ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta stavební
Katedra mechaniky

Analýza tradičních příkladů rozměrové optimalizace

Analysis of sizing optimization benchmarks

Bakalářská práce

Studijní program: Stavební inženýrství
Studijní obor: Konstrukce pozemních staveb

Vedoucí práce: Ing. Matěj Lepš, Ph.D.

Adéla Pospíšilová

Praha 2010

Zde je prostor pro zadání.

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pouze za odborného vedení vedoucího bakalářské práce Ing. Matěje Lepše, Ph.D.

Dále prohlašuji, že veškeré podklady, ze kterých jsem čerpala, jsou uvedeny v seznamu použité literatury.

Datum: Podpis:

Ráda bych touto cestou vyjádřila svůj dík Ing. Matějovi Lepšovi, Ph.D., za jeho cenné připomínky, trpělivost a ochotu při vedení mé bakalářské práce.

Tento projekt byl realizován za finanční podpory z prostředků státního rozpočtu prostřednictvím Ministerstva průmyslu a obchodu (projekt MPO FT-TA4/100).

Abstrakt

Tato práce se zabývá studiem klasických příkladů rozměrové diskrétní optimalizace. Z nepřeberného množství byly vybrány čtyři nejvíce 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, byly tedy 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. V budoucnu plánujeme tyto skripty využít ke spuštění optimalizace za pomoci metody větví a mezí (branch-and-bound methods).

Abstract

This thesis focuses on studies of classical sizing discrete optimization benchmarks. 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 thesis. Finally, our future vision is to use these scripts for the next optimization with branch-and bound methods.

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

Obsah

Seznam tabulek

Seznam obrázků

Kapitola 1

Úvod

V současné době neustále vznikají nové optimalizační algoritmy. S příchodem počítačů do stavební praxe se blíží doba reálné numerické optimalizace konstrukcí [Rammant, 2008]. Bude tedy nezbytné mít pro tyto účely vyvinuté vhodné optimalizační metody. Při návrhu takové metody je vždy nezbytné ji verifikovat a validovat. Oba postupy vyžadují sadu vzorových příkladů, tzv. benchmarků, u kterých obvykle známe přesné řešení.

Na tyto příklady jsou kladeny dva protichůdné požadavky - nejprve by měly být dostatečně reprezentativní neboli mít schopnost dostatečně přesně vystihovat chování reálných optimalizačních úloh. To obvykle vede k jejich velikosti a složitosti. Na druhou stranu by měly být příklady dostatečně malé, aby jejich vyhodnocení nebylo neúnosně výpočetně náročné.

V různých oblastech numerické optimalizace již vznikly rozličné sady benchmarků. Jmenujme například oblast vícekriteriální optimalizace, kde je stupeň standardizace na vysoké úrovni [Suganthan, 2010]. Taktéž v tradiční oblasti jednokriteriální optimalizace funkcí nad spojitými oblastmi byly snahy o vytvoření ucelených sad testovacích funkcí [Suganthan, 2010].

Naše pozornost je zaměřena na tzv. rozměrovou optimalizaci konstrukcí. Cílem je najít příčné řezy příhradové konstrukce tak, aby byla minimalizována hmotnost konstrukce, ale aby zároveň byla splněna některá omezení, nejčastěji formou limitních napětí a posunů. Speciální podtřídou jsou příklady tzv. diskrétní, kde příčné řezy nabývají jen diskrétních hodnot, obvykle z předem daného seznamu, např. válcovaných profilů. Zde existuje několik typových příkladů (ty nejčastější budou představeny v této práci), nicméně do této doby nebyly podrobně prozkoumány, což je naším dlouhodobým cílem.

Cílem optimalizace i studia benchmarků diskrétní rozměrové optimalizace je nejčastěji

skladba a vlastnosti globálního optima. I ty nejmodernější metody optimalizace založené na metodě větví a mezí [Demel, 2008] v některé své části vyžadují využití „hrubé výpočetní síly“. Tudíž by bylo vhodné, aby vyhodnocení takového benchmarku bylo co nejrychlejší.

Cílem této práce je tudíž prostudovat možné implementace rozměrové optimalizace s důrazem na dobu výpočtu. V současné době jsou velice oblíbené vědecké nástroje, jako je Matlab, Mathematica či Maple, které kromě interpretovaného programovacího jazyka nabízejí širokou škálu nástrojů na zpracování a vizualizaci vědeckých výsledků. Taktéž množství již naprogramovaných knihoven různých (nejen matematických) nástrojů je v těchto programech nepřeberné. Díky tomu bývá vývoj nových metod a vědeckých postupů v těchto programech rychlejší než např. v některých standardních kompilovaných jazycích, jako je C/C++, Fortran, Basic, apod. Naopak, interpretované jazyky by měly být teoreticky pomalejší než kompilované, neboť při vyhodnocování výrazu probíhá u interpretovaných jazyků zároveň kontrola správnosti, která u kompilovaných jazyků probíhá před vlastním startem programu. Pokusem obejít tuto nevýhodu je možnost zkompilovat interpretovaný kód. Takovou možnost nabízí např. Matlab a i my jsme podrobili tuto možnost našemu výzkumu.

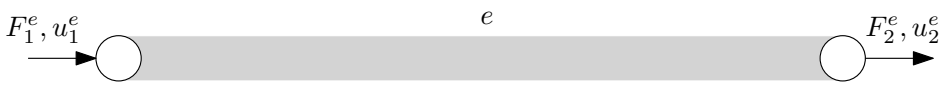
Kapitola 2

Analýza konstrukce

K získání deformace konstrukce (tzn. pole posunů a pootočení) nebo vnitřních sil v konstrukci je možno použít mnoho metod výpočtu. My se v této práci budeme zabývat metodou konečných prvků, jelikož právě tato metoda je vhodná pro její algoritmizaci.

2.1 Přímé odvození pro tažené a tlačené pruty

Základní myšlenkou této metody je vyjádření koncových uzlových sil v závislosti na koncových posunech prvku. Pro stručné odvození vezměme například prut, který je orientován v ose x, je přímý s konstantním průřezem po celé jeho délce a je zatížen pouze osovými silami v jeho uzlech. Tento model používá pro odvození i [Fish and Belytschko, 2007]. Známe jeho průřezovou plochu A, jeho délku l a Youngův modul pružnosti materiálu E, ze kterého je prut vyroben.



Obrázek 2.1: Prut orientovaný v ose x

Chceme-li si vyjádřit uzlové síly, je nejprve nutné zavést znaménkovou konvenci. V této práci budeme uvažovat tah jako kladný. Pak můžeme zapsat, že:

$$\begin{aligned} F_1^e &= -\sigma_x \cdot A, \\ F_2^e &= \sigma_x \cdot A. \end{aligned} \tag{2.1}$$

Dále potřebujeme vyjádření Hookova zákona, který nám říká, že napětí je lineární funkcí přetvoření ε:

$$\sigma_x = E \cdot \varepsilon. \tag{2.2}$$

A též potřebujeme definici deformace, a to jako poměrného protažení prutu:

$$\varepsilon = \frac{\Delta l}{l}. \tag{2.3}$$

Pokud dosadíme do první rovnice, získáme:

$$\begin{aligned} F_1^e &= -\sigma_x \cdot A = -E \cdot \varepsilon \cdot A = -E \cdot \frac{\Delta l}{l} \cdot A. \\ F_2^e &= \sigma_x \cdot A = E \cdot \varepsilon \cdot A = E \cdot \frac{\Delta l}{l} \cdot A. \end{aligned} \tag{2.4}$$

A při předpokladu, že $\Delta l = u_2^e - u_1^e$, kde u_1^e a u_2^e jsou koncové posuny uzlů, platí:

$$\begin{aligned} F_1^e &= -E \cdot \frac{u_2^e - u_1^e}{l} \cdot A, \\ F_2^e &= E \cdot \frac{u_2^e - u_1^e}{l} \cdot A. \end{aligned} \tag{2.5}$$

Vyjádříme-li si tuhost prutu jako $k^e = \frac{EA}{l}$, můžeme pak maticově zapsat

$$\begin{Bmatrix} F_1^e \\ F_2^e \end{Bmatrix} = \begin{bmatrix} k^e & -k^e \\ -k^e & k^e \end{bmatrix} \begin{Bmatrix} u_1^e \\ u_2^e \end{Bmatrix}, \tag{2.6}$$

nebo zkráceně $f^e = K^e \cdot r^e$, kde

$$K^e = \begin{bmatrix} k^e & -k^e \\ -k^e & k^e \end{bmatrix} = \frac{E.A}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \tag{2.7}$$

Matici K^e nazýváme maticí tuhosti prutu, r^e jsou neznámé posuny, f^e je vektor pravých stran neboli uzlová zatížení prutu.

2.2 Sestavení globální matice tuhosti konstrukce a lokalizace v 1D

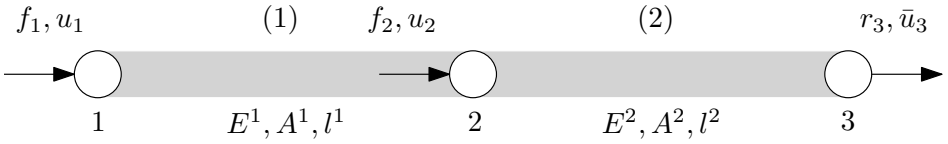
V předchozí kapitole jsme odvodili matici tuhosti pro tažený, respektive tlačený prut. Pokud však chceme sestavit matici tuhosti pro více prutů - tzv. globální matici tuhosti, je nutno objasnit problematiku lokalizace.

Uvažujeme konstrukci složenou z několika prutů orientovaných v ose x, s proměnnými délkami l^e, průřezovými plochami A^e a moduly pružnosti E^e. Předpokládáme, že uzly jsou jedinečně očíslovány. Pro každý prut určíme lokální matici tuhosti K^e a distribuční matici L^e. Distribuční matice má rozměr $n \times m$, kde počet řádků m je roven počtu lokálních posunů a počet sloupců n je roven počtu globálních posunů. (Jelikož se jedná o řešení 1D problému, počet posunů je totožný s počtem uzlů.) Získáme ji tak, že do nulové matice o rozměru $n \times m$ přičteme jedničku k prvkům a_{ij} tam, kde se i-tá pozice shoduje s pořadím lokálního posunu a j-tá pozice s pořadím globálního posunu. L^e nám v podstatě převádí lokální matici tuhosti prutu K^e o rozměru $m \times m$ na sčítanec globální matice tuhosti konstrukce $\tilde{K}^e$ o rozměru $n \times n$. Globální matici tuhosti K pak získáme posčítáním vzniklých sčítanců od všech lokálních matic tuhosti.

$$\tilde{K}^e = L^{eT} \cdot K^e \cdot L^e, \tag{2.8}$$

$$K = \sum_{e=1}^{n} \tilde{K}^e. \tag{2.9}$$

Například pro konstrukci na obrázku 2.2 platí toto:



Obrázek 2.2: Model dvouprutové příhradové konstrukce

$$K^1 = \begin{bmatrix} k^1 & -k^1 \\ -k^1 & k^1 \end{bmatrix}, \quad K^2 = \begin{bmatrix} k^2 & -k^2 \\ -k^2 & k^2 \end{bmatrix},$$

$$L^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad L^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\tilde{K}^1 = L^{1T} \cdot K^1 \cdot L^1 = \begin{bmatrix} k^1 & -k^1 & 0 \\ -k^1 & k^1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\tilde{K}^2 = L^{2T} \cdot K^2 \cdot L^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & k^2 & -k^2 \\ 0 & -k^2 & k^2 \end{bmatrix},$$

$$K = \sum_{e=1}^{n} \tilde{K}^e = \begin{bmatrix} k^1 & -k^1 & 0 \\ -k^1 & k^1 + k^2 & -k^2 \\ 0 & -k^2 & k^2 \end{bmatrix}.$$

2.3 Zavedení okrajových podmínek a způsob vyřešení soustavy rovnic $K \cdot r = f$

Globální matice tuhosti konstrukce je sama o sobě singulární. Předepíšeme-li však dostatečný počet okrajových podmínek, soustava $K \cdot r = f$ se stane regulární [Patzák, 2009]. Nyní rozdělíme soustavu rovnic na dvě části příslušné volným u a předepsaným $\bar{u}$ stupňům volnosti:

$$\begin{bmatrix} K_{uu} & K_{up} \\ K_{pu} & K_{pp} \end{bmatrix} \begin{Bmatrix} u \\ \bar{u} \end{Bmatrix} = \begin{Bmatrix} f \\ r \end{Bmatrix}, \tag{2.10}$$

