       m = 1:                                      m = 1:
       Stiff =  525 000         0  0                Stiff =  525 000  -525 000         0
                      0         0  0                         -525 000  1 365 000         0
                      0         0  0                               0         0         0
       m = 2:                                      m = 2:
       Stiff =  525 000  -525 000  0                Stiff =  525 000  -525 000         0
                      0         0  0                         -525 000  1 365 000 -840 000
                      0         0  0                               0         0         0
    j = 2:                                      j = 2:
       m = 1:                                      m = 1:
       Stiff =  525 000  -525 000  0                Stiff =  525 000  -525 000         0
                525 000         0  0                         -525 000  1 365 000 -840 000
                      0         0  0                               0  -840 000         0
       m = 2:                                      m = 2:
       Stiff =  525 000  -525 000  0                Stiff =  525 000  -525 000         0
                525 000  -525 000  0                         -525 000  1 365 000 -840 000
                      0         0  0                               0  -840 000   840 000

% Vypsání matice tuhosti konstrukce
Stiff
    Stiff =  525000  -525000       0
            -525000  1365000 -840000
                  0  -840000  840000
```

3.3 Parametrické vyjádření matice

V kapitole 2 bylo zmíněno, že budeme pravděpodobně nuceni spočítat všechny varianty kombinací průřezových ploch na všech prutech, abychom obdrželi optimální výsledek. Budeme tedy stále dokola sestavovat matici tuhosti konstrukce, počítat neznámé posuny a reakce a následně napětí v prutech nebo vnitřní síly v nich (záleží na zadaných omezujících podmínkách). Pokud budeme sestavovat matici tuhosti číselně, mohlo by to být zbytečně časově náročné. Jelikož má ale MATLAB symbolický toolbox (Symbolic Math Toolbox), který umožňuje sestavit parametrický zápis a poté s ním dále pracovat, můžeme si matici tuhosti, vektor neznámých posunů a vektor napětí pro danou konstrukci sestavit dopředu a pak už jen do tohoto zápisu dosazovat při hlavním běhu optimalizace.

[1] Proměnná `numdis` určuje počet možných posunů a pootočení na prutu.

Pro názornost si můžeme ukázat, jak by aplikace parametrického zápisu mohla vypadat při sestavení matice tuhosti na příkladu dvouprutové konstrukce vyobrazené na obrázku 1. Jako nejvhodnější parametr k použití se jeví tuhost jednotlivých prutů `ki`. Bylo by možné použít například i plochu A.

```
% Zadání parametrů konstrukce
   Area  =[0.01 0.02];[2]
   inode =[1 2];
   jnode =[2 3];f
   x     =[0 4 9];

% Sestavení kódových čísel
   ij=[inode',jnode'];                                        ij = 1   2
                                                                   2   3

% Zavedení parametrů pro výpočet pomocí symbolického toolboxu
   syms k1 k2 real
   ki = [k1 k2];

% Umístění lokální matice tuhosti prutu do globální matice tuhosti konstrukce
   numnod=length(x);                                          numnod = 3
   numelm=length(Area);                                       numelm = 2
   Stiff=sym(zeros(numnod));[3]

   for m=1:numelm,

     % Sestavení lokální matice tuhosti jednotlivých prutů
     i=inode(m)
     j=jnode(m)
     k=ki(m)*[1,-1;-1,1]

     n=ij(m,:)
     Stiff(n,n)=Stiff(n,n)+ k
   end

% Mezivýsledky for cyklu
 m = 1:                                    m = 2:
    i = 1                                     i = 2
    j = 2                                     j = 3
    k =      k1  -k1            k =           k2  -k2
            -k1   k1                         -k2   k2

    Stiff =  k1  -k1  0          Stiff =  k1  -k1       0
            -k1   k1  0                   -k1  k1+k2   -k2
              0    0  0                     0  -k2       k2
```

Nyní máme matici tuhosti v parametrickém zápisu a můžeme ji použít do nového skriptu. Do ní pak dosazujeme ze sady ploch, což pro nás bude jediná proměnná.

```
% Zadání parametrů konstrukce
   Emod = 210*10^6;
   Area  =[0.01 0.02];
   Length=[4 5]; [4]

```

[2] Plochy příčných průřezů jsou potřeba pro výpočet počtu prvků konstrukce.

[3] Symbolicky definujeme nulovou matici.

```
% Výpočet tuhosti prutů
   ki=Emod*Area./Length;

% Matice tuhosti a dosazení do ní
   Stiff = [  ki(1),     -ki(1),        0; ...
             -ki(1),  ki(1)+ki(2),  -ki(2); ...
                 0,     -ki(2),      ki(2)]
```

```
Stiff  =  525000  -525000       0
          -525000  1365000 -840000
                0  -840000  840000
```

3.4 Možnosti řešení soustavy rovnic

Řešení soustavy lineárních rovnic je jedním z největších problémů na poli technických výpočtů [The MathWorks, 2010a]. MATLAB umožňuje počítat tyto soustavy rovnic několika způsoby.

3.4.1 Klasické řešení $X = A \setminus B$

Vezměme soustavu $A \cdot X = B$. Matematicky nejjednodušší řešení je pomocí inverzní matice A^{-1}, kdy dostaneme zápis $X = A^{-1} \cdot B$. Sestavení inverzní matice je ale zdlouhavé a vede k zaokrouhlovacím nepřesnostem. Proto je v prostředí MATLAB daleko snadnější použít operátor zpětného lomítka (backslash operator), kde se inverzní matice nepočítá. Pak dostaneme řešení ze zápisu $X = A \setminus B$ [The MathWorks, 2010a].

Algoritmus zpětného lomítka záleží na typu matic A a B. Pro případ plné symetrické matice A se vektor X spočítá Choleského dekompozicí, pro pásovou symetrickou řídkou matici se pro výpočet vektoru X použije speciální pásový řešič v závislosti na šířce pásu [The MathWorks, 2010c].

3.4.2 LU faktorizace

LU faktorizace rozloží matici A na horní U a dolní L trojúhelníkovou matici tak, že $A = L \cdot U$ [Bubeník et al., 1997].

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} l_{ij} \cdot u_{jk} \quad \text{pro } i = 1, \ldots, k, \tag{1}$$

$$l_{ik} = \frac{1}{u_{kk}} \cdot (a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot u_{jk}) \quad \text{pro } i = k+1, \ldots, n, \tag{2}$$

Jelikož symbolický toolbox nepodporuje funkci `lu`, je nutno pro rozklad matice A na horní U a dolní L trojúhelníkovou matici sestavit kód pomocí `for` a `if` cyklů.

```
   for k=1:m
      if k==1
         for I=1:m
            U(k,I)=KK(k,I);
            L(I,k)=KK(I,k)/U(k,k);
         end
      else
         for I=1:m
```

[4]Konstrukce se nemění, není proto problém použít již předem vypočítané délky.

```
9             U(k,I)=KK(k,I)-sum(L(k,1:k-1)*U(1:k-1,I));
10          end
11          for I=1:m
12             L(I,k)=(KK(I,k)-sum(L(I,1:k-1)*U(1:k-1,k)))/U(k,k);
13          end
14       end
15    end
```

Poté se provede substituce $y = U \cdot x$ a následně se řeší soustavy $y = L^{-1} \cdot b$ a $x = U^{-1} \cdot y$:

$$\begin{aligned} A \cdot x &= b \quad | \quad A = L \cdot U, && (3)\\ L \cdot U \cdot x &= b \quad | \quad y = U \cdot x, && (4)\\ L \cdot y &= b, && (5)\\ y &= L^{-1} \cdot b, && (6)\\ x &= U^{-1} \cdot y. && (7) \end{aligned}$$

Pokud blíže prozkoumáme výše uvedený kód, zjistíme, že pro výpočet prvku `L(I,k)` musíme provést dělení prvkem `U(k,k)`. V parametrickém zápisu tak budou vznikat v obou maticích U i L velice dlouhé zápisy prvků.

3.4.3 Choleského dekompozice

Choleského dekompozice slouží pro rozklad na horní R a dolní trojúhelníkovou matici, kde dolní trojúhelníková matice je transponovaná horní trojúhelníková R^T. Matice A musí být symetrická a pozitivně definitní [5] [Bubeník et al., 1997], [The MathWorks, 2010a].

Jednotlivé prvky při rozkladu z matice A na dolní trojúhelníkovou matici L můžeme vyjádřit dle [Sváček and Feistauer, 2006] jako:

$$\begin{aligned} r_{11} &= \sqrt{a_{11}}, && (8)\\ r_{i1} &= \frac{a_{i1}}{r_{11}} \quad \text{pro } i = 2, \cdots, n, && (9)\\ r_{jj} &= \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2}, \quad \text{pro } j = 2, \cdots, n, && (10)\\ r_{ij} &= \frac{1}{r_{jj}} \cdot (a_{ij} - \sum_{k=1}^{j-1} r_{ik} \cdot r_{jk}) \quad \text{pro } i = j+1, \cdots, n. && (11) \end{aligned}$$

Řešení pak dostaneme z:

$$\begin{aligned} A \cdot x &= b \quad | \quad A = R^T \cdot R, && (12)\\ R^T \cdot R \cdot x &= b, && (13)\\ x &= R^{-1} \cdot ((R^T)^{-1} \cdot b). && (14) \end{aligned}$$

Dle kapitoly 3.4.1 ale není nutné inverzní matice počítat. V MATLABu můžeme vyjádřit (14) jako:

$$x = R \setminus (R^T \setminus b). \quad (15)$$

[5] Pozitivně definitní matice je taková, která má všechny prvky na diagonále kladné a ostatní prvky v matici nejsou několikanásobně větší.

Jak je vidět, Choleského dekompozice je v podstatě zvláštním případem LU faktorizace, přičemž ukládáme jen horní trojúhelníkovou matici R, což je jistě výhodou. Zde je problémem odmocnina v prvcích matice r_{jj} a dělení v prvcích r_{ij}, stejně jako v LU faktorizaci popsané v 3.4.2.

V programu MATLAB se dá použít pro Choleského dekompozici funkce `chol`.

3.4.4 LDL^T rozklad

Dalším způsobem, jak řešit soustavu rovnic $A \cdot x = b$ je LDL^T rozklad, kde L je dolní trojúhelníková matice a D je diagonální matice. Jednotlivé prvky matic získáme takto:

$$l_{ij} = \frac{1}{d_{jj}} \cdot (a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_{kk} \cdot l_{jk}), \tag{16}$$

$$d_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \cdot d_{kk}. \tag{17}$$

Řešení pak dostaneme podle [Jiroušek, 2006] takto:

$$A \cdot x = b \quad | \quad A = L \cdot D \cdot L^T, \tag{18}$$

$$L \cdot D \cdot L^T \cdot x = b. \tag{19}$$

Použijeme-li pomocný vektor $\bar{b}$, pro který platí $L \cdot \bar{b} = b$, pak

$$L \cdot D \cdot L^T \cdot x = L \cdot \bar{b}, \tag{20}$$

$$D \cdot L^T \cdot x = \bar{b}, \tag{21}$$

$$L^T \cdot x = D^{-1}\bar{b}. \tag{22}$$

Inverzní matice matice diagonální D^{-1} má na nenulových prvcích převrácenou hodnotu prvků matice diagonální D. Jelikož je L^T horní trojúhelníková matice, poslední rovnice soustavy je tedy rovnice o jedné neznámé. Ostatní neznámé získáme zpětnou substitucí.

Symbolický toolbox MATLABu opět funkci `ldl` nepodporuje. Řešení můžeme získat například takto:

```
for j=1:n

    if(j==1)
        for i=j:n
            D(j,j)=K(j,j);
            L(i,j)=K(i,j)/D(j,j);
        end

    elseif(j<n)
        for i=j:n-1
            D(j,j)=K(j,j)-sum((L(j,1:j-1)).*(L(j,1:j-1))*(D(1:j-1,1:j-1)));
            L(i+1,j)=(K(i+1,j)-sum(L(i+1,1:j-1).*L(j,1:j-1)*D(1:j-1,1:j-1)))/D(j,j);
        end

    elseif(j==n)
        D(j,j)=K(j,j)-sum((L(j,1:j-1)).*(L(j,1:j-1))*(D(1:j-1,1:j-1)));
    end
end
```

3.4.5 Metoda sdružených gradientů

Metoda sdružených gradientů je jednou z iteračních metod. To znamená, že nehledáme přesné analytické vyjádření řešení, ale snažíme se iteracemi ke správnému řešení přibližovat. Počet iteračních kroků pak ovlivní přesnost získaného řešení.

Máme soustavu rovnic $A \cdot x = b$, kde A je symetrická, pozitivně definitní matice. Odvození této metody je možno nalézt v mnoha zdrojích, [Ralston, 1978], [Shewchuk, 1994]. My zde uvedeme algoritmus výpočtu dle [Sváček and Feistauer, 2006].

Na počátku volme $d^0 = r^0 = b - A \cdot x^0$ a vektor x^0. Tento vektor může být buď nulový, nebo můžeme využít předpodmínění a zvolit ho tak, abychom dosáhli větší rychlosti řešení. Pokud je však x^0 zvolen špatně, může nám počet iterací vedoucí na správný výsledek zvýšit.

Pro $k = 0, 1, 2, \cdots, n$ pak:

$$\alpha_k = \frac{(r^k)^T \cdot d^k}{(d^k)^T \cdot A \cdot d^k}, \tag{23}$$

$$x^{k+1} = x^k + \alpha_k \cdot d^k, \tag{24}$$

$$r^{k+1} = b - A \cdot x^{k+1}, \tag{25}$$

$$\beta_k = \frac{(r^{k+1})^T \cdot A \cdot d^k}{(d^k)^T \cdot A \cdot d^k}. \tag{26}$$

$$d^{k+1} = r^{k+1} - \beta_k \cdot d^k, \tag{27}$$

Výsledné řešení x^{k+1} získáme z rovnice 24. Hodnota α_k z rovnice 23 vyjadřuje optimální krok ve směru d^k. Směry d^k volíme tak, aby byly A-ortogonální, tzn. aby platilo, že $d_i^T \cdot A \cdot d_j = 0$ pro $i \neq j$, což zaručuje rovnice 27. Reziduum r^{k+1} je určeno vztahem 25 a vyjadřuje, jak daleko jsme od správné hodnoty vektoru b [Shewchuk, 1994]. Volba β_k zaručuje A-ortogonalitu d^{k+1} vůči d^k.