Což představuje soustavu rovnic:

$$K_{uu} \cdot u + K_{up} \cdot \bar{u} = f, \tag{2.11}$$

$$K_{pu} \cdot u + K_{pp} \cdot \bar{u} = r. \tag{2.12}$$

Z první rovnice soustavy si vyjádříme neznámé posuny u:

$$K_{uu} \cdot u = f - K_{up} \cdot \bar{u}, \tag{2.13}$$

$$u = K_{uu}^{-1}(f - K_{up} \cdot \bar{u}). \tag{2.14}$$

A k vyjádření neznámých reakcí r na konstrukci už jen zbývá dosadit vyjádřené posuny do druhé rovnice.

$$K_{pu} \cdot u + K_{pp} \cdot \bar{u} = r, \tag{2.15}$$

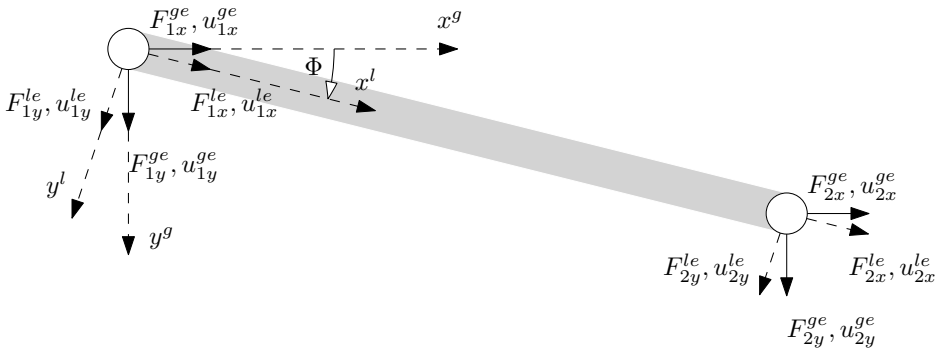
$$K_{pu} \cdot K_{uu}^{-1}(f - K_{up} \cdot \bar{u}) + K_{pp} \cdot \bar{u} = r. \tag{2.16}$$

Pokud použijeme konstrukci z obrázku 2.2, pak příslušná soustava rovnic vypadá následovně:

$$\left[\begin{array}{cc|c} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ \hline k_{31} & k_{32} & k_{33} \end{array}\right] \left\{\begin{array}{c} u_1 \\ u_2 \\ \hline \bar{u}_3 \end{array}\right\} = \left\{\begin{array}{c} f_1 \\ f_2 \\ \hline r_3 \end{array}\right\}$$

2.4 Transformace

V předchozích kapitolách jsme řešili pouze 1D problém. Pokud však máme prut pootočený, je nutno zavést prutový 2D prvek.



Obrázek 2.3: Prut natočený o úhel Φ

Máme prut natočený o úhel Φ, jak lze vidět na obrázku 2.3. Tento prut může být popsán lokálním a globálním souřadným systémem. Transformace převádí jeden systém na druhý a obráceně. V předchozích kapitolách jsme měli zavedenou soustavu pro 1D, tu nyní rozšíříme pro 2D tak, že $F_{1y}^{le} = F_{2y}^{le} = 0$.

$$\left\{\begin{array}{c} F_{1x}^{le} \\ F_{2x}^{le} \end{array}\right\} = \left[\begin{array}{cc} k^e & -k^e \\ -k^e & k^e \end{array}\right] \left\{\begin{array}{c} u_{1x}^{le} \\ u_{2x}^{le} \end{array}\right\}, \tag{2.17}$$

rozšíříme na

$$\left\{\begin{array}{c} F_{1x}^{le} \\ F_{1y}^{le} \\ F_{2x}^{le} \\ F_{2y}^{le} \end{array}\right\} = \left[\begin{array}{cccc} k^e & 0 & -k^e & 0 \\ 0 & 0 & 0 & 0 \\ -k^e & 0 & k^e & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \left\{\begin{array}{c} u_{1x}^{le} \\ u_{1y}^{le} \\ u_{2x}^{le} \\ u_{2y}^{le} \end{array}\right\}. \tag{2.18}$$

Vztah pro transformaci vektoru mezi dvěma souřadnými systémy můžeme vyjádřit jako:

$$u_{1x}^{le} = u_{1x}^{ge} \cdot \cos(\Phi^e) + u_{1y}^{ge} \cdot \sin(\Phi^e), \tag{2.19}$$

$$u_{1y}^{le} = -u_{1x}^{ge} \cdot \sin(\Phi^e) + u_{1y}^{ge} \cdot \cos(\Phi^e), \tag{2.20}$$

v maticovém zápisu pro oba konce prutu pak:

$$\begin{Bmatrix} u_{1x}^{le} \\ u_{1y}^{le} \\ u_{2x}^{le} \\ u_{2y}^{le} \end{Bmatrix} = \begin{bmatrix} \cos(\Phi^e) & \sin(\Phi^e) & 0 & 0 \\ -\sin(\Phi^e) & \cos(\Phi^e) & 0 & 0 \\ 0 & 0 & \cos(\Phi^e) & \sin(\Phi^e) \\ 0 & 0 & -\sin(\Phi^e) & \cos(\Phi^e) \end{bmatrix} \begin{Bmatrix} u_{1x}^{ge} \\ u_{1y}^{ge} \\ u_{2x}^{ge} \\ u_{2y}^{ge} \end{Bmatrix}, \tag{2.21}$$

$$\text{zkráceně } r^{le} = T^e \cdot r^{ge},$$

kde T^e je transformační matice, r^{le} jsou posuny vyjádřené v lokálním souřadném systému a r^{ge} v globálním souřadném systému (dále jen s.s.).

Pro koncové síly platí obdobné vztahy:

- pro transformaci z globálního s.s. do lokálního s.s.: $F^{le} = T^e \cdot F^{ge}$,
- pro transformaci z lokálního s.s. do globálního s.s.: $F^{ge} = (T^e)^T \cdot F^{le}$.

$(T^e)^T$ je transponovaná transformační matice, která je totožná s inverzní maticí $(T^e)^{-1}$, jelikož je transformační matice T^e ortogonální. Pokud bychom chtěli vyjádřit vztah mezi vektorem koncových sil a koncových posunů v globálním s.s., došli bychom k tomuto:

$$\begin{aligned} F^{ge} &= (T^e)^T \cdot F^{le}, \\ F^{ge} &= (T^e)^T \cdot K^{le} \cdot r^{le}, \\ F^{ge} &= (T^e)^T \cdot K^{le} \cdot T^e \cdot r^{ge}, \\ F^{ge} &= K^{ge} \cdot r^{ge}, \end{aligned} \tag{2.22}$$

kde $K^{ge} = (T^e)^T \cdot K^{le} \cdot T^e$ je vztah pro převod matice do globálního s.s. Dosazením z (2.21) bychom dospěli k tomu, že:

$$K^{ge} = \frac{EA}{l} \begin{bmatrix} \cos^2\Phi^e & \cos\Phi^e \sin\Phi^e & -\cos^2\Phi^e & -\cos\Phi^e \sin\Phi^e \\ \cos\Phi^e \sin\Phi^e & \sin^2\Phi^e & -\cos\Phi^e \sin\Phi^e & -\sin^2\Phi^e \\ -\cos^2\Phi^e & -\cos\Phi^e \sin\Phi^e & \cos^2\Phi^e & \cos\Phi^e \sin\Phi^e \\ -\cos\Phi^e \sin\Phi^e & -\sin^2\Phi^e & \cos\Phi^e \sin\Phi^e & \sin^2\Phi^e \end{bmatrix}.$$

2.5 Vnitřní osové síly

Máme vyjádřenou globální matici tuhosti prutu K^{ge}. Nyní můžeme provést lokalizaci dle pododdílu 2.2, a tím obdržíme matici tuhosti konstrukce K^g. Dále je potřeba sestavit vektor pravých stran a vektor neznámých posunů v globálním souřadném systému. Dle (2.14) vypočítáme neznámé posuny u a dle (2.16) reakce v podporách r. Normálové síly získáme výpočtem z $N_x = \sigma_x \cdot A$. Dosadíme-li za napětí σ_x vyjádření Hookova zákona a dále definici deformace, můžeme vyjádřit:

$$N_x = \sigma_x \cdot A, \tag{2.23}$$

$$N_x = E \cdot \varepsilon \cdot A, \tag{2.24}$$

$$N_x = E \cdot A \cdot \frac{\Delta l}{L}, \tag{2.25}$$

$$N_x = \frac{EA}{L} \cdot (u_{2x}^{le} - u_{1x}^{le}), \tag{2.26}$$

$$N_x = \frac{EA}{L} \cdot [-1 \quad 0 \quad 1 \quad 0] \begin{bmatrix} u_{1x}^{le} \\ u_{1y}^{le} \\ u_{2x}^{le} \\ u_{2y}^{le} \end{bmatrix}, \tag{2.27}$$

$$N_x = \frac{EA}{L} \cdot [-1 \quad 0 \quad 1 \quad 0] \cdot u^{le}. \tag{2.28}$$

Z kapitoly 2.4 víme, že transformace z globálního do lokálního souřadného systému vypadá takto: $u^{le} = T^e \cdot u^{ge}$. Dosadíme-li do (2.28), pak:

$$N_x = \frac{EA}{L} \cdot [-1 \quad 0 \quad 1 \quad 0] \cdot T^e \cdot u^{ge}, \tag{2.29}$$

$$N_x = \frac{EA}{L} \cdot [-1 \quad 0 \quad 1 \quad 0] \cdot \begin{bmatrix} \cos\Phi^e & \sin\Phi^e & 0 & 0 \\ -\sin\Phi^e & \cos\Phi^e & 0 & 0 \\ 0 & 0 & \cos\Phi^e & \sin\Phi^e \\ 0 & 0 & -\sin\Phi^e & \cos\Phi^e \end{bmatrix} \cdot u^{ge}, \tag{2.30}$$

$$N_x = \frac{EA}{L}[-cos(\Phi^e) \quad -sin(\Phi^e) \quad cos(\Phi^e) \quad sin(\Phi^e)] \cdot u^{ge}. \tag{2.31}$$

Rovnice (2.31) nám dává vztah pro výpočet normálových sil v prutech získaný z globálních posunů.

Kapitola 3

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. Principem je použít všechny kombinace ze sady ploch příčných průřezů na všech prutech konstrukce, spočítat deformace konstrukce a vnitřní osové síly, resp. napětí na jednotlivých prutech a vybrat tu kombinaci ploch, která se nejvíce blíží spodní hranici předem zadaných podmínek. Jelikož je nutno spočítat všechny varianty, je tato metoda optimalizace časově náročná. Je tedy vhodné najít pro podobné typy konstrukcí nejrychlejší výpočetní postupy, aby bylo vůbec reálné tuto metodu použít. Pro hledání těchto postupů bude použito programu MATLAB a programovacího jazyka C++.

Kapitola 4

Implementace v programu MATLAB

4.1 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.

4.2 Možnosti sestavení globální matice tuhosti

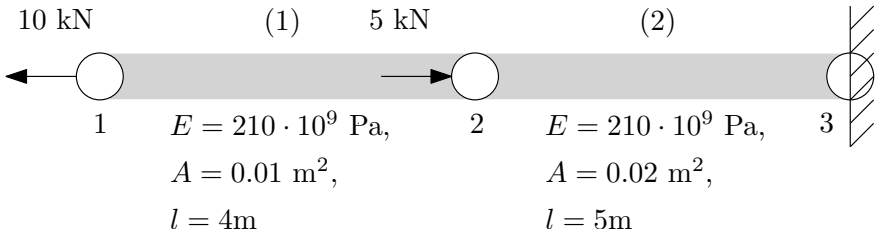
V kapitole 2.2 byla popsána problematika lokalizace přes distribuční matici. Program MATLAB však nabízí i jiné možnosti sestavení globální matice tuhosti, odpadá nám pak sestavování distribučních matic. Pokud definujeme okrajové podmínky k příkladu na obrázku 2.2 (vizte obrázek 4.1), nic nám pak nebrání k tomu, abychom globální matici tuhosti sestavili.