V MATLABu můžeme pro tento způsob výpočtu použít funkci `pcg`. Příkaz pak bude vypadat takto: `X = pcg(A,B,TOL,MAXIT,M1,M2,X0)`, kde `X` je řešení soustavy rovnic (pro nás `Disp`), `A` je matice (pro nás matice tuhosti konstrukce `Stiff`), `B` je pravá strana soustavy rovnic (pro nás vektor zatížení `f`), `TOL` je tolerance výpočtu (lze zadat [] pro výchozí hodnotu 1e-6), `MAXIT` je maximální počet iterací (lze zadat [] pro výchozí hodnotu min(N,20)), `M1, M2, X0` se používá pro předpodmínění (pro nulové hodnoty lze zadat []).

3.5 Kompilace do samostatně spustitelného souboru

Ne každý uživatel, který chce používat námi vytvořené kódy, má nainstalované prostředí MATLAB. Proto vyvinula společnost The MathWorks nástroj zvaný MATLAB Compiler, který umožňuje vytvořit samostatně spustitelné aplikace *.exe nebo knihovny jazyka C a C++. Ke spuštění aplikací mimo prostředí MATLAB pak stačí nainstalovat MCR (MATLAB Compiler Runtime) pomocí volně šiřitelného instalačního souboru MCRinstaller.exe. V takto vytvořených aplikacích ovšem nemůže být kód MATLABu viděn, nedá se tedy ani upravovat. Je-li potřeba cokoliv v aplikaci vytvořené Compilerem změnit, musí se provést editace v původním m-file a znovu ho zkompilovat.

Jsou dvě varianty kompilace souborů. Buď lze použít vestavěného nástroje, nebo provést kompilaci přímo z příkazového řádku v Command Window [The MathWorks, 2010b]. Prvně zmíněná varianta používá nástroj Deployment Tool, který se dá spustit pomocí zápisu příkazu

`deploytool` do příkazového řádku v Command Window. Kompilace probíhá v těchto krocích: Nejdříve je třeba sestavit m-file, který chceme kompilovat, a otestovat jeho funkčnost. Dále je třeba vytvořit samostatnou aplikaci (pomocí tlačítka Build) a k ní přibalit potřebné knihovny (případně i MCRinstaller.exe, pomocí tlačítka Package). Pro ověření funkčnosti zkompilované aplikace je vhodné ji spustit.

Druhou variantou je kompilace za pomoci příkazu `mcc`, která ovšem nenabízí možnost přibalení instalačního souboru MCRinstaller.exe. Bez něj není možné aplikaci mimo prostředí MATLAB spustit, jelikož jsou využívány jeho knihovny. Instalační soubor však lze stáhnout z webových stránek společnosti The MathWorks. Je však zapotřebí uvést verzi kompilátoru, která je používána. Lze zkompilovat buď samostatný m-file pomocí zápisu `mcc -m název_souboru.m`, nebo případně několik souborů, které jsou na sebe vázané, tzn. že jeden m-file volá druhý, například pomocí `function`. Toto se provede pomocí zápisu `mcc -m název_hlavního_souboru.m -a název_ podružného_souboru.m`.

4 Implementace v jazyce C++

Prostředí MATLAB je sice uživatelsky velmi příjemné, neboť obsahuje velké množství toolboxů a obsáhlou nápovědu, používá však interpretovaný jazyk a ten by měl být pomalejší než jazyk kompilovaný. V interpretovaném jazyce se překládá každá řádka za běhu skriptu, v kompilovaném jazyce se kontrola správnosti syntaxe a překlad dějí ještě před spuštěním. Kompilovaný jazyk je zase náročnější na správnou syntaxi zápisu a inicializaci všech proměnných před jejich následným užitím.

Pro editování kódů byl použit editor Eclipse [CDT Community, 2010], pro překlad kompilátor MinGW [MinGW team, 2010]. Jak editor, tak překladač jsou poskytovány jako freeware.

MATLAB nabízí převod symbolického kódu do syntaxe C/C++ pomocí příkazu `ccode`. Tento příkaz zároveň zajistí přeindexování jednotlivých prvků pole z 1 až n prvků na 0 až (n-1) prvků. To je značná výhoda, neboť odpadá zdlouhavé přepisování, jsou-li již kódy pro MATLAB hotové. I my jsme tohoto příkazu použili pro převod jednotlivých parametricky vyjádřitelných částí skriptů. Stejně tak jako v MATLABu, i zde je nutné zamyslet se nad řešením soustavy rovnic `Stiff*Disp=f`. Řešič pro plné vyjádření matice tuhosti byl převzat od doc. Kruise (LDL^T rozklad) [Jaroslav Kruis a kolektiv, 2010]. Dalším způsobem vyřešení soustavy rovnic je použití volně dostupné knihovny LAPACK++ (zde použita LU faktorizace). Posledním v práci použitým prostředkem je řešič ing. Vondráčka FEMCad [Vondráček, 2008a].

LAPACK++ (Linear Algebra PACKage in C++) je rozšíření knihovny LAPACKu do C++ [Dongarra et al., 1996]. LAPACK je napsaný v jazyce Fortran90 a dají se s ním řešit běžné úlohy numerické lineární algebry, např. řešení soustav lineárních rovnic, problém vlastních čísel a podobně [Anderson et al., 1999]. LAPACK je uzpůsoben k tomu, aby prováděl výpočty rychle a efektivně. Je to standard, který využívají i mnohé komerční programy, jako je MATLAB nebo Mathematica.

Pro práci s knihovnou LAPACK++ je nutno si uvědomit, že matici nebudeme zadávat jako prvky pole v zápisu jazyka C++, ale podobně jako prvky pole v jazyce Fortran. Nejprve je nutno si celou matici deklarovat. Pro plnou matici je možno využít zápisu, který nám vytvoří objekt `A`, `LaGenMatDouble A(M,N)`, kde `A` je název matice a M × N je její rozměr. Jednotlivé prvky matice se pak načítají pomocí `A(i,j)`, rozsah `(i,j)` je $0 \leq i < M$ a $0 \leq j < N$.

Jednotlivé prvky se tedy neindexují podle standardu jazyka Fortran, ale podle jazyka C. Vektor se může deklarovat pomocí `LaVectorDouble x(N)`. Po deklaraci matice a vektorů a definici známé matice `A` a vektoru pravé strany `b` lze použít funkci `LaLinearSolve(A,x,b);` k vyřešení neznámého vektoru řešení `x` [Dongarra et al., 1996].

Pro názornost uveďme malý příklad.

```
#include <lapackpp.h>
int main() {
    int N = 2;
    LaGenMatDouble A(N, N);
    LaVectorDouble x(N), b(N);

    A(0, 0) = 1.0;
    A(0, 1) = 2.0;
    A(1, 0) = 3.0;
    A(1, 1) = 4.0;
    b(0) = 17;
    b(1) = 39;

    LaLinearSolve(A,x,b);
    return 0;
}
```

Pokud chceme na terminálu vidět vstupy a výstupy, není problém vše vypsat například pomocí `for cyklů` a `printf()`.

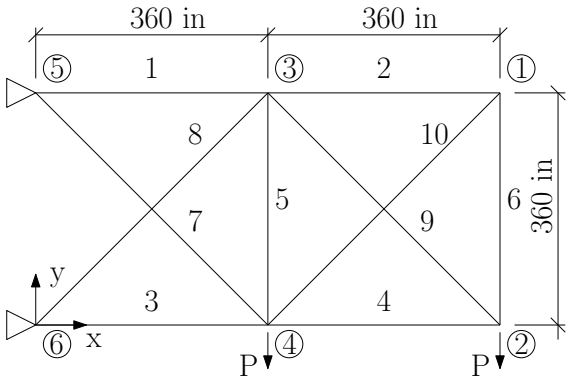
FEMCad [Vondráček, 2008a] je řešič soustavy lineárních rovnic pro řídké matice vyvinutý ing. Vondráčkem. Po zadání matice se nejprve provede analýza jejího typu, aby se mohlo provést optimální přeuspořádání do vhodného tvaru a mohl se vybrat vhodný řešič. Jelikož je matice tuhosti symetrická, stačí uložit do paměti jen její polovinu. Vhodné je též použít metodu kompresovaných řádků, kde uložíme hodnoty jednotlivých prvků dolní trojúhelníkové matice, jejich sloupcové indexy a pozici prvku ve vektoru hodnot, kde začíná nový řádek matice [Vondráček, 2008b].

5 Použité benchmarky

K testování výpočetních metod bylo zapotřebí vybrat několik vhodných modelů konstrukcí. Byla snaha vybrat takové modely, které už dříve byly spočítány a ke kterým existují výsledky k porovnání přesnosti. Proto jsme zvolili čtyři nejvíce používané. Jedná se o desetiprutovou příhradovou konzolu ve 2D, dvacetipětiprutovou příhradovou věž ve 3D, padesátidvouprutovou příhradovou konstrukci ve 2D a sedmdesátidvouprutovou příhradovou konstrukci ve 3D. Tyto konstrukce jsou používány i v [Lemonge and Barbosa, 2003].

5.1 Desetiprutová příhradová 2D konzola

Tato konstrukce byla poprvé zveřejněna Venkayyou v článku [Venkayya, 1971]. Konstrukce byla řešena spojitou rozměrovou optimalizací. Prvního případu diskrétní rozměrové optimalizace se nám bohužel dopátrat nepodařilo, ale nejstarší článek, který máme k dispozici a který tuto konstrukci zmiňuje, je [Cai and Thierauf, 1996]. Použité charakteristiky materiálu konstrukce, omezující podmínky a zatížení konstrukce jsou v tabulce 1, konstrukce



Obrázek 2: 10-prutová příhradová 2D konzola

Materiál:	hliník
Objemová hmotnost:	0,1 lb/in^3
Youngův modul pružnosti E:	10^7 psi
Limitní napětí:	25 000 psi
Limitní posun:	2 in
Zatížení P:	100 kips

Tabulka 1: Charakteristiky materiálu a omezující a okrajové podmínky 10-prutové konzoly

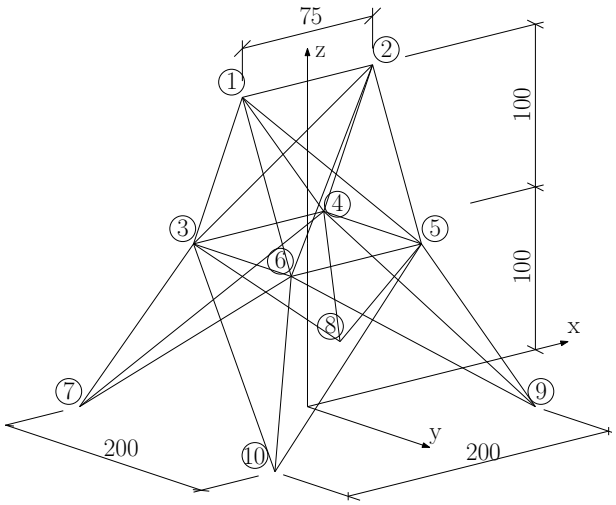
je vyobrazena na obrázku 2. Sada testovacích ploch příčných průřezů obsahuje tyto hodnoty (v in^2): 1,62, 1,80, 1,99, 2,13, 2,38, 2,62, 2,63, 2,88, 2,83, 3,09, 3,13, 3,38, 3,47, 3,55, 3,63, 3,84, 3,87, 3,88, 4,18, 4,22, 4,49, 4,59, 4,80, 4,97, 5,12, 5,74, 7,22, 7,97, 11,50, 13,50, 13,90, 14,20, 15,50, 16,00, 16,90, 18,80, 19,90, 22,00, 22,90, 26,50, 30,00, 33,50. Tato vstupní data byla převzata z [Lemonge and Barbosa, 2003]. Chtěli jsme dosáhnout co nejpřesnějšího porovnání, proto jsme záměrně nechali tyto hodnoty v původních anglosaských jednotkách. Přehled všech těchto jednotek včetně převodu do SI soustavy naleznete v příloze A.

5.2 Dvacetipětiprutová příhradová 3D věž

Dle [Venkayya, 1971] byla tato konstrukce poprvé zveřejněna Foxem a Schmitem ve článku [Fox and Schmit, 1966]. Nejstarší námi objevený článek uveřejňující tuto konstrukci s diskrétními proměnnými je z roku 1995 [Wu and Chow, 1995a]. Konstrukce je znázorněna na obrázku 3, charakteristiky materiálu jsou uvedené v tabulce 3 a zatížení konstrukce naleznete v tabulce 4. Jednotlivé plochy příčných průřezů byly vybírány z této množiny (v in^2): 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,9, 1,0, 1,1, 1,2, 1,3, 1,4, 1,5, 1,6, 1,7, 1,8, 1,9, 2,0, 2,1, 2,2, 2,3, 2,4, 2,5, 2,6, 2,8, 3,0, 3,2, 3,4 uveřejněné ve článku [Wu and Chow, 1995b]. Tato konstrukce je symetrická, proto byly jednotlivé pruty sdruženy do skupin, které můžete nalézt v tabulce 2.

5.3 Padesátidvouprutová příhradová 2D konstrukce

Nejstarší článek, který máme k dispozici a který tuto konstrukci zkoumá a popisuje, je [Wu and Chow, 1995b]. Sada ploch příčných průřezů prutů byla pro výpočet použita dle [Giger and Ermanni, 2006]. Jedná se o tuto množinu: 71,613, 90,968, 126,451, 161,290, 198,064,



Obrázek 3: 25-prutová příhradová 3D konstrukce věže

Skupina	Připojení prutů k uzlům
A_1	1-2
A_2	1-4, 2-3, 1-5, 2-6
A_3	2-5, 2-4, 1-3, 1-6
A_4	3-6, 4-5
A_5	3-4, 5-6
A_6	3-10, 6-7, 4-9, 5-8
A_7	3-8, 4-7, 6-9, 5-10
A_8	3-7, 4-8, 5-9, 6-10

Tabulka 2: Sdružení ploch příčných průřezů do skupin (25-prutová konstrukce)

Materiál:	hliník
Objemová hmotnost:	0,1 lb/in^3
Youngův modul pružnosti E:	10^7 psi
Limitní napětí:	40 000 psi
Limitní posun:	2 in

Tabulka 3: Charakteristiky materiálu a omezující podmínky 25-prutové věže

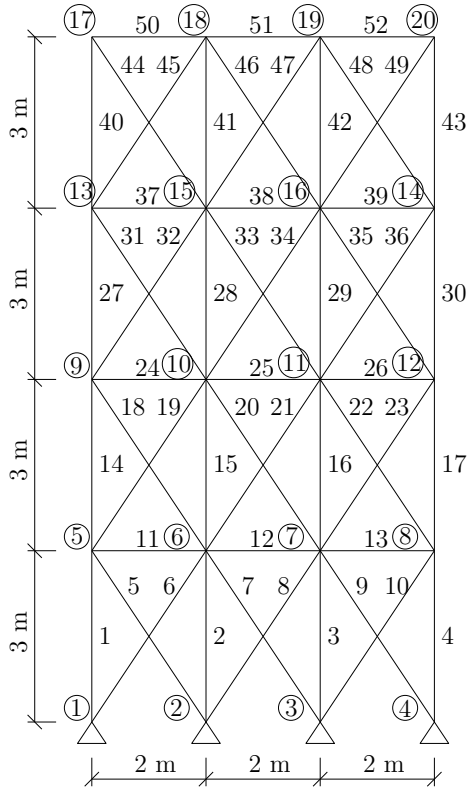
Uzel	F_x	F_y	F_z
1	1,0	-10,0	-10,0
2	0	-10,0	-10,0
3	0,5	0	0
6	0,6	0	0

Tabulka 4: Zatížení 25-prutové konstrukce v kip jednotkách

252,258, 285,161, 363,225, 388,386, 494,193, 506,451, 641,289, 645,160, 792,256, 816,773, 940,000, 1 008,385, 1 045,159, 1 161,288, 1 283,868, 1 374,191, 1 535,481, 1 690,319, 1 696,771, 1 858,061, 1 890,319, 1 993,544, 2 019,351, 2 180,641, 2 238,705, 2 290,318, 2 341,191, 2 477,414, 2 496,769, 2 503,221, 2 696,769, 2 722,575, 2 896,768, 2 961,284, 3 096,768, 3 206,445, 3 303,219, 3 703,218, 4 658,055, 5 141,925, 5 503,215, 5 999,998, 6 999,986, 7 419,340, 8 709,660, 8 967,724, 9 161,272, 9 999,980, 10 322,560, 10 903,204, 12 129,008, 12 838,684, 14 193,520, 14 774,164, 15 806,420, 17 096,740, 18 064,480, 19 354,800, 21 612,860 (v mm^2). Konstrukce je vyobrazena na obrázku 4, charakteristiky materiálu a omezující podmínky jsou uvedeny v tabulce 6 a zatížení naleznete v tabulce 7. Konstrukce je opět symetrická, proto jsou plochy sdruženy do skupin, které jsou uvedeny v tabulce 5 [Lemonge and Barbosa, 2003].