4.2.1 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 definujeme 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
```



Obrázek 4.1: Dvouprutová příhradová konstrukce se zadanými okrajovými podmínkami

Kód MATLABu: Lokalizace přes kódová čísla s adresováním řádků a sloupců na konstrukci z obrázku 4.1:

```
% Zadání parametrů konstrukce
   Emod  =[210*10^6 210*10^6];
   Area  =[0.01 0.02];
   inode =[1 2];
```

```
5     jnode =[2 3];
6     x     =[0 4 9];
7  % Sestavení kódových čísel
8     ij=[inode',jnode'];                                          ij = 1   2
9                                                                      2   3
10
11 % Umístění lokální matice tuhosti prutu do globální matice tuhosti konstrukce
12    numnod=length(x);                                          numnod = 3
13    numelm=length(Area);                                       numelm = 2
14    Stiff=zeros(numnod);
15
16    for m=1:numelm,
17      % Sestavení lokální matice tuhosti jednotlivých prutů
18      i=inode(m)
19      j=jnode(m)
20      L=abs(x(j)-x(i))
21      k=Area(m)*Emod(m)/L*[1,-1;-1,1]
22
23      n=ij(m,:)
24      Stiff(n,n)=Stiff(n,n)+ k
25    end
26
27 % Mezivýsledky for cyklu
28  m = 1:                                  m = 2:
29      i = 1                                   i = 2
30      j = 2                                   j = 3
31      L = 4                                   L = 5
32      k =      525 000  -525 000              k =      840 000  -840 000
33              -525 000   525 000                     -840 000   840 000
34      n = 1   2                               n = 2   3
35
36      Stiff =  525 000  -525 000  0           Stiff =  525 000  -525 000        0
37              -525 000   525 000  0                   -525 000  1 365 000 -840 000
38                    0         0  0                         0  -840 000   840 000
39
40 % Vypsání matice tuhosti konstrukce
41    Stiff
42     Stiff =  525 000  -525 000        0
43             -525 000  1 365 000 -840 000
44                   0  -840 000   840 000
```

4.2.2 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
                 525 000         0  0
                       0         0  0
        m = 2:
        Stiff =  525 000  -525 000  0
                 525 000  -525 000  0
                       0         0  0
```

```
 Stiff =  525 000  -525 000         0
         -525 000  1 365 000  -840 000
                0  -840 000         0
m = 2:
 Stiff =  525 000  -525 000         0
         -525 000  1 365 000  -840 000
                0  -840 000   840 000
```

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

4.3 Parametrické vyjádření matice

V kapitole 3 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.

Pro názornost si můžeme ukázat, jak by aplikace parametrického zápisu mohla vypadat při sestavení matice tuhosti na příkladu dvouprutové konstrukce vyobrazené na obrázku 4.1. 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];
   x     =[0 4 9];

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

```

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

```
% 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]

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

% Matice tuhosti a dosazení do ní
   Stiff = [  ki(1),     -ki(1),        0; ...      Stiff  =  525000  -525000       0
             -ki(1),  ki(1)+ki(2),  -ki(2); ...                -525000  1365000 -840000
                 0,     -ki(2),      ki(2)]                        0  -840000  840000
```

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

[3]Symbolicky definujeme nulovou matici.

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

4.4 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.

4.4.1 Klasické řešení $X = A \setminus B$

Vezměme soustavu $A \cdot X = B$. Matematicky nejjednodušší řešení je pomocí inverzní matice A^{-1}, pak 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].

4.4.2 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].

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} l_{ij} \cdot u_{jk} \quad \text{pro } i = 1, \ldots, k, \tag{4.1}$$

$$l_{ik} = \frac{1}{u_{kk}} \cdot (a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot u_{jk}) \quad \text{pro } i = k+1, \ldots, n, \tag{4.2}$$

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.

```
1   for k=1:m
2      if k==1
3         for I=1:m
4            U(k,I)=KK(k,I);
5            L(I,k)=KK(I,k)/U(k,k);
6         end
7      else
```

```
8           for I=1:m
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 $x = U^{-1} \cdot y$:

$$A \cdot x = b \quad | \quad A = L \cdot U, \tag{4.3}$$

$$L \cdot U \cdot x = b \quad | \quad y = U \cdot x, \tag{4.4}$$

$$L \cdot y = b, \tag{4.5}$$

$$y = L^{-1} \cdot b, \tag{4.6}$$

$$x = U^{-1} \cdot y. \tag{4.7}$$

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.

4.4.3 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 [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:

$$r_{11} = \sqrt{a_{11}}, \tag{4.8}$$

$$r_{i1} = \frac{a_{i1}}{r_{11}} \quad \text{pro } i = 2, \cdots, n, \tag{4.9}$$

$$r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2}, \quad \text{pro } j = 2, \cdots, n, \tag{4.10}$$

$$r_{ij} = \frac{1}{r_{jj}} \cdot (a_{ij} - \sum_{k=1}^{j-1} r_{ik} \cdot r_{jk}) \quad \text{pro } i = j+1, \cdots, n. \tag{4.11}$$

Řešení pak dostaneme z:

$$A \cdot x = b \quad | \quad A = R^T \cdot R, \tag{4.12}$$

$$R^T \cdot R \cdot x = b, \tag{4.13}$$

$$x = R^{-1} \cdot ((R^T)^{-1} \cdot b). \tag{4.14}$$

Dle kapitoly 4.4.1 ale není nutné inverzní matice počítat. V MATLABu můžeme vyjádřit (4.14) jako:

$$x = R \setminus (R^T \setminus b). \tag{4.15}$$

Jak je vidět, Choleského dekompozice je v podstatě zvláštním případem LU faktorizace, přičemž ukládáme jen horní trojúhelníkovou matici R, což je jistě výhodou. Problémem však zde je odmocnina v prvcích matice r_{jj} a dělení v prvcích r_{ij}, stejně jako v LU faktorizaci popsané v 4.4.2.

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

4.4.4 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:

$$l_{ij} = \frac{1}{d_{jj}} \cdot (a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_{kk} \cdot l_{jk}), \tag{4.16}$$

$$d_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \cdot d_{kk}. \tag{4.17}$$

Řešení pak dostaneme podle [Jiroušek, 2006] takto:

$$A \cdot x = b \quad | \quad A = L \cdot D \cdot L^T, \tag{4.18}$$

$$L \cdot D \cdot L^T \cdot x = b. \tag{4.19}$$

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

$$L \cdot D \cdot L^T \cdot x = L \cdot \bar{b}, \tag{4.20}$$

$$D \cdot L^T \cdot x = \bar{b}, \tag{4.21}$$

$$L^T \cdot x = D^{-1}\bar{b}. \tag{4.22}$$

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
```

4.4.5 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:

$$\alpha_k = \frac{(r^k)^T \cdot d^k}{(d^k)^T \cdot A \cdot d^k}, \tag{4.23}$$

$$x^{k+1} = x^k + \alpha_k \cdot d^k, \tag{4.24}$$

$$r^{k+1} = b - A \cdot x^{k+1}, \tag{4.25}$$

$$\beta_k = \frac{(r^{k+1})^T \cdot A \cdot d^k}{(d^k)^T \cdot A \cdot d^k}. \tag{4.26}$$

$$d^{k+1} = r^{k+1} - \beta_k \cdot d^k, \tag{4.27}$$

Výsledné řešení x^{k+1} získáme z rovnice 4.24. Hodnota α_k z rovnice 4.23 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 4.27. Reziduum r^{k+1} je určeno vztahem 4.25 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 []).

4.5 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`.

Kapitola 5

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`. 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. Dalším způsobem vyřešení soustavy rovnic je použití volně dostupné knihovny LAPACK++.

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 `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.