5.4 Sedmdesátidvouprutová příhradová 3D konstrukce

Konstrukce na obrázku 5 byla poprvé uveřejněna dle [Venkayya, 1971] Venkayyou, Knotem a Reddym v roce 1968. Jako u 10-prutové konstrukce je však řešena spojitou rozměrovou optimalizací. V článku [Wu and Chow, 1995b] je tato konstrukce diskrétní optimalizací řešena. Zároveň je to nejstarší článek, ve kterém jsme tuto konstrukci objevili. Jednotlivé plochy příčných průřezů byly vybírány z této množiny (v in^2): 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8,



Obrázek 4: 52-prutová příhradová 2D konstrukce

A_1	1, 2, 3, 4
A_2	5, 6, 7, 8, 9, 10
A_3	11, 12, 13
A_4	14, 15, 16, 17
A_5	18, 19, 20, 21, 22, 23
A_6	24, 25, 26
A_7	27, 28, 29, 30
A_8	31, 32, 33, 34, 35, 36
A_9	37, 38, 39
A_{10}	40, 41, 42, 43
A_{11}	44, 45, 46, 47, 48, 49
A_{12}	50, 51, 52

Tabulka 5: Sdružení ploch příčných průřezů do skupin (52-prutová konstrukce)

Objemová hmotnost:	7 860 kg/m^3
Modul pružnosti E:	2,07 $\times 10^5$ MPa
Limitní napětí:	180 MPa

Tabulka 6: Charakteristiky materiálu a omezující podmínky 52-prutové věže

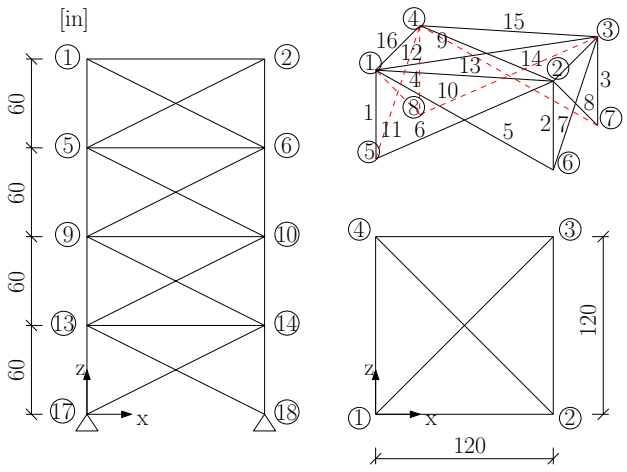
Uzel	F_x	F_y
17	100,0	200,0
18	100,0	200,0
19	100,0	200,0
20	100,0	200,0

Tabulka 7: Zatížení 52-prutové konstrukce v kN

0,9, 1,0, 1,1, 1,2, 1,3, 1,4, 1,5, 1,6, 1,7, 1,8, 1,9, 2,0, 2,1, 2,2, 2,3, 2,4, 2,5, 2,6, 2,7, 2,8, 2,9, 3,0, 3,1, 3,2 viz [Wu and Chow, 1995b]. Materiálové charakteristiky a omezující podmínky (vizte tabulku 10) a zatížení (vizte tabulku 8) byly převzaty z [Lemonge and Barbosa, 2003]. Plochy byly opět sdruženy do skupin z důvodu symetrie konstrukce, vizte tabulku 9.

Uzel	F_x	F_y	F_z
1	5	5	-5

Tabulka 8: Zatížení 72-prutové konstrukce [kip]



Obrázek 5: 72-prutová příhradová 3D konstrukce

Skupina	Pruty
A_1	1, 2, 3, 4
A_2	5, 6, 7, 8, 9, 10, 11, 12
A_3	13, 14, 15, 16
A_4	17, 18
A_5	19, 20, 21, 22
A_6	23, 24, 25, 26, 27, 28, 29, 30
A_7	31, 32, 33, 34
A_8	35, 36
A_9	37, 38, 39, 40
A_{10}	41, 42, 43, 44, 45, 46, 47, 48
A_{11}	49, 50, 51, 52
A_{12}	53, 54
A_{13}	55, 56, 57, 58
A_{14}	59, 60, 61, 62, 63, 64, 65, 66
A_{15}	67, 68, 69, 70
A_{16}	71, 72

Tabulka 9: Sdružení ploch průřezů do skupin (72-prutová konstrukce)

Objemová hmotnost:	0,1 lb/in^3
Youngův modul pružnosti E:	10^7 psi
Limitní napětí:	25 ksi
Limitní posun:	0,25 in

Tabulka 10: Charakteristiky materiálu a omezující podmínky 72-prutové věže

6 Testovací metody a jejich výsledky

6.1 MATLAB a m-files

Při tvorbě prvních kódů pro tuto práci v programu MATLAB byl použit jako referenční kód A. Rapoffa [Rapoff, 2006]. Jeho kód byl sestaven pro sedmiprutovou příhradovou konstrukci s pěti styčníky. Pro naše výzkumy ale tato konstrukce použita nebyla, jelikož není tak běžná v dostupné zahraniční literatuře jako námi zmíněné konstrukce v kapitole 5. Po úpravě kódu bylo nutno zjistit, které operace zabírají nejvíce času.

V tabulkách 11 a 12 jsou uvedeny časy jednotlivých operací při spuštění výpočtu jednou a tisíckrát. Jak je vidět, je nutno porovnat různé způsoby výpočtu soustavy rovnic $K \cdot r = f$. Operace `Disp=s\f;` pro výpočet neznámých posunů zabírá pro jedno spuštění výpočtu nejvíce času. Pokud spustíme výpočet tisíckrát, zjistíme, že nejvíce času zabírá sestavení matice tuhosti konstrukce. Pokud neměníme tvar konstrukce, ale pouze příčné průřezové plochy jednotlivých prutů, respektive jejich tuhosti, můžeme si matici tuhosti konstrukce vyjádřit parametricky v závislosti na tuhosti prvků dle pododdílu 3.3. Sestavení vektoru pravé strany zabírá též zbytečně mnoho paměti. Je proto vhodné sestavit tento vektor jinak.

řádek kódu	výpis kódu	počet volání	čas v s	čas v %
102	`Disp=s\f;`	1	0,006	40,0%
70	`Stiff=zeros(2*numnod);`	1	0,005	33,3%
ostatní			0,004	26,7%
celkem			0,015	100%

Tabulka 11: Výstup z Profileru MATLABu pro jedno spuštění kódu přímého řešiče pro desetiprutovou konstrukci

řádek kódu	výpis kódu	počet volání	čas v s	čas v %
80	`k=Area(m)*Emod(m)/Length(m)*[T...`	10 000	0,065	15,2%
64	`ij=[2*inode'-1,2*inode',2*jnod...`	1 000	0,046	10,7%
79	`T=[[cc,cs];[cs,ss]];`	10 000	0,040	9,3%
84	`Stiff(n,n)=Stiff(n,n)`_k;+	10 000	0,038	8,9%
93	`f([2*idfx'-1;2*idfy'])=[fx';fy...`	1 000	0,029	6,8%
ostatní			0,210	49,1%
celkem			0,428	100%

Tabulka 12: Výstup z Profileru MATLABu pro 1 000 spuštění kódu přímého řešiče pro 10-prutovou konstrukci

6.1.1 Parametrické vyjádření skriptu

Pro ušetření času bylo vhodné veškeré části skriptu, které se při běhu sestavují či počítají, eliminovat. Toho lze dosáhnout jednak zapsáním vektorů (např. délky nebo zatížení) přímo a jednak parametrickým vyjádřením všech početních a sestavovacích operací (např. sestavení matice tuhosti, výpočtu posunů nebo výpočtu vnitřních sil) tak, aby se v závislosti na parametru tuhosti prutů, mohly dosazovat jen jejich jednotlivé příčné průřezové plochy.

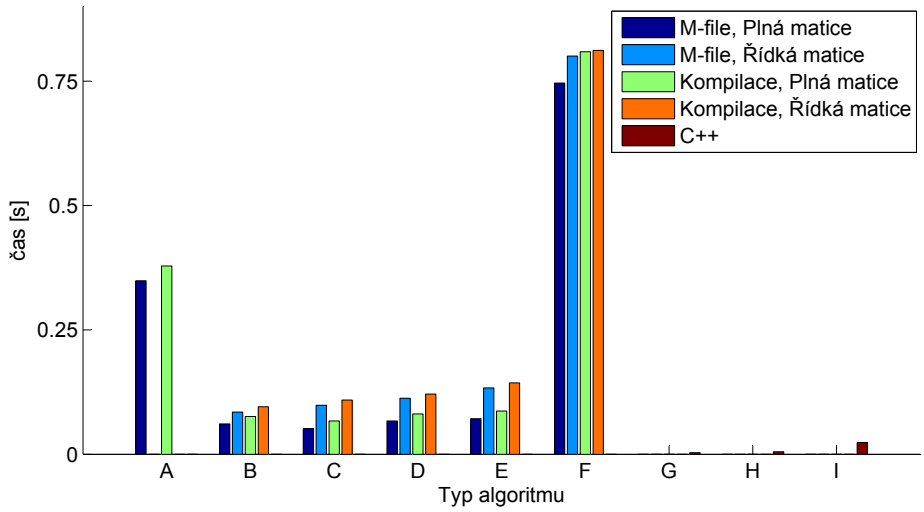