```
1  #include <lapackpp.h>
2  int main() {
3      int N = 2;
4      LaGenMatDouble A(N, N);
5      LaVectorDouble x(N), b(N);
6
7      A(0, 0) = 1.0;
8      A(0, 1) = 2.0;
9      A(1, 0) = 3.0;
10     A(1, 1) = 4.0;
11     b(0) = 17;
12     b(1) = 39;
13
14     LaLinearSolve(A,x,b);
15     return 0;
16 }
```

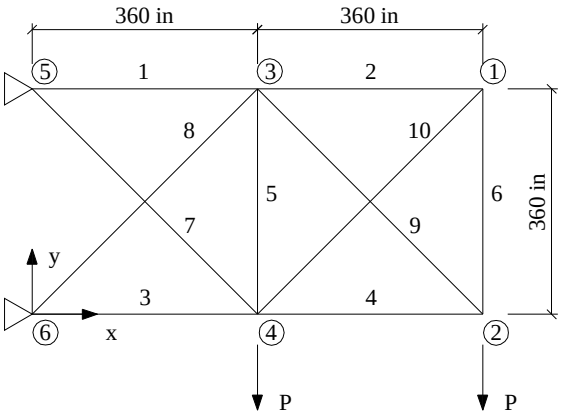
Pokud chceme na terminálu vidět vstupy a výstupy, není problém vše vypsat například pomocí `for cyklů` a `printf()`.

Kapitola 6

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].

6.1 Desetiprutová příhradová 2D konzola



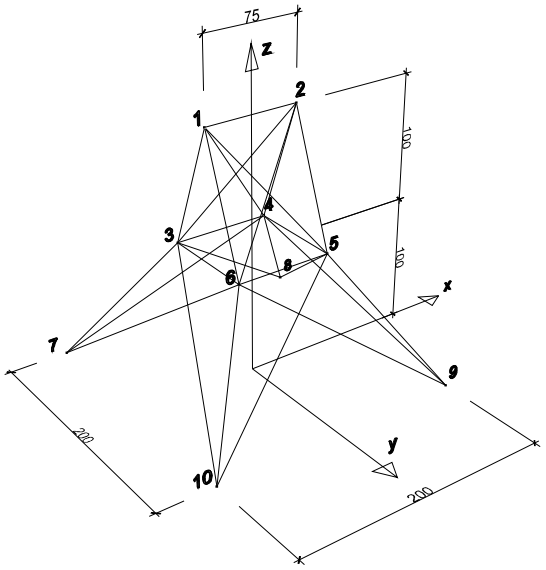
Obrázek 6.1: 10-prutová příhradová 2D konzola

Tato konstrukce byla poprvé zveřejněna Venkayyou v [Venkayya, 1971]. Konstrukce byla řešena spojitou rozměrovou optimalizací. Prvního případu diskrétní rozměrové optimalizace

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 6.1: Charakteristiky materiálu a omezující a okrajové podmínky 10-prutové konzoly

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 6.1, konstrukce je vyobrazena na obrázku 6.1. 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.



Obrázek 6.2: 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 6.2: 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 6.3: Charakteristiky materiálu a omezující podmínky 25-prutové věže

6.2 Dvacetipětiprutová příhradová 3D věž

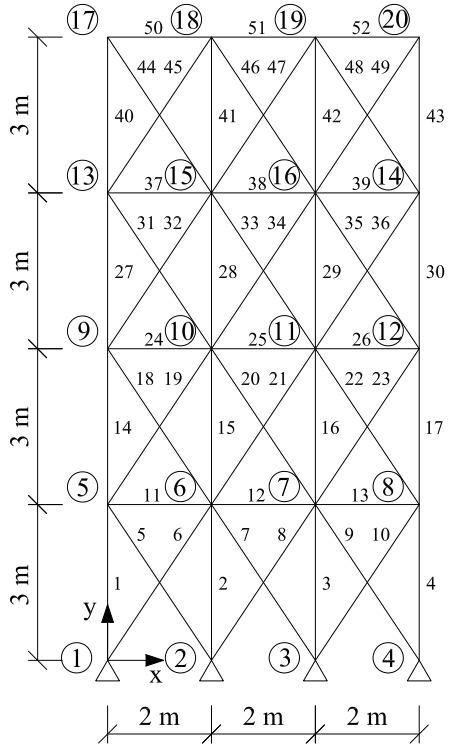
Dle [Venkayya, 1971] byla tato konstrukce poprvé zveřejněna Foxem a Schmitem v [Fox and Schmit, 1966]. Nejstarší námi objevený článek uveřejňující tuto konstrukci je [Wu and Chow, 1995a]. Je znázorněna na obrázku 6.2, charakteristiky materiálu jsou uvedené v tabulce 6.3 a zatížení konstrukce naleznete v tabulce 6.4. 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 v [Wu and Chow, 1995b]. Tato konstrukce je symetrická, proto byly jednotlivé pruty sdruženy do skupin, které můžete nalézt v tabulce 6.2.

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 6.4: Zatížení 25-prutové konstrukce v kip jednotkách

6.3 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 (v mm^2): 71,613, 90,968, 126,451, 161,290, 198,064, 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,



Obrázek 6.3: 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 6.5: Sdružení ploch příčných průřezů do skupin (52-prutová konstrukce)

14 193,520, 14 774,164, 15 806,420, 17 096,740, 18 064,480, 19 354,800, 21 612,860. Konstrukce je vyobrazena na obrázku 6.3, charakteristiky materiálu a omezující podmínky jsou uvedeny v tabulce 6.6 a zatížení naleznete v tabulce 6.7. Konstrukce je opět symetrická, proto jsou plochy sdruženy do skupin, které jsou uvedeny v tabulce 6.5 [Lemonge and Barbosa, 2003].

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

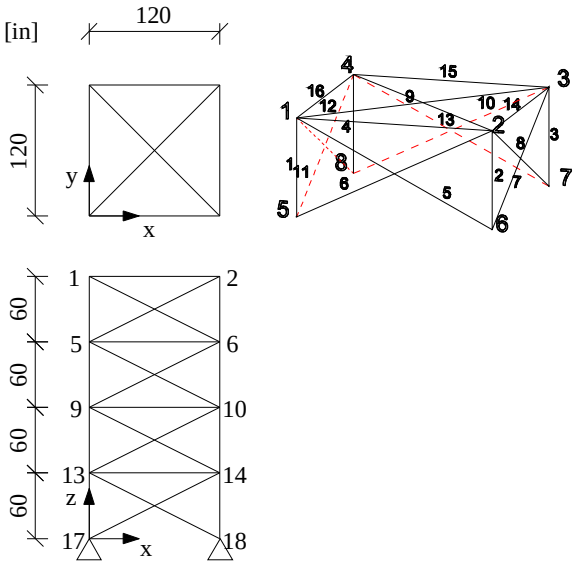
Tabulka 6.6: Charakteristiky materiálu a omezující podmínky 52-prutové věže

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 6.7: Zatížení 52-prutové konstrukce v kN

6.4 Sedmdesátidvouprutová příhradová 3D konstrukce

Konstrukce na obrázku 6.4 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 [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, 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 6.9) a zatížení (vizte tabulku 6.10) byly převzaty z [Lemonge and Barbosa, 2003]. Plochy byly opět sdruženy do skupin, jelikož je konstrukce symetrická, vizte tabulku 6.8.



Obrázek 6.4: 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 6.8: 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 6.9: Charakteristiky materiálu a omezující podmínky 72-prutové věže

Uzel	F_x	F_y	F_z
1	5	5	-5

Tabulka 6.10: Zatížení 72-prutové konstrukce v kipech

Kapitola 7

Testovací metody a jejich výsledky

7.1 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 6. Po úpravě kódu, vizte přílohu C, kde je uveden kód pro 10-prutovou příhradovou 2D konzolu, bylo nutno zjistit, které operace zabírají nejvíce času.

řádek kódu[5]	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 7.1: Výstup z Profileru MATLABu pro jedno spuštění kódu přímého řešiče pro desetiprutovou konstrukci

V tabulkách 7.1 a 7.2 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

[5]Řádek kódu odpovídá číslování kódu v příloze C.

řádek kódu[6]	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 7.2: Výstup z Profileru MATLABu pro 1 000 spuštění kódu přímého řešiče pro 10-prutovou konstrukci

konstrukce vyjádřit parametricky v závislosti na tuhosti prvků dle pododdílu 4.3. Sestavení vektoru pravé strany zabírá též zbytečně mnoho paměti. Je proto vhodné sestavit tento vektor jinak.

7.1.1 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 4.3) 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 4.4.4 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. Výsledný skript po provedených úpravách vizte v příloze D.

7.1.2 Plná a řídká matice

Jelikož matice tuhosti konstrukce je řídká, zkoumali jsme i vliv této vlastnosti na výsledný čas. Pro 10-prutovou příhradovou 2D konzolu vizte přílohu E. Skripty všech

[6]Řádek kódu odpovídá číslování kódu v příloze C.

	sestavení matice	matice	způsob řešení K*r=f	průměrný čas
MATLAB, m-file	při běhu skriptu	plná	zpětné lomítko	0,3489
	předem sestavena parametricky	plná	zpětné lomítko	0,0610
	předem sestavena parametricky	plná	LU faktorizace	**0,0516**
	předem sestavena parametricky	plná	Choleského dekompozice	0,0669
	předem sestavena parametricky	plná	LDL^T rozklad	0,0716
	předem sestavena parametricky	plná	metoda sdružených gradientů	0,7461
	předem sestavena parametricky	řídká	zpětné lomítko	0,0848
	předem sestavena parametricky	řídká	LU faktorizace	0,0986
	předem sestavena parametricky	řídká	Choleského dekompozice	0,1127
	předem sestavena parametricky	řídká	LDL^T rozklad	0,1334
	předem sestavena parametricky	řídká	metoda sdružených gradientů	0,8004
MATLAB, kompilace	při běhu skriptu	plná	zpětné lomítko	0,3785
	předem sestavena parametricky	plná	zpětné lomítko	0,0761
	předem sestavena parametricky	plná	LU faktorizace	**0,0669**
	předem sestavena parametricky	plná	Choleského dekompozice	0,0811
	předem sestavena parametricky	plná	LDL^T rozklad	0,0869
	předem sestavena parametricky	plná	metoda sdružených gradientů	0,8090
	předem sestavena parametricky	řídká	zpětné lomítko	0,0956
	předem sestavena parametricky	řídká	LU faktorizace	0,1091
	předem sestavena parametricky	řídká	Choleského dekompozice	0,1210
	předem sestavena parametricky	řídká	LDL^T rozklad	0,1435
	předem sestavena parametricky	řídká	metoda sdružených gradientů	0,8118
C++	předem sestavena parametricky	plná	LDL^T rozklad	**0,0033**
	předem sestavena parametricky	plná	LAPACK++	0,0051

Tabulka 7.3: Výsledky časů v sekundách na 10-prutové konstrukci

dříve zmiňovaných konstrukcí v kapitole 6 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 7.4 a 7.5. Řídkost jednotlivých matic je znázorněna na obrázcích H.1 až H.4 v příloze H.

7.1.3 Řešiče soustavy rovnic `Stiff*Disp=f`

K řešení soustavy rovnic `Stiff*Disp=f` bylo použito pět způsobů řešení. Metoda zpětného lomítka popsaná v pododdíle 4.4.1 by teoreticky měla být nejrychlejší, jelikož je celý algoritmus výpočtu optimalizován společností MathWorks. Nicméně tomu tak není. Dalo by

	sestavení matice	matice	způsob řešení K*r=f	průměrný čas
MATLAB, m-file	při běhu skriptu	plná	zpětné lomítko	1,2632
	předem sestavena parametricky	plná	zpětné lomítko	0,0970
	předem sestavena parametricky	plná	LU faktorizace	**0,0824**
	předem sestavena parametricky	plná	Choleského dekompozice	0,0936
	předem sestavena parametricky	plná	LDL^T rozklad	0,1019
	předem sestavena parametricky	plná	metoda sdružených gradientů	1,4402
	předem sestavena parametricky	řídká	zpětné lomítko	0,2492
	předem sestavena parametricky	řídká	LU faktorizace	0,2664
	předem sestavena parametricky	řídká	Choleského dekompozice	0,2536
	předem sestavena parametricky	řídká	LDL^T rozklad	0,2865
	předem sestavena parametricky	řídká	metoda sdružených gradientů	1,5945
MATLAB, kompilace	při běhu skriptu	plná	zpětné lomítko	1,2428
	předem sestavena parametricky	plná	zpětné lomítko	0,1406
	předem sestavena parametricky	plná	LU faktorizace	**0,1267**
	předem sestavena parametricky	plná	Choleského dekompozice	0,1403
	předem sestavena parametricky	plná	LDL^T rozklad	0,1491
	předem sestavena parametricky	plná	metoda sdružených gradientů	1,5176
	předem sestavena parametricky	řídká	zpětné lomítko	0,2822
	předem sestavena parametricky	řídká	LU faktorizace	0,3010
	předem sestavena parametricky	řídká	Choleského dekompozice	0,2881
	předem sestavena parametricky	řídká	LDL^T rozklad	0,3279
	předem sestavena parametricky	řídká	metoda sdružených gradientů	1,6063
C++	předem sestavena parametricky	plná	LDL^T rozklad	0,0171
	předem sestavena parametricky	plná	LAPACK++	**0,0118**

Tabulka 7.4: Výsledky časů v sekundách na 25-prutové konstrukci

se předpokládat, že se čas ztrácí na výběru typu matice (řídká, plná, symetrická apod.) a následném výběru vhodného způsobu výpočtu.

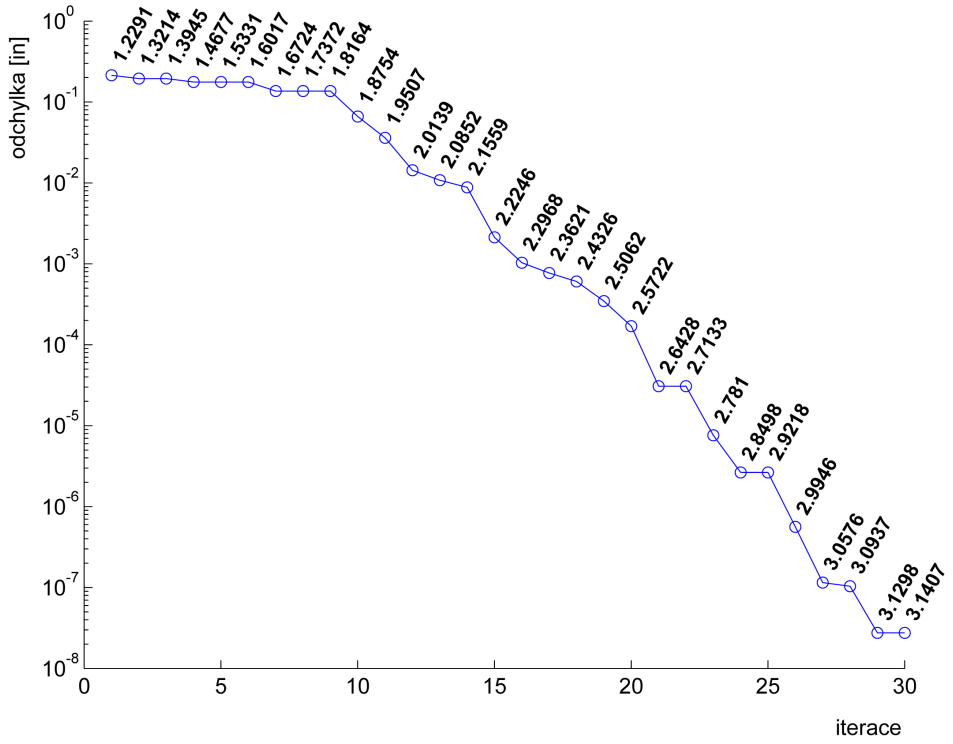
Jako další řešič byla použita LU faktorizace charakterizovaná v pododdíle 4.4.2. Způsob tohoto výpočtu je vhodný pro malé konstrukce, které mají málo nulových prvků v matici tuhosti, hodí se pro ně tedy plná matice tuhosti. Vizte tabulku 7.3 a 7.4 pro deseti a dvacetipětiprutovou konstrukci.

Choleského dekompozice popsaná v pododdíle 4.4.3 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 7.5).

LDL^T rozklad objasněný v pododdíle 4.4.4 byl shledán jako pomalejší než LU faktorizace 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 7.1 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.



Obrázek 7.1: Závislost iterace na chybě řešení s časem výpočtu pro 1000 spuštění v sekundách pro 72-prutovou konstrukci

7.2 MATLAB a kompilace

Původní předpoklad, že kompilace MATLAB kódu do samostatně spustitelné aplikace znatelně urychlí čas výpočtu, se nepotvrdil. Kompilace byla provedena podle pododdílu 4.5. Důvodem je nejspíše dlouhá inicializace aplikace. Výsledky jsou vidět v tabulkách 7.3 až 7.6.

	sestavení matice	matice	způsob řešení K*r=f	průměrný čas
MATLAB, m-file	při běhu skriptu	plná	zpětné lomítko	3,5938
	předem sestavena parametricky	plná	zpětné lomítko	1,2550
	předem sestavena parametricky	plná	LU faktorizace	1,0383
	předem sestavena parametricky	plná	Choleského dekompozice	1,0459
	předem sestavena parametricky	plná	LDL^T rozklad	1,0749
	předem sestavena parametricky	plná	metoda sdružených gradientů	2,56000
	předem sestavena parametricky	řídká	zpětné lomítko	0,6246
	předem sestavena parametricky	řídká	LU faktorizace	0,6696
	předem sestavena parametricky	řídká	Choleského dekompozice	**0,5931**
	předem sestavena parametricky	řídká	LDL^T rozklad	0,6904
	předem sestavena parametricky	řídká	metoda sdružených gradientů	2,0546
MATLAB, kompilace	při běhu skriptu	plná	zpětné lomítko	3,5380
	předem sestavena parametricky	plná	zpětné lomítko	1,1997
	předem sestavena parametricky	plná	LU faktorizace	1,0506
	předem sestavena parametricky	plná	Choleského dekompozice	1,0424
	předem sestavena parametricky	plná	LDL^T rozklad	1,0622
	předem sestavena parametricky	plná	metoda sdružených gradientů	2,6101
	předem sestavena parametricky	řídká	zpětné lomítko	0,6639
	předem sestavena parametricky	řídká	LU faktorizace	0,7101
	předem sestavena parametricky	řídká	Choleského dekompozice	**0,6348**
	předem sestavena parametricky	řídká	LDL^T rozklad	0,7460
	předem sestavena parametricky	řídká	metoda sdružených gradientů	2,1085
C++	předem sestavena parametricky	plná	LDL^T rozklad	0,2348
	předem sestavena parametricky	plná	LAPACK++	**0,0395**

Tabulka 7.5: Výsledky časů v sekundách na 72-prutové konstrukci

7.3 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`. Příkaz `ccode` zároveň zajistí přeindexování jednotlivých prvků pole z 1 až n prvků na 0 až (n-1) prvků. Dále bylo použito jednorozměrného pole pro vektor zatížení `f`. K řešení soustavy rovnic pro získání neznámých posunů `u` bylo využito dvou řešičů. Jako první můžeme zmínit řešič pana docenta Kruise, jako druhý pak řešič za pomoci knihovny LAPACK++[Dongarra et al., 1996].

	sestavení matice	matice	způsob řešení K*r=f	průměrný čas
MATLAB, m-file	při běhu skriptu	plná	zpětné lomítko	1,3548
	předem sestavena parametricky	plná	zpětné lomítko	0,2450
	předem sestavena parametricky	plná	LU faktorizace	0,2312
	předem sestavena parametricky	plná	Choleského dekompozice	**0,2245**
	předem sestavena parametricky	plná	LDL^T rozklad	0,2453
	předem sestavena parametricky	plná	metoda sdružených gradientů	1,7452
	předem sestavena parametricky	řídká	zpětné lomítko	0,4174
	předem sestavena parametricky	řídká	LU faktorizace	0,4387
	předem sestavena parametricky	řídká	Choleského dekompozice	0,4098
	předem sestavena parametricky	řídká	LDL^T rozklad	0,4624
	předem sestavena parametricky	řídká	metoda sdružených gradientů	1,8485
MATLAB, kompilace	při běhu skriptu	plná	zpětné lomítko	1,3151
	předem sestavena parametricky	plná	zpětné lomítko	0,3080
	předem sestavena parametricky	plná	LU faktorizace	0,2933
	předem sestavena parametricky	plná	Choleského dekompozice	**0,2776**
	předem sestavena parametricky	plná	LDL^T rozklad	0,2964
	předem sestavena parametricky	plná	metoda sdružených gradientů	1,8457
	předem sestavena parametricky	řídká	zpětné lomítko	0,4556
	předem sestavena parametricky	řídká	LU faktorizace	0,4774
	předem sestavena parametricky	řídká	Choleského dekompozice	0,4466
	předem sestavena parametricky	řídká	LDL^T rozklad	0,5147
	předem sestavena parametricky	řídká	metoda sdružených gradientů	1,9099
C++	předem sestavena parametricky	plná	LDL^T rozklad	0,0757
	předem sestavena parametricky	plná	LAPACK++	**0,0201**

Tabulka 7.6: Výsledky časů v sekundách na 52-prutové konstrukci

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 řešič soustavy rovnic s použitím knihovny LAPACK++ bylo třeba kód upravit dle kapitoly 5. Zejména vstupní matici tuhosti konstrukce `K`, vektor zatížení `f` a výstupní posuny jednotlivých uzlů `u`. Kód vizte v příloze F.

Kapitola 8

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 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, 2008]. 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 je také možné do budoucna spustit například metodu větví a mezí, která se použije pro diskrétní rozměrovou optimalizaci bez potřebné předem použité spojité rozměrové optimalizace, jak je často zmiňováno v zahraničních článcích [L.J.Li and nad F.Liu, 2009].

Literatura

[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/`.

[Demel, 2008] Demel, J. (2008). Operační výzkum.

[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`.

[Fish and Belytschko, 2007] Fish, J. and Belytschko, T. (2007). *A First Course in Finite Elements*. JohnWiley & Sons, Ltd.

[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). Evolutionary truss topology optimization using a graph-based parameterization concept. *Structural and Multidisciplinary Optimization*, 32(4):313–326.

[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*.

[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.

[L.J.Li and nad F.Liu, 2009] L.J.Li and nad F.Liu, Z. (2009). A heuristic particle swarm optimization method for truss structures with discrete variables. *Computers & Structures*, 87(7-8):435–443.

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

[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.

[Rammant, 2008] Rammant, J.-P. (2008). White paper: Optimal design of structures. `http://www.scia-online.com/WWW/WebSiteUS.nsf/0/b0950d97cf5dc205c12574200029b067/$FILE/OptimalDesignofStructures_SE2008_EN.pdf`.

[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, 2^{nd} edition. ISBN 978-3-540-42992-0.

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

[Suganthan, 2010] Suganthan, P. N. (2010). Benchmarks for evaluation of evolutionary algorithms. `http://www3.ntu.edu.sg/home/epnsugan/index_files/cec-benchmarking.htm`.

[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, 2008] Vondráček, R. (2008). DSSinC - Direct Sparse Solver in C++. `http://www.femcad.com/DSSinC/`.

[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.

Příloha 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 A.1: Přehled užitých anglosaských jednotek s převodem do SI soustavy

Příloha B

Parametry použitého počítače

Dell Studio XPS Desktop 435MT	
Procesor	Intel Core i7 920, 2.67 GHz
Cache	8 L3 MBytes, L2 4x256 kBytes
Chipset	Intel X58
Pamět	DDR3 Triple Channels 6x1026 MBytes
Grafická karta	ATI Radeon HD 4670
Operační systém	Windows 7 Enterprise (64-bitový)
Verze MATLABu	R2009b
Kompiler pro MATLAB	Microsoft Visual C++ 2008 Express
Editor C++	Eclipse IDE for C/C++ Developers
Kompiler pro C++	MinGW 5.1.6
Překladač LaTeXu	MiKTeX 2.8

Tabulka B.1: Použitá počítačová sestava

Příloha C

Kód pro desetiprutovou konstrukci pro přímý řešič v MATLABu