Sestavení matice tuhosti konstrukce nebyl problém. Matice se vyjádřila (vizte pododdíl 3.3) a „ořezala“ o řádky a sloupce se známými (nulovými) posuny v podporách. Vyjádření vektoru posunů v parametrickém zápisu MATLAB nebyl schopen. Například pro výpočetně nejméně náročný LDL^T rozklad popsaný v pododdíle 3.4.4 se čas 6. iterace oproti předchozí iteraci zvýšil zhruba 1 200krát (pro 10-prutovou konstrukci). Proto byl zvolen druhý parametr `Disp`, který tyto posuny vyjadřuje, a s ním pak opět nebyl problém sestavit vektor vnitřních sil v konstrukci.

6.1.2 Plná a řídká matice

Jelikož matice tuhosti konstrukce je řídká, zkoumali jsme i vliv této vlastnosti na výsledný čas. Skripty všech dříve zmiňovaných konstrukcí v kapitole 5 jsou tedy vyjádřeny jak pro plnou, tak pro řídkou matici tuhosti, abychom mohli výsledné časy porovnat. Bylo zjištěno, že záleží na velikosti této matice a poměru nulových a nenulových prvků. Pro malé konstrukce, jako je např. 10-prutová, se vyplatí použít matice plná, pro větší a složitější konstrukce, s větším počtem nulových prvků v této matici, pak matice řídká. Toto je vidět ve velkém kontrastu u 25-prutové a 72-prutové konstrukce v tabulkách 14 a 16.

		M-file, plná matice	M-file, řídká matice	Kompilace, plná matice	Kompilace, řídká matice	C++
A	Sestavení matice za běhu	0,3489		0,3785		
B	`K * r = f`: Zpětné lomítko	0,0610	0,0848	0,0761	0,0956	
C	`K * r = f`: LU faktorizace	0,0516	0,0986	0,0669	0,1091	
D	`K * r = f`: Chol. dekompozice	0,0669	0,1127	0,0811	0,1210	
E	`K * r = f`: LDL^T rozklad	0,0716	0,1334	0,0869	0,1435	
F	`K * r = f`: Metoda s. gradientů	0,7461	0,8004	0,8090	0,8118	
G	`K * r = f`: LDL^T rozklad					**0,0033**
H	`K * r = f`: LU faktorizace					0,0051
I	FemCAD					0,0236

Tabulka 13: Výsledky časů v sekundách na 10-prutové konstrukci



Obrázek 6: Graf porovnávající jednotlivé časy na 10-prutové konstrukci, legenda vizte tabulku 13

6.1.3 Řešiče soustavy rovnic `Stiff*Disp=f`

K řešení soustavy rovnic `Stiff*Disp=f` bylo použito pět způsobů řešení. Metoda zpětného lomítka popsaná v pododdíle 3.4.1 by teoreticky měla být nejrychlejší, jelikož je celý algoritmus výpočtu optimalizován společností MathWorks. Nicméně tomu tak není. Dalo by se předpokládat, že se čas ztrácí na výběru typu matice (řídká, plná, symetrická apod.) a následném výběru vhodného způsobu výpočtu.

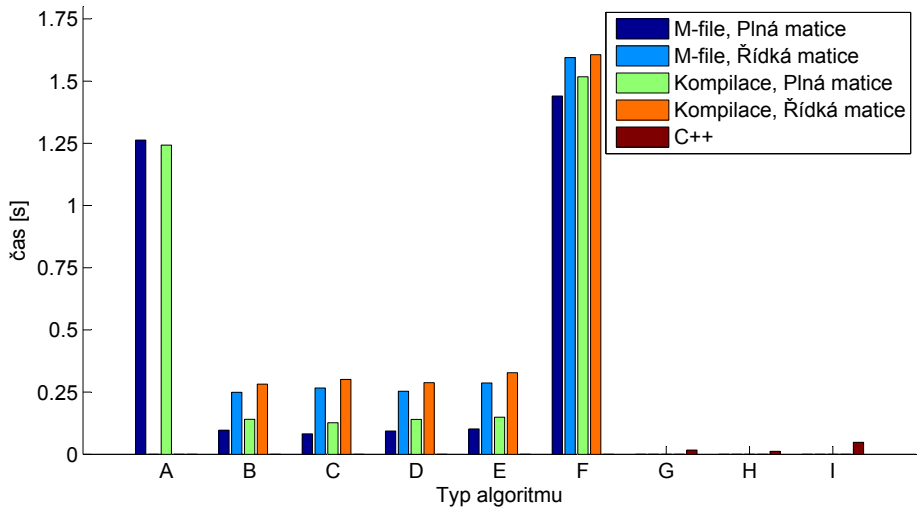
Jako další řešič byla použita LU faktorizace charakterizovaná v pododdíle 3.4.2. Způsob tohoto výpočtu je vhodný pro malé konstrukce, které mají málo nulových prvků v matici tuhosti, hodí se pro ně tedy plná matice tuhosti. Vizte tabulku 13 a 14 pro deseti a dvacetipětiprutovou konstrukci.

Choleského dekompozice popsaná v pododdíle 3.4.3 by měla být teoreticky rychlejší, jelikož matice tuhosti konstrukce je symetrická. Uplatňuje se ve výpočtech tam, kde se již vyplatí použít řídkou matici tuhosti (tzn. pro 72-prutovou konstrukci, vizte tabulku 16).

LDL^T rozklad objasněný v pododdíle 3.4.4 byl shledán jako pomalejší než LU faktorizace

		M-file, plná matice	M-file, řídká matice	Kompilace, plná matice	Kompilace, řídká matice	C++
A	Sestavení matice za běhu	1,2632		1,2428		
B	`K * r = f`: Zpětné lomítko	0,0970	0,2492	0,1406	0,2822	
C	`K * r = f`: LU faktorizace	0,0824	0,2664	0,1267	0,3010	
D	`K * r = f`: Chol. dekompozice	0,0936	0,2536	0,1403	0,2881	
E	`K * r = f`: LDL^T rozklad	0,1019	0,2865	0,1491	0,3279	
F	`K * r = f`: Metoda s. gradientů	1,4402	1,5945	1,5176	1,6063	
G	`K * r = f`: LDL^T rozklad					0,0171
H	`K * r = f`: LU faktorizace					**0,0118**
I	FemCAD					0,0480

Tabulka 14: Výsledky časů v sekundách na 25-prutové konstrukci



Obrázek 7: Graf porovnávající jednotlivé časy na 25-prutové konstrukci, legenda vizte tabulku 14

a Choleského dekompozice u řídké i u plné matice tuhosti konstrukce.

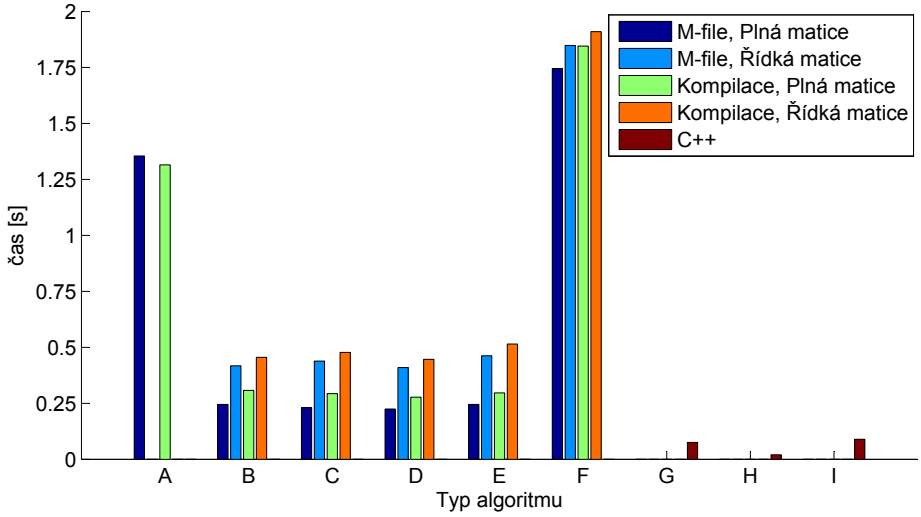
Čas výpočtu při použití metody konjugovaných gradientů je závislý na počtu iterací. Čím je méně iterací a tedy větší chyba v řešení rovnice `Stiff*Disp=f`, tím rychlejší je výpočet. Na obrázku 10 je graf závislosti počtu iterací na chybě v řešení. Pokud označíme vypočtené posuny přímým řešičem (např. `Disp=Stiff\f`) u_{orig} a posuny vypočtené metodou konjugovaných gradientů $u_{new,i}$, můžeme dostat maximální odchylku v řešení mezi přímou a iterační metodou pomocí $|u_{orig} - u_{new,i}|$. Popisky jednotlivých bodů v grafu znázorňují čas výpočtu při spuštění celého skriptu 1000krát v sekundách.

6.2 MATLAB a kompilace

Původní předpoklad, že kompilace MATLAB kódu do samostatně spustitelné aplikace znatelně urychlí čas výpočtu, se nepotvrdil. Kompilace byla provedena podle pododdílu 3.5. Důvodem je nejspíše dlouhá inicializace aplikace. Výsledky jsou vidět v tabulkách 13 až 16.

		M-file, plná matice	M-file, řídká matice	Kompilace, plná matice	Kompilace, řídká matice	C++
A	Sestavení matice za běhu	1,3548		1,3151		
B	`K * r = f`: Zpětné lomítko	0,245	0,4174	0,308	0,4556	
C	`K * r = f`: LU faktorizace	0,2312	0,4387	0,2933	0,4774	
D	`K * r = f`: Chol. dekompozice	0,2245	0,4098	0,2776	0,4466	
E	`K * r = f`: LDL^T rozklad	0,2453	0,4624	0,2964	0,5147	
F	`K * r = f`: Metoda s. gradientů	1,7452	1,8485	1,8457	1,9099	
G	`K * r = f`: LDL^T rozklad					0,0757
H	`K * r = f`: LU faktorizace					**0,0201**
I	FemCAD					0,0895

Tabulka 15: Výsledky časů v sekundách na 52-prutové konstrukci



Obrázek 8: Graf porovnávající jednotlivé časy na 52-prutové konstrukci, legenda vizte tabulku 15

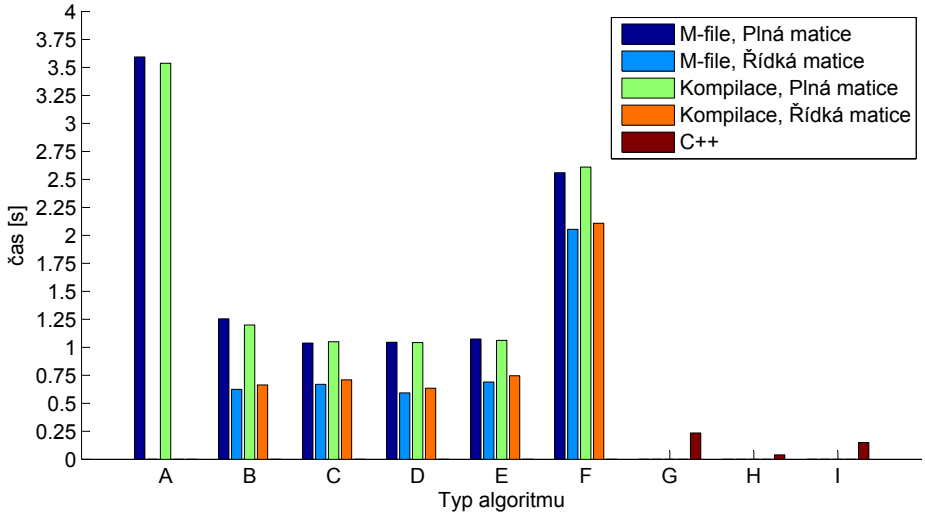
6.3 C++

Všechny kódy v jazyce C++ vycházejí z kódů MATLABu. Náhodný generátor čísel z množiny ploch vybere plochy příčných průřezů prutu v jejich potřebném počtu. Jsou definovány tuhosti `k` a k nim přiřazeny jejich hodnoty s parametrem plochy `A`. Pokud v původním skriptu v MATLABu obsahovaly jednotlivé tuhosti prvky s odmocninami, použil se v něm příkaz `vpa` k převedení reálného čísla na číslo s desetinným rozvojem. Použitá přesnost na 29 platných číslic by neměla ovlivnit hodnotu výsledného řešení. Matice tuhosti `K` byla převzata z kódu MATLABu pomocí příkazu `ccode`. Dále bylo použito jednorozměrného pole pro vektor zatížení `f`. K řešení soustavy rovnic pro získání neznámých posunů `u` bylo využito tří již zmíněných řešičů v kapitole 4. Po výpočtu posunů `u` se vypočítají jednotlivá napětí v prutech `S` v závislosti na tuhostech `k`, posunech `u` a plochách `A`. Nakonec jsou nalezeny maximální hodnoty posunů `u` a napětí `S` a spočítána celková váha konstrukce `maxw`, která je opět vyjádřena parametricky ze skriptu MATLABu.

Pro všechny tři řešiče bude odlišný zápis matice tuhosti konstrukce `K`, vektoru zatížení `f`

		M-file, plná matice	M-file, řídká matice	Kompilace, plná matice	Kompilace, řídká matice	C++
A	Sestavení matice za běhu	3,5938		3,5380		
B	`K * r = f`: Zpětné lomítko	1,2550	0,6246	1,1997	0,6639	
C	`K * r = f`: LU faktorizace	1,0383	0,6696	1,0506	0,7101	
D	`K * r = f`: Chol. dekompozice	1,0459	0,5931	1,0424	0,6348	
E	`K * r = f`: LDL^T rozklad	1,0749	0,6904	1,0622	0,7460	
F	`K * r = f`: Metoda s. gradientů	2,5600	2,0546	2,6101	2,1085	
G	`K * r = f`: LDL^T rozklad					0,2348
H	`K * r = f`: LU faktorizace					**0,0395**
I	FemCAD					0,1494

Tabulka 16: Výsledky časů v sekundách na 72-prutové konstrukci



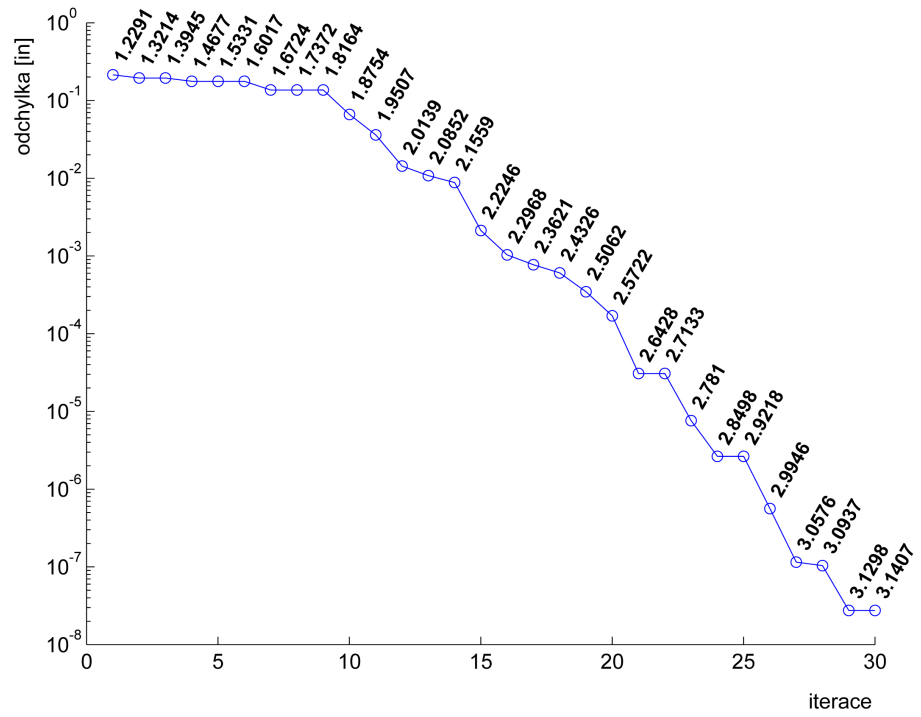
Obrázek 9: Graf porovnávající jednotlivé časy na 72-prutové konstrukci, legenda vizte tabulku 16

a výstupních posunů jednotlivých uzlů `u`. Většina nutných úprav byla zmíněna v kapitole 4.

7 Závěr

Při použití všech zamýšlených metod na zvolených konstrukcích můžeme dospět k závěru, že přestože je MATLAB v dnešní době velmi rozvíjený, stále se jedná o interpretovaný jazyk a je tedy pomalejší než jazyk C++ jakožto jazyk kompilovaný. Ani s pomocí kompilace se nám nepodařilo zvýšit jeho rychlost. Nicméně MATLAB je vhodné prostředí k testování nových metod a k analyzování chování skriptu a třeba i konstrukce vůbec. Nabízí totiž nejen vestavěné funkce, které není třeba algoritmizovat a optimalizovat, a k nim i obsáhlou nápovědu, ale i grafické rozhraní a různé nástroje pro analýzu skriptů.

Použité matematické metody jsou závislé na velikosti konstrukce a její složitosti. Čím je konstrukce větší a složitější, tím je matice tuhosti konstrukce více řídká. Pro malé konstrukce se ale vyplatí použít matice plná. Na typu uložení matice pak závisí volba vhodné matematické metody. Pro plnou malou matici je vhodné použít LU faktorizaci, pro řídkou matici zase



Obrázek 10: Závislost iterace na chybě řešení s časem výpočtu pro 1000 spuštění v sekundách pro 72-prutovou konstrukci

Choleského dekompozici. Metoda sdružených gradientů se nám neosvědčila, předpokládáme, že je vhodná pro daleko větší konstrukce s řidšími maticemi tuhosti, než byly užity v této práci.

Dalším typem uložení matice tuhosti je například pásová matice nebo za pomoci kompresovaných řádků, sloupců nebo jejich kombinace [Vondráček, 2008a]. Do budoucna hodláme prozkoumat i tyto varianty.

Do úvahy též připadají další konstrukce pro prozkoumání efektivnosti i ostatních metod, a to nejen větší příhradové konstrukce, ale i rámové konstrukce apod. Díky optimalizaci skriptů budeme schopni do budoucna spustit například metodu větví a mezí, která se použije pro diskrétní rozměrovou optimalizaci. Tato metoda je totiž velmi výpočetně náročná, přestože použijeme horní odhad (z publikovaných článků) i dolní odhad (získaný spojitou rozměrovou optimalizací). Pokud bychom tyto odhady nepoužili, počet řešení by u nejmenší konstrukce trval přibližně 1787 let při použití nejrychlejší výpočetní metody (zde LTL^T rozkladu), vizte tabulku 17. Pro názornost jsou pak v téže tabulce ukázány počty řešení a jejich časová náročnost pro metodu větví a mezí, provedou-li 2 iterace nahoru a dvě dolů, kde se pohybujeme například kolem předem zvoleného řešení.

Jelikož tato metoda nebude možná spustit pouze na jednom jádře jednoho počítače, jako se tomu dělo v této práci, přichází v úvahu nejen paralelizace na jednotlivá jádra procesoru, ale i použití počítačového clusteru. V dnešní době se rozvíjí i paralelní výpočty prováděné na grafických kartách, které zvládnou jednoduché operace. Zmiňme na příklad program CUDA od NVIDIA [NVIDIA, 2011]. Bude proto vhodné podrobit analýze i tento typ výpočtu.

Počet prutů	10	25	52	72
Počet skupin prutů	10	8	12	16
Počet prvků v množině	42	32	64	21
Počet možných zadání	$42^{10} = 1,7 \cdot 10^{16}$	$32^{8} = 1,1 \cdot 10^{12}$	$64^{12} = 4,7 \cdot 10^{21}$	$32^{16} = 1,2 \cdot 10^{24}$
Počet zadání - B&B[6]	$42^{4} = 3,1 \cdot 10^{6}$	$32^{4} = 1,1 \cdot 10^{6}$	$64^{4} = 1,7 \cdot 10^{7}$	$32^{4} = 1,1 \cdot 10^{6}$