```
%% Výpočet napětí a posunů na konstrukci
% Konstrukce            - 10-prutová 2D konzola
% Matice tuhosti        - sestavuje se při běhu skriptu, plná
% Řešič soustavy rovnic - zpětné lomítko

%% Zadání hodnot:
% Emod    - Youngův modul pružnosti
% L       - vektor délek jednotlivých prvků
% f       - vektor pravých stran (zatížení)
% inode   - začáteční uzel prutu
% jnode   - koncový uzel prutu
% x       - x-ová souřadnice uzlu
% y       - y-ová souřadnice uzlu
% idu     - uzel prutu, na kterém je předepsaný posun uzlu ve směru x
% u       - předepsaný posun uzlu ve směru x
% idv     - uzel prutu, na kterém je předepsaný posun uzlu ve směru y
% v       - předepsaný posun uzlu ve směru y
% idfx    - uzel prutu, na kterém je uzlové zatížení ve směru x
% fx      - uzlové zatížení ve směru x
% idfy    - uzel prutu, na kterém je uzlové zatížení ve směru y
% fy      - uzlové zatížení ve směru y

%% Výpočet:
% xx      - vodorovný průmět prutu
% yy      - svislý průmět prutu
% Length  - délka prutu
% ij      - matice kódových čísel
% k       - lokální matice tuhosti prutů
% Stiff   - globální matice tuhosti konstrukce
% f       - vektor zatížení
% Disp    - posuny uzlů
% React   - reakce v podporách
```

```
% ki      - tuhost jednotlivých prutů
% Mforces - normálové síly
% maxs    - maximální hodnota napětí ze všech prutů
% maxw    - maximální posun ze všech styčníků
% w       - váha konstrukce

%% ---------------------------------------------------------------

function example1(Area)

% clear;
% clc;
format long g

%% Zadání parametrů konstrukce
Emod  =[1.e4 1.e4 1.e4 1.e4 1.e4 1.e4 1.e4 1.e4 1.e4 1.e4];
inode =[5 3 6 4 4 2 4 6 3 4];
jnode =[3 1 4 2 3 1 5 3 2 1];
x     =720 720 360 360 0 0];
y     =[360 0 360 0 360 0];
idu   =[5 6];   idfx=[];
u     =[0 0];   fx  =[];
idv   =[5 6];   idfy=[1 2 3 4 5 6];
v     =[0 0];   fy  =[0 -100 0 -100 0 0 ];

%% Výpočet vodorovných a svislých průmětů prutů a jejich délky
xx=(x(jnode)-x(inode));
yy=(y(jnode)-y(inode));
Length=sqrt(xx.^2+yy.^2);

%% Sestavení matice kódových čísel
ij=[2*inode'-1,2*inode',2*jnode'-1,2*jnode'];

%% Umístění lokální matice tuhosti prutu do globální matice tuhosti konstrukce

numnod=length(x);
numelm=length(Area);
Stiff=zeros(2*numnod);

for m=1:numelm

    % Sestavení lokální matice tuhosti jednotlivých prutů
    i=inode(m);  j=jnode(m);
    c=(x(j)-x(i))/Length(m);
    s=(y(j)-y(i))/Length(m) ;
    cc=c*c ; cs=c*s; ss=s*s;
    T=[[cc,cs];[cs,ss]];
    k=Area(m)*Emod(m)/Length(m)*[T,-T;-T,T];
    % % % % % % % % % % % % % % % % % % %

    n=ij(m,:);
```

```
84      Stiff(n,n)=Stiff(n,n)+ k;
85  end
86
87  %% Řešení soustavy rovnic K*r=f
88  ns=size(Stiff,1);
89  f=zeros(ns,1);
90  [ndc,ndr]=size([u';v']);
91  irow=[2*idu'-1;2*idv'];
92  % Sestavení vektoru zatížení
93  f([2*idfx'-1;2*idfy'])=[fx';fy'];
94  f(irow)=[u';v'];
95  %Výměna sloupců matice tuhosti konstrukce v závislosti na známých posunech
96  diagv=zeros(ns,1);
97  diagv(irow)=ones(ndc,1);
98  s=Stiff;
99  s(irow,:)=zeros(ndc,ns);
100 s=s+diag(diagv);
101 % Řešení neznámých posunů a reakcí v podporách
102 Disp=s\f;
103 React=Stiff*Disp;
104
105 %% Řešení vnitřních sil
106 cs=xx'./Length';
107 sn=yy'./Length'; ki=(Area'.*Emod')./Length';
108 iu=Disp(2*inode-1); iv=Disp(2*inode);
109 ju=Disp(2*jnode-1); jv=Disp(2*jnode);
110 Mforces=ki.*((ju-iu).*cs+(jv-iv).*sn);
111
112 %% Maximální napětí, maximální posun a váha konstrukce
113 maxs=max(max(abs(Mforces./Area'))) ;
114 maxw=max(max(abs(Disp)));
115 w=0.1*sum(Area*Length');
116
117 end
118
```

Příloha D

Kód pro desetiprutovou konstrukci se symbolickou plnou maticí v programu MATLAB

```
% % Výpočet napětí a posunů na konstrukci
% Konstrukce            - 10-prutová 2D konzola
% Matice tuhosti        - v symbolickém zápisu, plná
% Řešič soustavy rovnic - zpětné lomítko

% % Zadání hodnot:
% Emod    - Youngův modul pružnosti
% L       - vektor délek jednotlivých prvků
% f       - vektor pravých stran (zatížení)

% % Výpočet:
% ki      - tuhost jednotlivých prutů
% K       - globální matice tuhosti konstrukce
% Disp    - posuny uzlů
% Mforces - vnitřní síly
% maxs    - maximální hodnota napětí ze všech prutů
% maxw    - maximální posun ze všech styčníků
% w       - váha konstrukce

%---------------------------------------------
function example1(Area)
format long g

%% Zadání parametrů konstrukce:
Emod = 1.e4;
L=100*[3.6 3.6 3.6 3.6 3.6 3.6 5.091168824543143 5.091168824543143 ...
       5.091168824543143 5.091168824543143];
f=[0 0 0 -100 0 0 0 -100]';

```

```
%% Vlastní výpočet:
ki=Emod*Area./L;

K=[ 0.5*ki(10) + ki(2), 0.5*ki(10), 0, 0, -ki(2), 0, (-0.5)*ki(10), (-0.5)*ki(10); ...
    0.5*ki(10), 0.5*ki(10) + ki(6), 0, -ki(6), 0, 0, (-0.5)*ki(10), (-0.5)*ki(10); ...
    0, 0, ki(4) + 0.5*ki(9), (-0.5)*ki(9), (-0.5)*ki(9), 0.5*ki(9), -ki(4), 0; ...
    0, -ki(6), (-0.5)*ki(9), ki(6) + 0.5*ki(9), 0.5*ki(9), (-0.5)*ki(9), 0, 0; ...
    -ki(2), 0, (-0.5)*ki(9), 0.5*ki(9), ki(1) + ki(2) + 0.5*ki(8) + 0.5*ki(9), ...
    0.5*ki(8) - 0.5*ki(9), 0, 0; 0, 0, 0.5*ki(9), (-0.5)*ki(9), ...
    0.5*ki(8) - 0.5*ki(9), ki(5) + 0.5*ki(8) + 0.5*ki(9), 0, -ki(5); ...
    (-0.5)*ki(10), (-0.5)*ki(10), -ki(4), 0, 0, 0, ...
    0.5*ki(10) + ki(3) + ki(4) + 0.5*ki(7), 0.5*ki(10) - 0.5*ki(7); ...
    (-0.5)*ki(10), (-0.5)*ki(10), 0, 0, 0, -ki(5), ...
    0.5*ki(10) - 0.5*ki(7), 0.5*ki(10) + ki(5) + 0.5*ki(7)];