Doba výpočtu (1 000x)	0,0033 s	0,0118 s	0,0201 s	0,0395 s
Doba výpočtu na 1 jádře	1787,3 let	0,41 let	$3,01 \cdot 10^{9}$ let	$1,51 \cdot 10^{12}$ let
Doba výpočtu (B&B)	10,269 s	12,373 s	337,222 s	41,419 s

Tabulka 17: Diskuse počtu řešení a doby trvání

Reference

[Anderson et al., 1999] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, třetí edition.

[Bittnar, 1983] Bittnar, Z. (1983). *Metody numerické analýzy konstrukcí*. Skriptum. České vysoké učení technické v Praze.

[Bubeník et al., 1997] Bubeník, F., Pultar, M., and Pultarová, I. (1997). *Matematické vzorce a metody*. Skriptum. České vysoké učení technické v Praze.

[Cai and Thierauf, 1996] Cai, J. and Thierauf, G. (1996). Evolution strategies for solving discrete optimization problems. *Advances in Engineering Software*, 25:177–183.

[CDT Community, 2010] CDT Community (2010). Eclipse C/C++ Development Tooling - CDT. `http://www.eclipse.org/cdt/`.

[Dongarra et al., 1996] Dongarra, J., Pozo, R., and Walker, D. (1996). LAPACK++ V. 1.1: High performance linear algebra users' guide. `http://math.nist.gov/lapack++/lapackppman1_1.ps.gz`.

[Fox and Schmit, 1966] Fox, R. L. and Schmit, L. A. (1966). Advances in the integrated approach to structural synthesis. *Journal of Spacecraft and Rockets*, 3(6):858–866.

[Giger and Ermanni, 2006] Giger, M. and Ermanni, P. (2006). Evolutionery truss topology optimization using a graph-based parameterization concept. *Structural and Multidisciplinary Optimization*, 32(4):313–326.

[Jaroslav Kruis a kolektiv, 2010] Jaroslav Kruis a kolektiv (2010). Homepage of SIFEL. `http://mech.fsv.cvut.cz/~sifel/`.

[Jiroušek, 2006] Jiroušek, O. (2006). Metoda konečných prvků: poznámky k přednáškám. `http://mech.fd.cvut.cz/education/master/k618y2m1/download/ymkp_fem.pdf`.

[Kirsch, 1995] Kirsch, U. (1995). Layout optimization using reduction and expansion processes. *First World Congress of Structural and Multidisciplinary Optimization*.

[6] B&B je zkratka Branch and bound, v překladu Metoda větví a mezí.

[Lemonge and Barbosa, 2003] Lemonge, A. C. C. and Barbosa, H. J. C. (2003). An adaptive penalty scheme for genetic algorithms in structural optimization. *International Journal for Numerical Methods in Engineering*, 59:703–736.

[Lepš, 2004] Lepš, M. (2004). *Single and Multi-Objective Optimization in Civil Engineering with Applications*. PhD thesis, ČVUT v Praze.

[MinGW team, 2010] MinGW team (2010). MinGW: Minimalist GNU for Windows. `http://www.mingw.org/`.

[NVIDIA, 2011] NVIDIA (2011). CUDA Zone. `http://www.nvidia.com/object/cuda_home_new.html`.

[Patzák, 2009] Patzák, B. (2009). Úvodní přednáška do předmětu NAK1. `https://mech.fsv.cvut.cz/homeworks/student/NAK1/lecture01.pdf`.

[Ralston, 1978] Ralston, A. (1978). *Základy numerické matematiky*. Academia, nakladatelství Československé akademie věd.

[Rapoff, 2006] Rapoff, A. J. (2006). MER 311 advanced strength of materials course page [online]. `http://engineering.union.edu/ rapoffa/MER311/laboratories/`.

[Shewchuk, 1994] Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical report, School od Computer Science, Carnegie Mellon University.

[Sigmund and Bendsœ, 2003] Sigmund, O. and Bendsœ, M. P. (2003). *Topology Optimization: Theory, Methods and Applications*. Springer-Verlag, 2nd edition. ISBN 978-3-540-42992-0.

[Steven, 2003] Steven, G. (2003). Product and system optimization in engineering simulation. *FENet Newsletter*.

[Sváček and Feistauer, 2006] Sváček, P. and Feistauer, M. (2006). *Metoda konečných prvků*. Skriptum. České vysoké učení technické v Praze.

[The MathWorks, 2010a] The MathWorks (2010a). Matlab 7: Mathematics. `http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/math.pdf`.

[The MathWorks, 2010b] The MathWorks (2010b). Matlab Compiler 4: User's Guide. `http://www.mathworks.com/access/helpdesk/help/pdf_doc/compiler/compiler.pdf`.

[The MathWorks, 2010c] The MathWorks (2010c). mldivide \, mrdivide / [online]. `http://www.mathworks.com/access/helpdesk/help/techdoc/ref /mldivide.html#f90-1002049`.

[Venkayya, 1971] Venkayya, V. B. (1971). Design of optimum structures. *Computers & Structures*, 1:265–309.

[Vondráček, 2008a] Vondráček, R. (2008a). DSSinC - direct sparse solver in c++. `http://www.femcad.com/DSSinC/`.

[Vondráček, 2008b] Vondráček, R. (2008b). *Use of a Sparse Direct Solver in Engineering Applications of the Finite Element Method*. PhD thesis, České vysoké učení technické v Praze.

[Wu and Chow, 1995a] Wu, S.-J. and Chow, P.-T. (1995a). Integrated discrete and configuration optimization of trusses using genetic algorithms. *Computers & Structures*, 55(4):495–702.

[Wu and Chow, 1995b] Wu, S.-J. and Chow, P.-T. (1995b). Steady-state genetic algorithms for discrete optimization of trusses. *Computers & Structures*, 56(6):979–991.

A Přehled užitých anglosaských jednotek s převodem do SI soustavy

Jednotka	Název anglický	Název český	Převod do SI soustavy
in	inch	palec	1 in = 25,4 mm
psi	pound per square inch	libra síly na čtverečný palec	1 psi = 6.894,757 Pa
kp	kip	-	1 kip = 4.448,222 N
lb/in^3	pound per cubic inch	libra síly na kubický palec	$3{,}613 \cdot 10^{-5}$ $lb/in^3 = 1$ kg/m^3

Tabulka 18: Přehled užitých anglosaských jednotek s převodem do SI soustavy