Disp=K\f;

Disp=[Disp;0;0;0;0];

Mforces=[ ki(1)*Disp(5), ...
          ki(2)*(Disp(1)-Disp(5)), ...
          ki(3)*Disp(7), ...
          ki(4)*(Disp(3)-Disp(7)), ...
          ki(5)*(Disp(6)-Disp(8)), ...
          ki(6)*(Disp(2)-Disp(4)), ...
          ki(7)*(1/2*Disp(7)*2^(1/2)-1/2*Disp(8)*2^(1/2)), ...
          ki(8)*(1/2*Disp(5)*2^(1/2)+1/2*Disp(6)*2^(1/2)), ...
          ki(9)*(1/2*(Disp(3)-Disp(5))*2^(1/2)-1/2*(Disp(4)-Disp(6))*2^(1/2)), ...
          ki(10)*(1/2*(Disp(1)-Disp(7))*2^(1/2)+1/2*(Disp(2)-Disp(8))*2^(1/2))]';

maxs = max(max(abs(Mforces./Area'))) ;
maxw=max(max(abs(Disp)));
w=0.1*sum(Area*L');

end
```

Příloha E

Kód pro desetiprutovou konstrukci se symbolickou řídkou maticí v programu MATLAB

```
% % Výpočet napětí a posunů na konstrukci
% Konstrukce            - 10-prutová 2D konzola
% Matice tuhosti        - v symbolickém zápisu, řídká
% Řešič soustavy rovnic - zpětné lomítko

% % Zadání hodnot:
% Emod    - Youngův modul pružnosti
% L       - vektor délek jednotlivých prvků
% f       - vektor pravých stran (zatížení)

% % Výpočet:
% ki      - tuhost jednotlivých prutů
% K       - globální matice tuhosti konstrukce
% Disp    - posuny uzlů
% Mforces - vnitřní síly
% maxs    - maximální hodnota napětí ze všech prutů
% maxDisp - maximální posun ze všech styčníků
% w       - váha konstrukce

%-----------------------------------------------
function example1(Area)
format long g

%% Zadání parametrů konstrukce:
Emod = 1.e4;
L=100*[3.6 3.6 3.6 3.6 3.6 3.6 5.091168824543143 5.091168824543143 ...
       5.091168824543143 5.091168824543143];
f=[0 0 0 -100 0 0 0 -100]';

```

```
%% Vlastní výpočet:
ki=Emod*Area./L;

su1=0.5*ki(7)+0.5*ki(10);
su2=-0.5*ki(7)+0.5*ki(10);
su3=0.5*ki(10);
su4=0.5*ki(9);
su5=0.5*ki(8);

K=sparse([1 2 5 7 8 1 2 4 7 8 3 4 5 6 7 2 3 4 5 6 1 3 4 5 6 3 4 5 6 ...
          8 1 2 3 7 8 1 2 6 7 8],...
         [1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6 6 6 ...
          6 7 7 7 7 7 8 8 8 8 8],...
 [ki(2)+su3,       su3,    -ki(2),               -su3,      -su3, ...
        su3, ki(6)+su3,    -ki(6),               -su3,      -su3, ...
  ki(4)+su4,      -su4,      -su4,                su4,    -ki(4), ...
     -ki(6),      -su4, ki(6)+su4,                su4,      -su4, ...
     -ki(2),      -su4,       su4, ki(1)+ki(2)+su5+su4,   su5-su4, ...
        su4,      -su4,   su5-su4,       ki(5)+su5+su4,    -ki(5), ...
       -su3,      -su3,    -ki(4),     ki(3)+ki(4)+su1,       su2, ...
       -su3,      -su3,    -ki(5),                su2, ki(5)+su1]);

Disp=K\f;

Mforces=[ki(1)*Disp(5); ...
         ki(2)*(Disp(1)-Disp(5)); ...
         ki(3)*Disp(7); ...
         ki(4)*(Disp(3)-Disp(7)); ...
         ki(5)*(Disp(6)-Disp(8)); ...
         ki(6)*(Disp(2)-Disp(4)); ...
         ki(7)*(0.5*Disp(7)*2^(0.5)-0.5*Disp(8)*2^(0.5)); ...
         ki(8)*(0.5*Disp(5)*2^(0.5)+0.5*Disp(6)*2^(0.5)); ...
         ki(9)*(0.5*(Disp(3)-Disp(5))*2^(0.5)-0.5*(Disp(4)-Disp(6))*2^(0.5)); ...
         ki(10)*(0.5*(Disp(1)-Disp(7))*2^(0.5)+0.5*(Disp(2)-Disp(8))*2^(0.5))];

maxs = max(max(abs(Mforces./Area'))) ;
maxDisp=max(max(abs(Disp)));
w=0.1*sum(Area*L');

end
```

Příloha F

Kód pro desetiprutovou konstrukci v C++ s využitím knihovny LAPACK++

```
1  #include <time.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #include <lapackpp.h>
6  #include <sybfd.h>
7
8  #define VYPISUJ
9
10 # ifndef RAND_MAX
11 #   define RAND_MAX 32767
12 # endif
13
14 const int neq = 8 ;  // počet rovnic v soustavě
15 const int narea = 42 ; // počet prvků v množině, ze které se vybírají plochy
16 const int ncharea = 10; // počet vybraných ploch
17
18
19 /* generátor náhodných čísel */
20
21 static int give_rnd_int ( int omax )
22 {
23    if ( omax==1 )
24       return ( 0 ) ;
25    else if ( omax>0 )
26    {
27       return ( (long)floor((double)omax * ((double)(rand() - 1)/(double)( RAND_MAX ))) ) ;
28    }
29    else
```

```
30    {
31       fprintf ( stderr, "\n\n Omax should be bigger than zero. \n" ) ;
32       fprintf ( stderr, " give_rnd_int (%s, line %d)\n",__FILE__,__LINE__) ;
33       exit ( 1 ) ;
34    }
35
36    return( 0 ) ;
37 }
38
39 /* z narea možných ploch příčných průřezů se vybere ncharea prvků */
40
41 static void get_solution ( double* A )
42 {
43    double M[narea] = {1.62, 1.80, 1.99, 2.13, 2.38, 2.62, 2.63, 2.88, 2.83, 3.09, 3.13, \
44           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, \
45           5.12, 5.74, 7.22, 7.97, 11.5, 13.5, 13.9, 14.2, 15.5, 16.0, 16.9, 18.8, 19.9, \
46           22.0, 22.9, 26.5, 30.0, 33.5};
47
48    for ( int i=0 ; i<ncharea ; i++ )
49       A[i]=M[give_rnd_int( narea )] ;
50 }
51
52 /* funkce, která řeší posuny, napětí a váhu konstrukce */
53
54 static void compute ( double* A )
55 {
56    double A1, A2, A3, A4, A5, A6, A7, A8, A9, A10 ;
57    int i ;  int j;
58
59    A1= A[0];  A2= A[1];  A3= A[2];  A4= A[3];  A5= A[4];
60    A6= A[5];  A7= A[6];  A8= A[7];  A9= A[8];  A10= A[9];
61
62    double k1 = 250.0/9.0*A1;  double k2 = 250.0/9.0*A2;
63    double k3 = 250.0/9.0*A3;  double k4 = 250.0/9.0*A4;
64    double k5 = 250.0/9.0*A5;  double k6 = 250.0/9.0*A6;
65    double k7 = 0.17592186044416E18/0.8956478914489369E16*A7;
66    double k8 = 0.17592186044416E18/0.8956478914489369E16*A8;
67    double k9 = 0.17592186044416E18/0.8956478914489369E16*A9;
68    double k10 = 0.17592186044416E18/0.8956478914489369E16*A10;
69
70    LaGenMatDouble K(neq,neq);
71    for(i=0;i<neq;i++){
72       for(j=0;j<neq;j++){
73          K(i,j)=0;
74       }
75    }
76
77    K(0,0) = k10*(1.0/2.0)+k2;       K(0,1) = k10*(1.0/2.0);
78    K(0,4) = -k2;                    K(0,6) = k10*(-1.0/2.0);
79    K(0,7) = k10*(-1.0/2.0);         K(1,0) = k10*(1.0/2.0);
80    K(1,1) = k10*(1.0/2.0)+k6;       K(1,3) = -k6;
```

```
81     K(1,6) = k10*(-1.0/2.0);         K(1,7) = k10*(-1.0/2.0);
82     K(2,2) = k4+k9*(1.0/2.0);        K(2,3) = k9*(-1.0/2.0);
83     K(2,4) = k9*(-1.0/2.0);          K(2,5) = k9*(1.0/2.0);
84     K(2,6) = -k4;                    K(3,1) = -k6;
85     K(3,2) = k9*(-1.0/2.0);          K(3,3) = k6+k9*(1.0/2.0);
86     K(3,4) = k9*(1.0/2.0);           K(3,5) = k9*(-1.0/2.0);
87     K(4,0) = -k2;                    K(4,2) = k9*(-1.0/2.0);
88     K(4,3) = k9*(1.0/2.0);           K(4,4) = k1+k2+k8*(1.0/2.0)+k9*(1.0/2.0);
89     K(4,5) = k8*(1.0/2.0)-k9*(1.0/2.0);      K(5,2) = k9*(1.0/2.0);
90     K(5,3) = k9*(-1.0/2.0);          K(5,4) = k8*(1.0/2.0)-k9*(1.0/2.0);
91     K(5,5) = k5+k8*(1.0/2.0)+k9*(1.0/2.0);   K(5,7) = -k5;
92     K(6,0) = k10*(-1.0/2.0);         K(6,1) = k10*(-1.0/2.0);
93     K(6,2) = -k4;                    K(6,6) = k10*(1.0/2.0)+k3+k4+k7*(1.0/2.0);
94     K(6,7) = k10*(1.0/2.0)-k7*(1.0/2.0);     K(7,0) = k10*(-1.0/2.0);
95     K(7,1) = k10*(-1.0/2.0);         K(7,5) = -k5;
96     K(7,6) = k10*(1.0/2.0)-k7*(1.0/2.0);     K(7,7) = k10*(1.0/2.0)+k5+k7*(1.0/2.0);
97
98     LaVectorDouble f(neq), u(neq);
99     f(0)=0; f(1)=0; f(2)=0; f(3)=-100; f(4)=0; f(5)=0; f(6)=0; f(7)=-100;
100
101    LaLinearSolve(K,u,f);
102
103    double u1, u2, u3, u4, u5, u6, u7, u8 ;
104    u1= u(0); u2= u(1); u3= u(2); u4= u(3); u5= u(4); u6= u(5); u7= u(6); u8= u(7);
105
106    double S[ncharea] ;
107
108    S[0] = fabs(k1*u5/A1);      S[1] = fabs(k2*(u1-u5)/A2);
109    S[2] = fabs(k3*u7/A3);      S[3] = fabs(k4*(u3-u7)/A4);
110    S[4] = fabs(k5*(u6-u8)/A5); S[5] = fabs(k6*(u2-u4)/A6);
111    S[6] = fabs(k7*(u7*sqrt(2.0)/2.0-u8*sqrt(2.0)/2.0)/A7);
112    S[7] = fabs(k8*(u5*sqrt(2.0)/2.0+u6*sqrt(2.0)/2.0)/A8);
113    S[8] = fabs(k9*((u3/2.0-u5/2.0)*sqrt(2.0)-(u4/2.0-u6/2.0)*sqrt(2.0))/A9);
114    S[9] = fabs(k10*((u1/2.0-u7/2.0)*sqrt(2.0)+(u2/2.0-u8/2.0)*sqrt(2.0))/A10);
115
116    double maxw = 0 ;
117
118    for ( i=0 ; i<neq ; i++ )
119       if ( fabs(u(i)) > maxw ) maxw = fabs(u(i)) ;
120
121    maxw = 0 ;
122
123    for ( i=0 ; i<ncharea ; i++ )
124       if ( S[i] > maxw ) maxw = S[i] ;
125
126    maxw = 36.0*A1+36.0*A2+36.0*A3+36.0*A4+36.0*A5+36.0*A6
127    +0.8956478914489369E16/175921860444160.0*A7
128    +0.8956478914489369E16/175921860444160.0*A8
129    +0.8956478914489369E16/175921860444160.0*A9
130    +0.8956478914489369E16/175921860444160.0*A10;
131 }
```

```
132 int main ( int , char * )
133 {
134   /* inicializace generátoru náhodných čísel*/
135    time_t t;  t = time(&t);  int rseed=t ;
136    srand( rseed ) ;
137
138   /* hlavní cyklus */
139   const int iter = 1000 ;
140   double AA[narea] ;
141
142    for ( int j=0 ; j<iter ; j++ ) {
143       get_solution( AA ) ;
144       compute( AA ) ;
145    }
146
147   return 0 ;
148 }
```

Příloha G

Výsledné časy jednotlivých skriptů

V této příloze je použita následující legenda:

Způsob uložení matice:	
01	matice se sestavuje při běhu skriptu a je plná
02	matice je uložena jako parametrická a je plná
03	matice je uložena jako parametrická a je řídká
Způsob výpočtu řešení soustavy K*r=f:	
a	pomocí zpětného lomítka (vizte podkapitolu 4.4.1)
b	pomocí LU faktorizace (vizte podkapitolu 4.4.2)
c	pomocí Choleského dekompozice (vizte podkapitolu 4.4.3)
d	LDL^T rozkladu (vizte podkapitolu 4.4.4)
e	pomocí metody sdružených gradientů (vizte podkapitolu 4.4.5)

Bylo provedeno 10 měření, ze kterých se našla minimální a maximální hodnota a byl spočítán průměr. Veškeré uvedené časy jsou v sekundách. Odchylky mezi minimální a maximální hodnotou v měření jsou nejspíše způsobeny operačním systémem a jeho nezbytnými systémovými procesy.

typ	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	0,3471	0,3479	0,3507	0,3536	0,3472	0,3487	0,3473	0,3482	0,3504	0,3478	0,3471	0,3536	0,3489
02a	0,0608	0,0607	0,0609	0,0608	0,0612	0,0608	0,0629	0,0607	0,0607	0,0605	0,0605	0,0629	0,0610
02b	0,0520	0,0514	0,0516	0,0519	0,0515	0,0516	0,0513	0,0518	0,0516	0,0514	0,0513	0,0520	0,0516
02c	0,0667	0,0669	0,0677	0,0665	0,0670	0,0665	0,0669	0,0667	0,0669	0,0667	0,0665	0,0677	0,0669
02d	0,0712	0,0713	0,0712	0,0721	0,0719	0,0720	0,0715	0,0716	0,0716	0,0719	0,0712	0,0721	0,0716
02e	0,7457	0,7452	0,7553	0,7461	0,7429	0,7453	0,7483	0,7459	0,7429	0,7438	0,7429	0,7553	0,7461
03a	0,0850	0,0853	0,0850	0,0848	0,0844	0,0852	0,0850	0,0842	0,0848	0,0847	0,0842	0,0853	0,0848
03b	0,0987	0,0981	0,0989	0,0990	0,0988	0,0984	0,0986	0,0986	0,0987	0,0983	0,0981	0,0990	0,0986
03c	0,1130	0,1122	0,1124	0,1124	0,1142	0,1124	0,1128	0,1124	0,1126	0,1125	0,1122	0,1142	0,1127
03d	0,1329	0,1339	0,1341	0,1353	0,1327	0,1327	0,1332	0,1330	0,1329	0,1335	0,1327	0,1353	0,1334
03e	0,8007	0,7966	0,8067	0,8050	0,8007	0,8011	0,7970	0,7994	0,8005	0,7966	0,7966	0,8067	0,8004

Tabulka G.1: Výsledky časů na 10-prutové konstrukci (MATLAB)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	1,2683	1,2612	1,2590	1,2677	1,2596	1,2622	1,2664	1,2643	1,2600	1,2637	1,2590	1,2683	1,2632
02a	0,0972	0,0969	0,0966	0,0970	0,0962	0,0967	0,0995	0,0968	0,0969	0,0965	0,0962	0,0995	0,0970
02b	0,0825	0,0823	0,0827	0,0819	0,0833	0,0825	0,0822	0,0821	0,0825	0,0824	0,0819	0,0833	0,0824
02c	0,0934	0,0938	0,0933	0,0932	0,0938	0,0934	0,0939	0,0932	0,0939	0,0944	0,0932	0,0944	0,0936
02d	0,1021	0,1025	0,1020	0,1021	0,1020	0,1011	0,1019	0,1016	0,1022	0,1016	0,1011	0,1025	0,1019
02e	1,4395	1,4439	1,4447	1,4399	1,4407	1,4343	1,4472	1,4420	1,4358	1,4342	1,4342	1,4472	1,4402
03a	0,2505	0,2479	0,2484	0,2482	0,2497	0,2483	0,2502	0,2471	0,2500	0,2514	0,2471	0,2514	0,2492
03b	0,2658	0,2638	0,2677	0,2642	0,2642	0,2661	0,2700	0,2681	0,2671	0,2672	0,2638	0,2700	0,2664
03c	0,2567	0,2559	0,2508	0,2554	0,2554	0,2522	0,2537	0,2518	0,2509	0,2534	0,2508	0,2567	0,2536
03d	0,2879	0,2856	0,2906	0,2856	0,2849	0,2855	0,2869	0,2856	0,2845	0,2883	0,2845	0,2906	0,2865
03e	1,5932	1,5975	1,5899	1,5926	1,5908	1,5951	1,5979	1,5958	1,5962	1,5963	1,5899	1,5979	1,5945

Tabulka G.2: Výsledky časů na 25-prutové konstrukci (MATLAB)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	3,4290	3,6114	3,4355	3,4903	3,6091	3,6453	3,5412	3,6916	3,7615	3,7229	3,4290	3,7615	3,5938
02a	1,3082	1,2598	1,3279	1,2417	1,2516	1,2498	1,2141	1,1843	1,2808	1,2321	1,1843	1,3279	1,2550
02b	1,0406	1,0408	1,0363	1,0366	1,0390	1,0415	1,0389	1,0352	1,0395	1,0348	1,0348	1,0415	1,0383
02c	1,0447	1,0461	1,0476	1,0425	1,0431	1,0478	1,0495	1,0420	1,0487	1,0472	1,0420	1,0495	1,0459
02d	1,0733	1,0779	1,0751	1,0733	1,0727	1,0741	1,0729	1,0790	1,0793	1,0710	1,0710	1,0793	1,0749
02e	2,5581	2,5501	2,5593	2,5630	2,5557	2,5534	2,5730	2,5760	2,5576	2,5537	2,5501	2,5760	2,5600
03a	0,6273	0,6239	0,6227	0,6251	0,6232	0,6227	0,6230	0,6261	0,6283	0,6243	0,6227	0,6283	0,6246
03b	0,6727	0,6673	0,6666	0,6702	0,6671	0,6688	0,6731	0,6730	0,6703	0,6672	0,6666	0,6731	0,6696
03c	0,6015	0,5950	0,5917	0,5909	0,5894	0,5935	0,5895	0,5946	0,5919	0,5935	0,5894	0,6015	0,5931
03d	0,6956	0,6914	0,6973	0,6925	0,6852	0,6883	0,6906	0,6882	0,6871	0,6876	0,6852	0,6973	0,6904
03e	2,0657	2,0565	2,0552	2,0522	2,0562	2,0518	2,0499	2,0501	2,0572	2,0509	2,0499	2,0657	2,0546

Tabulka G.3: Výsledky časů na 72-prutové konstrukci (MATLAB)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	1,3422	1,3047	1,3132	1,3168	1,3495	1,4496	1,3845	1,4387	1,2762	1,3723	1,2762	1,4496	1,3548
02a	0,2444	0,2494	0,2452	0,2441	0,2444	0,2452	0,2442	0,2438	0,2444	0,2454	0,2438	0,2494	0,2450
02b	0,2316	0,2301	0,2309	0,2311	0,2311	0,2302	0,2306	0,2311	0,2309	0,2348	0,2301	0,2348	0,2312
02c	0,2244	0,2243	0,2235	0,2248	0,2247	0,2243	0,2238	0,2247	0,2248	0,2255	0,2235	0,2255	0,2245
02d	0,2446	0,2467	0,2476	0,2443	0,2457	0,2435	0,2439	0,2462	0,2457	0,2445	0,2435	0,2476	0,2453
02e	1,7565	1,7438	1,7442	1,7454	1,7492	1,7415	1,7425	1,7426	1,7441	1,7422	1,7415	1,7565	1,7452
03a	0,4179	0,4211	0,4196	0,4195	0,4185	0,4166	0,4159	0,4156	0,4146	0,4147	0,4146	0,4211	0,4174
03b	0,4390	0,4389	0,4408	0,4379	0,4386	0,4362	0,4369	0,4404	0,4379	0,4401	0,4362	0,4408	0,4387
03c	0,4094	0,4100	0,4121	0,4127	0,4070	0,4103	0,4078	0,4094	0,4088	0,4109	0,4070	0,4127	0,4098
03d	0,4641	0,4692	0,4621	0,4639	0,4616	0,4624	0,4597	0,4598	0,4606	0,4603	0,4597	0,4692	0,4624
03e	1,8472	1,8452	1,8519	1,8475	1,8460	1,8514	1,8424	1,8574	1,8467	1,8494	1,8424	1,8574	1,8485

Tabulka G.4: Výsledky časů na 52-prutové konstrukci (MATLAB)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	0,3755	0,3806	0,3739	0,3710	0,3706	0,3773	0,4075	0,3783	0,3746	0,3758	0,3706	0,4075	0,3785
02a	0,0760	0,0773	0,0763	0,0764	0,0765	0,0742	0,0753	0,0763	0,0778	0,0751	0,0742	0,0778	0,0761
02b	0,0672	0,0683	0,0674	0,0670	0,0664	0,0668	0,0665	0,0665	0,0672	0,0661	0,0661	0,0683	0,0669
02c	0,0812	0,0805	0,0811	0,0803	0,0808	0,0799	0,0840	0,0819	0,0812	0,0804	0,0799	0,0840	0,0811
02d	0,0871	0,0883	0,0876	0,0859	0,0861	0,0889	0,0860	0,0860	0,0862	0,0865	0,0859	0,0889	0,0869
02e	0,8401	0,8169	0,8117	0,8019	0,8026	0,7998	0,8050	0,8033	0,8029	0,8054	0,7998	0,8401	0,8090
03a	0,0972	0,0940	0,0943	0,0956	0,0951	0,0960	0,0957	0,0975	0,0942	0,0964	0,0940	0,0975	0,0956
03b	0,1074	0,1099	0,1105	0,1082	0,1110	0,1087	0,1115	0,1087	0,1079	0,1071	0,1071	0,1115	0,1091
03c	0,1192	0,1197	0,1208	0,1242	0,1204	0,1185	0,1233	0,1217	0,1221	0,1203	0,1185	0,1242	0,1210
03d	0,1475	0,1464	0,1433	0,1423	0,1413	0,1418	0,1438	0,1430	0,1435	0,1421	0,1413	0,1475	0,1435
03e	0,8185	0,8164	0,8042	0,8109	0,8127	0,8108	0,8094	0,8082	0,8148	0,8121	0,8042	0,8185	0,8118

Tabulka G.5: Výsledky časů na 10-prutové konstrukci (kompilace)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	1,2487	1,2330	1,2492	1,2370	1,2595	1,2526	1,2405	1,2360	1,2373	1,2339	1,2330	1,2595	1,2428
02a	0,1419	0,1411	0,1409	0,1409	0,1409	0,1400	0,1391	0,1404	0,1421	0,1384	0,1384	0,1421	0,1406
02b	0,1301	0,1261	0,1243	0,1260	0,1281	0,1269	0,1257	0,1258	0,1261	0,1281	0,1243	0,1301	0,1267
02c	0,1414	0,1388	0,1371	0,1391	0,1428	0,1435	0,1394	0,1386	0,1410	0,1409	0,1371	0,1435	0,1403
02d	0,1493	0,1481	0,1515	0,1486	0,1486	0,1448	0,1482	0,1516	0,1513	0,1485	0,1448	0,1516	0,1491
02e	1,5218	1,5168	1,5045	1,5197	1,5192	1,5136	1,5132	1,5238	1,5187	1,5246	1,5045	1,5246	1,5176
03a	0,2834	0,2826	0,2809	0,2776	0,2826	0,2837	0,2820	0,2836	0,2835	0,2815	0,2776	0,2837	0,2822
03b	0,2995	0,2994	0,3016	0,3000	0,2969	0,2998	0,3036	0,3004	0,3037	0,3045	0,2969	0,3045	0,3010
03c	0,2865	0,2877	0,2881	0,2886	0,2858	0,2898	0,2886	0,2877	0,2909	0,2876	0,2858	0,2909	0,2881
03d	0,3270	0,3274	0,3322	0,3279	0,3248	0,3272	0,3274	0,3275	0,3314	0,3260	0,3248	0,3322	0,3279
03e	1,6034	1,6065	1,5992	1,6048	1,6463	1,6030	1,5932	1,6017	1,6073	1,5981	1,5932	1,6463	1,6063

Tabulka G.6: Výsledky časů na 25-prutové konstrukci (kompilace)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	3,5606	3,4531	3,4822	3,6186	3,7159	3,4623	3,4078	3,5303	3,6622	3,4872	3,4078	3,7159	3,5380
02a	1,1749	1,1839	1,2971	1,1622	1,2564	1,2204	1,1473	1,1503	1,2265	1,1785	1,1473	1,2971	1,1997
02b	1,0550	1,0509	1,0547	1,0484	1,0448	1,0463	1,0495	1,0524	1,0553	1,0492	1,0448	1,0553	1,0506
02c	1,0372	1,0447	1,0383	1,0415	1,0431	1,0410	1,0466	1,0474	1,0431	1,0412	1,0372	1,0474	1,0424
02d	1,0574	1,0677	1,0622	1,0590	1,0683	1,0618	1,0624	1,0577	1,0640	1,0617	1,0574	1,0683	1,0622
02e	2,6324	2,5946	2,6034	2,6031	2,6085	2,6027	2,6111	2,6178	2,6150	2,6126	2,5946	2,6324	2,6101
03a	0,6637	0,6603	0,6624	0,6638	0,6665	0,6611	0,6628	0,6627	0,6633	0,6726	0,6603	0,6726	0,6639
03b	0,7113	0,7101	0,7045	0,7087	0,7112	0,7107	0,7173	0,7079	0,7097	0,7095	0,7045	0,7173	0,7101
03c	0,6341	0,6333	0,6388	0,6374	0,6322	0,6311	0,6363	0,6346	0,6331	0,6374	0,6311	0,6388	0,6348
03d	0,7466	0,7461	0,7463	0,7458	0,7449	0,7479	0,7437	0,7465	0,7470	0,7453	0,7437	0,7479	0,7460
03e	2,1117	2,1058	2,1126	2,1060	2,1023	2,1119	2,1007	2,1180	2,1194	2,0968	2,0968	2,1194	2,1085

Tabulka G.7: Výsledky časů na 72-prutové konstrukci (kompilace)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
01	1,2338	1,3780	1,3560	1,2796	1,5017	1,3030	1,3279	1,1706	1,3006	1,2994	1,1706	1,5017	1,3151
02a	0,3056	0,3093	0,3073	0,3076	0,3093	0,3072	0,3040	0,3087	0,3103	0,3109	0,3040	0,3109	0,3080
02b	0,2915	0,2914	0,2940	0,2959	0,2937	0,2934	0,2929	0,2930	0,2943	0,2930	0,2914	0,2959	0,2933
02c	0,2780	0,2799	0,2796	0,2759	0,2778	0,2799	0,2731	0,2773	0,2746	0,2797	0,2731	0,2799	0,2776
02d	0,3018	0,2944	0,2979	0,2952	0,2954	0,2957	0,2942	0,2955	0,2955	0,2981	0,2942	0,3018	0,2964
02e	1,8484	1,8350	1,8572	1,8573	1,8336	1,8476	1,8387	1,8596	1,8406	1,8393	1,8336	1,8596	1,8457
03a	0,4570	0,4548	0,4527	0,4612	0,4538	0,4542	0,4541	0,4548	0,4589	0,4550	0,4527	0,4612	0,4556
03b	0,4769	0,4903	0,4723	0,4763	0,4734	0,4755	0,4814	0,4757	0,4744	0,4782	0,4723	0,4903	0,4774
03c	0,4482	0,4523	0,4469	0,4409	0,4420	0,4472	0,4459	0,4469	0,4472	0,4482	0,4409	0,4523	0,4466
03d	0,5153	0,5153	0,5144	0,5098	0,5143	0,5126	0,5149	0,5162	0,5161	0,5177	0,5098	0,5177	0,5147
03e	1,9068	1,9127	1,9129	1,9058	1,9092	1,9155	1,9181	1,9117	1,8999	1,9061	1,8999	1,9181	1,9099

Tabulka G.8: Výsledky časů na 52-prutové konstrukci (kompilace)

	1	2	3	4	5	6	7	8	9	10	min	max	průměr
10-bar	0,0033	0,0033	0,0033	0,0033	0,0033	0,0033	0,0034	0,0033	0,0033	0,0034	0,0033	0,0034	0,00333
25-bar	0,0171	0,0187	0,0165	0,0166	0,0166	0,0171	0,0172	0,0179	0,0172	0,0164	0,0164	0,0187	0,01713
72-bar	0,2341	0,2495	0,2372	0,2364	0,2282	0,2292	0,2345	0,2358	0,2291	0,2370	0,2280	0,2495	0,2348
52-bar	0,0755	0,0753	0,0769	0,0757	0,0753	0,0754	0,0763	0,0755	0,0752	0,0761	0,0752	0,0769	0,07572

Tabulka G.9: Výsledky časů v jazyce C++

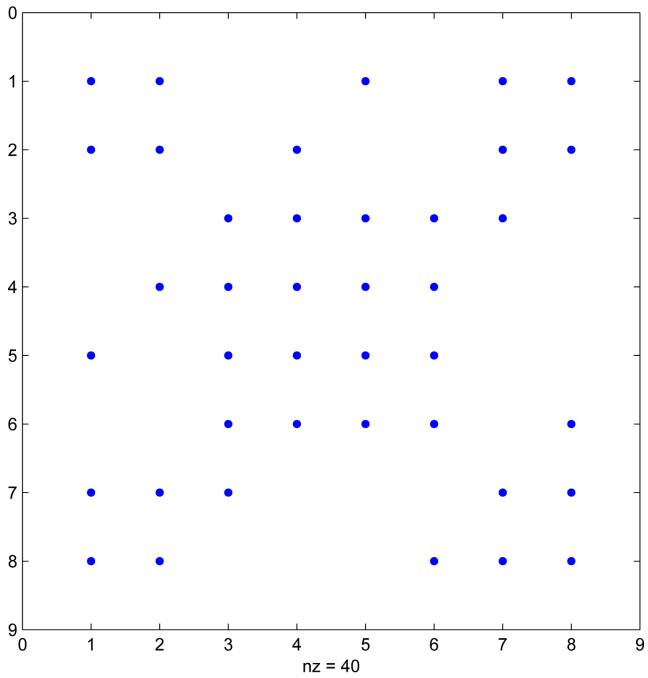
	1	2	3	4	5	6	7	8	9	10	min	max	průměr
10-bar	0,0046	0,0061	0,0049	0,0051	0,0053	,0048	0,0046	0,0052	0,0044	0,0060	0,0044	0,0061	0,0051
25-bar	0,0117	0,0119	0,0114	0,0119	0,0113	0,0115	0,0131	0,0113	0,0114	0,0120	0,0113	0,0131	0,0118
72-bar	0,0396	0,0405	0,0385	0,0387	0,0369	0,0390	0,0460	0,0385	0,0390	0,0387	0,0369	0,0460	0,0395
52-bar	0,0187	0,0153	0,0200	0,0232	0,0272	0,0151	0,0177	0,0186	0,0207	0,0243	0,0151	0,0272	0,0201

Tabulka G.10: Výsledky časů v jazyce C++ s použitím knihovny LAPACK++

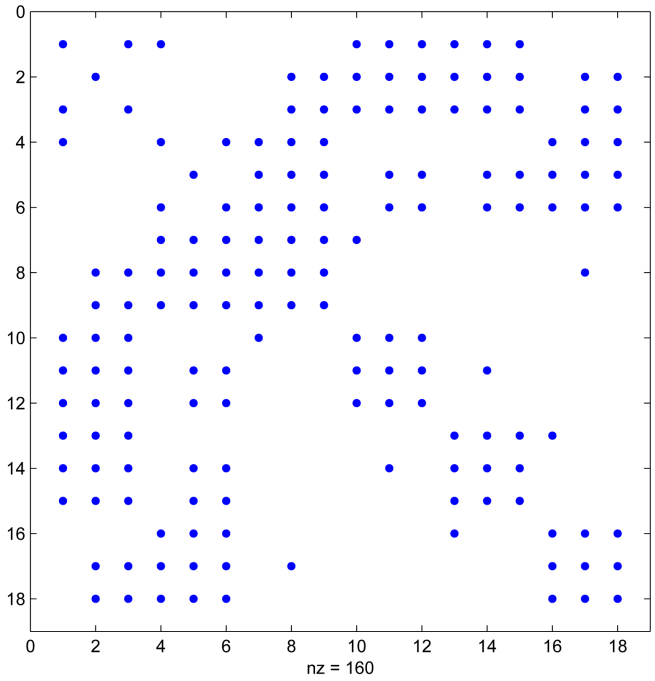
Příloha H

Znázornění nenulových prvků v řídkých submaticích tuhosti

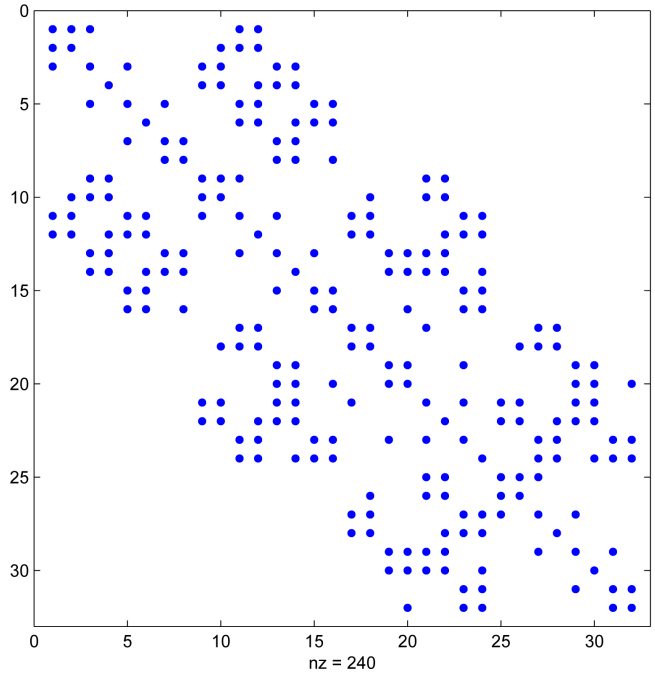
V konstrukcích uvedených v kapitole 6 jsou takové podpory, kde jsou posuny v nich nulové. Z rovnice 2.14 v pododdíle 2.3 tedy vypadne člen $K_{up} \cdot \bar{u}$, jelikož $\bar{u} = 0$, a pro výpočet u pak zůstane $K_{uu} \cdot u = f$. K_{uu} je submatice znázorněná na obrázcích H.1 až H.4.



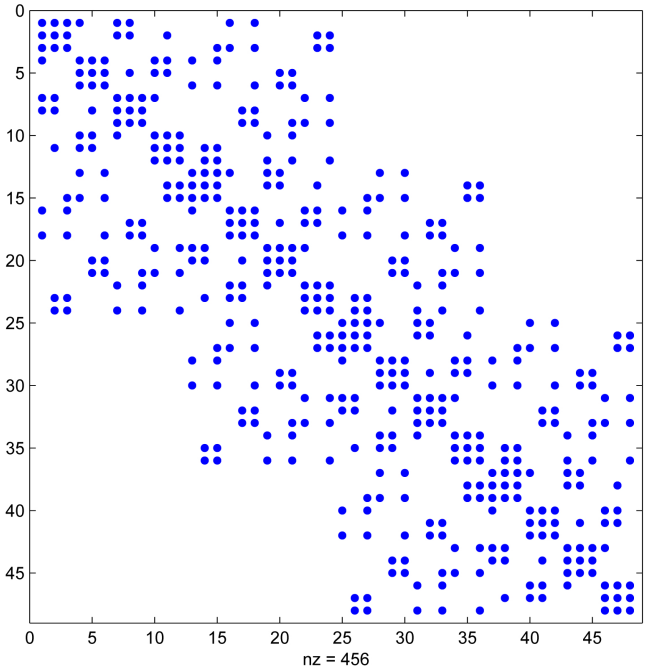
Obrázek H.1: Řídká submatice tuhosti pro 10-prutovou konstrukci



Obrázek H.2: Řídká submatice tuhosti pro 25-prutovou konstrukci



Obrázek H.3: Řídká submatice tuhosti pro 52-prutovou konstrukci



Obrázek H.4: Řídká submatice tuhosti pro 72-prutovou konstrukci

Příloha I

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy.