TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

DISERTAČNÍ PRÁCE

# Aplikace metody konečných prvků v problémech piezoelektrických a feroelektrických struktur

2004

Josef Novák

# TECHNICKÁ UNIVERZITA V LIBERCI

**Fakulta mechatroniky a mezioborových inženýrských studií**


Studijní program: P 2612 Elektrotechnika a informatika
Studijní obor: 3901V025 Přírodovědné inženýrství

# Aplikace metody konečných prvků v problémech piezoelektrických a feroelektrických struktur

Disertační práce


Ing. Josef Novák

Školitel: Doc. Dr. Ing. Jiří Maryška, CSc.


Rozsah práce a příloh:

| | |
|---|---|
| Počet stran textu: | 78 |
| Počet příloh: | 5 stran + CD |
| Počet obrázků: | 44 |
| Počet tabulek: | 10 |



Datum: 29. 10. 2004

## Anotace

Disertační práce se zabývá aplikací metody konečných prvků v modelování vybraných problémů piezoelektrických a feroelektrických struktur. Obsahuje fyzikální popis piezoelektrik a feroelektrik, matematický popis modelovaných dějů, odvození slabého řešení problému. Je ukázáno testování úlohy na základních úlohách a aplikace modelu. Model je aplikován na výpočet elektrických a elastických polí v okolí $90^0$ doménové stěny v tenkém filmu feroelektrika a na analýzu mechanických poměrů v piezoelektrickém měniči po jeho polarizaci. Hlavním přínosem je počítačová implementace modelu, jeho testování a aplikace.



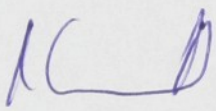
## Abstract

The aim of the thesis is the application of the finite element method in modelling of the problems of piezoelectric and ferroelectric structures. It includes physical description of piezoelectric and ferroelectric materials, mathematical description of modelled phenomenons and derivation of weak formulation of the problem. It deals with testing of the model on simple examples and application of the model. Model is applied in modelling of spatial distributions of electric and elastic fields in ferroelectric thin film with two domains separated by a $90^0$ domain wall and in analysis of mechanical properties in piezoelectric transducer after its polarisation. The main results are computer implementation of the model, its testing and application.

## Prohlášení

Disertační práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací se školitelem.

V Liberci dne 29.10.2004

Podpis: 

## Poděkování

Velmi rád bych poděkoval svému školiteli Doc. Dr. Ing. Jiřímu Maryškovi, CSc. za jeho lidský přístup a cenné rady. Dále bych rád poděkoval všem lidem ve svém blízkém okolí, kteří mě při psaní této práce podporovali.

# Obsah

# Seznam nejdůležitějších symbolů

| | |
|---|---|
| $R$ | množina reálných čísel |
| $L^2(\Omega)$ | Lebesgueův prostor funkcí na oblasti $\Omega$ integrovatelných s druhou mocninou |
| $C(\Omega)$ | Prostor spojitých funkcí na oblasti $\Omega$ |
| $C^i(\Omega)$ | Prostor funkcí na oblasti $\Omega$ se spojitými derivacemi do i-tého řádu |
| $\Omega$ | souvislá oblast v $R^3$, ve které je úloha řešena |
| $\Gamma$ | hranice oblasti $\Omega$ |
| $\mathbf{T}$ | tenzor elastických napětí |
| $\mathbf{S}$ | tenzor deformace |
| $\mathbf{E}$ | vektor elektrického pole |
| $\mathbf{D}$ | vektor elektrické indukce |
| $\mathbf{P}$ | vektor polarizace |
| $\mathbf{u}$ | vektor posunutí |
| $\varphi$ | elektrický potenciál |
| $\mathbf{c}$ | tenzor elastických modulů |
| $\mathbf{s}$ | tenzor elastických koeficientů |
| $\mathbf{e}$ | tenzor piezoelektrických modulů |
| $\mathbf{d}$ | tenzor piezoelektrických koeficientů |
| $\varepsilon$ | tenzor permitivity |

# Kapitola 1

# Úvod

Výzkum vlastností piezoelektrických prvků a jejich aplikace má na Technické univerzitě v Liberci dlouholetou tradici, která je spojena především se jménem prof. Jiřího Zelenky. O dosahovaných výsledcích svědčí i úspěšnost při získávání prostředků na financování výzkumu, z nejvýznamnějších jmenujme MŠMT VS 96006 (Fousek, Nosek), MSM 242200002 (Nosek).

Vzhledem k výjimečným vlastnostem feroelektrických a piezoelektrických látek roste zájem o jejich využití v reálných aplikacích a stává se stále aktuálnějším další výzkum jejich vlastností. Z aplikací můžeme jmenovat například využití v oblasti paměťových médií, kdy při zvyšování kapacity velkou roli hraje jejich fyzická velikost, piezoelektrické měniče z piezoelektrických keramik užívané v řadě mechatronických systémů nebo laditelné obvodové prvky.

Vlastnosti paměťových médií z tenkých feroelektrických vrstev jsou z velké části limitovány chováním domén a doménových stěn, které tvoří vlastní paměťové buňky. Podstatným vlivem je rozložení elektrických a elastických polí, které na tyto struktury působí.

Praktické nasazení piezoelektrických měničů je omezeno technologickými možnostmi jejich výroby. Při polarizaci měničů, která je důležitá pro jejich funkci, může vlivem změn v strukturní mřížce docházet ke vzniku mechanických defektů.

Analýza všech uvedených jevů za použití prostředků současné výpočetní techniky a numerických metod je velmi efektivním nástrojem ke zlepšení parametrů piezoelektrických a feroelektrických prvků. Oblasti zkoumání těchto vlastností za pomoci modelů byla věnována pozornost již před několika desítkami let, jak je patrné např. z [1]. V této práci je publikován model popisující rozložení elektrického pole v okolí devadesátistupňové doménové stěny. Principem modelu je nahrazení domén a jejich rozhraní sítí vodičů s vodivostí rozdílnou ve dvou na sebe kolmých směrech.

Podobný model je uveden i v [2], kde je ale již na základě energetických úvah popsán princip přepolarizace domén. V práci [3] je uveden model pohybu doménových stěn v PZT keramikách ve slabém elektrickém poli. Tento je založen na popisu jevu obyčejnou diferenciální rovnicí - pohybovou rovnicí. Jsou též samozřejmě publikovány práce, viz [4], ve kterých je feroelektrikum popsáno pomocí svého náhradního elektrického obvodu složeného z diskrétních R, L, C prvků.

S rozvojem možností výpočetní techniky byly v posledních letech publikovány modely, které již využívají metodu konečných diferencí [5] či metodu konečných prvků [6], [7]. V těchto pracech je ale pro výpočet užito komerčního MKP systému ANSYS, který má implementovány základní moduly pro práci s piezoelektrickými strukturami. Tento software má velmi příjemné uživatelské rozhraní, neumožňuje ale přímo zasahovat do implementovaného programového kódu a upravovat ho pro speciální účely.

Přínosem a cílem disertační práce je implementace vlastního programového kódu primární formulace MKP a jeho aplikace při výzkumu vlastností feroelektrických struktur a optimalizaci výrobního procesu piezoelektrických měničů. Konkrétně se jedná o prostorová rozložení elektrických a mechanických polí v okolí doménových rozhraní a minimalizaci elastických napětí při polarizaci piezoelektrických měničů.

Obsah disertační práce je koncipován s ohledem na hlavní přínosy a cíle práce, kterým je věnována největší pozornost, do následujících kapitol.

Nejprve jsou ve druhé kapitole popsány základní fyzikální vlastností krystalů, které užijeme při návrhu modelu. Popsány jsou především vlastnosti, kterými se projevují piezoelektrické a feroelektrické materiály. Podrobně popíšeme dielektrické, elastické a piezoelektrické vlastnosti.

Třetí kapitola je zaměřena na matematickou formulaci řešených a odvození jejich slabého řešení. V první části kapitoly je popsána matematickofyzikální podstata modelovaných dějů. Poté je formulováno slabé řešení problému definovaném v první části. Závěrečná část kapitoly je věnována aproximaci řešení metodou konečných prvků.

Čtvrtá kapitola obsahuje stručný popis implementace navrženého modelu. Popsány jsou především formáty vstupních a výstupních souborů a hlavní datové struktury.

Úkolem páté kapitoly je otestování všech částí modelu na jednoduchých úlohách tak, aby byla zaručena relevantnost výsledků, které model poskytuje. Vzhledem k primární MKP formulaci, která je užita k sestavení modelu budou testovány především primární veličiny (elektrický potenciál, mechanická posunutí).

Disertační práce je uplatněna ve dvou z mnoha aplikací, které lze postihnout obdobným matematicko-fyzikálním popisem. Jedna je z oblasti teoretického materiálového výzkumu feroelektrických tenkých filmů s jednou nebo více doménovými stěnami. Druhá aplikace je tématem z průmyslové praxe a je zaměřena na optimalizaci technologických parametrů výroby piezoelektrických měničů. Obě témata úzce navazují na institucionální výzkum Fakulty mechatroniky a mezioborových inženýrských studií.

Šestá kapitola se zabývá výpočtem elektrických a elastických polí na doménovém rozhraní feroelektrik. V důsledku jejich dielektrických, piezoelektrických a elastických vlastností patří mezi v současné době nejvíce zkoumané materiály. Zatížení vzorků materiálu obsahujících různé doménové stavy elektrickým polem či elastickým napětím vede k rozložení elektrických a elastických polí, která vykazují značnou prostorovou nehomogenitu. Je nanejvýš vhodné, uvažovat zásadní roli prostorového rozložení elektrických a elastických polí na jejich materiálové vlastnosti.

Sedmá kapitola je zaměřena na analýzu poměrů v piezoelektrickém měniči při jeho polarizaci. V důsledku nehomogenního rozložení elektrického pole v piezoelektrických měničích dochází při jejich polarizaci vlivem změny strukturní mřížky materiálu ke vzniku elastických deformací a napětí. Tato elastická napětí jsou zodpovědná za mechanické defekty v měniči.

Eliminace nejvýraznějších nehomogenit v rozložení elektrického pole povede k rovnoměrnější polarizaci a tím i k rovnoměrnějšímu rozložení elastických napětí, k menšímu riziku vzniku strukturních defektů.

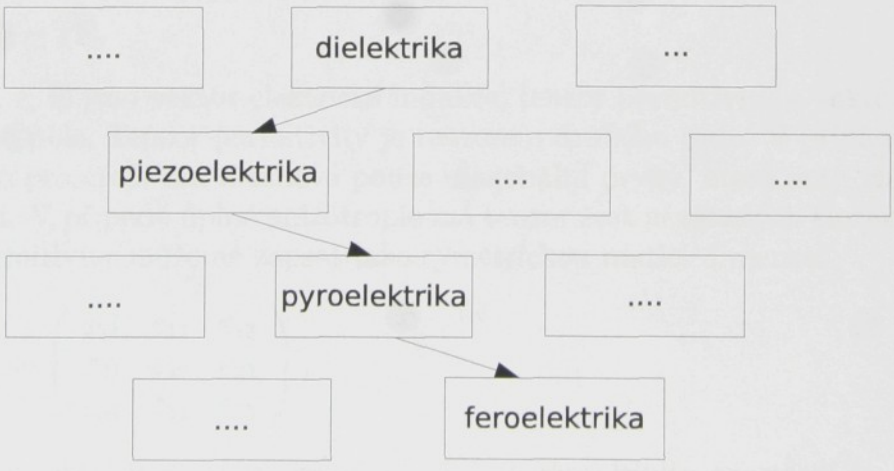
# Kapitola 2

# Fyzikální vlastnosti krystalů

Obsahem této kapitoly je popis základních fyzikálních vlastností krystalů, které užijeme při návrhu modelu. Popíšeme dielektrické, elastické a piezoelektrické vlastnosti. Vynecháme popis vlastností, které sice piezoelektrické a feroelektrické materiály vykazují, ale nebudou implementovány v modelu. V této kapitole je čerpáno z publikací [8], [9], [10], [11], kde lze nalézt další podrobnosti.

## 2.1 Jevy charakterizující piezoelektrické a feroelektrické materiály

Schématické rozdělění vlastností, které materiály z okruhu našeho zájmu vykazují je naznačeno na obr. 2.1. Schéma nám říká, že pouze některé látky ze




Obrázek 2.1: Souvislosti mezi jednotlivými skupinami materiálů

skupiny dielektrik vykazují piezoelektrický jev, viz [25]. Stejně tak pouze některá skupina látek ze skupiny piezoelektrických vykazuje jev pyroelektrický. A jen část pyroelektrik má vlastnosti, které označujeme za feroelektrické, podrobnosti např. v [23].

Látkám, ve kterých dochází ke změně polarizace také na základě mechanického působení, říkáme látky piezoelektrické. Podrobnosti jsou uvedeny v části 2.4.

Pyroelektrické látky jsou takové látky, u nichž lze pozorovat polarizaci i při nulovém přiloženém elektrickém poli. Takovou polarizaci nazýváme spontánní. Změny spontánní polarizace lze v pyroelektrikách dosáhnout změnou teploty, viz [25]. Tento jev lze pro malé změny teploty popsat rovnicí, viz [9]

$$\Delta P_i = p_i \Delta\Theta, \quad i = 1, 2, 3, \tag{2.1}$$

kde $\Delta P_i$ jsou změny složek vektoru elektrické polarizace, $p_i$ nazýváme pyroelektrickými koeficienty a $\Delta\Theta$ je změna teploty.

Feroelektrické látky jsou takové, u nichž lze směr spontánní polarizace změnit působením elektrického pole.

Většina feroelektrik má teplotu přechodu, tzv. Curieho teplotu $T_c$, při jejímž překročení dochází k vymizení spontánní polarizace.

## 2.2 Dielektrické vlastnosti

Působením vnějšího elektrického pole na dielektrikum dochází k elektrické indukci. Je definován vztah mezi vektorem elektrického pole a vektorem elektrické indukce

$$\mathbf{D} = \varepsilon \mathbf{E}, \tag{2.2}$$

kde $\mathbf{D}$, $\varepsilon$, $\mathbf{E}$ jsou vektor elektrické indukce, tenzor permitivity a vektor elektrického pole. Tenzor permitivity je tenzorem druhého řádu. V případě izotropního prostředí má nenulové pouze diagonální prvky, které mají shodnou velikost. V případě úplné anizotropie má tenzor šest nezávislých složek. Tenzor permitivity můžeme zapsat jako symetrickou matici dimenze:

$$\varepsilon = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \varepsilon_{33} \end{pmatrix}. \tag{2.3}$$

Tedy platí, že $\varepsilon_{ij} = \varepsilon_{ji}$ pro $i, j = 1, 2, 3$. Permitivita prostředí je definována jako součin

$$\varepsilon = \varepsilon_0 \varepsilon_r, \tag{2.4}$$

kde $\varepsilon_0$ je permitivita vakua a $\varepsilon_r$ je relativní permitivita prostředí. Vektor elektrického pole je definován jako záporný gradient elektrostatického potenciálu $\varphi$, tedy vztahem

$$\mathbf{E} = -\nabla\varphi. \tag{2.5}$$

Elektricky nevodivé látky - dielektrika - obsahují stejně jako vodiče nabité částice, jejichž náboje se v makroskopickém měřítku vzájemně ruší. Dielektrika se proto jeví jako elektricky neutrální. Náboje však nejsou v dielektriku vázány na zcela neproměnné polohy. Jejich rozložení se může působením vnějšího elektrického pole měnit. Tím se poruší i vzájemná kompenzace kladných a záporných nábojů. Tento jev nazýváme polarizací dielektrika. Kvantifikaci tohoto jevu vyjadřujeme pomocí vektoru elektrické polarizace $\mathbf{P}$. Elektrická polarizace vyvolaná vnějším elektrickým polem $\mathbf{E}$ je definovaná jako

$$\mathbf{P} = \mathbf{D} - \varepsilon\mathbf{E} = \varepsilon_0\kappa\mathbf{E}, \tag{2.6}$$

kde $\kappa = \varepsilon_r - 1$ je dielektrická susceptibita.

Působí-li na polarizovatelný krystal elektrické pole $\mathbf{E_a}$, je v krystalu indukováno vnitřní, tzv. depolarizační pole $\mathbf{E_i}$. Výsledné elektrické pole je dáno součtem těchto elektrických polí:

$$\mathbf{E} = \mathbf{E_a} + \mathbf{E_i}. \tag{2.7}$$

U paramagnetických a diamagnetických látek, u kterých dielektrická susceptibilita je nízká, je vliv krystalu na elektrické pole v souladu se vztahem 2.6 nízký. Toto však neplatí pro feromagnetické látky, kde dielektrická susceptibilita je vysoká. Elektrická pole $\mathbf{E_a}$ i $\mathbf{E_i}$ jsou často stejného řádu. Velikost $\mathbf{E_i}$ je závislé na tvaru krystalu.

Platnost výše uvedených vztahů je zaručena pro krystaly, které se chovají jako ideální izolátor. Tedy pro krystaly, pro které jejich elektrický odpor je $R = \infty$. Reálně dochází vlivem příměsí v každém materiálu k pohybu elektrického náboje, který způsobuje alespoň minimální vodivost.

Ke změně elektrického pole v objemové jednotce dielektrika je třeba vynaložit práci, která je definována jako

$$dw_D = E_i dD_i. \tag{2.8}$$

Provedeme-li integraci předchozího vztahu v objemu uvažovaného vzorku, získáme výraz pro hustotu elektrostatické energie:

$$w_D = \frac{1}{2}E_i D_i. \tag{2.9}$$

## 2.3 Elastické vlastnosti

Plošné síly působící na jednotkovou plochu povrchu elementu budeme, viz např. [11], značit $\tilde{\mathbf{T}}$.

K plnému určení stavu napjatosti stačí znát vektory elastických napětí ve třech vzájemně kolmých rovinách procházejících daným bodem. Roviny zpravidla volíme tak, aby jejich normály byly rovnoběžné s ortogonálním systémem os $x_i$, $i = 1, 2, 3$. Každý z vektorů elastických napětí potom můžeme vyjádřit jako

$$\tilde{\mathbf{T}}^{\mathbf{i}} = T_{i1}\mathbf{n_1} + T_{i2}\mathbf{n_2} + T_{i3}\mathbf{n_3}, \quad i = 1, 2, 3, \tag{2.10}$$

kde $\mathbf{n_i}$ jsou jednotkové vektory rovnoběžné s osami $x_i$. Napjatost tělesa v okolí bodu je tedy popsána devíti složkami $T_{ij}$. Tyto složky označujeme jako složky tenzoru napětí, např. [11].

Elastické napětí lze vyjádřit ve formě symetrického tenzoru druhého řádu. Tenzor napětí, který označíme $\mathbf{T}$, zapíšeme

$$\mathbf{T} = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix}. \tag{2.11}$$

Ze zákona zachování momentů sil, viz [11] plyne symetrie tenzoru napětí, platí tedy $T_{ij} = T_{ji} \quad i, j = 1, 2, 3$. Má tedy šest nezávislých složek.

Vlivem napětí dojde k posunutí elementárních částí, které vyjadřujeme vektorem posunutí

$$\mathbf{u} = (u_1, u_2, u_3). \tag{2.12}$$

Složky vektoru posunutí vyjadřují posunutí ve směrech os $x_1, x_2, x_3$ ortogonálního systému.

Tato posunutí vyvolají deformaci, která je reprezentována symetrickým tenzorem druhého řádu

$$\mathbf{S} = \begin{pmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{pmatrix}. \tag{2.13}$$

Tenzor je symetrický a proto platí $S_{ij} = S_{ji} \quad i, j = 1, 2, 3$. Má tedy také šest nezávislých složek, stejně jako tenzor napětí.

Budeme předpokládat, že gradienty posunutí budou dostatečně malé. Potom můžeme jednotlivé složky tenzoru deformace vyjádřit jako

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{2.14}$$

V případě obecné napjatosti je napětí a deformace v každém bodě napjatého tělesa charakterizována tenzorem napětí a tenzorem deformace. Lineární teorie pružnosti předpokládá, že každá složka tenzoru napětí je lineární funkcí všech složek tenzoru deformace. Můžeme tedy psát, že

$$T_{ij} = c_{ijkl} S_{kl}. \tag{2.15}$$

Koeficienty $c_{ijkl}$ nazýváme elastickými moduly. Jsou složkami tenzoru čtvrtého řádu. Tenzor čtvrtého řádu má obecně 81 prvků. Vzhledem k symetrii tenzoru napětí i deformace a z energetických úvah, viz [11] plyne, že tenzor elastických modulů je symetrický a platí, že

$$c_{ijkl} = c_{ijlk} = c_{jikl} = c_{jilk}. \tag{2.16}$$

Celkem je tedy nejvýše 21 nezávislých elastických modulů. Se vzrůstající symetrií krystalové mříže počet nezávislých složek tenzoru elastických modulů dále klesá, viz [8]. Tenzor elastických modulů můžeme zapsat jako symetrickou matici

$$\mathbf{c} = \begin{pmatrix} c_{1111} & c_{1122} & c_{1133} & c_{1123} & c_{1131} & c_{1112} \\ c_{2211} & c_{2222} & c_{2233} & c_{2223} & c_{2231} & c_{2212} \\ c_{3311} & c_{3322} & c_{3333} & c_{3323} & c_{3331} & c_{3312} \\ c_{3211} & c_{3222} & c_{3233} & c_{3223} & c_{3231} & c_{3212} \\ c_{3111} & c_{3122} & c_{3133} & c_{3123} & c_{3131} & c_{3112} \\ c_{2111} & c_{2122} & c_{2133} & c_{2123} & c_{2131} & c_{2112} \end{pmatrix}. \tag{2.17}$$

Vztah mezi elastickým napětím a deformací lze též vyjádřit takto

$$S_{ij} = s_{ijkl} T_{kl}, \tag{2.18}$$

kde $s_{ijkl}$ jsou složky tenzoru elastických koeficientů. Tenzor elastických koeficientů má též nejvýše 21 nezávislých složek. Užijeme-li maticového zápisu nezávislých složek tenzorů elastických koeficientů a elastických modulů, potom ze vztahů 2.15 a 2.18 vyplývá

$$\mathbf{s} = \mathbf{c}^{-1}. \tag{2.19}$$

Tenzor elastických modulů lze interpretovat tak, že charakterizuje tuhost materiálu, tenzor elastických koeficientů chrakterizuje poddajnost materiálu.

Při změně deformace se uvnitř objemové jednotky vykoná elastická práce definovaná vztahem

$$dw_S = T_{ij} dS_{ij} \tag{2.20}$$

Integrujeme-li uvedený vztah v objemu uvažovaného vzorku, získáme výraz pro hustotu elastické energie

$$w_D = \frac{1}{2} T_{ij} S_{ij}. \tag{2.21}$$

## 2.4 Piezoelektrické vlastnosti

U piezoelektrických látek je elektrická polarizace vyvolána nejen působením elektrického pole, ale též působením elastického napětí, resp. v důsledku deformace. Tento jev nazýváme přímým piezoelektrickým jevem a můžeme jej vyjádřit jako

$$P_i = d_{ijk} T_{jk}, \tag{2.22}$$

nebo jako

$$P_i = e_{ijk} S_{jk}, \tag{2.23}$$

kde $d_{ijk}$, $e_{ijk}$ jsou prvky tenzoru piezoelektrických koeficientů resp modulů. Obdobně lze vyjádřit i tzv. převrácený piezoelektrický jev, který vyvolává elastické napětí resp. deformaci v důsledku působení elektrického pole na piezoelektrický materiál

$$T_{ij} = -e_{kij} E_k, \tag{2.24}$$

$$S_{ij} = d_{kij} E_k. \tag{2.25}$$

Piezoelektrické moduly a koeficienty jsou složkami tenzoru třetího řádu. Tenzory piezoelektrických modulů a koeficientů jsou symetrické a mají maximálně 18 nezávislých složek. Můžeme je tedy výhodně vyjádřit formou matice, např. tenzor piezoelektrických modulů

$$\mathbf{e} = \begin{pmatrix} e_{111} & e_{122} & e_{133} & e_{123} & e_{131} & e_{112} \\ e_{211} & e_{222} & e_{233} & e_{223} & e_{231} & e_{212} \\ e_{311} & e_{322} & e_{333} & e_{323} & e_{331} & e_{312} \end{pmatrix}. \tag{2.26}$$

Mezi piezoelektrickými koeficienty a moduly platí vztahy

$$e_{ijk} = d_{ilm} c_{lmjk} \quad i, j, k, l, m = 1, 2, 3 \tag{2.27}$$

$$d_{ijk} = e_{ilm} s_{lmjk} \quad i, j, k, l, m = 1, 2, 3 \tag{2.28}$$

## 2.5 Piezoelektrické stavové rovnice

Jak již bylo uvedeno výše, u piezoelektrických látek lze elektrickou polarizaci vyvolat nejen působením elektrického pole, ale též působením elastického napětí resp. v důsledkudeformace. Vzájemný vztah mezi těmito dvěma silovými poli lze vyjádřit takzvanými piezoelektrickými stavovými rovnicemi. Při odvození těchto vztahů se vychází např. z vnitřní energie, jak je ukázáno v [8], či z jiného termodynamického potenciálu.

Nebudeme zde tyto rovnice odvozovat, pouze se omezíme na konstatování, že budeme uvažovat lineární případ, tedy že koeficienty vystupující v

těchto rovnicích jsou nezávislé na velikosti nezávisle proměnných a budeme uvažovat pouze izotermické a adiabatické děje. Potom se piezoelektrické stavové rovnice redukují na čtyři možné dvojice, kde rovnice ve dvojici popisují vždy přímý a převrácený piezoelektrický jev. Tyto dvojice můžeme zapsat jako

$$T_{ij} = c^S_{ijkl} S_{kl} - e_{kij} E_k, \tag{2.29}$$
$$D_i = e_{ijk} S_{jk} + \varepsilon^S_{ij} E_j, \tag{2.30}$$

$$T_{ij} = c^D_{ijkl} S_{kl} - d_{kij} D_k, \tag{2.31}$$
$$E_i = -d_{ijk} S_{jk} + \beta^S_{ij} D_j, \tag{2.32}$$

$$S_{ij} = s^E_{ijkl} T_{kl} + d_{kij} E_k, \tag{2.33}$$
$$D_i = d_{ijk} T_{jk} + \varepsilon^T_{ij} E_j, \tag{2.34}$$

$$S_{ij} = s^D_{ijkl} T_{kl} + g_{kij} D_k, \tag{2.35}$$
$$E_i = -g_{ijk} T_{jk} + \beta^T_{ij} E_j, \tag{2.36}$$

kde $\beta$, $\mathbf{g}$ jsou tenzory impermeability a piezoelektrických koeficientů. Horní indexy u koeficientů v rovnicích znamenají veličinu, pro jejíž konstantní hodnotu jsou definovány. Přitom platí, že

$$e^D_{ijk} = d^E_{ijk} = d_{ijk}, \tag{2.37}$$
$$d^D_{ijk} = g^E_{ijk} = g_{ijk}, \tag{2.38}$$
$$e^E_{ijk} = e_{ijk}. \tag{2.39}$$

Povšimněme si, že rovnice 2.29, 2.31 jsou v podstatě pouze rozšířením rovnice 2.15. Stejný závěr můžeme udělat i pro rovnice 2.30, 2.34 a 2.2.

Uvedené piezoelektrické stavové rovnice budeme dále využívat při formulaci úlohy. Budeme však muset respektovat všechna omezení, která limitují platnost lineárních piezoelektrických stavových rovnic. Přesto jsou však pro řadu aplikací, ke kterým je model určen, lineární stavové rovnice vyhovující.

# Kapitola 3

# Matematická formulace úlohy

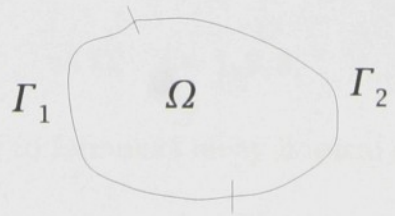
Tato kapitola je věnována matematicko-fyzikálním popisu řešených úloh a odvození jejich slabého řešení. V první části kapitoly je shrnut matematicko-fyzikální popis modelovaných dějů. V druhé části kapitoly je formulováno slabé řešení problému definovaného v první části. Závěrečná část kapitoly je věnována aproximaci řešení metodou konečných prvků.

## 3.1 Matematicko-fyzikální popis elektrického a elastického pole

Chování elektrického pole v obecném anizotropním dielektriku popisují Maxwellovy rovnice, např. [10]. Nechť $\Omega \subset R^3$ je souvislá oblast, ve které je úloha řešena. Její hranici označme $\Gamma$ a nechť je hranice lipschitzovská, viz např. [11]. Vlastnosti elektrického pole v oblasti $\Omega$ popisuje jedna z Maxwellových rovnic, parciální diferenciální rovnice

$$\nabla \cdot \mathbf{D} = \frac{\partial \mathrm{D_i}}{\partial x_i} = 0, \quad \text{na } \Omega \quad i = 1, 2, 3, \tag{3.1}$$

kde $\mathbf{D}$ je vektor elektrické indukce.




Obrázek 3.1: Oblast $\Omega$ a rozdělení její hranice.

Předpokládáme, že hranice oblasti je rozdělena na dvě části $\Gamma = \Gamma_1 \cup \Gamma_2$, na kterých jsou zadány okrajové podmínky (obr. 3.1)

Na části hranice $\Gamma_1$ jsou definovány Dirichletovy okrajové podmínky a jsou značeny indexem $D$, na části hranice $\Gamma_2$ jsou definovány Nemannovy okrajové podmínky a jsou značeny indexem $N$ ($n_j$ je j-tá složka jednotkového normálového vektoru na hranici oblasti $\Omega$).

$$\varphi = \varphi_D \quad \text{na} \quad \Gamma_1, \tag{3.2}$$

$$\mathrm{D_k n_k} = \mathrm{D_N} \quad \text{na} \quad \Gamma_2. \tag{3.3}$$

kde $\varphi_D$ a $D_N$ jsou dostatečně hladké funkce.

**Poznámka 1:** Pojem dostatečně hladké vymezíme v části 3.2 zavedením prostorů funkcí, ze kterých budou funkce eralizující okrajové podmínky zadávány. Postačující pro formulaci úlohy i pro reálné aplikace je, aby tyto funkce byly po částech spojité.

**Poznámka 2:** V obecných úlohách lze uvažovat i Newtonovu okrajovou podmínku, která je nejfyzikálnější interpretací reality. Přesto jsou na hranici $\Gamma$ definovány pouze Dirichletova a Neumannova okrajová podmínka. Identifikace Newtonovy okrajové podmínky je vzhledem k charakteru úloh, na které je model koncipován, velmi obtížná a navíc pro námi volené aplikace Dirichletovy a Neumannovy okrajové podmínky s dostatečnou přesností vystihují děje na hranicích zkoumaných oblastí. Rovněž jsou Dirichletova a Neumannova okrajová podmínka snadnější při vlastní počítačové implementaci modelu. Ze stejných důvodů není Newtonova okrajová podmínka zadávána ani u elastických rovnic.

Vlastnosti elastického tělesa jsou popsány Newtonovým zákonem zachování sil, např. [10]. V obecném tvaru pro vícedimenzionální oblast lze zákon vyjádřit

$$\nabla \cdot \mathbf{T} = \frac{\partial \mathrm{T_{ij}}}{\partial x_j} = 0 \qquad \text{na } \Omega, \quad i = 1, 2, 3, \tag{3.4}$$

kde $\mathbf{T}$ je tenzor napětí. Pro formulaci úlohy lineární elasticity je nutné zadat ještě okrajové podmínky.

Na části hranice $\Gamma_1$ jsou definovány Dirichletovy okrajové podmínky a jsou značeny indexem $D$, na části hranice $\Gamma_2$ jsou definovány Neumannovy okrajové podmínky a jsou značeny indexem $N$ ($n_j$ je j-tá složka jednotkového

normálového vektoru na hranici oblasti $\Omega$).

$$u_i = u_{iD} \quad i = 1,2,3 \quad \text{na} \quad \Gamma_1, \tag{3.5}$$

$$\mathrm{T_{ij}n_j} = t_{iN} \quad i = 1,2,3 \quad \text{na} \quad \Gamma_2. \tag{3.6}$$

kde $u_{iD}$ a $t_{iD}$ jsou dostatečně hladké funkce. Vlastnosti těchto funkcí vymezíme v části 3.2, viz Poznámka 1.

**Poznámka 3:** Okrajové podmínky 3.2, 3.5 resp. 3.3, 3.6 jsou v předchozím textu definovány na stejných částech hranice $\Gamma$. Jedná se pouze o zjednodušení zadání z důvodu větší přehlednosti odvození formulace řešení úlohy. Obecně mohou být elektrické i elastické okrajové podmínky definovány na různých částech hranice. Tímto způsobem je koncipována i samotná implementace modelu.

Nyní využijeme znalosti materiálových vlastností, které jsou uvedeny v kapitole 2. V rovnici 3.1 lze vektor elektrické indukce $\mathbf{D}$ vyjádřit pomocí vztahu 2.30 pro přímý piezoelektrický jev následovně

$$\nabla \cdot \mathbf{D} = \frac{\partial}{\partial x_k}\left(e_{kij} \cdot \mathrm{S_{ij}} + \varepsilon^{\mathrm{S}}_{\mathrm{kj}} \cdot \mathrm{E_j}\right) = 0. \tag{3.7}$$

Pro zjednodušení zápisu budeme dále užívat zjednodušené značení $\varepsilon^S_{ij} = \varepsilon_{ij}$. Pomocí definičních vztahů 2.5 a 2.14 pro vektor intenzity elektrického pole a pro tenzor deformace a po dosazení do 3.7 odvodíme

$$\nabla \cdot \mathbf{D} = \frac{\partial}{\partial x_k}\left(e_{kij} \cdot \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \varepsilon_{kj} \cdot \frac{\partial \varphi}{\partial x_j}\right) = 0. \tag{3.8}$$

Podobný postup budeme aplikovat i na rovnice 3.4. Tenzor napětí $\mathbf{T}$ vyjádříme z rovnice 2.29

$$\nabla \cdot \mathbf{T} = \frac{\partial}{\partial x_j}\left(c^E_{ijkl} \cdot \mathrm{S_{kl}} - \mathrm{e_{kij}} \cdot \mathrm{E_k}\right) = 0. \tag{3.9}$$

Pro zjednodušení zápisu a nahradíme $c^E_{ijkl} = c_{ijkl}$. Použijeme vztahů 2.5 a 2.14 a dosadíme do 3.9

$$\nabla \cdot \mathbf{T} = \frac{\partial}{\partial x_j}\left(c_{ijkl} \cdot \frac{1}{2}\left(\frac{\partial u_k}{\partial x_l} + \frac{\partial u_l}{\partial x_k}\right) + e_{kij} \cdot \frac{\partial \varphi}{\partial x_k}\right) = 0. \tag{3.10}$$

## 3.2 Slabé řešení

V tomto odstavci zformulujeme slabé řešení úlohy definované v předchozí části. Řešení podobné úlohy je podrobně matematicky formulováno např. v [15] a není cílem disertační práce jeho odvození, proto zde zavedeme pouze základní pojmy nutné k formulaci slabého řešení, nebudeme se zabývat rozborem vlastností samotného slabého řešení. Otázky jednoznačnosti řešení a konvergence aproximací jsou diskutovány např. v [13].

Nejdříve zaveďme potřebné prostory funkcí. Při zavedení značení i pojmů vycházíme z [13].

**Definice 1:** Označme $C_0^{(\infty)}(\overline{\Omega})$ prostor funkcí nekonečněkrát spojitě diferencovatelné v $\overline{\Omega}$.

**Definice 2:** Řekneme, že reálná funkce $u(x)$ je integrovatelná v oblasti $\Omega$ s druhou mocninou, jsou-li konvergentní a jsou-li konečné integrály

$$\int_\Omega u(x)\,dx, \quad \int_\Omega u^2(x)\,dx.$$

Nechť $u(x)$, $v(x)$ jsou dvě funkce integrovatelné s druhou mocninou v oblasti $\Omega$. Jejich skalárním součinem rozumíme číslo

$$(u,v)_\Omega = \int_\Omega u(x)v(x)\,dx. \tag{3.11}$$

Normou funkce $u(x)$ integrovatelné s druhou mocninou v oblasti $\Omega$ rozumíme nezáporné číslo

$$||u|| = \sqrt{(u,u)_\Omega} = \sqrt{\int_\Omega u^2(x)\,dx}. \tag{3.12}$$

Vzdálenost dvou funkcí $u(x)$, $v(x)$ integrovatelných s druhou mocninou v oblasti $\Omega$ rozumíme normu jejich rozdílu

$$\varrho(u,v) = ||u-v|| = \sqrt{\int_\Omega [u(x)-v(x)]^2\,dx}. \tag{3.13}$$

Prostor funkcí integrovatelných s druhou mocninou v oblasti $\Omega$ s metrikou danou předpisem 3.13 nazýváme metrickým prostorem $L_2(\Omega)$. Skalární součin a norma v tomto prostoru jsou dány vztahy 3.11, 3.12.

Prostor funkcí $C_0^{(\infty)}(\Omega)$ s metrikou definovanou vztahem 3.13 tvoří metrický prostor. Ten není úplný. Provedeme-li jeho zúplnění (doplníme prostor o limity všech cauchyovských posloupností z $(C_0^{(\infty)}, \varrho)$, viz [13]), dostaneme Hilbertův prostor $L^2(\Omega)$.

Úplný obal metrického prostoru $(C_0^{(\infty)}, \varrho_1)$ se nazývá *Sobolevův prostor* $W_2^{(1)}$, kde metrika $\varrho_1$ je definována jako

$$\varrho_1(u,v) = \sqrt{\int_\Omega [u(x) - v(x)]^2 + [\nabla u - \nabla v]^2 \, dx}.$$

Pro naši úlohu je oblast $\Omega$ podmnožinou $\mathbf{R^3}$. Sobolevův prostor má tvar

$$W_2^{(1)}(\Omega) = \{\varphi \in C_0^{(\infty)}(\overline{\Omega}) | \varphi \in L_2(\Omega), \nabla\varphi \in [L_2(\Omega)]^3\}.$$

**Poznámka 4:** Prostor $C_0^{(\infty)}(\overline{\Omega})$ jsou funkce z $C_0^{\infty}(\Omega)$, které lze spojitě prodloužit z vnitřku oblasti $\Omega$ na hranici $\partial\Omega$. Dále budeme pracovat z prostoru $W_2^{(1)}$, které mají definované stopy na $\Gamma$. K definici operátoru stopy blíže viz [13], kapitola 30.

Dále označme

$$V(\Omega) = \{\mathbf{v} | \mathbf{v} \in W_2^{(1)}(\Omega), \quad \mathbf{v}|_{\Gamma_1} = 0 \quad ve \quad smyslu \quad stop\}.$$

prostor těch funkcí z $W_2^{(1)}(\Omega)$, které na části hranice $\Gamma_1$ oblasti splňují homogenní Dirichletovu podmínku (ve smyslu stop).

**Poznámka 5:** Integrál přes hranici budeme psát

$$\langle f, g \rangle_\Gamma = \int_\Gamma f g d\Gamma.$$

Nejdříve upravme rovnici pro elektrické pole 3.1. Vynásobme postupně tuto rovnici libovolnými funkcemi $\phi \in V$ a integrujme přes $\Omega$. Dostáváme

$$\int_\Omega \frac{\partial D_i}{\partial x_i} \cdot \phi \, d\Omega = 0. \tag{3.14}$$

Použitím Greenovy věty, viz např. [14], rovnost upravíme na

$$\left(D_i, \frac{\partial \phi}{\partial x_i}\right)_\Omega = \left\langle D_i n_i, \phi \right\rangle_{\Gamma_2}. \tag{3.15}$$

Dosazením za $D_i$ z rovnice 2.30 přímého piezoelektrického jevu a použitím okrajové podmínky 3.3 odvodíme rovnici

$$\left(e_{ijk} S_{jk} + \varepsilon_{ij} E_j, \frac{\partial \phi}{\partial x_i}\right)_\Omega = \left\langle D_N, \phi \right\rangle_{\Gamma_2}. \tag{3.16}$$

Dále dosazením za $E_j$ z rovnice 2.5, za $S_{jk}$ ze vztahu 2.14 a s využitím linearity skalárního součinu odvodíme integrální rovnost

$$\left(e_{ijk}\frac{1}{2}\left(\frac{\partial u_j}{\partial x_k}+\frac{\partial u_k}{\partial x_j}\right),\frac{\partial \phi}{\partial x_i}\right)_\Omega+\left(\varepsilon_{ij}\frac{\partial \varphi}{\partial x_j},\frac{\partial \phi}{\partial x_i}\right)_\Omega=\left\langle D_N,\phi\right\rangle_{\Gamma_2}. \tag{3.17}$$

Obdobným způsobem budeme postupovat i v případě elastické rovnice 3.4. Označme $\mathbf{w}=(w_1,w_2,w_3)\in V^3$, kde složky $w_1,w_2,w_3$ jsou libovolně zvolené. Vynásobme postupně tyto rovnice libovolnými funkcemi $w_i\in V$, $i=1,2,3$, sečtěme je a integrujme přes $\Omega$. Dostáváme

$$\int_\Omega \frac{\partial \mathrm{T_{ij}}}{\partial x_j}\cdot w_i\,d\Omega=0. \tag{3.18}$$

Použijeme Greenovy věty a získáme

$$\left(\mathrm{T_{ij}},\frac{\partial w_i}{\partial x_j}\right)_\Omega=\left\langle \mathrm{T_{ij}}\cdot n_j,w_i\right\rangle_{\Gamma_2}. \tag{3.19}$$

Dosazením za $T_{ij}$ z rovnice 2.29 převráceného piezoelektrického jevu a použitím okrajové podmínky 3.6 upravíme rovnici na tvar

$$\left(c_{ijkl}\cdot \mathrm{S_{kl}}-\mathrm{e_{kij}}\cdot\ \mathrm{E_k},\frac{\partial w_i}{\partial x_j}\right)_\Omega=\left\langle t_{iN},w_i\right\rangle_{\Gamma_2}. \tag{3.20}$$

Vzhledem k symetrii tenzorů $\mathbf{c}$ a $\mathbf{e}$ lze rovnici zapsat jako

$$\left(c_{ijkl}\cdot \mathrm{S_{kl}}-\mathrm{e_{kij}}\cdot\ \mathrm{E_k},\frac{1}{2}\left(\frac{\partial w_i}{\partial x_j}+\frac{\partial w_j}{\partial x_i}\right)\right)_\Omega=\left\langle t_{iN},w_i\right\rangle_{\Gamma_2}. \tag{3.21}$$

využijeme linearitu skalárního součinu a dosadíme za $E_k$ z rovnice 2.5 a za $S_{kl}$ ze vztahu 2.14. Dostáváme integrální rovnost

$$\begin{aligned}&\left(c_{ijkl}\cdot\frac{1}{2}\left(\frac{\partial u_k}{\partial x_l}+\frac{\partial u_l}{\partial x_k}\right),\frac{1}{2}\left(\frac{\partial w_i}{\partial x_j}+\frac{\partial w_j}{\partial x_i}\right)\right)_\Omega+\\&+\left(e_{kij}\cdot\ \frac{\partial \varphi}{\partial x_k},\frac{1}{2}\left(\frac{\partial w_i}{\partial x_j}+\frac{\partial w_j}{\partial x_i}\right)\right)_\Omega=\left\langle t_{iN},w_i\right\rangle_{\Gamma_2}.\end{aligned} \tag{3.22}$$

Zaveďme označení pro integro-diferenciální operátory:

$$\begin{aligned}\mathbf{A}(\mathbf{u},\mathbf{w}) &= \left(c_{ijkl}\cdot\frac{1}{2}\left[\frac{\partial u_k}{\partial x_l}+\frac{\partial u_l}{\partial x_k}\right],\frac{1}{2}\left[\frac{\partial w_i}{\partial x_j}+\frac{\partial w_j}{\partial x_i}\right]\right)_\Omega,\\ \mathbf{D}(\varphi,\mathbf{w}) &= \left(e_{ijk}\cdot\frac{\partial \varphi}{\partial x_k},\frac{1}{2}\left[\frac{\partial w_i}{\partial x_j}+\frac{\partial w_j}{\partial x_i}\right]\right)_\Omega,\\ \widetilde{\mathbf{D}}(\mathbf{u},\phi) &= \left(e_{ijk}\cdot\frac{1}{2}\left[\frac{\partial u_j}{\partial x_k}+\frac{\partial u_k}{\partial x_j}\right],\frac{\partial \phi}{\partial x_i}\right)_\Omega,\\ \mathbf{E}(\varphi,\phi) &= \left(\varepsilon_{ij}\frac{\partial \varphi}{\partial x_j},\frac{\partial \phi}{\partial x_i}\right)_\Omega,\end{aligned} \tag{3.23}$$

kde $\mathbf{u},\mathbf{w}\in\ [V(\Omega)]^3$ , $\varphi,\phi\in V(\Omega)$.

Dále zaveďme označení pro funkcionály charakterizující slabé splnění Neumannových okrajových podmínek:

$$\mathbf{r}(\mathbf{w}) = \left\langle t_{iN}, w_i \right\rangle_{\Gamma_2}, \tag{3.24}$$

$$\mathbf{q}(\phi) = \left\langle D_N, \phi \right\rangle_{\Gamma_2}. \tag{3.25}$$

Rovnice 3.17, 3.22 přepišme do operátorového tvaru:

$$\begin{aligned} &\mathbf{A}(\mathbf{u}, \mathbf{w}) + \mathbf{D}(\varphi, \mathbf{w}) = \mathbf{r}(\mathbf{w}), \\ &\widetilde{\mathbf{D}}(u, \mathbf{w}) + \mathbf{E}(\varphi, \phi) = \mathbf{q}(\phi). \end{aligned} \tag{3.26}$$

**Definice**: Nechť jsou dány funkce $\mathbf{u}^* \in [W_2^{(1)}(\Omega)]^3$ a $\varphi^* \in W_2^{(1)}(\Omega)$ jejichž stopy zajišťují na hranici splnění Dirichletových okrajových podmínek, tedy

$$u^* = u_D \quad \text{na } \Gamma_1,$$

$$\varphi^* = \varphi_D \quad \text{na } \Gamma_1.$$

Nechť funkce $\mathbf{u_0} = (\mathbf{u_1}, \mathbf{u_2}, \mathbf{u_3}) \in [\mathbf{V}(\Omega)]^3$ a $\varphi_0 \in V(\Omega)$ splňují rovnice 3.26 pro všechny testovací funkce $\mathbf{w} = (w_1, w_2, w_3) \in [V(\Omega)]^3, \quad \phi \in V(\Omega)$. Potom funkce

$$\mathbf{u} = \mathbf{u_0} + \mathbf{u}^*, \quad \varphi = \varphi_0 + \varphi^*$$

nazveme slabým řešením úlohy 3.1, 3.4 s okrajovými podmínkami 3.2, 3.3, 3.5, 3.6, viz např. [13].

## 3.3 Aproximace úlohy elasto-elektrických polí

Oblast $\Omega$ rozdělíme na konečnou množinu $E_h$ disjunktních otevřených čtyřstěnů (simplexů v $R^3$) pokrývajících oblast $\Omega$.

$$\overline{\Omega} = \bigcup_{e \in E_h} \overline{e}. \tag{3.27}$$

Pro každý simplex $e \in E_h$ zavedeme prostory funkcí s lokálním nosičem

$$V(e) = \phi \in W_2^1(e), \quad \phi \in P^1(e), \tag{3.28}$$

kde $P^1(e)$ je prostor polynomů prvního stupně na simplexu $e$. Báze tohoto prostoru

$$\Phi^e = \{\phi_i^e(x, y, z) | i = 1, 2, 3, 4\} \subset V(e) \tag{3.29}$$

je tvořena čtyřmi bazickými funkcemi takovými, že pro každou z nich platí:

$$\phi_i^e(s_j^e) = \delta_{ij}, \qquad i,j = 1,2,3,4, \tag{3.30}$$

kde $s_j^e$ $j = 1,2,3,4$ jsou vrcholy simplexu $e$. Každá funkce $\phi_i^e$ je tedy zadána svými hodnotami ve vrcholech simplexu. Funkci $\phi_i^e$ nazveme bazickou funkcí příslušející i-tému vrcholu simplexu $e$.

Dělení na čtyřstěny bylo voleno z důvodu jednoduchosti konstrukce interpolačních funkcí. Konkrétní tvar interpolační funkce $\phi_i^e$ je

$$\phi_i^e(x,y,z) = \alpha_{0i}^e + \alpha_{1i}^e x + \alpha_{2i}^e y + \alpha_{3i}^e z, \tag{3.31}$$

kde koeficienty $\alpha_{ji}^e$ z definice funkce $\phi_i^e$ 3.31 jsou prvky matice inverzní k matici obsahující souřadnice vrcholů čtyřstěnu, přesněji

$$\begin{pmatrix} \alpha_{01}^e & \alpha_{02}^e & \alpha_{03}^e & \alpha_{04}^e \\ \alpha_{11}^e & \alpha_{12}^e & \alpha_{13}^e & \alpha_{14}^e \\ \alpha_{21}^e & \alpha_{22}^e & \alpha_{23}^e & \alpha_{24}^e \\ \alpha_{31}^e & \alpha_{32}^e & \alpha_{33}^e & \alpha_{34}^e \end{pmatrix} = \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{pmatrix}^{-1}. \tag{3.32}$$

Aproximace posunutí $\mathbf{u}$ a potenciálu $\varphi$ na zvoleném simplexu $e$ mají tvar

$$u_{hi}^e(\mathbf{x}) = \sum_{j=1}^{4} u_{ji}^e \phi_j^e(\mathbf{x}), \quad i = 1,2,3, \tag{3.33}$$

$$\varphi_h^e(\mathbf{x}) = \sum_{j=1}^{4} \varphi_j^e \phi_j^e(\mathbf{x}). \tag{3.34}$$

Z definice funkcí $\phi_i$ plyne, že koeficienty lineární kombinace $u_i^e$ a $\varphi_i^e$ jsou (neznámé) hodnoty veličin $\mathbf{u}_h(x)$ a $\varphi_h$ v i-tém uzlu příslušného prvku.

Pro parciální derivace posunutí $\mathbf{u}$ a potenciálu $\varphi$ platí:

$$\frac{\partial}{\partial x_j} u_{hi}^e(\mathbf{x}) = \sum_{k=1}^{4} u_e^{ki} \frac{\partial}{\partial x_j} \psi_k^e(\mathbf{x}), \tag{3.35}$$

$$\frac{\partial}{\partial x_j} \varphi_h^e(\mathbf{x}) = \sum_{k=1}^{4} \varphi_j^e \frac{\partial}{\partial x_j} \psi_k^e(\mathbf{x}). \tag{3.36}$$

Takto je aproximováno posunutí $\mathbf{u}$ a potenciál $\varphi$ na všech simplexech rozkladu $E_h$. Množina všech bazických funkcí z jednotlivých simplexů pak tvoří bázi $\Phi$ prostoru $V_h(\Omega)$. Pro funkce

$$f, g \in V_h(\Omega), \quad (\text{supp } f \cap \text{supp } g = 0 \Rightarrow (f,g)_\Omega = 0,)$$

Nenulové budou členy se skalárním součinem funkcí, které mají společný nosič. To jsou vždy bazické funkce na tomtéž simplexu, jejichž nosiče mají neprázdný průnik.

## 3.4 Algoritmus sestavení stavové matice

Omezíme-li se na jeden simplex, můžeme takto sestavit lokální matici simplexu. Pro větší přehlednost bude v následujícím postupu užito maticového zápisu.

Zaveďme diferenciální operátor **L**:

$$\mathbf{L} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{2\partial z} & \frac{\partial}{2\partial y} \\ \frac{\partial}{2\partial z} & 0 & \frac{\partial}{2\partial x} \\ \frac{\partial}{2\partial y} & \frac{\partial}{2\partial x} & 0 \end{pmatrix}$$

Nechť $e$ je libovolný pevně zvolený simplex z $E^h$, $\Phi^e = \{\phi_i^e | i = 1,2,3,4\}$ je jeho báze. Funkce $\phi_i$ příslušící $i$-tému vrcholu je definována vztahem 3.31. Označme

$$\mathbf{w^1}_i = \begin{pmatrix} \phi_i \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{w^2}_i = \begin{pmatrix} 0 \\ \phi_i \\ 0 \end{pmatrix}, \quad \mathbf{w^3}_i = \begin{pmatrix} 0 \\ 0 \\ \phi_i \end{pmatrix},$$

$$\mathbf{W}_i^e = \begin{pmatrix} \mathbf{w^1}_i, & \mathbf{w^2}_i, & \mathbf{w^3}_i \end{pmatrix}$$

matici bazických funkcí pro i-tý vrchol simplexu $e$. Zaveďme matici deformací pro $i$-tý vrchol simplexu $e$:

$$\mathbf{B}^{ie} = \mathbf{LW_i} = \begin{pmatrix} \alpha_{1i} & 0 & 0 \\ 0 & \alpha_{2i} & 0 \\ 0 & 0 & \alpha_{3i} \\ 0 & \frac{1}{2}\alpha_{3i} & \frac{1}{2}\alpha_{2i} \\ \frac{1}{2}\alpha_{3i} & 0 & \frac{1}{2}\alpha_{1i} \\ \frac{1}{2}\alpha_{2i} & \frac{1}{2}\alpha_{1i} & 0 \end{pmatrix} \tag{3.37}$$

**Lokální matice tuhosti $\mathbf{K^e}$** simplexu $e$ v blokovém zápisu má tvar:

$$\mathbf{K^e} = \begin{pmatrix} \mathbf{K_{11}}^e & \dots & \mathbf{K_{14}}^e \\ \vdots & \ddots & \vdots \\ \mathbf{K_{41}}^e & \dots & \mathbf{K_{44}}^e \end{pmatrix},$$

$$\mathbf{K_{ij}}^e = \int_{\Omega^h} [\mathbf{B}^{je}]^{\mathrm{T}} \mathbf{C} \mathbf{B}^{ie} d\Omega \qquad \mathbf{i,j = 1,2,3,4.} \tag{3.38}$$

Pro $\forall i, j \in \widehat{4}$, kde $\widehat{4} = \{1, 2, 3, 4\}$ je $\mathbf{K}_{ij}^e \in \mathrm{R}^{3,3}$, tedy $\mathbf{K^e} \in \mathrm{R}^{12,12}$. $\mathbf{K^e}$ je symetrická a $\mathbf{C}$ je matice tenzoru elastických koeficientů, viz 2.17.

**Lokální matice $\mathbf{P^e}$** simplexu $e$ vyjadřující piezoelektrickou vazbu má tvar:

$$\mathbf{P}^e = \begin{pmatrix} \mathbf{P_{11}}^e & \dots & \mathbf{P_{14}}^e \\ \vdots & \ddots & \vdots \\ \mathbf{P_{41}}^e & \dots & \mathbf{P_{44}}^e \end{pmatrix},$$

$$\mathbf{P_{ij}}^e = \left(\mathbf{DB^{je}}, \nabla \cdot \mathbf{W_i^e}\right)_{\Omega} \qquad \mathbf{i, j = 1, 2, 3, 4.}$$

Pro $\forall i, j \in \widehat{4}$ je $\mathbf{P_{ij}}^e \in \mathrm{R}^{1,3}$, tedy $\mathbf{P}^e \in \mathrm{R}^{4,12}$. $\mathbf{D}$ je matice tenzoru piezoelektrických modulů.

**Lokální matice $\mathbf{E}^e$** simplexu $e$ vyjadřující elektrickou vazbu má tvar:

$$\mathbf{E}^e = \begin{pmatrix} \mathbf{E_{11}}^e & \dots & \mathbf{E_{14}}^e \\ \vdots & \ddots & \vdots \\ \mathbf{E_{41}}^e & \dots & \mathbf{E_{44}}^e \end{pmatrix},$$

$$\mathbf{E_{ij}}^e = \left(\Sigma \nabla \cdot \mathbf{W_j^e}, \nabla \cdot \mathbf{W_i^e}\right)_{\Omega} \qquad \mathbf{i, j = 1, 2, 3, 4.}$$

$\mathbf{E^e} \in \mathrm{R}^{4,4}$ je opět symetrická matice, $\Sigma$ je matice tenzoru permitivity.

Označme vektory pravé strany

$$\mathbf{R} = \left( \mathbf{R_1^1},\ \mathbf{R_2^1},\ \mathbf{R_3^1},\ \mathbf{R_4^1},\ \mathbf{R_1^2},\ \mathbf{R_2^2},\ \mathbf{R_3^2},\ \mathbf{R_4^2},\ \mathbf{R_1^3},\ \mathbf{R_2^3},\ \mathbf{R_3^3},\ \mathbf{R_4^3} \right)^{\mathbf{T}},$$

kde

$$\mathbf{R_i^m = r(w_i^m)} \qquad \mathbf{i = 1, 2, 3, 4,} \qquad \mathbf{m = 1, 2, 3.}$$

Označme

$$\mathbf{Q} = \left( \mathbf{Q_1},\ \mathbf{Q_2},\ \mathbf{Q_3},\ \mathbf{Q_4} \right)^{\mathbf{T}},$$

kde

$$\mathbf{Q_i = q}(\phi_\mathbf{i}), \qquad \mathbf{i = 1, 2, 3, 4.}$$

Nyní ukažme algoritmus sestavení globální matici pro celou oblast $\Omega^h$. Princip sestavení globální matice uveďme např. pro matici tuhosti. U piezoelektrické a elektrické matice se postupuje obdobně. Níže popsaný algoritmus je i algoritmem programu na sestavení globální matice.

1. Proveď diskretizaci oblasti na množinu simplexů $E^h$.
2. Očísluj všechny vrcholy čísly 1,...,n.
3. Pro každý simplex e $\in E^h$proveď:

3*a*. Nechť má simplex vrcholy s čísly $a, b, c, d \in \widehat{n}$.
Proveď lokální očíslování vrcholů čísly $1 - 4$.

3*b*. Načti souřadnice vrcholů.

3*c*. Sestroj bazické funkce $\phi_i^e$, $i \in \widehat{4}$ podle 3.32.

3*d*. Sestav matice deformací $\mathbf{B}^{ie}$, $i \in \widehat{4}$.

3*e*. $\forall i, j \in \widehat{4}$ : sestav submatici $\mathbf{K}_{ij}^e$, lokální matice tuhosti.
Nechť vrcholy $i, j$ mají v globálním číslování označení $a, b$.
Přičti submatici $\mathbf{K}_{ij}^e$ na místo $\mathbf{K}_{ab}$ v globální matici tuhosti.

Postupně procházíme všechny simplexy, počítáme jejich lokální matice a její prvky přičítáme na příslušné místo v globální matici.

Stejným způsobem se vytvoří i globální submatice piezoelektrické a elektrické a pravá strana. Nyní, použijeme-li těchto výsledků, můžeme celou úlohu symbolicky zapsat ve tvaru:

$$\begin{pmatrix} \mathbf{K} & \mathbf{D}^{\mathrm{T}} \\ \mathbf{D} & \mathbf{E} \end{pmatrix} \begin{pmatrix} \mathbf{U} \\ \Phi \end{pmatrix} = \begin{pmatrix} \mathbf{R} \\ \mathbf{Q} \end{pmatrix}. \tag{3.39}$$

Při sestavování jednotlivých lokálních matic je třeba výrazy integrovat přes oblast $\Omega$.

Mějme libovolný simplex $e$, kde i-tý vrchol má souřadnice $(x_1^i, x_2^i, x_3^i)$, a na něm dvě bazické funkce $\phi_r^e, \phi_s^e$ (lineární funkce definované podle 3.31). Označme $(t_1, t_2, t_3)$ souřadnice těžiště simplexu.

Objem simplexu $e$ lze spočítat jako (viz [16])

$$|e| = \frac{1}{6} \begin{vmatrix} 1 & x_1^1 & x_2^1 & x_3^1 \\ 1 & x_1^2 & x_2^2 & x_3^2 \\ 1 & x_1^3 & x_2^3 & x_3^3 \end{vmatrix}$$

V lokálních maticích se vyskytují integrály tří typů funkcí.

Integrál typu

$$\int_e \phi_r^e(\mathbf{x}) \phi_s^e(\mathbf{x}) d\Omega$$

je integrálem ze součinu dvou lineárních funkcí, tedy kvadratické funkce.

Integrál typu

$$\int_e \frac{\partial}{\partial x_i} \phi_r^e \mathbf{x}) \frac{\partial}{\partial x_j} \phi_s^e(\mathbf{x}) d\Omega$$

je integrálem ze součinu parciálních derivací lineárních funkcí, tedy ze součinu konstant.

Integrál typu

$$\int_e \frac{\partial}{\partial x_i} \phi_r^e(\mathbf{x}) \phi_s^e(\mathbf{x}) d\Omega$$

je integrálem z lineární funkce.

Při integraci byly použity vzorce (viz [16]):

$$\begin{aligned}
\int_e \mathrm{C} d\mathbf{x} &= \mathrm{C}.|e| \\
\int_e x_i d\mathbf{x} &= |e|.t_i \\
\int_e x_i^2 d\mathbf{x} &= \frac{|e|}{20} \sum_{l=1}^{4} (x_i^l - t_i)^2 + |e|.t_i^2 \\
\int_e x_i x_j d\mathbf{x} &= \frac{|e|}{20} \sum_{l=1}^{4} (x_i^l - t_i)(x_j^l - t_j) + |e|.t_i t_j.
\end{aligned}$$

## 3.5 Výpočet elektrické idukce a tenzoru napětí

Řešením systému 3.39 jsou vektor posunutí **U** a elektrické potenciály $\Phi$ v uzlech sítě konečných prvků. Tenzor elastických napětí **T**, a vektor elektrické indukce **D** lze dopočítat z vektoru posunutí, resp. z elektrických potenciálů.

Označme $u^i(e) = (u_1^i, u_2^i, u_3^i)$ $i = 1, 2, 3$ vektor posunutí a $\varphi^i(e)$ elektrický potenciál v $i$-tém uzlu elementu $e$, viz obr. 3.2.

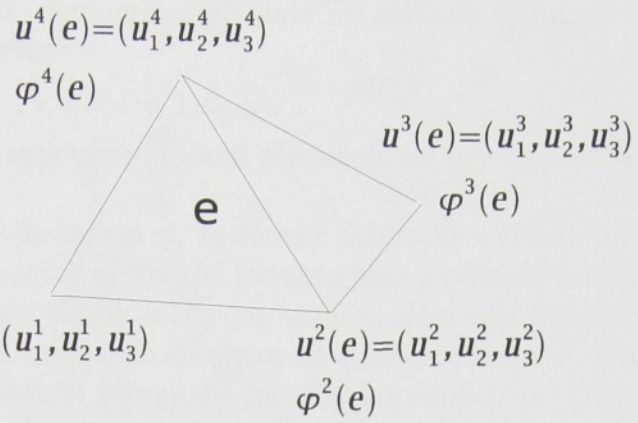
Potom můžeme na každém elementu $e$ spočítat složky vektoru elektrického pole, viz [12],

$$\begin{aligned}
E_x^e &= -\frac{\partial \varphi^e}{\partial x} = -(\alpha_{02}^e \varphi_1^e + \alpha_{12}^e \varphi_2^e + \alpha_{22}^e \varphi_3^e + \alpha_{32}^e \varphi_4^e), \\
E_y^e &= -\frac{\partial \varphi^e}{\partial y} = -(\alpha_{03}^e \varphi_1^e + \alpha_{13}^e \varphi_2^e + \alpha_{23}^e \varphi_3^e + \alpha_{33}^e \varphi_4^e), \\
E_z^e &= -\frac{\partial \varphi^e}{\partial z} = -(\alpha_{04}^e \varphi_1^e + \alpha_{14}^e \varphi_2^e + \alpha_{24}^e \varphi_3^e + \alpha_{34}^e \varphi_4^e),
\end{aligned}$$

kde $\alpha_{kl}$ jsou koeficienty interpolačních funkcí zavedených vztahem 3.32.

Obdobným způsobem lze spočítat složky tenzoru deformace na elementu

$$S_{11}^e = \frac{1}{2} \left( \frac{\partial u_1^e}{\partial x} + \frac{\partial u_1^e}{\partial x} \right) = (\alpha_{02}^e u_1^1 + \alpha_{12}^e u_1^2 + \alpha_{22}^e u_1^3 + \alpha_{32}^e u_1^4)$$




Obrázek 3.2: Vektory posunutí $u^i(e)$ a elektrické potenciály $\varphi^i(e)$ v uzlech elementu $e$..

$$S_{12}^e = \frac{1}{2}\left(\frac{\partial u_1^e}{\partial y} + \frac{\partial u_2^e}{\partial y}\right) = \frac{1}{2}(\alpha_{02}^e u_1^1 + \alpha_{12}^e u_1^2 + \alpha_{22}^e u_1^3 + \alpha_{32}^e u_1^4 +$$

$$+\alpha_{03}^e u_2^1 + \alpha_{13}^e u_2^2 + \alpha_{23}^e u_2^3 + \alpha_{33}^e u_2^4),$$

$$S_{13}^e = \frac{1}{2}\left(\frac{\partial u_1^e}{\partial y} + \frac{\partial u_2^e}{\partial y}\right) = \frac{1}{2}(\alpha_{02}^e u_1^1 + \alpha_{12}^e u_1^2 + \alpha_{22}^e u_1^3 + \alpha_{32}^e u_1^4 +$$

$$+\alpha_{04}^e u_3^1 + \alpha_{14}^e u_3^2 + \alpha_{24}^e u_3^3 + \alpha_{34}^e u_3^4),$$

$$S_{22}^e = \frac{1}{2}\left(\frac{\partial u_1^e}{\partial x} + \frac{\partial u_1^e}{\partial x}\right) = (\alpha_{02}^e u_2^1 + \alpha_{12}^e u_2^2 + \alpha_{22}^e u_2^3 + \alpha_{32}^e u_2^4)$$

$$S_{23}^e = \frac{1}{2}\left(\frac{\partial u_1^e}{\partial y} + \frac{\partial u_2^e}{\partial y}\right) = \frac{1}{2}(\alpha_{02}^e u_2^1 + \alpha_{12}^e u_2^2 + \alpha_{22}^e u_2^3 + \alpha_{32}^e u_2^4 +$$

$$+\alpha_{04}^e u_3^1 + \alpha_{14}^e u_3^2 + \alpha_{24}^e u_3^3 + \alpha_{34}^e u_3^4),$$

$$S_{33}^e = \frac{1}{2}\left(\frac{\partial u_1^e}{\partial y} + \frac{\partial u_2^e}{\partial y}\right) = (\alpha_{04}^e u_3^1 + \alpha_{14}^e u_3^2 + \alpha_{24}^e u_3^3 + \alpha_{34}^e u_3^4).$$

Vektor elektrické indukce na elementu spočítáme na základě vztahu 2.2, který upravíme na tvar

$$D_i^e = \varepsilon_{ij}^e E_j^e, \quad i,j = 1,2,3,$$

kde $\varepsilon^e$ je tenzor permitivity příslušný dle topologie sítě elementu $e$.

Tenzor napětí na elementu spočítáme na základě Hookova zákona 2.15, který upravíme na tvar

$$T_{ij}^e = c_{ijkl}^e S_{kl}^e, \quad i,j,k,l = 1,2,3,$$

kde $c_{ijkl}^e$ je tenzor elastických modulů příslušný dle topologie sítě elementu $e$.

**Poznámka 6:** Všimněme si, že tenzor deformace, tenzor napětí, vektor elektrického pole a vektor elektrické indukce jsou v případě použití lineárních funkcí pro aproximaci složek $\mathbf{u}$ a $\varphi$ na každém elementu konstantními funkcemi. Není tedy zaručena jejich spojitost na stěnách elementů. Tato vlastnost, která vyplývá z primární formulace metody konečných prvků má praktické důsledky při nasazení modelu, které jsou diskutovány především v kapitole 6.

## 3.6 Vstupní parametry modelu

V této části uvedeme všechny parametry, které vstupují do modelu. V první řadě se jedná o materiálové parametry modelu:

- tenzor permitivity,
- tenzor elastických modulů,
- tenzor piezoelektrických modulů.

V případě modelování kompozitů je do modelu zadáváno tolik tenzorů elastických modulů, piezoelektrických modulů a permitivity, z kolika materiálů je kompozit složen. V našem případě se může jednat např. o dva materiály v případě modelování jevů ve feroelektrické vrstvě, ve které se vyskytují dva doménové stavy. Definovány tedy musí být i oblasti, ve kterých je ta která sada materiálových parametrů užita.

Další skupinou údajů, které musíme zadat, jsou geometrické parametry modelované oblasti:

- definice geometrie modelované oblasti,
- definice rozkladu geometrie modelované oblasti na podoblasti - konečné prvky,
  - definice elementů pomocí uzlů, které tvoří jeho vrcholy,
  - definice souřadnic jednotlivých uzlů.
- definice hranic oblasti, kde budou zadány různé typy okrajových podmínek.

Dalšími vstupními parametry jsou hodnoty okrajových podmínek.

## 3.7 Výstupy modelu

V případě modelu na bázi primární formulace MKP jsou přímými výstupy modelu následující veličiny:

- vektory posunutí,
- elektrické potenciály

v uzlech diskretizace.

Pokud do výstupů modelu zahrneme i následné zpracování primárních veličin, můžeme za výstupy označit

- vektory elektrického pole,
- vektory elektrické indukce,
- tenzory napětí
- tenzory deformace

na jednotlivých elementech.

Dále to mohou být různé funkční závislosti vstupních a výstupních parametrů modelu. Např. průběhy výstupních veličin v závislosti na poloze uzlů diskretizace apod.

# Kapitola 4

# Počítačová implementace

Tato kapitola obsahuje stručný popis implementace navrženého modelu. Implementace je provedena v jazyce C. Použitým kompilátorem je GCC ve verzi 3.2.2 pod operačním systémem Linux. Jazyk C je zvolen z důvodu jeho vysoké portability mezi různými operačními systémy. Z tohoto důvodu je též koncipován jako konsolová aplikace bez grafického rozhraní. Všechny parametry modelu jsou předávány buď formou vstupních datových souborů, nebo jako parametry spouštěného programu.

Pro řešení vzniklé soustavy algebraických rovnic je využito příslušných funkcí z knihovny LAPACK ([20]), která je součástí mocného balíku GSL ([21]) pro vědecko-technické výpočty pod OS Linux.

Preprocessing, tedy tvorba geometrie modelu, tvorba sítě konečných prvků, zadání okrajových podmínek, definice oblastí s různými materiálovými vlastnostmi je realizována pomocí freewarového prostředku GMSH, viz. [19]. Z tohoto důvodu je naprogramován interface mezi výstupními strukturami GMSH systému a vlastním softwarem. Pomocí GMSH je řešeno i grafické zobrazení výsledků a je naprogramováno tedy i rozhraní mezi vytvořeným softwarem a postprocessingovým modulem GMSH.

## 4.1 Formát vstupních souborů

Důležitým znakem všech vstupních souborů je to, že jsou textové. Tím je zaručena snadná editovatelnost souborů a tím i jednoduchá změna vstupních parametrů modelu.

### 4.1.1 Soubor *.mtr

Tento typ souboru obsahuje hodnoty hustoty materiálu a hodnoty složek tenzoru elastických modulů, tenzoru piezoelektrických modulů a tenzoru permitivity.

Složky tenzoru elastických modulů jsou vypsány v matici 6x6, [8]. Ve stejné konvenci je psán tenzor piezoelektrických modulů, který tvoří matici 3x6, i tenzor permitivity, matice 3x3.

Jeho struktura je následující:

1. řádek: hustota $[kg/m^3]$
2. řádek: prázdný
3.-8. řádek: elastické moduly $[m^{-2}N]$
9. řádek: prázdný
10.-12. řádek: piezoelektrické moduly $[m^{-1}V]$
13. řádek: prázdný
14.-16. řádek: permitivita $[Fm^{-1}]$
konec souboru

Příklad takového souboru je uveden v příloze A.1

### 4.1.2 Soubor *.geo

Je typem souboru, ve kterém jsou definovány geometrické charakteristiky modelované oblasti. Jedná se o nativní formát systému GMSH. Obsahuje definice tzv. elementárních entit - bodů, křivek, ploch, objemů, viz [19]. Další jeho částí jsou definice tzv. fyzikálních entit, které se skládají z entit elementárních. Fyzikální entity jsou posléze užívány k definici okrajových podmínek na různých částech hranice zkoumané oblasti a k definici různých materiálových vlastností podoblastí.

Soubor je strukturován po záznamech na řádcích. Každý záznam obsahuje definici elementární či fyzikální entity, které jsou vzestupně číslovány. Záznamy jsou odděleny středníkem.

Soubor je možné vytvořit buď pomocí libovolného textového editoru, nebo užitím prostředí programu GMSH.

Příklad takového souboru je uveden v příloze A.2.

### 4.1.3 Soubor *.msh

Soubory s příponou `msh` obsahují definici sítě konečných prvků a jejich příslušnost k fyzikálním entitám definovaným v příslušném `geo` souboru. Opět se jedná o nativní formát systému GMSH.

Jeho struktura je následující:

1. Klíčové slovo `$NOD`
2. Počet uzlů sítě

3. Definice uzlů sítě
4. Klíčové slovo `$ENDNOD`
4. Klíčové slovo `$ELM`
5. Počet elementů sítě
6. Definice konečných prvků sítě
7. Klíčové slovo `$ENDELM`

konec souboru

Soubor lze editovat v textovém editoru, ale vzhledem k vysokému počtu elementů, které jsou zpravidla v síti obsaženy, je pohodlnější soubor editovat za použití programu GMSH.

Příklad takového souboru je uveden v příloze A.3.

## 4.2 Formát výstupních souborů

Model poskytuje vypočtené výsledky, tedy velikosti posunutí a elektrické potenciály v uzlech sítě, ve dvou podobách. Buď jako prostý textový soubor, nebo soubor ve formátu `pos`, který lze zobrazit programem GMSH.

Stejné formáty výstupu mají i soubory obsahující údaje o vypočtených složkách tenzoru deformace, tenzoru napětí, elektrického pole, elektrické indukce na jednotlivých elementech.

Dále jsou generovány `dat` soubory, které obsahují různé funkční závislosti, jež lze vykreslit například programem Gnuplot.

### 4.2.1 Soubory `posuny` a `potencialy`

Soubory s těmito názvy obsahují hodnoty vektorů posunutí a elektrických potenciálů v uzlech sítě. Jedná se o prostý textový výpis hodnot, který je v pořadí uzlů sítě, definovaném v příslušném `msh` souboru.

Soubor `posuny` tedy obsahuje $3 \cdot n$ reálných hodnot (složek vektorů posunutí v uzlech) a soubor `potencialy` obsahuje $n$ reálných hodnot (velikosti potenciálů v uzlech), kde $n$ je počet uzlů sítě.

### 4.2.2 Soubory `posuny.pos` a `potencialy.pos`

Tyto soubory obsahují stejné hodnoty jako soubory `posuny` a `potencialy`. Jsou však transformované do podoby, kterou na svém vstupu požaduje program GMSH. Využívá se přitom možnosti programu GMSH vizualizovat skalární a vektorové veličiny.

Soubory obsahují záznamy, které definují vektorovou, resp. skalární veličinu, kterou chceme vizualizovat, a její geometrické umístění do příslušného uzlu uvedené v `msh` souboru.

Příklad takových souborů je uveden v příloze A.4.

### 4.2.3 Soubory `E.pos`, `E_x.pos`, `E_y.pos`, `E_z.pos`

Soubory obsahují hodnoty vektoru elektrického pole na jednotlivých elementech sítě. Opět se jedná o soubory ve formátu pro zobrazení programem GMSH.

Soubory obsahují jednak úplný vektor elektrického pole (`E.pos`), tak jeho jednotlivé složky (`E_x.pos`, `E_y.pos`, `E_z.pos`). Vzhledem k tomu, že elektrické pole je na elementech konstantní, počátky zobrazovaných vektorů jsou umístěny do těžišť příslušných elementů.

Příklad těchto souborů je uveden v příloze A.5.

### 4.2.4 Soubory `strain_**.pos`, `stress_**.pos`

V souborech jsou obsažena data pro vizualizaci jednotlivých složek $S_{ij}, T_{ij}$ tenzorů deformace a napětí systémem GMSH. Složky tenzorů jsou zobrazovány po elementech konstantně a velikost je rozlišena barevnou škálou.

Příklad těchto souborů je uveden v příloze A.6.

### 4.2.5 Soubory `*.dat`

Vzhledem k tomu, že je mnohdy účelné prezentovat výsledky nejen formou barevných map promítnutých na geometrii řešeného vzorku, jsou v rámci postprocessingu generovány soubory s příponou `dat`. Obsahují data, která lze snadno prezentovat ve formě grafů různých funkčních závislostí.

Jedná se například o tečné a normálové složky vektoru elektrického pole v konkrétním řezu vzorkem, tečné a normálové síly apod. Soubor je uveden hlavičkou, která obsahuje informace o uložených datech. Data jsou v souboru uspořádána do sloupců, ve kterých jsou zaznamenány hodnoty jednotlivých veličin. Tento formát souboru je možné zpracovávat například za pomoci programu Gnuplot, viz [22].

Příklad tohoto typu souboru je uveden v příloze A.7.

## 4.3 Hlavní datové struktury a algoritmy modelu

Tato část obsahuje stručný popis členění vyvinutého programu a jeho hlavní datové struktury.

Jednotlivé funkce jsou rozděleny v několika souborech podle účelu, který plní. Soubor `matice.c` obsahuje funkce pro jednoduché operace s maticemi, které jsou využívány při tvorbě lokálních a globálních datových struktur. Součástí jsou i funkce realizující řešení soustavy lineárních rovnic Gaussovou eliminací, viz např. [17], které byly užity ve fázi testování programu.

Soubor `mkp.c` obsahuje funkce, které realizují samotnou metodu konečných prvků, tvoří globální matici a vektor pravé strany soustavy lineárních rovnic.

Soubor `gmsh.cc` obsahuje funkce zajišťující preprocessing, postprocessing a komunikaci s programem GMSH.

Kompletní zdrojové kódy programu jsou obsahem přílohy B.

Z datových struktur si všimněme struktur `lokal, lokale, lokalp` do kterých se ukládají lokální matice elastické, elektrické a piezoelektrické části.

Tyto struktury mají schéma shodné s uspořádáním elementu. Tzn. že každému ze čtyř vrcholů elementu přísluší matice dimenze $3 \times 3$, resp. $3 \times 1$ v piezoelektrické části a $1 \times 1$ v elektrické části.

Definici struktur a alokaci paměti nyní ukážeme, protože jejich správná definice a alokace paměti je klíčová pro správnou funkci programu.

Definice proměnné typu `lokal`:

```
typedef struct {
int u1, u2, u3, u4;
double ****lok;
} lokal;
```

`int u1, u2, u3, u4` - čísla uzlů, ze kterých je tvořen element

`double ****lok` - matice dimenze $4 \times 4$ s prvky typu $3 \times 3$.

alokace paměti pro prom. typu `lokal`:

```
lokal ml;
ml.lok=(double ****)malloc(4*sizeof(double ***));
for (i=0; i<4; i++)
ml.lok[i]=(double ***)malloc(4*sizeof(double **));
for (i=0; i<4; i++)
for (j=0; j<4; j++)
ml.lok[i][j]=(double **)malloc(3*sizeof(double *));
for (i=0; i<4; i++)
```

```
for (j=0; j<4; j++)
for (k=0; k<3; k++)
ml.lok[i][j][k]=(double *)malloc(3*sizeof(double));
```

Definice proměnné typu `lokale`:

```
typedef struct {
int u1, u2, u3, u4;
double ****lok;
} lokale;
```

`int u1, u2, u3` - čísla uzlů, ze kterých je tvořen simplex
`double ****lok` - matice dimenze $4\times4$ s prvky typu $1\times1$.
alokace paměti pro prom. typu `lokale`:

```
lokale ml;
ml.lok=(double ****)malloc(4*sizeof(double ***));
for (i=0; i<4; i++)
ml.lok[i]=(double ***)malloc(4*sizeof(double **));
for (i=0; i<4; i++)
for (j=0; j<4; j++)
ml.lok[i][j]=(double **)malloc(1*sizeof(double *));
for (i=0; i<4; i++)
for (j=0; j<4; j++)
ml.lok[i][j][0]=(double *)malloc(1*sizeof(double));
```

Definice proměnné typu `lokalp`:

```
typedef struct {
int u1, u2, u3, u4;
double ****lokp;
} lokalp;
```

`int u1, u2, u3, u4` - čísla uzlů, ze kterých je tvořen simplex
`double ****lok` - matice typu $4\times4$ s prvky typu $1\times3$.
alokace pameti pro prom. typu `lokalp`:

```
lokalp ml;
ml.lokp=(double ****)malloc(4*sizeof(double ***));
for (i=0; i<4; i++)
ml.lokp[i]=(double ***)malloc(4*sizeof(double **));
for (i=0; i<4; i++)
for (j=0; j<4; j++)
ml.lokp[i][j]=(double **)malloc(1*sizeof(double *));
for (i=0; i<4; i++)
for (j=0; j<4; j++)
ml.lokp[i][j][0]=(double *)malloc(3*sizeof(double));
```

Klíčové části implementace a algoritmy některých funkcí byly rozebrány již v části 3.3. Samotný popis algoritmů je pro čtenáře nezjímavý a mohl by být spíše obsahem referenčního manuálu k používání vytvořeného softwaru.

To je i jeden z autorových cílů po dokončení této práce, přiblížit vytvořené softwarové prostředky běžným uživatelům.

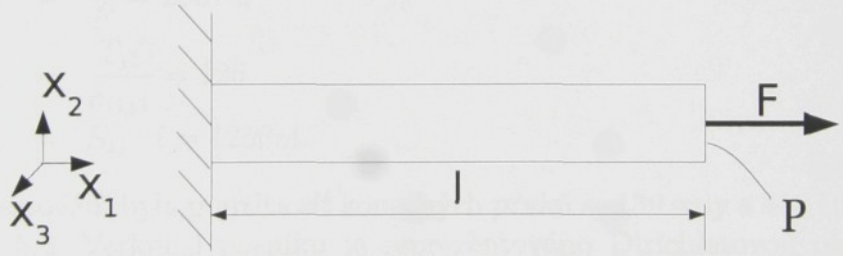
# Kapitola 5

# Testování

Úkolem této kapitoly je ověření všech částí modelu na jednoduchých úlohách tak, aby byla zaručena relevantnost výsledků, které model poskytuje. Vzhledem k MKP formulaci, která je užita k sestavení modelu budou testovány především primární veličiny (elektrický potenciál, mechanická posunutí).

## 5.1 Testování elastické části modelu

Pro testování elastické části byl zvolen případ vetknutého nosníku namáhaného prostým tahem, viz. obr. 5.1. Testy byly provedeny pro dva případy,




Obrázek 5.1: Schéma testovací úlohy - vetknutý nosník namáhaný prostým tahem.

lišící se zadáním různého tenzoru elastických modulů, viz. tab. 5.1.

První test odpovídá případu, kdy nedochází ve vzorku k přenosu působící síly do jiných směrů, než ve kterém působí. To znamená, že pokud bude vzorek namáhán silou v ose $x$, bude docházet k deformaci pouze ve směru osy $x$, v žádném jiném.

Parametry druhého testu již částečně odpovídají chování lineárního izotropního homogenního materiálu. Vzhledem k nenulovým hodnotám nediagonálních složek tenzoru $\mathbf{c}$ dojde k přenosu síly i do směru mimo směr působení

| | test č.1 | test č.2 |
|---|---|---|
| délka vzorku l [m] | 10 | 10 |
| příčný průřez P $[m^2]$ | 4 | 4 |
| Působící síla **F** [N] | 1000 | 1000 |
| tenzor elastických modulů **c** [Pa] | $\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ |

Tabulka 5.1: Parametry testů elastické části

síly. V případě podélného protažení vzorku tedy dojde i k jeho příčné kontrakci.

Analytické řešení prvního testovacího příkladu, se kterým budeme výsledky modelu porovnávat, je triviální. Vzhledem ke směru působící síly a tenzoru elastických modulů lze říci, že pouze složka $T_{11}$ tenzoru napětí, složka $S_{11}$ tenzoru deformace a složka $u_1$ vektoru posunutí budou nenulové. Uvedené veličiny lze vyjádřit jako:

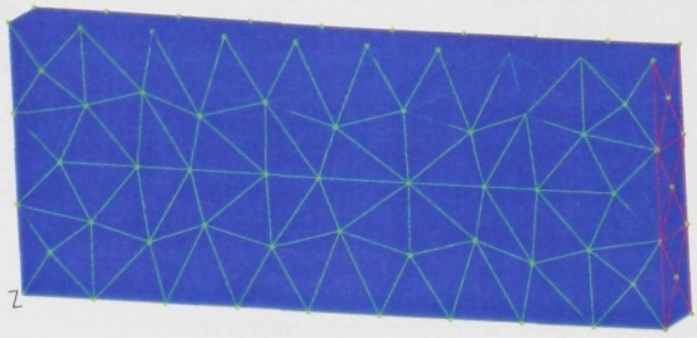
$$T_{11} = \frac{F}{P} = 250Pa \tag{5.1}$$

$$S_{11} = \frac{T_{11}}{c_{1111}} = 125 \tag{5.2}$$

$$u_1 = S_{11} \cdot l = 1250m. \tag{5.3}$$

Pro testování byla použita síť konečných prvků se 139 uzly a 413 čtyřstěny, viz. obr. 5.2. Vetknutí nosníku je reprezentováno Dirichletovou okrajovou podmínkou $u_D = 0$. Působící síla je zadána o velikosti $\frac{F}{n}$ v každém uzlu na straně protilehlé vetknutí, kde $n$ je počet uzlů na této straně. Na ostatních stěnách je zadána homogenní Neumannova okrajová podmínka $\mathbf{T} \cdot \mathbf{n} = 0$.

Posunutí vypočtená v uzlech na volném konci nosníku jsou uvedena v tab. 5.2. Jsou uvedena pouze posunutí ve směru osy $x_1$. Posunutí v ostatních směrech jsou zanedbatelně malá a vzhledem k užitému tenzoru **c** jsou pouze důsledkem numerické chyby řešení soustavy. V posledním sloupci tabulky je vypočten aritmetický průměr posunutí v uzlech. Jeho hodnota se liší o 2 % od hodnoty získané analytickým výpočtem. Tento výsledek lze označit za uspokojivý.



Obrázek 5.2: Síť konečných prvků pro testování elastické části.

| č. uzlu $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| posunutí $u^i_{x_1} \cdot 10^3 m$ | 1,390 | 1,387 | 1,410 | 1,400 | 1,194 | 1,184 |
| č. uzlu $i$ | 7 | 8 | 9 | 10 | 11 | $\frac{\sum_i^{max(i)} u_i}{max(i)}$ |
| posunutí $u^i_{x_1} \cdot 10^3 m$ | 1,194 | 1,190 | 1,180 | 1,254 | 1,253 | 1,276 |

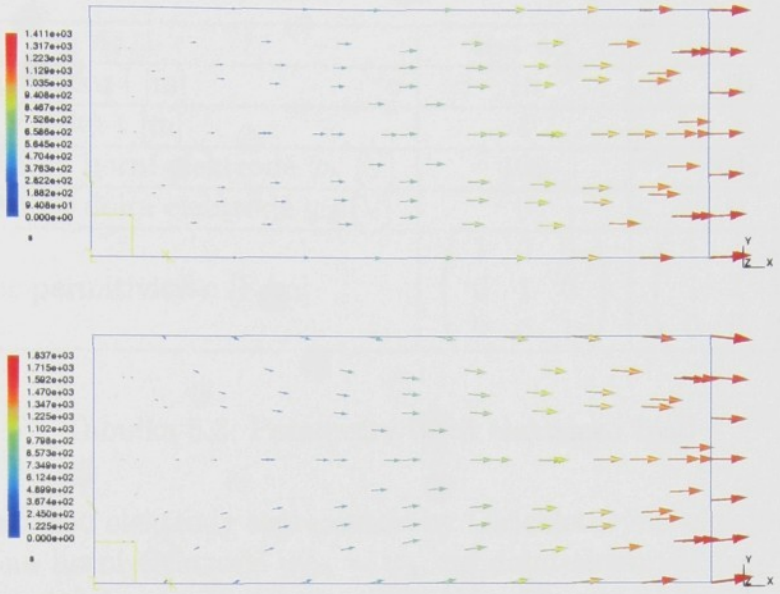
Tabulka 5.2: Hodnoty posunutí v uzlech - první test.

V případě druhého testu jsou vypočtená posunutí ekvivalentní testu prvnímu. Uplatňují se zde ale prvky tenzoru elastických modulů, které přenášejí účinky zatěžující síly i do směru kolmého ke směru jejího působení. Rozložení vektoru posunutí v prvním i druhém testu lze vidět na obr. 5.3.

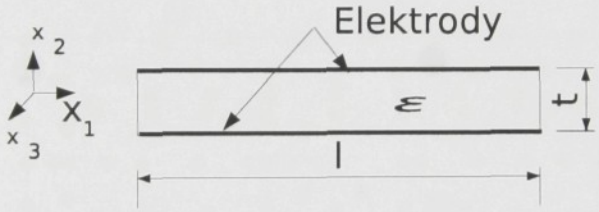
## 5.2 Testování elektrické části modelu

Elektrická část je ověřena na jednoduchém případu rozložení elektrického potenciálu a elektrického pole v deskovém kondenzátoru s dielektrikem o permitivitě $\varepsilon$. Schéma úlohy je naznačeno na obrázku 5.4. Pro testování je použita stejná geometrie vzorku i síť konečných prvků jako v případě elastických testů. Byly provedeny dva testy, lišící se použitým tenzorem permitivity. Parametry jsou uvedeny v tabulce 5.3.

Pokud nebudeme uvažovat přechodové jevy na levém a pravém okraji kondenzátoru, můžeme pro střední část vyjádřit analytické řešení. Materiál kondenzátoru je homogenní v celém svém objemu, proto rozložení elektric-



Obrázek 5.3: Rozložení vektoru posunutí pro první (nahoře) a druhý (dole) test.




Obrázek 5.4: Schéma úlohy pro testování elektrické části.

kého potenciálu bude:

$$\varphi(x_1, x_2, x_3) = \frac{1}{t} x_2(\varphi_h - \varphi_d) \qquad x_2 \in \langle 0, t \rangle \,. \tag{5.4}$$

Vzhledem k definici $E = -\nabla\varphi(x_1, x_2, x_3)$ platí:

$$E_1 = 0 \qquad E_2 = -\frac{1}{t}(\varphi_h - \varphi_d) = -25V/m \qquad E_3 = 0. \tag{5.5}$$

Vzhledem ke tvaru tenzoru permitivity bude v prvním případě pro elektrickou indukci platit:

$$D_1 = 0 \qquad D_2 = \varepsilon_{22} E_2 = -25C/m^2 \qquad D_3 = 0. \tag{5.6}$$

V druhém případě bude pro elektrickou indukci platit:

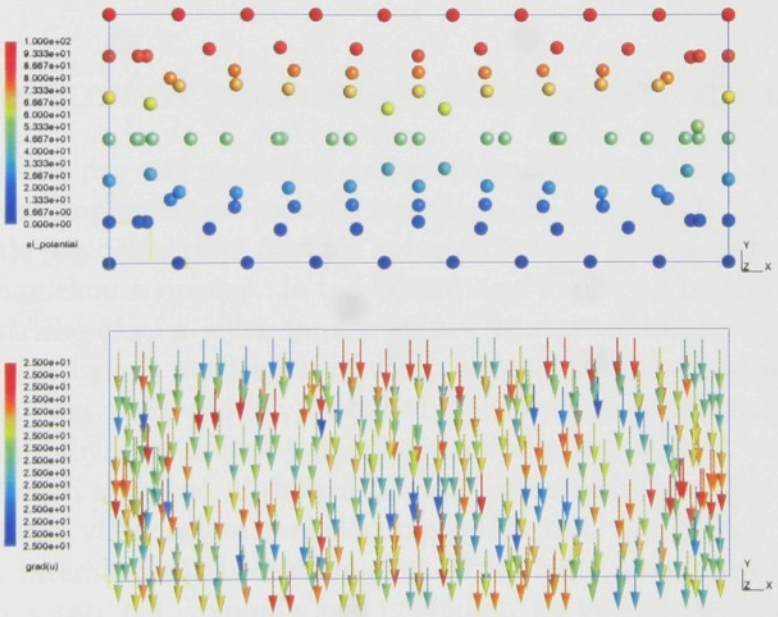
$$D_1 = \varepsilon_{12} E_2 = -25C/m^2 \qquad D_2 = \varepsilon_{22} E_2 = -25C/m^2 \qquad D_3 = 0. \tag{5.7}$$

| | test č.1 | test č.2 |
|---|---|---|
| délka vzorku l [m] | 10 | 10 |
| šířka vzorku t [m] | 4 | 4 |
| Napětí na horní elektrodě $\varphi_h$ [V] | 100 | 100 |
| Napětí na dolní elektrodě $\varphi_d$ [V] | 0 | 0 |
| tenzor permitivity **c** [F/m] | $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ |

Tabulka 5.3: Parametry testů elektrické části

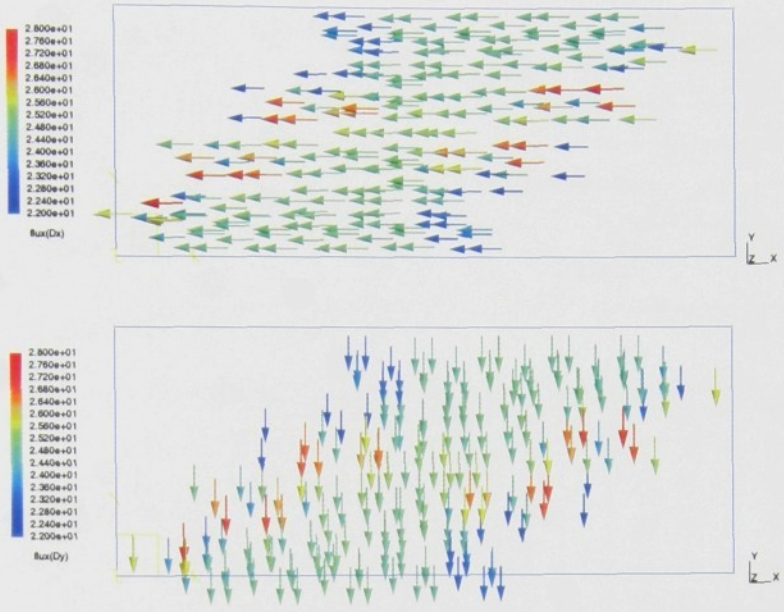
V modelu jsou elektrody reprezentovány Dirichletovými okrajovými podmínkami - na horní elektrodě $\varphi_{D1} = \varphi_h$, na dolní elektrodě $\varphi_{D2} = \varphi_d$. Na ostatních stěnách jsou zadány homogenní Neumannovy podmínky $D_N = 0$.

Na obrázku 5.5 je znázorněno rozložení elektrického potenciálu a elektrického pole vypočtené pro první test. Výsledek je plně v souladu se vztahy 5.4, 5.5. V druhém případě je pro nás zajímavé především rozložení vektoru



Obrázek 5.5: Rozložení elektrického potenciálu (nahoře) a elektrického pole (dole) pro první test.

elektrické indukce. Složky vektoru elektrické indukce jsou znázorněny na obr. 5.6. Jak už bylo naznačeno výše, zaměříme se na střední část vzorku, která není ovlivněna vlastnostmi okrajů. Je patrné, že vypočtená elektrická indukce




Obrázek 5.6: Rozložení složek $D_x$ (nahoře) a $D_y$ (dole) pro druhý test.

odpovídá vztahům 5.7.

## 5.3 Testování piezoelektrické části modelu

Zásadní význam pro celý model má ověření piezoelektrické části. Je testována nejen samotná implementace modulu reprezentující piezoelektrické vlastnosti struktury, ale piezoelektrická část též zprostředkovává přenos elektrické energie na mechanickou a opačně. Je tedy ověřována i správná interakce mechanické a elektrické části a jejich implementace ve stavové matici soustavy.

Pro ověření bylo z uvedených důvodů zvoleno několik testovacích úloh, které v první fázi ověřovacích výpočtů vliv elastických a elektrických vlastností omezují, aby bylo možné jednoduchým způsobem odhalit chyby v piezoelektrické části samotné. V dalších krocích jsou elektrické a elastické vlastnosti přiblíženy vlastnostem reálného materiálu tak, aby byla v plné míře odzkoušena interakce všech částí modelu. Parametry jednotlivých případů jsou shrnuty v tab. 5.4. Výpočty jsou prováděny na vzorku ve stejném uspořádání jako v části 5.2.

Pro první test můžeme vzhledem ke geometrii vzorku a materiálovým parametrům napsat analytické řešení. Nenulové složky tenzoru elastických napětí, tenzoru deformace a vektoru posunutí jsou vzhledem k zadání úlohy

| | test č.1 | test č.2 |
|---|---|---|
| délka vzorku l [m] | 10 | 10 |
| příčný průřez P $[m^2]$ | 4 | 4 |
| šířka vzorku t [m] | 4 | 4 |
| Působící síla **F** [N] | 0 | 0 |
| Napětí na horní elektrodě $\varphi_h$ [V] | 1000 | 1000 |
| Napětí na dolní elektrodě $\varphi_d$ [V] | 0 | 0 |
| tenzor elastických modulů **c** [Pa] | $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ |
| tenzor piezoelektrických modulů **e** [V/m] | $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -5 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ |
| tenzor permitivity $\varepsilon$ [F/m] | $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ |

Tabulka 5.4: Parametry testů piezoelektrické části

složky $T_{22}$, $S_{22}$, $u_2$. Můžeme psát:

$$E_2 = -\frac{\varphi_h - \varphi_d}{t} = -250V/m \tag{5.8}$$

$$d_{222} = e_{222} \cdot c_{2222} = 20m/V \tag{5.9}$$

$$S_{22} = d_{222}E_2 = -5000 \tag{5.10}$$

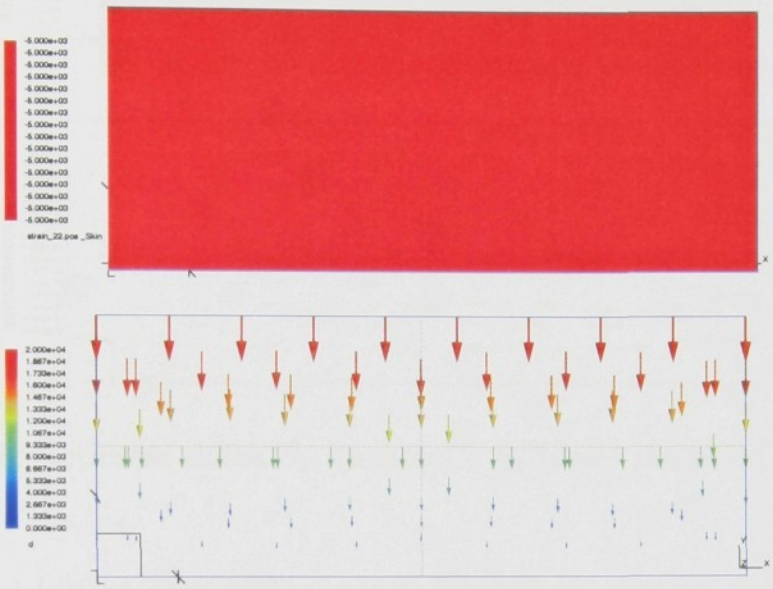
$$u_{2max} = S_{22}t = -20000m, \tag{5.11}$$

kde $d_{222}$ je prvek tenzoru piezoelektrických koeficientů.

Vzhledem k charakteru řešené úlohy - kombinaci elektrických a elastických polí - je nutné zadat okrajové podmínky jak pro posunutí tak pro potenciál. Vzorek je mechanicky upevněn na dolní elektrodě - zde je zadána Dirichletova okrajová podmínka $u_D = 0$. Na ostatních stěnách je zadána homogenní Neumannova okrajová podmínka $\mathbf{T} \cdot \mathbf{n} = 0$. V části popisující elektrické vlastnosti jsou na elektrodách zadány Dirichletovy okrajové podmínky

- na horní elektrodě $\varphi_{D1} = \varphi_h$, na dolní elektrodě $\varphi_{D2} = \varphi_d$. Na ostatních stěnách je zadána homogenní Neumannova podmínka, tedy $D_N = 0$.

Prostorové rozložení složek vypočtených veličin můžeme vidět na obr. 5.7. Zobrazené výsledky modelu se plně shodují s analytickým řešením.



Obrázek 5.7: Rozložení složek $S_{22}$ (nahoře) a $u_2$ (dole) pro první piezoelektrický test.

Druhý test se od prvního liší obsazením prvku $e_{211}$ v matici piezoelektrických modulů. Tento prvek transformuje přenos složky $E_2$ elektrického pole do mechanického působení ve směru osy $x_1$. Mechanické okrajové podmínky jsou: $u_D = 0$ na levé straně vzorku, ostatní stěny $\mathbf{T} \cdot \mathbf{n} = 0$. Elektrické okrajové podmínky jsou stejné jako v prvém případě. Opět můžeme psát analytické řešení:

$$E_2 = -\frac{\varphi_h - \varphi_d}{t} = -250V/m \tag{5.12}$$

$$d_{211} = e_{211} \cdot c_{1111} = -5m/V \tag{5.13}$$

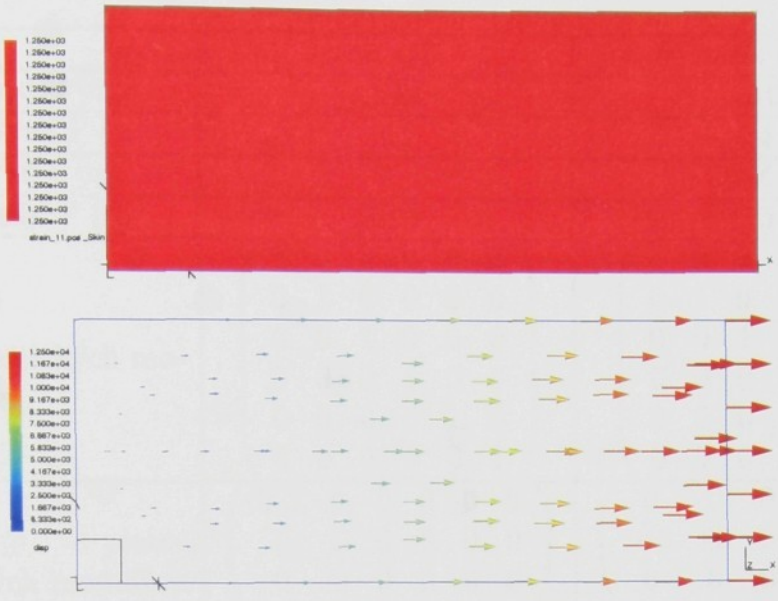
$$S_{11} = d_{211} E_2 = 1250 \tag{5.14}$$

$$u_{2max} = S_{22} t = 12500m, \tag{5.15}$$

kde $d_{211}$ je prvek tenzoru piezoelektrických koeficientů.

Vypočtené veličiny jsou na obr. 5.8. Výsledky se shodují s analytickým řešením.

Uvedené ověřovací testy byly zaměřeny na verifikaci převráceného piezoelektrického jevu, tedy přenosu elektrické energie na mechanickou. Další



Obrázek 5.8: Rozložení složek $S_{11}$ (nahoře) a $u_1$ (dole) pro první piezoelektrický test.

testy piezoelektrické části budou určeny k verifikaci přímého piezoelektrického jevu. Na vzorek bude působit mechanická síla a budeme zkoumat její vliv na elektrické veličiny. Výpočty budou prováděny na stejné konfiguraci vzorku jako v části 5.1. Parametry testů jsou uvedeny v tabulce 5.5.

Pro test č. 3 můžeme vzhledem ke tvaru materiálových tenzorů a zatěžující síly napsat analytické řešení:

$$s_{1111} = c_{1111} = 1\ m^2/N \tag{5.16}$$

$$d_{111} = e_{111} \cdot s_{1111} = 0,2\ m/V \tag{5.17}$$

$$T_{11} = \frac{F}{P} = 250\ Pa \tag{5.18}$$

$$D_1 = d_{111} \cdot T_{11} = 50\ C/m^2 \tag{5.19}$$

$$S_{1111} = s_{1111} \cdot T_{11} = 250 \tag{5.20}$$

$$u_1(l) = S_{11} \cdot l = 2500\ m, \tag{5.21}$$

kde $s_{1111}$ je prvek tenzoru elastických koeficientů, $d_{111}$ je prvek tenzoru piezoelektrických koeficientů.

Vetknutí nosníku charakterizuje Dirichletova okrajová podmínka $u_D = 0$. Působící síla je zadána o velikosti $\frac{F}{n}$ v každém uzlu na straně protilehlé vetknutí, kde $n$ je počet uzlů na této straně. Na ostatních stěnách je zadána homogenní Neumannova okrajová podmínka $\mathbf{T} \cdot \mathbf{n} = 0$. Vzorek je elektricky uzemněn v místě vetknutí - Dirichletova okrajová podmínka $\varphi_D = 0$.

| | test č.3 | test č.4 |
|---|---|---|
| délka vzorku l [m] | 10 | 10 |
| příčný průřez P $[m^2]$ | 4 | 4 |
| šířka vzorku t [m] | 4 | 4 |
| Působící síla **F** [N] | 1000 | 0 |
| tenzor elastických modulů **c** [Pa] | $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ |
| tenzor piezoelektrických modulů **e** [V/m] | $\begin{pmatrix} 0,2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -5 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ |
| tenzor permitivity $\varepsilon$ [F/m] | $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ |

Tabulka 5.5: Parametry testů piezoelektrické části

Na všech zbývajících stěnách je zadána homogenní Neumannova podmínka $D_N = 0$.

Výsledky výpočtu vektoru posunutí $u$ a vektoru elektrické indukce $D$ jsou znázorněny na obr. 5.9. V případě elastických veličin se jedná o případ, který je ekvivalentní s testováním elastické části (odstavec 5.1). Průměrná hodnota posunutí v koncových uzlech $u(l)$ je s tolerancí 2% rovna analyticky zjištěné hodnotě. V případě elektrické indukce je výpočet shodný s analytickým řešením v celé oblasti vyjma okolí hranice, kde je zadána zatěžující síla.

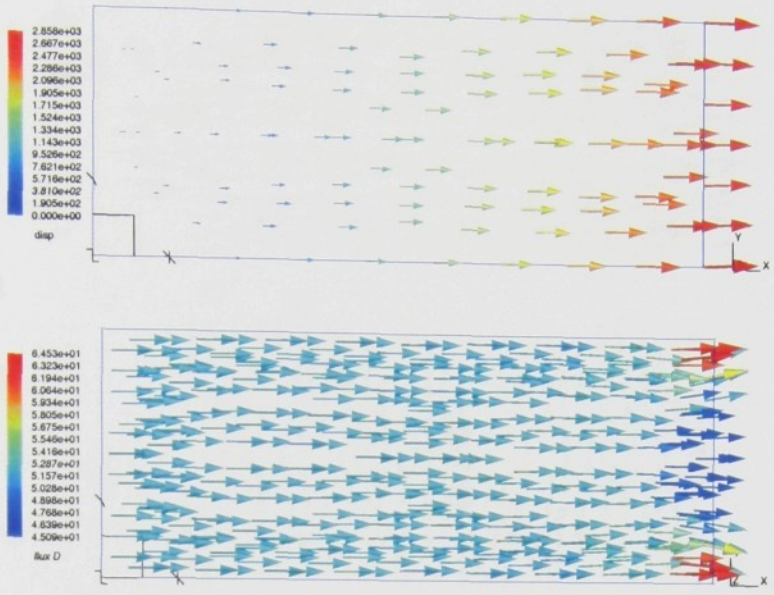
Test č.4 je zaměřen na zjištění přenosu mechanických veličin i do směrů různých od jejich působení. V matici piezoelektrických modulů je na rozdíl od třetího testu nenulový prvek $e_{211}$. Pro tento případ platí vztahy 5.16 - 5.21 s následujícím rozšířením pro složku $D_2$ elektrické indukce. Zanedbáme-li vliv hranic s mechanickými okrajovými podmínkami, můžeme pro střední část vzorku psát přibližné analytické řešení:

$$s_{2222} = c_{2222} = 1\ m^2/N, \tag{5.22}$$

$$d_{211} = e_{211} \cdot s_{2222} = 0,1\ m/V, \tag{5.23}$$

$$D_2 = d_{211} \cdot T_{11} = 25\ C/m^2. \tag{5.24}$$

Na obr. 5.10 jsou znázorněny složky vektoru elektrické indukce. V souladu




Obrázek 5.9: Rozložení složek $u_1$ (nahoře) a $D_1$ (dole) pro třetí piezoelektrický test.

s předpokladem odpovídají hodnoty složek elektrické indukce analytickému řešení ve střední části vzorku.

Následující výpočty budou ověřovat chování modelu při rotaci tenzorů jejich materiálových vlastností. Fyzikálně tato rotace tenzorů odpovídá například změně směru polarizace feroelektrika. Tato fyzikální interpretace je pro nás vzhledem k určení modelu jako prostředku pro analýzu polí ve vícedoménových vzorcích rozhodující. Proto jsou zařazeny testy této vlastnosti. Jsou zároveň zvoleny materiálové tenzory, které se kvalitativně i kvantitativně více blíží vlastnostem reálného feroelektrika. Parametry jsou uvedeny v tabulce 5.6.

Testy budou prováděny na vzorku ve stejném uspořádání jako v části 5.2. Vzorek je mechanicky upevněn na dolní elektrodě - je zadána Dirichletova okrajová podmínka $u_D = 0$. Na ostatních stěnách je zadána homogenní Neumannova okrajová podmínka $\mathbf{T} \cdot \mathbf{n} = 0$. V elektrické části jsou elektrody reprezentovány Dirichletovými okrajovými podmínkami - na horní elektrodě $\varphi_{D1} = \varphi_h$, na dolní elektrodě $\varphi_{D2} = \varphi_d$. Na ostatních stěnách je zadána homogenní Neumannova podmínka $D_N = 0$.

Vzhledem k tomu, že tyto testy ověřují pouze vlastnosti modelu při rotaci materiálových tenzorů, omezíme se pouze na kvalitativní popis, nebudeme analyticky vyčíslovat řešení. Budeme hledat mechanickou odezvu - elastická posunutí - na elektrické buzení vzorku. Vzhledem k umístění elektrod má vektor elektrického pole pouze jednu nenulovou složku $\mathbf{E} = (0, E_2, 0)$. Pro


Obrázek 5.10: Rozložení složek $D_1$ (nahoře) a $D_2$ (dole) pro třetí piezoelektrický test.

testy č.5 a 6. platí:

$$S_{11} = d_{211} \cdot E_2, \tag{5.25}$$

$$S_{22} = d_{222} \cdot E_2, \tag{5.26}$$

$$S_{12} = S_{13} = S_{23} = S_{33} = 0. \tag{5.27}$$

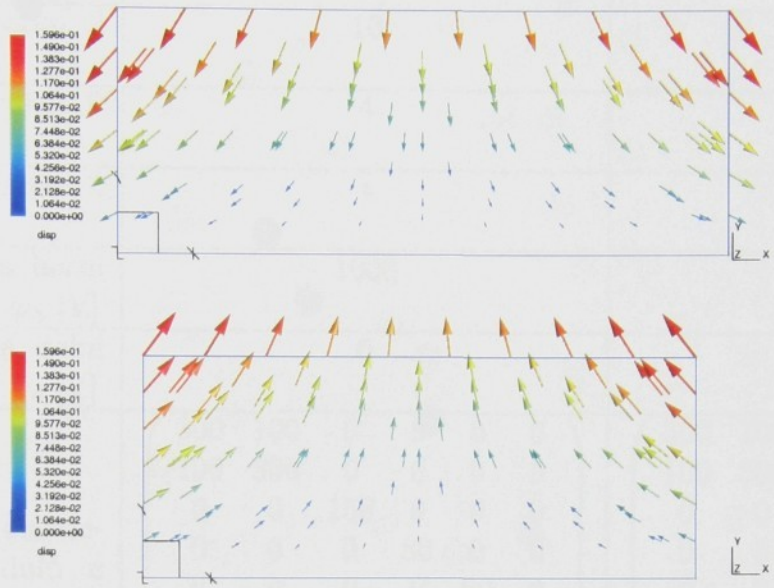
Pro test č. 5 platí, že vzhledem ke směru složky $E_2$ elektrického pole a znaménkům koeficientů $d_{211}$, $d_{222}$, resp. $e_{211}$, $e_{222}$, dojde k dilataci vzorku ve směru osy $x_1$ a kontrakci ve směru osy $x_2$. V případě testu č. 6 dojde ke kontrakci ve směru osy $x_1$ a dilataci ve směru osy $x_2$. Výsledky výpočtu posunutí pro testy č. 5 a 6 jsou na obr. 5.11. Výsledky plně odpovídají předpokladům.

Pro testy č. 7 a 8 platí:

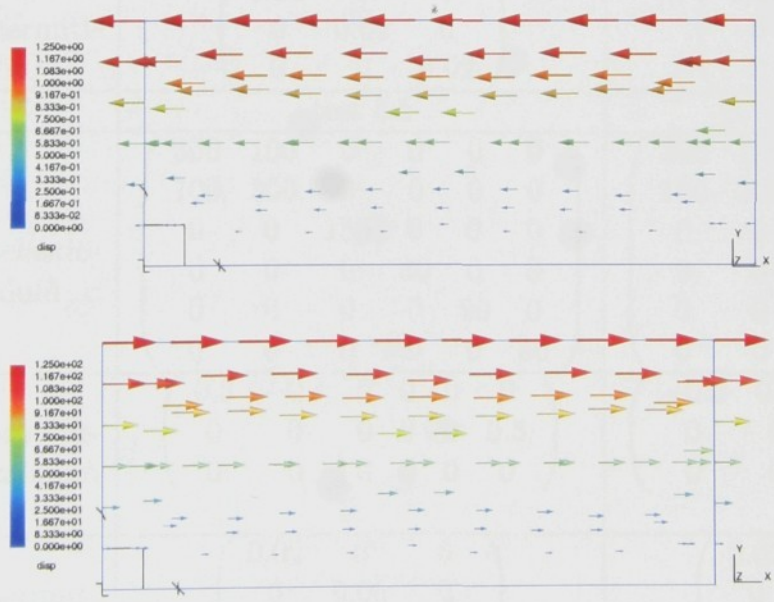
$$S_{12} = d_{212} \cdot E_2, \tag{5.28}$$

$$S_{11} = S_{13} = S_{22} = S_{23} = S_{33} = 0. \tag{5.29}$$

Vzhledem ke směru složky $E_2$ elektrického pole a znaménku koeficientu $d_{212}$, resp. $e_{212}$, dojde v případě testu č. 7 ke střižné deformaci v rovině $x_1, x_2$. Vzorek bude střižně deformován v kladném směru osy $x_1$. V testu č. 8 dojde také ke střižné deformaci v rovině $x_1, x_2$. Vzorek bude střižně deformován v záporném směru osy $x_1$.Výsledky výpočtu posunutí pro testy č. 7 a 8 jsou na obr. 5.12. Vypočtené hodnoty se shodují s teoretickým předpokladem.




Obrázek 5.11: Vizualizace vektorů posunutí pro testy č. 5 (nahoře) a 6 (dole).




Obrázek 5.12: Vizualizace vektorů posunutí pro testy č. 7 (nahoře) a 8 (dole).

| | test č.3 | test č.4 |
|---|---|---|
| délka vzorku l [m] | 10 | 10 |
| příčný průřez P $[m^2]$ | 4 | 4 |
| šířka vzorku t [m] | 4 | 4 |
| Napětí na horní elektrodě $\varphi_h$ [V] | 1000 | 1000 |
| Napětí na dolní elektrodě $\varphi_d$ [V] | 0 | 0 |
| tenzor elastických modulů c [Pa] | $\begin{pmatrix} 300 & 100 & 0 & 0 & 0 & 0 \\ 100 & 300 & 0 & 0 & 0 & 0 \\ 0 & 0 & 150 & 0 & 0 & 0 \\ 0 & 0 & 0 & 80 & 0 & 0 \\ 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$ | $\begin{pmatrix} 300 & 100 & 0 & 0 & 0 & 0 \\ 100 & 300 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 150 & 0 & 0 \\ 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$ |
| tenzor piezoelektrických modulů e [V/m] | $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0.5 \\ -0.1 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -0.5 \\ 0.1 & -0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ |
| tenzor permitivity $\varepsilon$ [F/m] | $\begin{pmatrix} 0.05 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{pmatrix}$ | $\begin{pmatrix} 0.05 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{pmatrix}$ |
| | test č.5 | test č.6 |
| tenzor elastických modulů c [Pa] | $\begin{pmatrix} 300 & 100 & 0 & 0 & 0 & 0 \\ 100 & 300 & 0 & 0 & 0 & 0 \\ 0 & 0 & 150 & 0 & 0 & 0 \\ 0 & 0 & 0 & 80 & 0 & 0 \\ 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$ | $\begin{pmatrix} 300 & 100 & 0 & 0 & 0 & 0 \\ 100 & 300 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 150 & 0 & 0 \\ 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$ |
| tenzor piezoelektrických modulů e [V/m] | $\begin{pmatrix} 0.3 & -0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} -0.3 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ |
| tenzor permitivity $\varepsilon$ [F/m] | $\begin{pmatrix} 0.02 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.02 \end{pmatrix}$ | $\begin{pmatrix} 0.02 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.02 \end{pmatrix}$ |

Tabulka 5.6: Parametry testů piezoelektrické části

# Kapitola 6

# Aplikace modelu pro výpočet elektrických a elastických polí na doménovém rozhraní feroelektrika

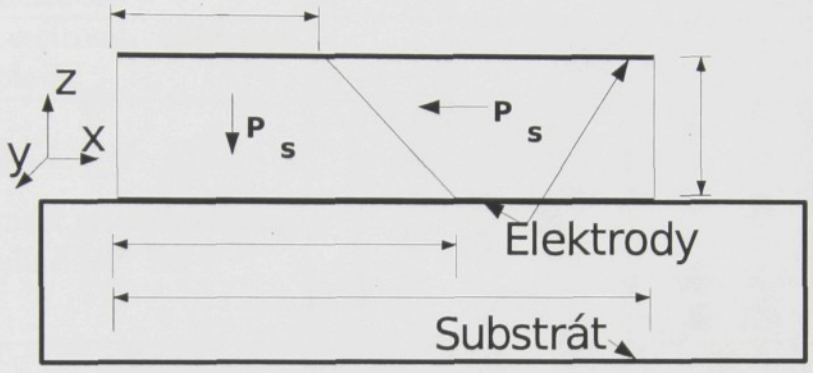
Makroskopické vlastnosti feroelektrických a feroelastických materiálů hrají důležitou roli v jejich aplikacích. V důsledku jejich dielektrických, piezoelektrických a elastických vlastností patří mezi v současné době nejvíce zkoumané materiály. Zatížení vzorků materiálu obsahujících různé doménové stavy elektrickým polem či elastickým napětím vede k rozložení elektrických a elastických polí, která vykazují značnou prostorovou nehomogenitu. Je nanejvýš vhodné uvažovat zásadní roli prostorového rozložení elektrických a elastických polí na jejich materiálové vlastnosti.

Tato charakterizace plně platí pro monokrystaly a z praktického hlediska hraje zásadní roli v tenkých filmech stejně jako ve feroelektrických keramikách. Navzdory tomu, v mnoha vysoce ceněných teoretických a prakticky velmi užitečných publikovaných pracech není uvedená skutečnost, nehomogenní rozložení polí, brána v potaz. Makroskopické vlastnosti vícedoménových vzorků jsou velmi často diskutovány jako veličiny průměrované ze středních hodnot elektrických a elastických polí.

Tato část práce obsahuje jednu z aplikací, ve které byl model použit. Aplikace je z oblasti teoretického materiálového výzkumu feroelektrických tenkých filmů. Jedná se o výpočet prostorového rozložení vektoru intenzity elektrického pole, elektrické indukce, mechanických posunutí, elastických deformací a elastických napětí v okolí $90^0$ doménové stěn v okolí $90^0$ doménové stěny.

## 6.1 Zadání úlohy

Schéma modelu je na obr. 6.1. Na obrázku je znázorněn řez vzorkem. Symbol $P_s$ se šipkou značí směr spontánní polarizace. Jedná se o jednotlivý případ, který se běžně vyskytuje v multidoménových strukturách, $90^0$ doménovou stěnu v tzv. "tail to head" uspořádání spontánní polarizace. Podobný případ je popsán i v literatuře, např. [5], je ale řešen metodou konečných diferencí, nikoliv metodou konečných prvků.




Obrázek 6.1: Schéma zadání úlohy.

Jak je patrné z obrázku, feroelektrický film je upevněn na tuhém substrátu. Na elektrodách, které jsou umístěny na protilehlých stranách vzorku je přivedeno elektrické napětí. Použitý materiál je $BaTiO_3$. Důležité parametry vzorku, naznačené na obr. 6.1 a materiálové vlastnosti jsou uvedeny v tabulce 6.1. Materiálové parametry byly převzaty z publikace [18].

Úkolem je zjistit rozložení elektrického pole, mechanických posunutí, elastických napětí a deformací v vzorkem, který je specifikován schématem na obr. 6.1 a jeho parametry uvedenými v tabulce 6.1.

## 6.2 Příprava modelu

Vzhledem k zadání úlohy, by úloha mohla být dimenzionálně redukována ze tří do dvou prostorových dimenzí. Protože však je model koncipován jak prostorově třídimenzionální, bude provedena pouze zdánlivá redukce dimenze. Ta spočívá v užití geometrie modelu, která bude mít oproti rozměrům ve směru os $x, z$ zanedbatelné rozměry ve směru osy $y$. Směry souřadných os jsou naznačeny na obr. 6.1. Rozměr vzorku ve směru osy $y$ byl zvolen jako

$$w = 2 \cdot 10^{-4}\ m.$$

| parametr | hodnota |
|---|---|
| a | $9 \cdot 10^3\ m$ |
| b | $8 \cdot 10^3\ m$ |
| t | $1 \cdot 10^3\ m$ |
| l | $10 \cdot 10^3\ m$ |
| napětí na horní elektrodě | 10 V |
| napětí na dolní elektrodě | 0 V |
| koercitivní elektrické pole $E_c$ | 50 V/mm |
| tenzor elastických modulů c [$10^9$ Pa] | $\begin{pmatrix} 222 & 108 & 111 & 0 & 0 & 0 \\ 108 & 222 & 111 & 0 & 0 & 0 \\ 111 & 111 & 151 & 0 & 0 & 0 \\ 0 & 0 & 0 & 61 & 0 & 0 \\ 0 & 0 & 0 & 0 & 61 & 0 \\ 0 & 0 & 0 & 0 & 0 & 134 \end{pmatrix}$ |
| tenzor piezoelektrických modulů e [V/m] | $\begin{pmatrix} 0 & 0 & 0 & 0 & 34.2 & 0 \\ 0 & 0 & 0 & 34.2 & 0 & 0 \\ -0.7 & -0.7 & 6.7 & 0 & 0 & 0 \end{pmatrix}$ |
| tenzor permitivity $\varepsilon$ [$10^{-}10$ F/m] | $\begin{pmatrix} 195 & 0 & 0 \\ 0 & 195 & 0 \\ 0 & 0 & 4.96 \end{pmatrix}$ |

Tabulka 6.1: Parametry modelované úlohy

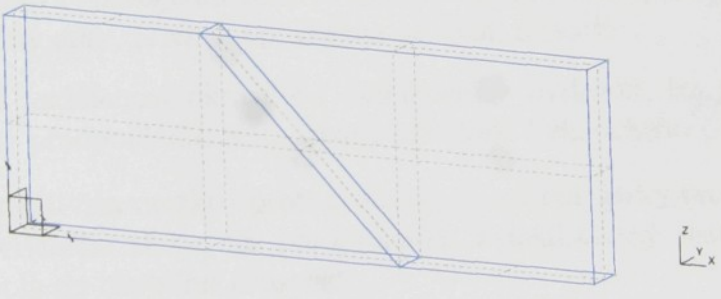
Daším úkolem, který je třeba řešit, je transformace tenzorů materiálových parametrů. Hodnoty uvedené v tabulce 6.1 odpovídají orientaci materiálu v doméně se spontánní polarizací ve směru souřadné osy $z$, tedy levé domény. Tenzory materiálových parametrů musíme pro dosažení orientace materiálu v pravé doméně rotovat o $90^0$. Hodnoty materiálových parametrů po provedení rotace jsou uvedeny v tab. 6.2.

Geometrie vzorku a síť konečných prvků byla tvořena ve volně šiřitelném generátoru GMSH, viz [19]. GMSH umožňuje vytvářenou geometrii dělit na tzv. fyzikální entity, na kterých lze zadávat okrajové podmínky či podle nich definovat oblasti s různými materiálovými vlastnostmi. Toho bylo s výhodou při tvorbě modelu užito. V geometrii modelu není zahrnut pevný substrát ani okolní prostředí vzorku. Tyto budou nahrazeny příslušnými okrajovými podmínkami.

| parametr | hodnota |
|---|---|
| tenzor elastických modulů $\mathbf{c}^{\mathbf{rot}}$ [$10^9$ Pa] | $\begin{pmatrix} 151 & 111 & 111 & 0 & 0 & 0 \\ 111 & 222 & 108 & 0 & 0 & 0 \\ 111 & 108 & 222 & 0 & 0 & 0 \\ 0 & 0 & 0 & 134 & 0 & 0 \\ 0 & 0 & 0 & 0 & 61 & 0 \\ 0 & 0 & 0 & 0 & 0 & 61 \end{pmatrix}$ |
| tenzor piezoelektrických modulů $\mathbf{e}^{\mathbf{rot}}$ [V/m] | $\begin{pmatrix} -6,7 & 0,7 & 0,7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -34,2 \\ 0 & 0 & 0 & 0 & -34,2 & 0 \end{pmatrix}$ |
| tenzor permitivity $\varepsilon^{rot}$ [$10^-10$ F/m] | $\begin{pmatrix} 4.96 & 0 & 0 \\ 0 & 195 & 0 \\ 0 & 0 & 195 \end{pmatrix}$ |

Tabulka 6.2: Materiálové parametry transformované pro pravou doménu.

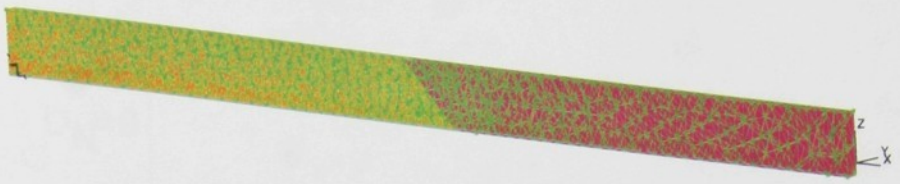
Geometrie modelu je rozdělena na dvě objemové entity, z nichž každá reprezentuje jednu doménu. Dále bylo provedeno dělení stěn na plošné entity tak, aby bylo možné zadat různé okrajové podmínky na různých stěnách. Rozdělení geometrie modelu na objemové a plošné entity je patrné z obr. 6.2.




Obrázek 6.2: Rozdělení geometrie modelu na objemové a plošné entity.

Bylo generováno několik variant sítí konečných prvků. Jako nejlepší řešení byla zvolena síť s vyšší hustotou diskretizace v okolí doménového rozhraní. Důvodem je výskyt velkých nehomogenit a tím i gradientů v prostorovém rozložení elektrických a elastických polí. Výsledná použitá síť je na obr. 6.3. Barevně jsou odlišeny jednotlivé objemové entity.

Jak již bylo řečeno výše, geometrie modelu nezahrnuje ani substrát, na kterém je vzorek upevněn, ani okolní prostředí vzorku. Nahradíme je okra-



Obrázek 6.3: Použitá síť konečných prvků pro model rozložení polí v okolí doménové stěny.

jovými podmínkami. Vzhledem k charakteru řešené úlohy budou zadávány jak okrajové podmínky elektrické, tak elastické.

Budeme uvažovat substrát za ideálně tuhý a zanedbáme vrstvu elektrody mezi substrátem a vzorkem. Elektrodu můžeme zanedbat z důvodu její řádově menší tloušťky než je diskretizace uvažované oblasti. Za těchto podmínek můžeme vliv substrátu na příslušné stěně nahradit Dirichletovou okrajovou podmínkou $u_D = 0$.

Na ostatních vnějších stěnách nepůsobí na vzorek žádná síla a bude tedy zadána homogenní Neumannova okrajová podmínka $t_N = 0$. Na stěnách reprezentujících řez vzorkem bude v důsledku rovnováhy sil zadána též homogenní Neumannova podmínka $t_{iN} = 0$.

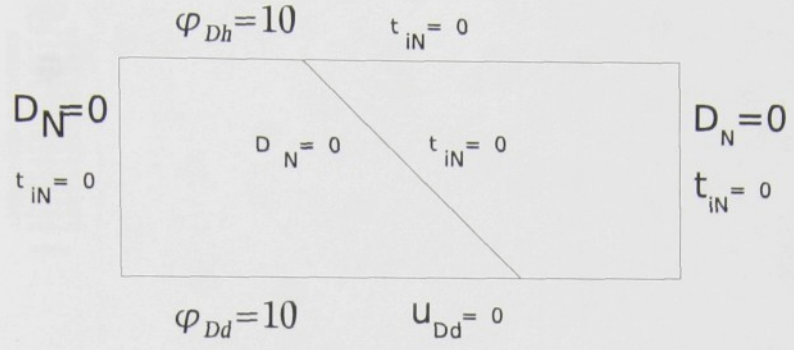
Okrajové podmínky pro elektrickou část jsou následující. Na elektrodách jsou zadány Dirichletovy okrajové podmínky. Na horní elektrodě je podmínka reprezentována hodnotou $\varphi_{Dh} = 10V$ a na dolní elektrodě podmínka $\varphi_{Dd} = 0V$. Na ostatních stěnách, které jsou ve styku s okolním prostředím je zadána Neumannova homogenní podmínka $D_N = 0$. Tato okrajová podmínka idealizuje skutečný stav, je však oprávněna ze dvou důvodů:

- dostatečná vzdálenost okrajů od doménového rozhraní, lze již v této vzdálenosti předpokládat homogenní rozložení elektrického pole,
- vysoká permitivita vzorku oproti okolnímu prostředí, který tvoří vzduch za atmosferického tlaku, lze předpokládat zanedbatelný rozptyl elektrického pole do okolního prostředí.

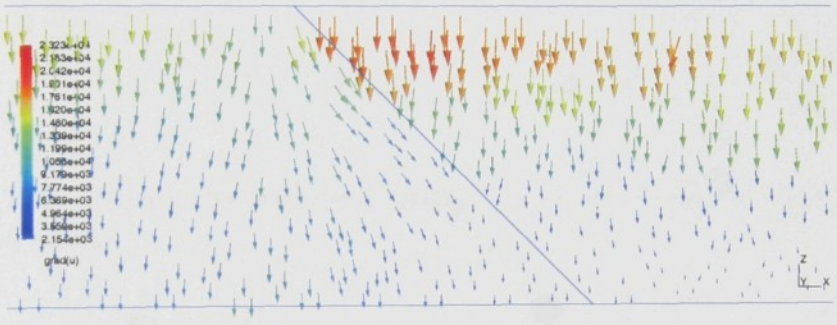
Schéma zadání okrajových podmínek je patrné z obr. 6.4.

## 6.3 Výsledky a jejich diskuse

Úvodem této části se zaměřme na výsledky rozložení elektrických veličin v okolí doménového rozhraní. Na obr. 6.5 je znázorněno prostorové rozložení vektoru elektrického pole. Je zobrazen pouze detail v okolí doménové stěny, v okrajových částech vzorku je elektrické pole již homogenní a tedy již méně zajímavé z hlediska jevů, který zkoumáme.




Obrázek 6.4: Zadání okrajových podmínek ve vzorku s jednou $90^0$ doménocvou stěnou.

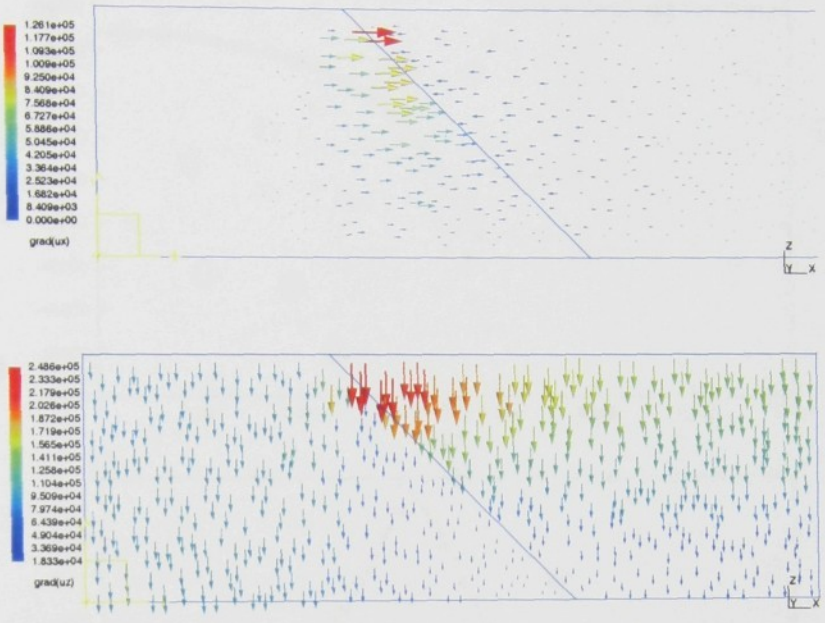


Obrázek 6.5: Prostorové rozložení vektoru elektrického pole v okolí $90^0$ doménové stěny.

Z grafických výsledků je patrné, že výrazné nehomogenity elektrického pole jsou především v místech styku domén s elektrodou. Zajímavé je též rozložení elektrického pole podél celého doménového rozhraní, kde je výrazná složka elektrického pole kolmá k elektrickému poli působícímu na vzorek. Lepší představu o velikosti jednotlivých složek získáme z obr. 6.6, kde jsou zobrazeny složky $E_x$ a $E_z$ pole.

Všimněme si především horního okraje vzorku v okolí doménové stěny. V této části je srovnatelná velikost obou diskutovaných složek elektrického pole. Tento jev lze vysvětlit právě rozhraním dvou anizotropních materiálů.

Z obrázků, kde jsou znázorněny vektory elektrického pole a jejich složky získáváme přehled o oblastech, ve kterých je jejich rozložení z hlediska vlastností struktury nejzajímavější. Nyní právě těmto oblastem věnujme zvýšenou pozornost. Jedná se především o oblasti s extrémními hodnotami vektoru elektrického pole, tedy oblastem v okolí doménového rozhraní a elektrod. V grafech na obrázku 6.7 jsou znázorněny složky $E_y$ elektrického pole v okolí horní a dolní elektrody. Doménové rozhraní probíhá v těchto grafech mezi




Obrázek 6.6: Prostorové rozložení složek $E_x$ (nahoře) a $E_z$ (dole) v okolí doménové stěny.

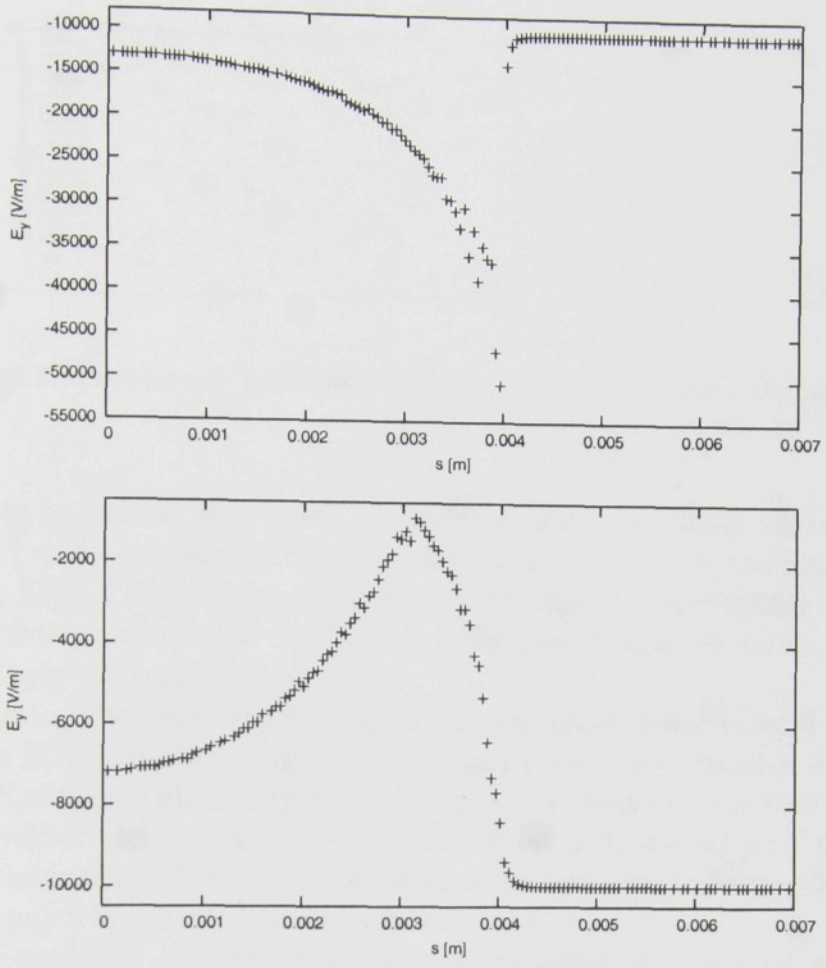
délkovými souřadnicemi 0,003m - 0,004m.

V grafech názorně vidíme oblasti, kde je překročena velikost koercitivního elektrického pole $E_c$. Důsledkem překročení koercitivního elektrického pole je $180^0$ změna směru spontánní polarizace feroelektrika. Pokud uvážíme přepolarizaci v oblastech s polem $E_y > E_c$, bude na doménové stěně generován elektrický náboj, který vyvolá dodatečné elektrické pole působící na vzorek. Pokud přepolarizaci uvážíme, model ztrácí vypovídací schopnost, protože náboje na stěnách neuvažujeme.

Budeme tedy dále uvažovat, že ke změně polarizace nedojde. Potom můžeme v důsledku sil, které elektrické pole vyvolává očekávat deformaci doménové stěny. Tento předpoklad bude diskutován dále, kde se budeme věnovat prostorovému rozložení mechanických veličin. Uvedené výsledky a závěry jsou publikovány v [28], [29].

Dále ukážeme výsledky rozložení mechanických veličin. Na obr. 6.8 jsou znázorněny vektory posunutí.

Vzhledem k zadání okrajových podmínek - nulové posunutí zadané na dolní elektrodě - dochází k výraznějším posunutím pouze v oblasti horní elektrody. Významný gradient posunutí na rozhraní domén u rozhraní elektrod může vést k deformaci doménové stěny. Tuto hypotézu budeme dále sledovat při interpretaci rozložení dalších veličin. Z tohoto důvodu ukážeme i rozložení vybraných složek tenzoru napětí a deformací a sil vypočtených



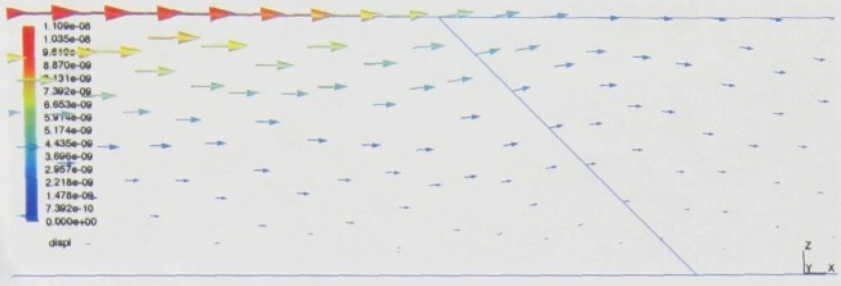

Obrázek 6.7: Rozložení složek $E_y$ podél horní elektrody (nahoře) a podél dolní elektrody (dole) v okolí doménové stěny. Doménová stěna je mezi délkovými souřadnicemi $s$ 0,003m - 0,004m.

podél doménové stěny.

Vzhledem k výrazným složkám posunutí ve směru souřadné osy $x$ budeme sledovat rozložení složky tenzoru napětí $T_{11}$ a složky $S_{11}$ tenzoru deformací. Tyto veličiny jsou prezentovány na obr. 6.9.

Vysoké gradienty složek $S_{11}$ a $T_{11}$ v okolí doménového rozhraní u horní elektrody korespondují s předchozím závěrem, že v uvedené oblasti může dojít k deformaci doménové stěny. Ukážeme rozložení normálových sil podél doménového rozhraní. Síly vypočítáme v řezech naznačených na obr. 6.10 vždy v první vrstvě elementů podél doménového rozhraní. Na obrázku je též naznačena normála $\vec{n}$, v jejímž směru se budou síly počítat a směr a počátek délkové souřadnice $s$ podle níž budou síly vynášeny v grafu.

V grafu na obrázku 6.11 jsou vyneseny síly vztažené na jednotku plochy podél doménového rozhraní tak, jak bylo řečeno výše.



Obrázek 6.8: Prostorové rozložení vektoru posunutí v okolí $90^0$ doménové stěny.

Z grafu je patrné, že v místě upevnění vzorku - na dolní elektrodě jsou si síly na pravé i levé straně doménového rozhraní rovny. Se vzrůstající souřadnicí $s$, blíže k horní elektrodě rozdíl sil vzrůstá. To opět svědčí o tom, že v horní části vzorku může docházet k deformaci doménové stěny. Uvedené výsledky byly též publikovány v [29].

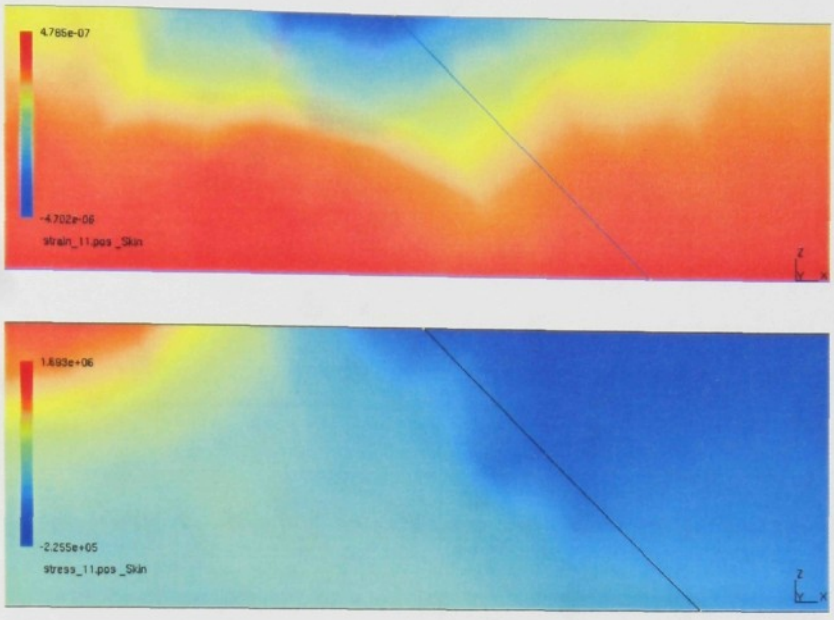
Na tomto místě však musíme upozornit na úskalí, které přináší primární formulace MKP. Spojitost řešení je zaručena pouze pro primární veličiny, v našem případě pro elektrický potenciál $\varphi$ a pro vektory posunutí $u$. Další veličiny, vektor elektrického pole, tenzor napětí a deformací jsou dopočítávány z primárních veličin a jsou konstantní na každém elementu. Není tedy zaručena jejich spojitost na hranicích elementů.

Proto rozdíl ve vypočtených silách na doménovém rozhraní může být způsoben nikoliv fyzikální podstatou modelovaného jevu, ale vlastností použité metody či numerickým řešením stavové soustavy. Odhad chyby metody a numerické chyby řešení přesahuje rámec této práce, která je věnována pouze tvorbě prostředků pro analýzu polí ve feroeletrických a piezoelektrických strukturách.

Jak již bylo uvedeno, stejným efektem - tedy nespojitostí - na hranicích elementů je zatížen i vypočtený vektor elektrické indukce. Častou otázkou odborníků, kteří se zabývají vlastnostmi doménových rozhraní je, zda je splněna spojitost normálových složek vektoru elektrické indukce na rozhraní domén. V limitním případě, kdy parametr diskretizace $h \rightarrow \infty$ bude spojitost splněna. V praktických výpočetních případech je rozdíl normálových složek elektrické indukce závislý na parametru diskretizace $h$. Uvedené charakteristiky můžeme ukázat na obrázku 6.12 pro diskretizaci užitou v našem případě.

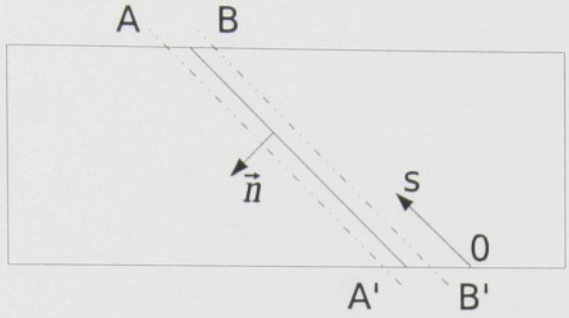
Z obrázku je patrné, že spojitost je téměř dodržena, zřejmé jsou rozdíly v oblastech, kde dochází k velkým gradientům $\varphi$, tedy u horní elektrody.

Na základě prezentovaných výsledků můžeme však tvrdit, že rozložení

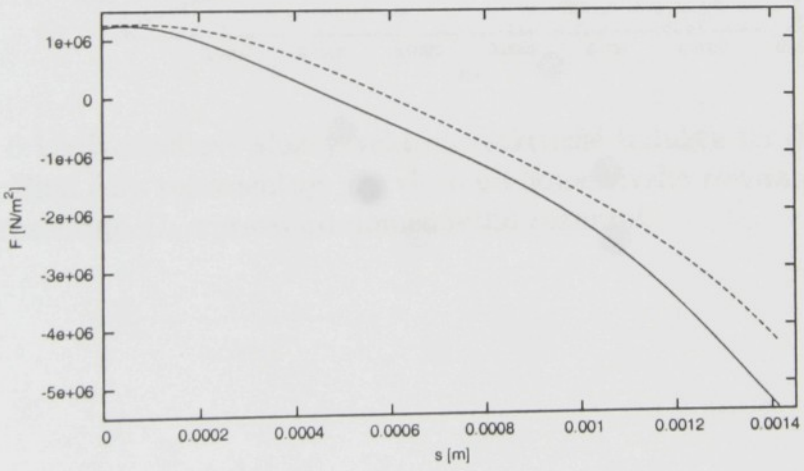



Obrázek 6.9: Prostorové rozložení složek $S_{11}$ tenzoru deformace (nahoře) a $T_{11}$ tenzoru napětí (dole) v okolí doménové stěny.

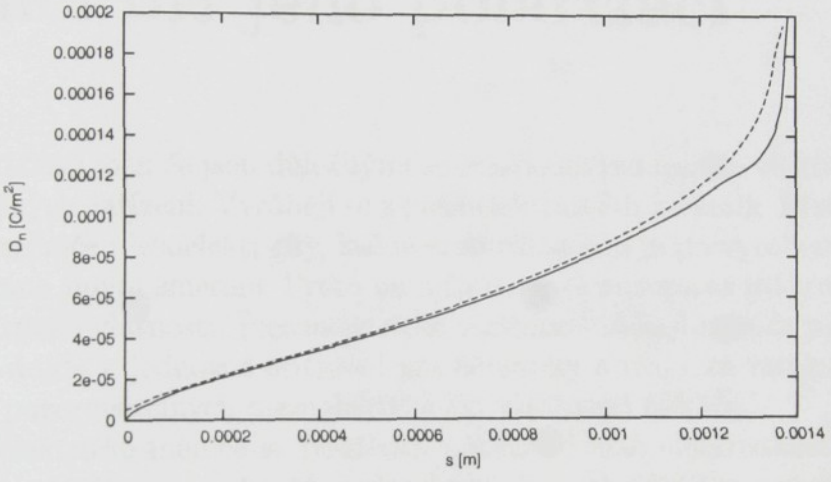
elektrických a mechanických polí má významné důsledky na makroskopické vlastnosti feroelektrických tenkých filmů. Velmi diskutabilní je tedy homogenizace vlastností podle průměrných hodnot polí, která se v uvedených strukturách vyskytují pouze v nezanedbatelné vzdálenosti od doménového rozhraní a nejsou tedy z hlediska aplikačního zajímavá.




Obrázek 6.10: Řezy, ve kterých budou počítány síly podél doménového rozhraní.




Obrázek 6.11: Síly na jednotku plochu podél doménového rozhraní. Plná čára reprezentuje sílu vlevo od doménového rozhraní, přerušovaná reprezentuje sílu napravo od doménového rozhraní.




Obrázek 6.12: Normálové složky vektoru elektrické indukce na doménovém rozhraní. Plná čára reprezentuje $D_n$ vlevo od doménového rozhraní, přerušovaná reprezentuje $D_n$ vpravo od doménového rozhraní.

# Kapitola 7

# Aplikace modelu při analýze poměrů v piezoelektrickém měniči při jeho polarizaci

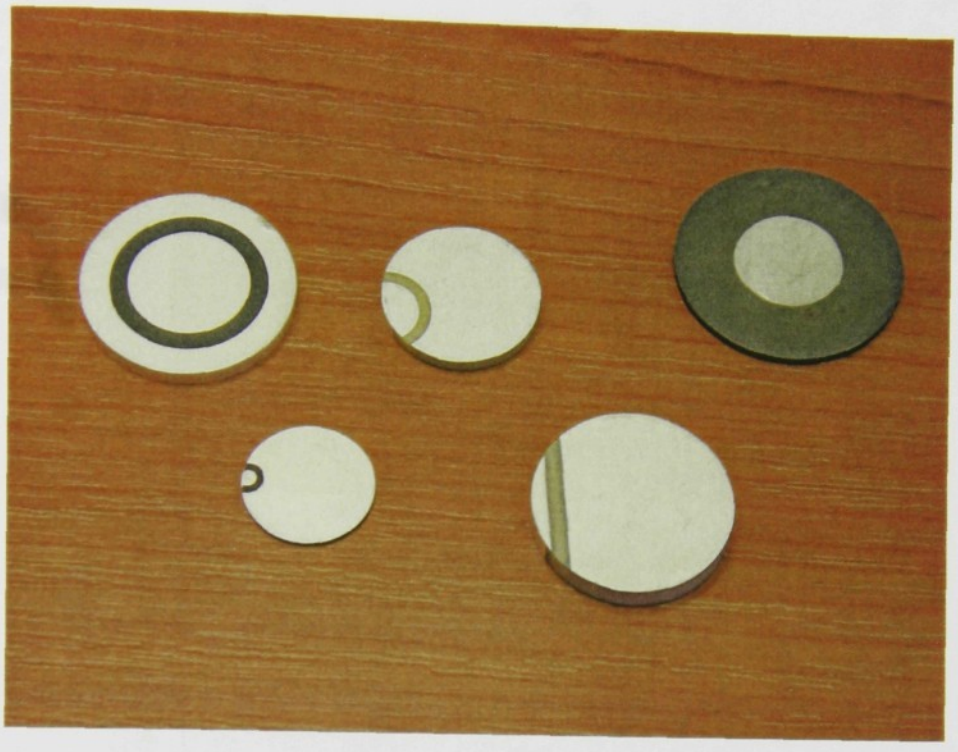
Piezoelektrické měniče jsou důležitými součástkami moderních elektrických a elektronických zařízení. Vyrábějí se z piezoelektrických keramik. Přesto, že je materiál měniče piezoelektrický, každé zrno materiálu je po vyrobení měniče polarizováno jiným směrem. Proto na měniči nelze pozorovat makroskopické piezoelektrické vlatnosti. Piezoelektrické vlastnosti získají měniče polarizací, při které dojde k jednotné orientaci zrn keramiky a tím i ke vzniku makroskopicky pozorovatelných piezoelektrických vlastností měniče.

Piezoelektrické měniče se používají například jako elektroakustické měniče. Tyto měniče mají obvykle jednoduchý geometrický tvar, např. kvádr, válec. Na měniči jsou zpravidla z obou stran elektrody, které slouží jednak jako přívody elektrického napětí při vlastní činnosti měniče, jednak jako elektrody, kterými se přivádí elektrické napětí při polarizaci měniče. Příklady sériově vyráběných měničů jsou na obr. 7.1. Tyto konkrétní měniče jsou z produkce firmy Piezoceram a.s.

V důsledku nehomogenního rozložení elektrického pole v piezoelektrických měničích dochází při jejich polarizaci vlivem změny strukturní mřížky materiálu ke vzniku elastických deformací a napětí. Tato elastická napětí vedou k mechanickým defektům v měniči.

Eliminace nejvýraznějších nehomogenit v rozložení elektrického pole při polarizaci povede k rovnoměrnosti polarizaci a k rovnoměrnějšímu rozložení elastických napětí, k menšímu riziku vzniku strukturních defektů. Jedná se především o potlačení složek elektrického pole v nežádoucích směrech polarizace měniče.

Analýza rozložení elektrického pole v měniči při jeho polarizaci je prvním nutným krokem k modelu, který bude poskytovat údaje o elastických



Obrázek 7.1: Sériově vyráběné piezoelektrické měniče.

poměrech v měniči.

## 7.1 Analýza elektrického pole

### 7.1.1 Zadání úlohy

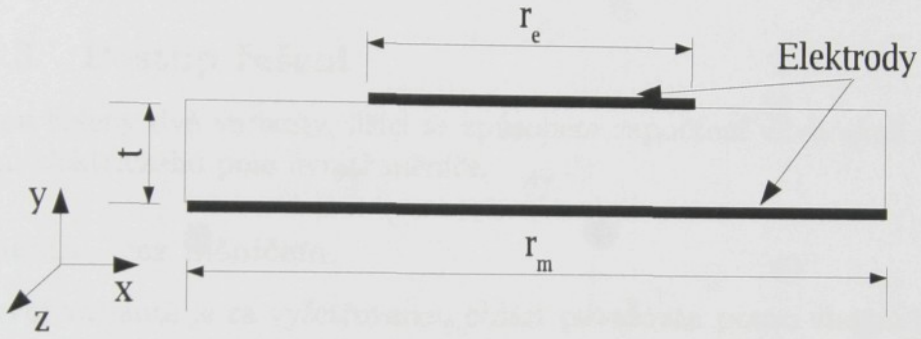
Úkolem je analyzovat rozložení elektrického pole v piezoelektrickém měniči, který je znázorněn na obr. 7.2.

U tohoto měniče dolní elektroda pokrývá celou jeho podstavu, průměr horní elektrody je však menší než průměr měniče. Řez měničem je uveden na obr. 7.3.

Horní elektroda je umístěna symetricky k ose souměrnosti měniče. Významné rozměry parametry měniče a procesu jeho polarizace jsou uvedeny v tab. 7.1.

Před započetím polarizace má vzhledem k náhodné polarizaci jednotlivých zrn materiálu měnič téměř izotropní vlastnosti. Ty se polarizací změní na vlastnosti anizotropní. My budeme v našem modelu pro zjednodušení uvažovat vlastnosti izotropní i v průběhu polarizace. Proto má tenzor permitivity všechny diagonální složky shodné.

Obrázek 7.2: Analyzovaný piezoelektrický měnič.




Obrázek 7.3: Řez analyzovaným měničem. Znázorněny jsou nejdůležitější rozměry a umístění elektrod.

### 7.1.2 Příprava modelu

Vzhledem k parametrům úlohy, především vzhledem k osové symetrii vyšetřované oblasti, může být úloha dimenzionálně redukována ze tří do dvou prostorových dimenzí. Proto bude řešen pouze řez měničem, naznačený na obr. 7.3.

Protože však je model koncipován jak prostorově třídimenzionální, bude provedena pouze zdánlivá redukce dimenze. Ta spočívá v užití geometrie modelu, která bude mít oproti rozměrům ve směru os $x, y$ zanedbatelné rozměry ve směru osy $z$. Směry souřadných os jsou naznačeny na obr. 7.3. Rozměr vzorku ve směru osy $z$ byl zvolen jako

$$w = 1 \cdot 10^{-4}\ m.$$

| parametr | hodnota |
|---|---|
| průměr měniče $r_m$ [mm] | 20 |
| průměr horní elektrody $r_e$ [mm] | 12 |
| tloušťka měniče $t$ [mm] | 1 |
| polarizační elektrické pole $E_{app}$ [kV/mm] | 3 |
| koercitivní elektrické pole $E_c$ [kV/mm] | 1 |
| tenzor permititvity $\varepsilon$ [$10^{-9}$ F/m] | $\begin{pmatrix} 17,6 & 0 & 0 \\ 0 & 17,6 & 0 \\ 0 & 0 & 17,6 \end{pmatrix}$ |

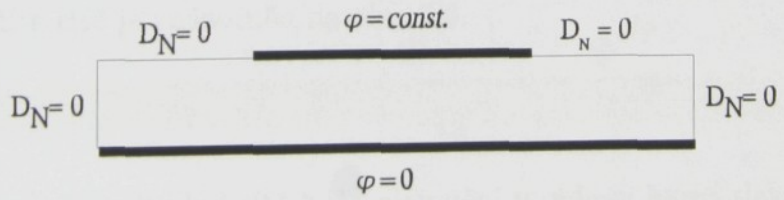
Tabulka 7.1: Parametry zkoumaného piezoelektrického měniče.

### 7.1.3 Postup řešení

Budou řešeny dvě varianty, lišící se způsobem započtení vlivu okolí na rozložení elektrického pole uvnitř měniče.

#### Varianta - řez měničem

V první variantě je za vyšetřovanou oblast považován pouze vlastní řez měničem, při zadání příslušných okrajových podmínek (OKP). Na elektrodách jsou zadány konstantní elektrické potenciály. Rozdíl potenciálů na elektrodách je roven $E_{app} = 3kV/m$. Na zbývajících částech hranice měniče jsou definovány okrajové podmínky $\mathbf{D} \cdot \mathbf{n} = D_N = 0$, kde $\mathbf{n} = (n_x, n_y, n_z)$ je jednotková vnější normála. Jsou tedy definovány nulové složky vektoru elektrické indukce k hranici měniče. Uvedené OKP jsou znázorněny na obr. 7.4.




Obrázek 7.4: Schéma zadání okrajových podmínek na hranici měniče.

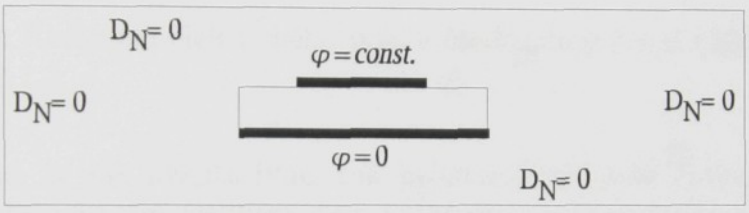
Tento předpoklad o vlivu vnějšího okolí je z hlediska tvorby sítě konečných prvků a výpočetní náročnosti výhodný, protože v modelu je obsažena pouze oblast reprezentující samotný měnič. Díky tomu klesají nároky jak na kvalitu

sítě konečných prvků tak, na vlastnosti užitého řešiče algebraických systémů. Uvedený postup může ale zkreslovat skutečný vliv okolí na rozložení el. pole.

### Varianta - řez měničem a jeho okolím

Opačné výhody a nevýhody než postup uvedený v 7.1.3 - tedy vyšší výpočetní náročnost, naproti tomu však pravdivější interpretace reálného vlivu okolí - má druhý způsob zadání okrajových podmínek. Spočívá v modelu, který obsahuje nejen samotný řez měničem, ale i jeho okolím, ve vzdálenosti rovné zhruba desetinásobku rozměrů měniče.

Potom jsou na hranici měniče zadány okrajové podmínky reprezentující vliv elektrod, nulové složky vektoru elektrické indukce k hranici jsou definovány na hranici sítě. Schématicky je tato varianta znázorněna na obr. 7.5.




Obrázek 7.5: Schéma zadání okrajových podmínek, v případě uvážení okolního prostředí.

### Tvorba sítě konečných prvků

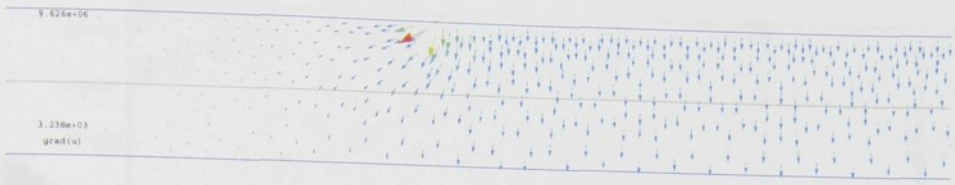
Dalším krokem k řešení je tvorba sítě konečných prvků. Pro dobrou aproximaci výsledků je nutné generovat jemnou síť v oblastech, kde se očekávají velké gradienty počítaných veličin. Velmi často se používá opakované tvorby sítě, která je založena na kontrolním výpočtu na hrubé síti. Tohoto postupu bylo užito i v případě, kdy byla zjemněna oblast v okolí horní elektrody. Příklad užité sítě je znázorněn na obr. 7.6.

Obrázek 7.6: Příklad sítě KP - zjemnění v oblasti horní elektrody.

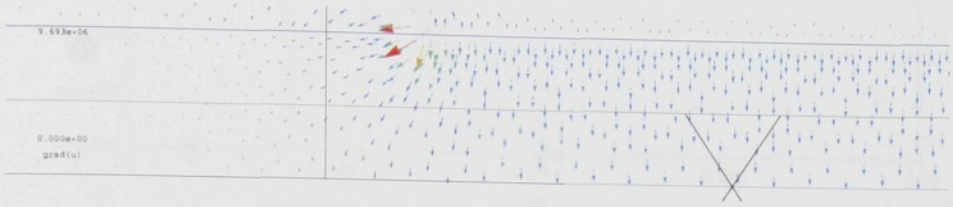
Pro tvorbu byl užit volně šiřitelný generátor GMSH, viz [19].

### Porovnání variant OKP

Obrázky 7.7 a 7.8 dokumentují rozložení elektrického pole v okolí kritického místa, v okolí hrany horní elektrody.



Obrázek 7.7: Rozložení elektrického pole v okolí hrany horní elektrody, varianta podle 7.1.3.



Obrázek 7.8: Rozložení elektrického pole v okolí hrany horní elektrody, varianta podle 7.1.3.

Je patrné, že jak kvalitativně, tak kvantitativně jsou výsledky srovnatelné, samozřejmě s tím rozdílem, že v druhé variantě je vypočteno elektrické pole i v okolí měniče.

Téměř shodné výsledky obou modelů lze odůvodnit vhodně zvoleným okrajovým podmínkám v první variantě a velkému rozdílu permitivit materiálu měniče a vzduchu, který tvoří okolní prostředí. V následujících úvahách bude použit model, který vliv okolního prostředí nahrazuje volbou vhodných okrajových podmínek.

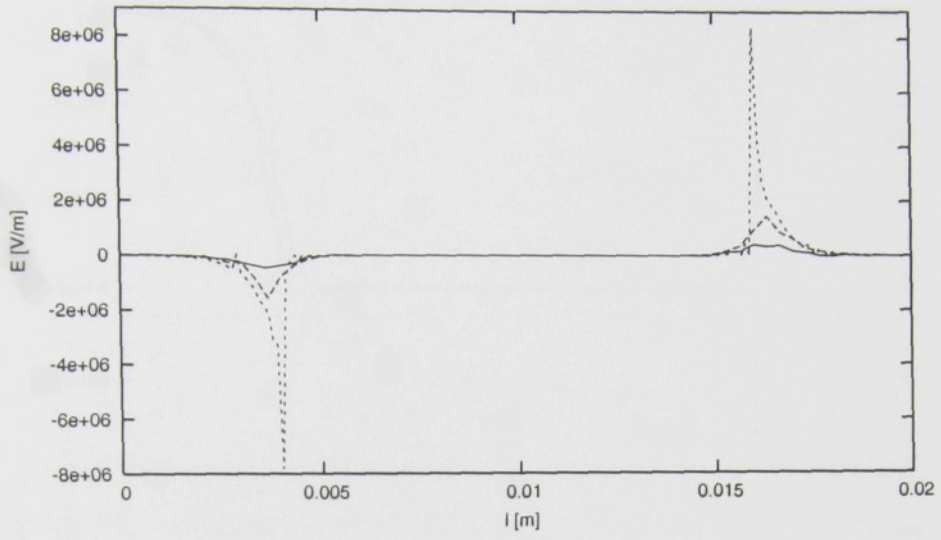
### 7.1.4 Výsledky a jejich diskuse

Výsledky výpočtu lze zobrazit ve dvou formách. Buď jako obrázky, ve kterých je šipkami znázorněn směr a velikost elektrického pole, tak jako obr. 7.7 a 7.8, nebo formou grafu. V grafu mohou být lépe znázorněny složky elektrického pole v různých řezech měničem, proto je zvolena tato varianta.

V grafu na obr. 7.9 je znázorněna složka $E_x$ pole v řezech vedených podél dolní elektrody, v polovině tloušťky měniče a podél horního okraje měniče.

Vzhledem k tomu, že elektrické pole je na každém konečném prvku počítáno v těžišti, nelze (vzhledem k tomu, že prvky jsou čtyřstěny), zobrazit pole přímo na elektrodách či okrajích měniče. Zobrazené elektrické pole v okolí dolní elektrody je zobrazeno ve vzdálenosti $0,1t$ od dolní elektrody, u horního okraje ve vzdálenosti $0,03t$ od horního okraje. Uvedené platí i pro graf na obrázku 7.10, kde je zobrazena $E_y$ složka pole.

Z grafů je patrné, že u horního okraje v okolí hrany horní elektrody bude pravděpodobně docházet k polarizaci ve vodorovném směru, protože složka




Obrázek 7.9: Rozložení složky $E_x$ elektrického pole. Plná čára podél dolní elektrody, tečkovaná v polovině tloušťky, dvojitě tečkovaná podél horního okraje měniče.

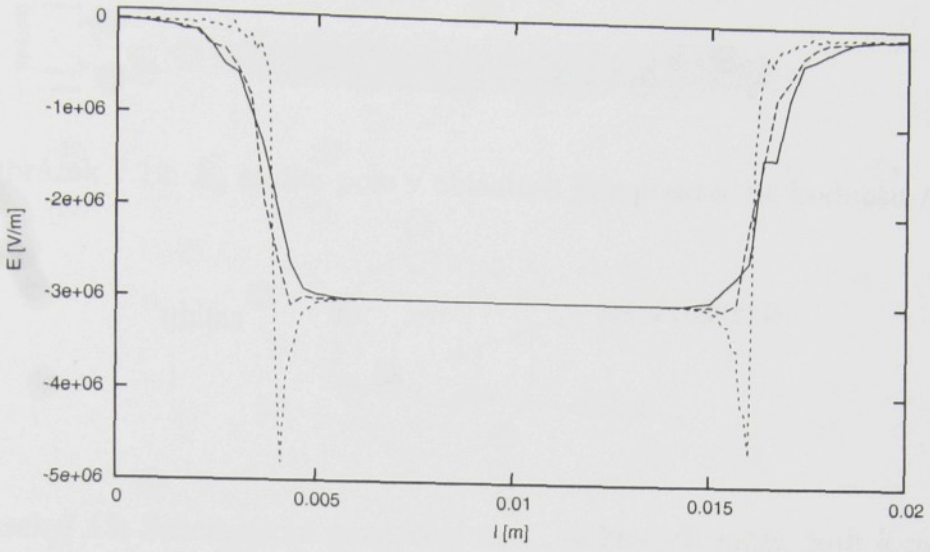
$E_x$ má vyšší amplitudu než složka $E_y$. Zároveň je dodržena podmínka velikosti pole vyššího než $E_c$. K polarizaci ve směru působení aplikovaného pole, ve směru svislém dojde v celé oblasti pod horní elektrodou a v úzkém, cca 1mm okolí.

Pro lepší názornost jsou oblasti polarizace v jednotlivých směrech znázorněny na obrázcích 7.11 a 7.12, kde jsou zobrazeny složky elektrického pole pouze v oblastech, ve kterých překračují $E_c > 1 \cdot 10^6 V/m$.

Z prezentovaných výsledků lze vyvodit závěr, že k největším mechanickým napětím a deformacím bude docházet v prstenci, který bude mít průměr cca 12-15 mm v celé tloušťce měniče. Napětí jsou způsobena jak polarizací ve vodorovném směru, tak nízkou polarizací ve svislém směru mimo oblast ležící pod horní elektrodou.

Tato napětí lze přirovnat k působení hydrostatického tlaku tekutiny na stěnu tlakové nádoby ve tvaru válce. I důsledky tohoto působení budou ekvivalentní. Dojde pravděpodobně k roztržení měniče, které je schématicky znázorněno na obr. 7.13.

Jde však o pouhý odhad důsledků nehomogenity elektrického pole. K podrobnější a přesnější analýze je nutné model doplnit o část, která bude popisovat mechanické vlastnosti uvažované struktury v důsledku působení elektrického pole.




Obrázek 7.10: Rozložení složky $E_y$ elektrického pole. Plná čára podél dolní elektrody, tečkovaná v polovině tloušťky, dvojitě tečkovaná podél horního okraje měniče.

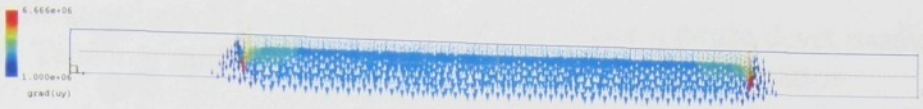
Obrázek 7.11: $E_x$ složka pole v oblastech kde překračuje hodnotu $E_c$.

## 7.2 Analýza elastických polí

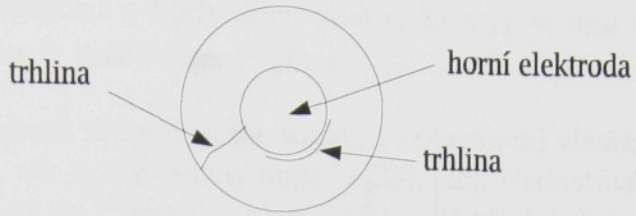
Jak již bylo uvedeno v předchozí části, při polarizaci piezoelektrického měniče dochází vlivem změny polarizace ke vzniku elastických napětí. Tato elastická napětí vznikají jako důsledek změny směru spontánní deformace, která přímo souvisí se změnou směru spontánní polarizace, viz např. [25].

Změna polarizace, resp. spontánní deformace, přímo je přímo ovlivněna elektrickým polem, které na materiál působí. Čím vyšším elektrickým polem na materiál působíme, tím více jeho zrn se přepolarizuje do směru působícího elektrického pole. V tím větším počtu zrn dojde ke změně směru spontánní polarizace a budeindukováno větší elastické napětí.

Tento jev svými důsledky připomíná jev piezoelektrický, tedy že na základě změny elektrické polarizace dojde ke změně elastického napětí. Vliv tohoto jevu na elastická napětí v materiálu je však o několik řádů vyšší, než u jevu piezoelektrického, jak je ukázáno např. v [26].



Obrázek 7.12: $E_y$ složka pole v oblastech kde překračuje hodnotu $E_c$.




Obrázek 7.13: Schématické naznačení míst, ve kterých může dojít k mechanickým defektům v měniči.

Je-li teplota materiálu nižší než Curieova teplota $T_c$, směr spontánní polarizace např. ve směru osy $x_1$, velikost spontánní deformace označme $S^s$. Pokud na těleso nepůsobí žádné vnější síly, můžeme potom o deformacích vzorku předpokládat, že $S_{11} = S^s$, $S_{22} = -\frac{1}{2}S^s$, $S_{33} = -\frac{1}{2}S^s$, viz [26].

Uvedených faktů, tedy ekvivalence s jevem piezoelektrickým a kvantitativní převahy nad jevem piezoelektrickým, využijeme při konstrukci modelu, který bude popisovat rozložení elastických napětí v závislosti na polarizačním elektrickém poli.

Vytvořme jednoduchý matematicko-fyzikální popis uvažovaného jevu. Vyjdeme z faktu, že elastická napětí vznikají na základě změny polarizaci při působení elektrického pole na vzorek. Obecně tedy lze napsat

$$T_{ij} = \eta_{kij} E_k, \tag{7.1}$$

kde $\eta_{kij}$ jsou složky tenzoru třetího řádu, které definují vztah mezi komponentami tenzoru elastických napětí a vektorem elektrického pole. Nyní vytvoříme hypotézu o vlivu elektrického pole na složky tenzoru napětí.

**Ke změně elastických napětí dojde při změně polarizace vždy pouze ve směru působícího elektrického pole a ve směrech na něj kolmých. Elektrické pole tedy bude indukovat nenulové takové složky tenzoru napětí $T_{ij}$, pro které platí, že $i = j$. Elektrické pole tedy nebude indukovat smyková napětí.**

Podle zavedených předpokladů bude mít tenzor $\eta$ pouze devět nezávislých složek. Tenzor $\eta$ můžeme tedy přehledně zapsat ve formě matice

$$\eta = \begin{pmatrix} \eta_{111} & \eta_{122} & \eta_{133} & 0 & 0 & 0 \\ \eta_{211} & \eta_{222} & \eta_{233} & 0 & 0 & 0 \\ \eta_{311} & \eta_{322} & \eta_{333} & 0 & 0 & 0 \end{pmatrix}.$$

Pro složky tenzoru $\eta$ bude navíc platit, že $\eta_{111} = \eta_{222} = \eta_{333}$ a $\eta_{122} = \eta_{133} = \eta_{211} = \eta_{233} = \eta_{311} = \eta_{322} = \frac{1}{2}\eta_{111}$.

Všechna uvedená tvrzení jsou v souladu s obecnými vlastnostmi dielektrik a PZT keramik, jak jsou uváděny např. v [25], [26]. Nedostáváme se tedy uvedenými tvrzeními do rozporu s platnými fyzikálními zákony a vlastnostmi, které zkoumané struktury mají.

Na základě zavedených předpokladů můžeme k výpočtu mechanických napětí, ke kterým dojde při polarizaci měniče, užít model, který je popsán výsledkem této práce. Model je určen pro výpočet elektrických a elastických polí v piezoelektrických strukturách. Pozorně se podíváme na rovnici převráceného piezoelektrického jevu, která je užita při formulaci modelu

$$T_{ij} = c^S_{ijkl} S_{kl} - e_{kij} E_j$$

a srovnáme ji se vztahem 7.1. Zjišťujeme, že pokud v modelu nahradíme tenzor piezoelektrických modulů **e** tenzorem $\eta$, můžeme vytvořený model pro výpočet napětí v důsledku změny spontánní polarizace použít.

### 7.2.1 Zadání úlohy

Úkolem je vyšetřit rozložení mechanických napětí a deformací v piezoelektrickém měniči vyrobeném z PZT keramiky. Jedná se o stejný měnič jako v části 7.1. Proto tabulkou 7.2 pouze doplníme údaje, které nebyly v části 7.1 zmíněny. Hodnoty uvedené v tabulce 7.2 jsou čerpány z [26].

| parametr | hodnota |
|---|---|
| Youngův modul pružnosti $E$ [GPa] | 80 |
| Poissonovo číslo $\nu$ | 0,25 |

Tabulka 7.2: Parametry vyšetřovaného piezoelektrického měniče.

Jak již bylo uvedeno v části 7.1, měnič má před zahájením samotné polarizace izotropní vlastnosti. My se tohoto konstatování přidržíme a budeme

uvažovat materiál měniče za izotropní a budeme zanedbávat změny jeho vlastností v průběhu polarizace.

Při respektování uvedených předpokladů má tenzor elastických modulů tvar, viz [27]:

$$\mathbf{c} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{pmatrix}.$$

Při procesu polarizace nedochází k jednorázové změně spontánní polarizace v celém objemu materiálu. Procento zpolarizovaného materiálu je úměrné velikosti polarizačního pole, jak je uvedeno v [24]. Tato závislost je nelineární, pro potřeby našeho modelu ji zlinearizujeme. Linearizaci můžeme provést bez toho, aniž by byly výrazně zkresleny výsledky modelu. Gradienty reálné závislosti jsou v rozmezí 5% od gradientu linearizované závislosti.

Podle grafu 2.26, který je v [24], je při elektrickém poli $E_{app}$ o velikosti 40 kV/cm indukovaná deformace $S_i$ ve směru působení elektrického pole $1 \cdot 10^{-3}$. Při nulovém elektrickém poli je indukovaná deformace nulová. Z těchto dvou údajů sestavíme vztah, který bude udávat velikost indukované deformace na velikosti elektrického pole

$$S_i = 2,5 \cdot 10^{-10} \cdot E_{app}$$

Nyní můžeme sestavit tenzor $\zeta$, který bude mít stejnou strukturu jako tenzor $\eta$, pouze bude udávat závislost mezi elektrickým polem a deformací, na rozdíl od tenzoru $\eta$, který udává závislost elektrického pole a elastického napětí,

$$\zeta = \begin{pmatrix} 2,5 & 1,25 & 1,25 & 0 & 0 & 0 \\ 1,25 & 2,5 & 1,25 & 0 & 0 & 0 \\ 1,25 & 1,25 & 2,5 & 0 & 0 & 0 \end{pmatrix} \cdot 10^{-10}\ m/V.$$

Jedná se v podstatě o tenzory, které jsou ekvivalentní tenzorům piezoelektrických modulů a koeficientů. Použijeme-li této ekvivalence, můžeme ze znalosti tenzoru elastických modulů a tenzoru $\zeta$ spočítat složky tenzoru $\eta$ jako, viz např. [8],

$$\eta_{kij} = \zeta_{klm} \cdot c_{lmij}, \quad k,i,j = 1,2,3.$$

Pro úplnost zde tenzor elastických modulů a tenzor $\eta$ uvedeme

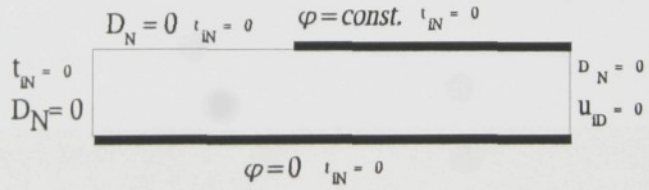
$$\mathrm{c} = \begin{pmatrix} 96 & 32 & 32 & 0 & 0 & 0 \\ 32 & 96 & 32 & 0 & 0 & 0 \\ 32 & 32 & 96 & 0 & 0 & 0 \\ 0 & 0 & 0 & 32 & 0 & 0 \\ 0 & 0 & 0 & 0 & 32 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 \end{pmatrix} \cdot 10^9 \; Pa,$$

$$\eta = \begin{pmatrix} 32 & 24 & 24 & 0 & 0 & 0 \\ 24 & 32 & 24 & 0 & 0 & 0 \\ 24 & 24 & 32 & 0 & 0 & 0 \end{pmatrix} Pa \; mV^{-1}.$$

### 7.2.2 Příprava modelu

Úloha bude vzhledem k symetrii vyšetřované oblasti řešena stejně jako v části 7.1 ve třech prostorových dimenzích. Navíc budeme řešit pouze jednu polovinu řezu, naznačeného na obr. 7.3, protože i samotný řez je symetrický.

Budeme uvažovat, že měnič je pevně uchycen ve svém geometrickém středu. Tento předpoklad nemusí být prakticky dodržen, my ho ale použijeme, abychom vyloučili vliv okrajových podmínek v oblasti, která nás zajímá, tedy v okolí hrany horní elektrody. Kompletní zadání okrajových podmínek je patrné z obr. 7.14

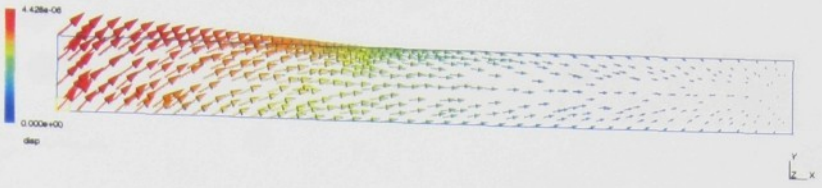



Obrázek 7.14: Schéma zadání okrajových podmínek na hranici měniče.

### 7.2.3 Výsledky a jejich diskuse

Prvním výsledkem, který budeme diskutovat je rozložení posunutí v měniči, viz obr. 7.15.

V okolí uchycení měniče jsou posunutí nulová. V oblasti, která je mezi oběmi elektrodami, tedy v oblasti, kde je vytvořeno homogenní elektrické pole, jsou posunutí rovnoměrná. V této oblasti dochází k rovnoměrné polarizaci materiálu ve svislém směru, proto nelze výrazné nehomogenity v poli



Obrázek 7.15: Rozložení posunutí v měniči.

posunutí očekávat. Je zřejmé, že v okolí hrany horní elektrody dojde v důsledku významné složky $E_x$ elektrického pole k ohybu měniče. V krajní části, kde je elektrické pole prakticky nulové k výrazným změnám posunutí nedochází.

Dalšími výsledky jsou prostorová rozložení složek $S_{11}, S_{12}, S_{22}$ tenzoru deformace, viz obr. 7.16 a složek $T_{11}, T_{12}, T_{22}$ tenzoru napětí, viz obr. 7.17.

Obrázek 7.16: Prostorové rozložení složek $S_{11}, S_{12}, S_{22}$ tenzoru deformace.

Jako první si povšimněme skutečnosti, že složka $S_{12}$ resp. $T_{12}$ je mimo svých extrémních hodnot v okolí okraje elektrody řádově menší než složky $S_{11}, S_{22}$ resp. $T_{11}, T_{22}$. To je logickým důsledkem zavedení předpokladu, že elektrické pole nebude indukovat smyková napětí. Existence smykových napětí a deformací je pouze důsledkem vazby hlavních směrů napětí a deformací se smykovými dané tenzorem elastických modulů.

Již díky znalosti rozložení vektoru posunutí jsme mohli odhadnout, že k největším deformacím a napětím dojde v okolí hrany horní elektrody. Toto

Obrázek 7.17: Prostorové rozložení složek $T_{11}, T_{12}, T_{22}$ tenzoru napětí.

místo je kritické a je v souladu s předpokladem z části 7.1, že ke strukturním defektům bude docházet pravděpodobně právě v okolí hrany horní elektrody.

Na základě získaných výsledků můžeme provést návrh optimalizace umístění elektrod za účelem eliminace napětí, která způsobují praskání měničů při jejich výrobě. Kritériem by tedy byla minimalizace sumy napětí přes celou oblast měniče. Tento úkol ale již přesahuje rámec této práce a proto ho nebudeme v tuto chvíli diskutovat. Přesto však již prezentované výsledky jsou jistým vodítkem k návrhu nového konstrukčního uspořádání měniče.

# Kapitola 8

# Závěr

Disertační práce představuje výsledky výzkumu, jehož hlavními přínosy jsou především implementace kódu primární formulace úlohy metody konečných prvků a jeho testování na základních úlohách a aplikace na vybrané problémy piezoelektrických a feroelektrických struktur.

Přínosem disertační práce je vytvoření funkčního prostředku pro analýzu elektrických a elastických polí v piezoelektrických a feroelektrických strukturách. Tento je založen na primární formulaci metody konečných prvků a implementován v jazyce C. Dalším přínosem je použití tohoto nástroje ve dvou reálných konkrétních aplikacích.

Tyto aplikace ukazují široké možnosti použití modelu, které jsou demonstrovány výpočtem elektrických a elastických polí na doménovém rozhraní feroelektrika a analýzou poměrů v piezoelektrickém měniči při jeho polarizaci.

První aplikace je z oblasti teoretického materiálového výzkumu tenkých vrstev feroelektrik. Výsledky mají i praktické dopady, např. v oblasti konstrukce nových typů paměťových médií. V disertační práci je poukázáno na významnou nehomogenitu elastických a elektrických polí v okolí devadesátistupňové doménové stěny.

Tento jev ovlivňuje makroskopické vlastnosti takových struktur. Lze tedy vyslovit domněnku, že je velmi diskutabilní homogenizace vlastností podle průměrných hodnot polí, která se v uvedených strukturách vyskytují pouze v nezanedbatelné vzdálenosti od doménového rozhraní a nejsou tedy z hlediska aplikačního zajímavá. Výsledky uvedené v disertační práci byly publikovány např. v [28], [29].

V budoucnu by model mohl být rozšířen pro řešení multidoménových struktur a měl by vhodně doplňovat experimentální výzkum.

Druhá aplikace modelu je zaměřena do oblasti průmyslové výroby piezoelektrických měničů. Dotýká se problému mechanických defektů v měniči,

které vznikají v průběhu polarizace. Tyto jsou způsobeny nehomogenním rozložením polarizačního elektrického pole a tím nerovnoměrné polarizace materiálu.

Na konkrétním piezoelektrickém měniči jsou analyzována elektrická a elastická pole a vymezeny oblasti, ve kterých může docházet k mechanickým defektům při polarizaci měniče. Uvedené výsledky se shodují s praktickými zkušenostmi při výrobě piezoelektrických měničů tohoto typu.

Práce též otevřela řadu možných směrů dalšího výzkumu a vývoje modelovacího nástroje. Jedná se především o možnosti upřesnění modelu doplněním jednak nelineárních závislostí mezi veličinami vystupujícími v modelu a tím rozšířit možnosti nasazení tohoto prostředku a dále i napojením na modely, které budou sledovat další přidružené stavy jako je například teplotní pole. To jsou však úkoly, které je třeba řešit vzhledem k jejich rozsahu ve větším kolektivu a tedy přesahují rámec disertační práce. Autorovy publikace [30], [31] ukazují řešení tématiky příbuzné disertační práci.

Pro tyto modely zahrnující nelinearity, respektive neustálené procesy, bude nutné propojit řešení stavových soustav v iteračním procesu. V těchto případech volání externího řešiče může vést k časovému zhroucení výpočtů. Nejinak tomu je i v případě tvarových optimalizací, které matematicky charakterizují geometrické nelinearity a vedou tedy na obdobné numerické potíže, jako nelinearity fyzikální.

Model může být dále rozšířen o nadstavbové moduly, které budou optimalizovat návrh piezoelektrického měniče tak, aby nedocházelo ke strukturním defektům a přitom aby zůstaly zachovány jeho provozní parametry. Jedná se především o optimalizaci tvaru a umístění elektrod.

Dalšími pracemi, které budou na předkládanou disertační práci bezprostředně navazovat, bude srovnání modelu prezentovaného v této práci s modelem na bázi smíšené hybridní formulace metody konečných prvků uvedeným v disertační práci Ing. Jiřiny Královcové.

Každý z modelů má své výhody a nevýhody a jejich vhodnou kombinací lze dosáhnout na řešení úloh, které nejsou samy o sobě dosažitelné pouze jedním typem modelu.

# Příloha A

# Datové soubory

## A.1 Příklad vstupního datového souboru *.mtr

```
2649

86.7400000e9 -8.2538730e9 27.1538730e9 -3.6595873e9 0.0000 0.0000
-8.2538730e9  129.766335e9  -7.4184748e9   5.7004309e9  0.0000  0.0000
27.1538730e9  -7.4184748e9  102.806139e9   9.9212759e9  0.0000  0.0000
-3.6595873e9  5.7004309e9   9.9212759e9    38.611552e9  0.0000  0.0000
0.00000000 0.0000000  0.0000000  0.0000000   68.80e9 2.56e9
0.00000000 0.0000000  0.0000000  0.0000000 2.537e9 29.1e9

0.17100000 -0.1524060 -0.0185940 0.0670099   0.0000000  0.00000
0.00000000  0.0000000  0.0000000 0.0000000   0.1077388  -0.099
0.00000000  0.0000000  0.0000000 0.0000000  -0.0761422  0.06703

39.2100000 0.00000000   0.0000000
0.00000000 39.8162355  0.8578037
0.00000000 0.8578037 40.4237645
```

## A.2 Příklad vstupního datového souboru *.geo

```
d = 0.02;
de = 0.012;
t = 0.001;
w = 0.00010;
h = 0.00025;
Point(1) = {0,0,0,h};
```

```
Point(2) = {d,0,0,h};
Point(3) = {d,t,0,h};
Point(4) = {(0.5*(d+de)),t,0,h/3};

...

Point(11) = {(0.5*(d-de)),t,w,h/3};
Point(12) = {0,t,w,h};
Line(1) = {7,8};
Line(2) = {8,9};
Line(3) = {9,10};
Line(4) = {10,11};

...

Line(17) = {10,4};
Line(18) = {11,5};
Line Loop(19) = {1,2,3,4,5,6};
Plane Surface(20) = {19};
Line Loop(21) = {12,7,8,9,10,11};
Plane Surface(22) = {21};
Line Loop(23) = {16,-12,-15,6};
Plane Surface(24) = {23};

...

Line Loop(33) = {7,-13,-1,16};
Plane Surface(34) = {33};
Surface Loop(35) = {28,26,-20,-34,22,24,32,30};
Volume(36) = {35};
Physical Surface(37) = {34};
Physical Surface(38) = {28};
Physical Volume(39) = {36};
```

## A.3 Příklad vstupního datového souboru *.msh

```
$NOD
2236
1 0 0 0
2 0.02 0 0
```

```
3 0.02 0.001 0
4 0.016 0.001 0
5 0.004 0.001 0

...

3219 0.01501633969600815 0.000504377791182868 4.973027274485881e-05
3220 0.007614784577950692 0.000276628037422205 4.951343504731317e-05
$ENDNOD
$ELM
7279
1 2 37 34 3 36 313 37
2 2 37 34 3 313 36 312
3 2 37 34 3 311 34 310
4 2 37 34 3 311 35 34
5 2 37 34 3 348 349 72

...

7278 4 39 36 4 1552 924 620 1286
7279 4 39 36 4 1282 1433 628 1807
$ENDELM
```

## A.4 Příklad výstupního datového souboru potencialy.pos, posuny.pos

potencialy.pos

```
View " phi " {
SP(0.000000, 0.000000, 0.000000){0.000000};
SP(10.000000, 0.000000, 0.000000){126.428800};
SP(10.000000, 4.000000, 0.000000){120.072700};
SP(0.000000, 4.000000, 0.000000){0.000000};

...

SP(0.661493, 2.567040, 0.473631){10.830290};
SP(9.338780, 1.473871, 0.474056){121.657700};
};
```

posuny.pos

```
View " disp " {
VP(0.00e+00, 0.00e+00, 0.00e+00){0.00e+00, 0.00e+00, 0.00e+00};
VP(1.00e+01, 0.00e+00, 0.00e+00){2.18e+03, 8.93e+01, -1.43e+01};
VP(1.00e+01, 4.00e+00, 0.00e+00){2.13e+03, -8.16e+01, -1.23e+01};
VP(0.00e+00, 4.00e+00, 0.00e+00){0.00e+00, 0.00e+00, 0.00e+00};

...

VP(6.32e-01, 2.55e+00, 4.09e-01){1.78e+02, 1.15e-02, -3.41e-02};
VP(9.80e+00, 1.41e+00, 4.57e-01){2.38e+03, 4.12e+00, -1.21e+01};
};
```

## A.5 Příklad výstupního datového souboru E.pos

```
View " grad(u) " {
VP(0.395663, 2.308542, 0.248747){52.087765, -0.002836, -0.001863};
VP(0.395659, 1.691807, 0.248757){52.087799, 0.002810, -0.001861};
VP(0.447571, 2.308691, 0.748747){52.087455, -0.002454, 0.001214};
VP(0.447568, 1.691956, 0.748757){52.087477, 0.002425, 0.001198};

...

VP(9.733119, 3.329396, 0.871396){56.305156, 12.140325, -1.729539};
VP(9.733123, 0.667668, 0.871254){56.116610, -12.196200, -1.418024};
};
```

## A.6 Příklad výstupního datového souboru strain_11.pos

```
View " strain_11.pos " {
SS(0.000193, 0.000124, 0.000050,  0.000152, 0.000260,
0.000050,  0.000093, 0.000176, 0.000050,  0.000138,
0.000173, 0.000100){-1.748728e-06, -1.748728e-06,
-1.748728e-06, -1.748728e-06};
SS(0.000193, 0.000124, 0.000050,  0.000093, 0.000176,
0.000050,  0.000152, 0.000260, 0.000050,  0.000138,
0.000174, 0.000000){-1.748599e-06, -1.748599e-06,
-1.748599e-06, -1.748599e-06};

...
```

```
SS(0.006439, 0.000631, 0.000100,  0.006787, 0.000501,
0.000100,  0.006517, 0.000311, 0.000100,  0.006595,
0.000497, 0.000000){-3.750473e-04, -3.750473e-04,
-3.750473e-04, -3.750473e-04};
};
Plugin(Skin).Run;
View.Visible=0;
```

## A.7 Příklad výstupního datového souboru force.dat

```
# soubor site:  sample_3.msh
# soubor vektoru: posuny
# rozliseni:  7
# souradnice   x-slozka  y-slozka  z-slozka  normalova slozka
# x_1 = 0.008900 ; y_1 = 0.000000 ; z_1 = 0.000000
# x_2 = 0.007900 ; y_2 = 0.000000 ; z_2 = 0.001000
# x_3 = 0.007900 ; y_3 = 0.001000 ; z_3 = 0.001000

0.000000e+00  -1.210914e+06 3.607393e-313 -5.095725e+05 1.216567e+06
2.020305e-04  -1.450557e+06 3.607393e-313 -7.275049e+05 1.540122e+06
4.040610e-04  1.348635e+05  3.607393e-313 -5.751353e+04 -5.469471e+04
6.060915e-04  5.938303e+05  3.607393e-313 -9.821865e+04 -3.504504e+05
8.081220e-04  1.709235e+06  3.607393e-313 3.428069e+05  -1.451013e+06
1.010153e-03  1.921415e+06  3.607393e-313 -7.777689e+04 -1.303649e+06
1.212183e-03  4.292784e+06  3.607393e-313 8.948169e+05  -3.668188e+06
1.414214e-03  5.826070e+06  3.607393e-313 1.897092e+06  -5.461100e+06
```

# Příloha B

# Zdrojové kódy programu

B.1 Soubor `matice.c`

B.2 Soubor `mkp.c`

B.3 Soubor `gmsh.c`

B.4 Zdrojový kód programu pro výpočet vektoru posunutí a elektrických potenciálů

B.5 Zdrojový kód programu pro výpočet elektrického pole a indukce

B.6 Zdrojový kód programu pro výpočet elastických napětí a deformací

# Literatura


[1] G.L. Link,*Motion of c Domain Centers in BaTiO3*, J. Appl. Phys. 32 (1961) 2566.a

[2] M. Omura, H. Adachi, Y. Ishibashi, *Simulation of Polarization Reversals by a Two-Dimensional Lattice Model*, Jpn. J. Appl. Phys. Vol. 31, pp. 3238-3240, September 1992.

[3] V.N. Nesterov, A.V. Shilnikov, *The Computer Analyses of Dynamics of Domain Boundaries in Ferroelectrics - Ferroelastics*, Ferroelectrics, Vol. 265, pp. 153-159, 2002.

[4] D.B.A. Rep, M.W.J. Prins, *Equivalent-circuit modelling of ferroelectric switching devices*, J. Appl. Phys., Vol. 85, pp. 7923-7930, 1999.

[5] Ricinschi D., Ishibashi Y., Okuyama M., *Electrostatic Model for the Dielectric Permitivity of Ferroelectric Films with* $90^0$ *Domain Structures*, Jpn. J. Appl. Phys. Vol. 42, pp. 6183 - 6187 (2003).

[6] Th. Steinkopff, *Micromechanical Modeling of Ferroelasticity and Ferroelectricity and Finite-Element Results for Nonlinear Piezoelectric Applications*, Ferroelectrics, Vol. 222, pp. 126-129, 1999.

[7] S. Kovalev, M. Sakai, *Numerical Modeling of Electro-Elastic Field in Ferroelectric Crystal Containing* $90^0$ *Twin Boundary*, Acta mater., Vol. 46, pp. 3015-3026, 1998.

[8] J. Zelenka, *Piezoelektrické rezonátory a jejich použití*, Academia, Praha 1983.

[9] J. F. Nye, *Physical Properties of Crystals*, Oxford University Press 2003, ISBN 0 19 851165 5.

[10] Z. Horák, F. Krupka, *Fyzika*, SNTL, Praha 1976.

[11] I. Hlaváček, J. Nečas, *Úvod do matematické teorie pružných a pružně plastických těles*, Praha, SNTL 1983.

[12] S. Míka, A. Kufner, *Parciální diferenciální rovnice I*, SNTL Praha 1983.

[13] K. Rektorys *Variační metody v inženýrských problémech a v problémech matematické fyziky*, Academia, Praha 1999.

[14] J. Nečas, *Les methodes directes en theorie des equations elliptiques*, Masson et Cie, Editeurs, Paris, 1967.

[15] H. Allik, T. J. R. Hughes, *Finite Element Method for Piezoelectric Vibrations*, International Journal for Numerical Methods in Engineering, Vol 2, pp. 151-157, 1970.

[16] O.C. Zienkiewicz, R.L. Taylor, *The Finite Element Method - 4th ed.*, McGraw-Hill Book Company (UK) Limited 1989

[17] D.S. Watkins, *Fundamentals of Matrix Computations*, John Wiley & Sons 1991

[18] M. Zgonik, P. Bernasconi, M. Duelli, R. Schlesser, P.Gunter, *Dielectric, elastic, piezoelectric, electro-optic and elasto-optic tensors of $BaTiO_3$ crystals*, Physical Review B, Vol. 50, No. 9, September 1994.

[19] *GMSH - mesh generator home page*, http://www.geuz.org/gmsh.

[20] *LAPACK - Linear Algebra Package*, http://www.netlib.org/lapack.

[21] *GSL - The GNU Scientific Library, Development Page*, http://sources.redhat.com/gsl.

[22] *Gnuplot Homepage*, http://www.gnuplot.info

[23] K. Uchino, *Ferroelectric Devices*, Marcel Dekker Inc., New York 2000, ISBN 0-8247-8133-3.

[24] K. Uchino, *Piezoelectric Actuators and Ultrasonic motors*, Kluwer Academic Publishers, 1996, ISBN 0792398114.

[25] J. Fousek, *Základy fysiky dielektrik a ferroelektrik*, ČSAV, Praha 1961.

[26] F. Kroupa, K. Nejezchleb, I. Saxl, *Anisotropy of Internal Stresses in Poled PZT Ceramics*, Ferroelectrics, 1988, Vol. 88, pp. 123-137.

[27] Okrouhlík M, *Technická mechanika II.*, ČVUT Praha 1984

[28] J. Novák, J. Maryška, J. Fousek, *Modelling of Electric Field in Ferroelectrics*, Proceedings of ECMS 2003, Liberec, Czech Republic, June 2-4, 2003, pp. 427 - 431. ISBN 80-7083-708-X.

[29] J. Novák, J. Fousek, J. Maryška, M. Marvan, *Distributions of Electric and Elastic Fields at Domain Boundaries*, Material Science & Engineering - B, přijato k publikaci, vyjde v roce 2005.

[30] J. Maryška, J. Novák, P. Rálek, *Modelling of the Resonance Characteristics of the Piezoelectric Resonators*, Proceedings of EFTF 02, St. Petetersburg Russia 2002, ISBN 5-8088-0082-X.

[31] J. Maryška, J. Novák, P. Rálek, J. Šembera, *Finite Element Model of Piezoelectric Resonator*, Current Trends in Scientific Computing, Contemporary Mathematics, Vol. 329, pp. 263-270, 2003.

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

# AUTOREFERÁT DISERTAČNÍ PRÁCE

## Aplikace metody konečných prvků v problémech piezoelektrických a feroelektrických struktur

2004

Josef Novák

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: P 2612 Elektrotechnika a informatika
Studijní obor: 3901V025 Přírodovědné inženýrství

# Aplikace metody konečných prvků v problémech piezoelektrických a feroelektrických struktur

Ing. Josef Novák

Školitel: Doc. Dr. Ing. Jiří Maryška, CSc.

Rozsah práce a příloh:

Počet stran textu: 78
Počet příloh: 5 stran + CD
Počet obrázků: 44
Počet tabulek: 10

Datum: 29. 10. 2004

## Anotace

Disertační práce se zabývá aplikací metody konečných prvků v modelování vybraných problémů piezoelektrických a feroelektrických struktur. Obsahuje fyzikální popis piezoelektrik a feroelektrik, matematický popis modelovaných dějů, odvození slabého řešení problému. Je ukázáno testování úlohy na základních úlohách a aplikace modelu. Model je aplikován na výpočet elektrických a elastických polí v okolí $90^0$ doménové stěny v tenkém filmu feroelektrika a na analýzu mechanických poměrů v piezoelektrickém měniči po jeho polarizaci. Hlavním přínosem je počítačová implementace modelu, jeho testování a aplikace.

## Abstract

The aim of the thesis is the application of the finite element method in modelling of the problems of piezoelectric and ferroelectric structures. It includes physical description of piezoelectric and ferroelectric materials, mathematical description of modelled phenomenons and derivation of weak formulation of the problem. It deals with testing of the model on simple examples and application of the model. Model is applied in modelling of spatial distributions of electric and elastic fields in ferroelectric thin film with two domains separated by a $90^0$ domain wall and in analysis of mechanical properties in piezoelectric transducer after its polarisation. The main results are computer implementation of the model, its testing and application.

# Obsah





# 1 Úvod

Výzkum vlastností piezoelektrických prvků a jejich aplikace má na Technické univerzitě v Liberci dlouholetou tradici, která je spojena především se jménem prof. Jiřího Zelenky. O dosahovaných výsledcích svědčí i úspěšnost při získávání prostředků na financování výzkumu, z nejvýznamnějších jmenujme MŠMT VS 96006 (Fousek, Nosek), MSM 242200002 (Nosek).

Vzhledem k výjimečným vlastnostem feroelektrických a piezoelektrických látek roste zájem o jejich využití v reálných aplikacích a stává se stále aktuálnějším další výzkum jejich vlastností. Z aplikací můžeme jmenovat například využití v oblasti paměťových médií, kdy při zvyšování kapacity velkou roli hraje jejich fyzická velikost, piezoelektrické měniče z piezoelektrických keramik užívané v řadě mechatronických systémů nebo laditelné obvodové prvky.

Vlastnosti paměťových médií z tenkých feroelektrických vrstev jsou z velké části limitovány chováním domén a doménových stěn, které tvoří vlastní paměťové buňky. Podstatným vlivem je rozložení elektrických a elastických polí, které na tyto struktury působí.

Praktické nasazení piezoelektrických měničů je omezeno technologickými možnostmi jejich výroby. Při polarizaci měničů, která je důležitá pro jejich funkci, může vlivem změn v strukturní mřížce docházet ke vzniku mechanických defektů.

Analýza všech uvedených jevů za použití prostředků současné výpočetní techniky a numerických metod je velmi efektivním nástrojem ke zlepšení parametrů piezoelektrických a feroelektrických prvků. Oblasti zkoumání těchto vlastností za pomoci modelů byla věnována pozornost již před několika desítkami let, jak je patrné např. z [1]. V této práci je publikován model popisující rozložení elektrického pole v okolí devadesátistupňové doménové stěny. Principem modelu je nahrazení domén a jejich rozhraní sítí vodičů s vodivostí rozdílnou ve dvou na sebe kolmých směrech.

Podobný model je uveden i v [2], kde je ale již na základě energetických úvah popsán princip přepolarizace domén. V práci [3] je uveden model pohybu doménových stěn v PZT keramikách ve slabém elektrickém poli. Tento je založen na popisu jevu obyčejnou diferenciální rovnicí - pohybovou rovnicí. Jsou též samozřejmě publikovány práce, viz [4], ve kterých je feroelektrikum popsáno pomocí svého náhradního elektrického obvodu složeného z diskrétních R, L, C prvků.

S rozvojem možností výpočetní techniky byly v posledních letech publikovány modely, které již využívají metodu konečných diferencí [5] či metodu konečných prvků [6], [7]. V těchto pracech je ale pro výpočet užito komerčního MKP systému ANSYS, který má implementovány základní moduly pro práci s piezoelektrickými strukturami. Tento software má velmi příjemné uživatelské rozhraní, neumožňuje ale přímo zasahovat do implementovaného programového kódu a upravovat ho pro speciální účely.

Přínosem a cílem disertační práce je implementace vlastního programového kódu

primární formulace MKP a jeho aplikace při výzkumu vlastností feroelektrických struktur a optimalizaci výrobního procesu piezoelektrických měničů. Konkrétně se jedná o prostorová rozložení elektrických a mechanických polí v okolí doménových rozhraní a minimalizaci elastických napětí při polarizaci piezoelektrických měničů.

Obsah disertační práce je koncipován s ohledem na hlavní přínosy a cíle práce, kterým je věnována největší pozornost, do následujících kapitol.

Nejprve jsou ve druhé kapitole popsány základní fyzikální vlastností krystalů, které užijeme při návrhu modelu. Popsány jsou především vlastnosti, kterými se projevují piezoelektrické a feroelektrické materiály. Podrobně popíšeme dielektrické, elastické a piezoelektrické vlastnosti.

Třetí kapitola je zaměřena na matematickou formulaci řešených a odvození jejich slabého řešení. V první části kapitoly je popsána matematicko-fyzikální podstata modelovaných dějů. Poté je formulováno slabé řešení problému definovaném v první části. Závěrečná část kapitoly je věnována aproximaci řešení metodou konečných prvků.

Čtvrtá kapitola obsahuje stručný popis implementace navrženého modelu. Popsány jsou především formáty vstupních a výstupních souborů a hlavní datové struktury.

Úkolem páté kapitoly je otestování všech částí modelu na jednoduchých úlohách tak, aby byla zaručena relevantnost výsledků, které model poskytuje. Vzhledem k primární MKP formulaci, která je užita k sestavení modelu budou testovány především primární veličiny (elektrický potenciál, mechanická posunutí).

Disertační práce je uplatněna ve dvou z mnoha aplikací, které lze postihnout obdobným matematicko-fyzikálním popisem. Jedna je z oblasti teoretického materiálového výzkumu feroelektrických tenkých filmů s jednou nebo více doménovými stěnami. Druhá aplikace je tématem z průmyslové praxe a je zaměřena na optimalizaci technologických parametrů výroby piezoelektrických měničů. Obě témata úzce navazují na institucionální výzkum Fakulty mechatroniky a mezioborových inženýrských studií.

Šestá kapitola se zabývá výpočtem elektrických a elastických polí na doménovém rozhraní feroelektrik. V důsledku jejich dielektrických, piezoelektrických a elastických vlastností patří mezi v současné době nejvíce zkoumané materiály. Zatížení vzorků materiálu obsahujících různé doménové stavy elektrickým polem či elastickým napětím vede k rozložení elektrických a elastických polí, která vykazují značnou prostorovou nehomogenitu. Je nanejvýš vhodné, uvažovat zásadní roli prostorového rozložení elektrických a elastických polí na jejich materiálové vlastnosti.

Sedmá kapitola je zaměřena na analýzu poměrů v piezoelektrickém měniči při jeho polarizaci. V důsledku nehomogenního rozložení elektrického pole v piezoelektrických měničích dochází při jejich polarizaci vlivem změny strukturní mřížky materiálu ke vzniku elastických deformací a napětí. Tato elastická napětí jsou zodpovědná za mechanické defekty v měniči.



Eliminace nejvýraznějších nehomogenit v rozložení elektrického pole povede k rovnoměrnější polarizaci a tím i k rovnoměrnějšímu rozložení elastických napětí, k menšímu riziku vzniku strukturních defektů.

# 2 Fyzikální vlastnosti krystalů

Obsahem této kapitoly v disertační práci je popis základních fyzikálních vlastností, které jsou užity při návrhu modelu. Je čerpáno z publikací [8], [9], [10], [11], kde lze nalézt další podrobnosti.

## 2.1 Dielektrické vlastnosti

Působením vnějšího elektrického pole na dielektrikum dochází k elektrické indukci. Jsou definovány vztahy mezi vektorem elektrického pole a vektorem elektrické indukce a mezi vektorem elektrického pole a elektrickým potenciálem, které jsou uvedeny v části 2.2 práce.

Elektrická polarizace vyvolaná vnějším elektrickým polem $\mathbf{E}$ je definovaná jako rozdíl vektoru elektrické indukce a vektoru vnějšího elekrického pole.

Působí-li na polarizovatelný krystal elektrické pole $\mathbf{E_a}$, je v krystalu indukováno vnitřní, tzv. depolarizační pole $\mathbf{E_i}$. Výsledné elektrické pole je dáno součtem těchto elektrických polí.

U paramagnetických a diamagnetických látek, u kterých dielektrická susceptibilita je nízká, je vliv krystalu na elektrické pole nízký. Toto však neplatí pro feromagnetické látky, kde dielektrická susceptibilita je vysoká. Elektrická pole $\mathbf{E_a}$ i $\mathbf{E_i}$ jsou často stejného řádu. Velikost $\mathbf{E_i}$ je závislé na tvaru krystalu.

Platnost výše uvedených závislostí je zaručena pro krystaly, které se chovají jako ideální izolátor. Tedy pro krystaly, pro které jejich elektrický odpor je $R = \infty$. Reálně dochází vlivem příměsí v každém materiálu k pohybu elektrického náboje, který způsobuje alespoň minimální vodivost.

## 2.2 Elastické vlastnosti

Napjatost tělesa v okolí bodu je tedy popsána devíti složkami $T_{ij}$. Tyto složky označujeme jako složky tenzoru napětí $\mathbf{T}$, např. [11].

Ze zákona zachování momentů sil, viz [11] plyne symetrie tenzoru napětí, platí tedy $T_{ij} = T_{ji} \quad i, j = 1, 2, 3$. Má tedy šest nezávislých složek.

Vlivem napětí dojde k posunutí elementárních částí, které vyjadřujeme vektorem posunutí $\mathbf{u}$. Složky vektoru posunutí vyjadřují posunutí ve směrech os $x_1, x_2, x_3$ ortogonálního systému.

Tato posunutí vyvolají deformaci, která je reprezentována symetrickým tenzorem deformace $\mathbf{S}$. Má obecně také šest nezávislých složek, stejně jako tenzor napětí.

V případě obecné napjatosti je napětí a deformace v každém bodě tělesa charakterizována tenzorem napětí a tenzorem deformace. Lineární teorie pružnosti předpokládá, že každá složka tenzoru napětí je lineární funkcí všech složek tenzoru defor-

mace. Můžeme tedy psát, že $T_{ij} = c_{ijkl}S_{kl}$. Koeficienty $c_{ijkl}$ nazýváme elastickými moduly. Jsou složkami tenzoru čtvrtého řádu, který má obecně 81 prvků. Vzhledem k symetrii tenzoru napětí i deformace a z energetických úvah, viz [11] plyne, že tenzor elastických modulů je symetrický. Celkem je tedy nejvýše 21 nezávislých elastických modulů. Se vzrůstající symetrií krystalové mříže počet nezávislých složek tenzoru elastických modulů dále klesá, viz [8].

### 2.3 Piezoelektrické vlastnosti

U piezoelektrických látek je elektrická polarizace vyvolána nejen působením elektrického pole, ale též působením elastického napětí, resp. v důsledku deformace. Tento jev nazýváme přímým piezoelektrickým jevem.

Obdobně lze vyjádřit i tzv. převrácený piezoelektrický jev, který vyvolává elastické napětí resp. deformaci v důsledku působení elektrického pole na piezoelektrický materiál

### 2.4 Piezoelektrické stavové rovnice

Jak již bylo uvedeno výše, u piezoelektrických látek lze elektrickou polarizaci vyvolat nejen působením elektrického pole, ale též působením elastického napětí resp. v důsledku deformace. Vzájemný vztah mezi těmito dvěma silovými poli lze vyjádřit takzvanými piezoelektrickými stavovými rovnicemi. Při odvození těchto vztahů se vychází např. z vnitřní energie, jak je ukázáno v [8], či z jiného termodynamického potenciálu.

Budeme uvažovat lineární případ, tedy že koeficienty vystupující v těchto rovnicích jsou nezávislé na velikosti nezávisle proměnných a budeme uvažovat pouze izotermické a adiabatické děje. Potom se piezoelektrické stavové rovnice redukují na čtyři možné dvojice, kde rovnice ve dvojici popisují vždy přímý a převrácený piezoelektrický jev.

Piezoelektrické stavové rovnice budeme dále využívat při formulaci úlohy. Budeme však muset respektovat všechna omezení, která limitují platnost lineárních piezoelektrických stavových rovnic. Přesto jsou však pro řadu aplikací, ke kterým je model určen, lineární stavové rovnice vyhovující.

## 3 Matematická formulace úlohy

Tato kapitola práce je věnována matematicko-fyzikálním popisu řešených úloh a odvození jejich slabého řešení. V první části kapitoly je shrnut matematicko-fyzikální popis modelovaných dějů. V druhé části kapitoly je formulováno slabé řešení problému definovaném v první části. Závěrečná část kapitoly je věnována aproximaci řešení metodou konečných prvků.



### 3.1 Matematicko-fyzikální popis elektrického a elastického pole

Chování elektrického pole v obecném anizotropním dielektriku popisují Maxwellovy rovnice, např. [10]. Předpokládáme, že hranice řešené oblasti je rozdělena na dvě části na kterých jsou zadány okrajové podmínky. Na části hranice jsou definovány Dirichletovy okrajové podmínky a na části hranice jsou definovány Neumannovy okrajové podmínky.

Vlastnosti elastického tělesa jsou popsány Newtonovým zákonem zachování sil, např. [10]. Na části hranice jsou definovány Dirichletovy okrajové podmínky a jsou značeny indexem, na části hranice jsou definovány Neumannovy okrajové podmínky.

### 3.2 Slabé řešení

V práci je v tomto odstavci je zformulováno slabé řešení úlohy definované v předchozí části. Řešení podobné úlohy je podrobně matematicky formulováno např. v [15] a není cílem disertační práce jeho odvození, proto jsou zde zavedeny pouze základní pojmy nutné k formulaci slabého řešení, nejsou zde diskutovány vlastnosti samotného slabého řešení. Otázky jednoznačnosti řešení a konvergence aproximací lze nalézrt např. v [13].

Nejprve jsou zavedeny prostory funkcí, kterých je dále využíváno. Při zavedení značení vycházíme i pojmů vycházíme z [13]. Jsou to především prostory funkcí $C_0^{(\infty)}(\overline{\Omega})$, $L_2(\Omega)$, $W_2^{(1)}$.

Dále jsou odvozeny následující integrální rovnosti pro rovnice popisující elektrické a elastické chování, které zapsány v operátorovém tvaru jsou

$$\begin{aligned} \mathbf{A}(\mathbf{u},\mathbf{w}) + \mathbf{D}(\varphi,\mathbf{w}) &= \mathbf{r}(\mathbf{w}), \\ \widetilde{\mathbf{D}}(u,\mathbf{w}) + \mathbf{E}(\varphi,\phi) &= \mathbf{q}(\phi), \end{aligned} \tag{1}$$

kde $\mathbf{A}$ je operátor elastické části, $\mathbf{D}$ operátor piezoelektrické části, $\mathbf{E}$ operátor elektrické části. $\mathbf{r}$ a $\mathbf{q}$ jsou funkcionály charakterizující slabé splnění Neumannových okrajových podmínek.

Stavovou matici problému lze zapsat ve tvaru

$$\begin{pmatrix} \mathbf{K} & \mathbf{D}^{\mathrm{T}} \\ \mathbf{D} & \mathbf{E} \end{pmatrix} \begin{pmatrix} \mathbf{U} \\ \Phi \end{pmatrix} = \begin{pmatrix} \mathbf{R} \\ \mathbf{Q} \end{pmatrix}. \tag{2}$$

kde submatice $\mathbf{K}$ vyjadřuje elastické vlastnosti modelovaného systému, $\mathbf{P}$ piezoelektrické vlastnosti, $\mathbf{E}$ vlastnosti dielektrické. Vektory $\mathbf{R}$ a $Q$ reprezentují Neumannovy okrajové podmínky.

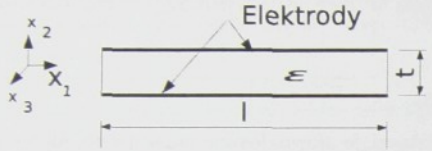
## 4 Testování

V této kapitole jsou ověřeny všechny části modelu na základních úlohách tak, aby byla zaručena relevantnost výsledků, které model poskytuje. V autoreferátu je ukázána pouze část testů piezoelektrické části modelu.

## 4.1 Testování piezoelektrické části modelu

Zásadní význam pro celý model má testování piezoelektrické části. Je testována nejen samotná implementace modulu reprezentující piezoelektrické vlastnosti struktury, ale piezoelektrická část též zprostředkovává přenos elektrické energie na mechanickou a opačně. Je tedy testována i správná interakce mechanické a elektrické části a jejich implementace v globální matici soustavy.

Parametry jednotlivých testů jsou shrnuty v tab. 5.1 v příslušné kapitole práce. Testy budou prováděny na vzorku, jehož schéma je na obrázku 1.




Obrázek 1: Schéma první úlohy pro testování piezoelektrické části.

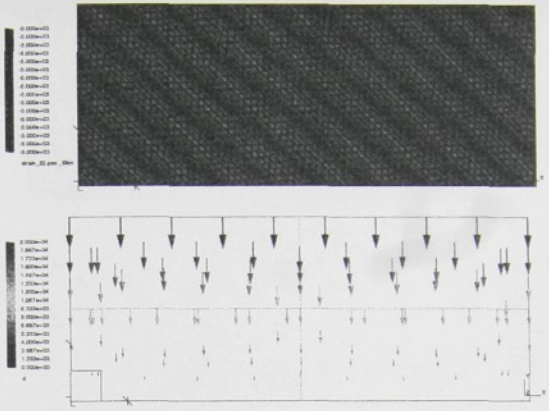
Pro první test můžeme vzhledem ke geometrii vzorku a materiálovým parametrům napsat analytické řešení. Vzhledem k charakteru řešené úlohy - kombinaci elektrických a elastických polí - je nutné zadat jak elastické tak elektrické okrajové podmínky. Vzorek je mechanicky upevněn na dolní elektrodě - je zadána Dirichletova okrajová podmínka $u_D = 0$. Na ostatních stěnách je zadána homogenní Neumannova okrajová podmínka $\mathbf{T} \cdot \mathbf{n} = 0$. V elektrické části jsou elektrody reprezentovány Dirichletovými okrajovými podmínkami - na horní elektrodě $\varphi_{D1} = \text{const}_1$, na dolní elektrodě $\varphi_{D2} = \text{const}_2$. Na ostatních stěnách je zadána homogenní Neumannova podmínka $D_N = 0$.

Prostorové rozložení složek vypočtených veličin můžeme vidět na obr. 2. Zobrazené výsledky modelu se plně shodují s analytickým řešením.

*Druhý test se od prvního liší obsazením prvku* $e_{211}$ *v matici piezoelektrických* modulů. Tento prvek zprostředkovává přenos složky $E_2$ elektrického pole do mechanického působení ve směru osy $x_1$. Mechanické okrajové podmínky jsou: $u_D = 0$ na levé straně vzorku, ostatní stěny $\mathbf{T} \cdot \mathbf{n} = 0$. Elektrické okrajové podmínky jsou stejné jako v případě prvního testu. Opět můžeme vyjádřit analytické řešení, které je uvedeno v kapitole 5.3 práce.

Vypočtené veličiny jsou na obr. 3. Výsledky se opět shodují s analytickým řešením.





Obrázek 2: Rozložení složek $S_{22}$ (nahoře) a $u_2$ (dole) pro první piezoelektrický test.

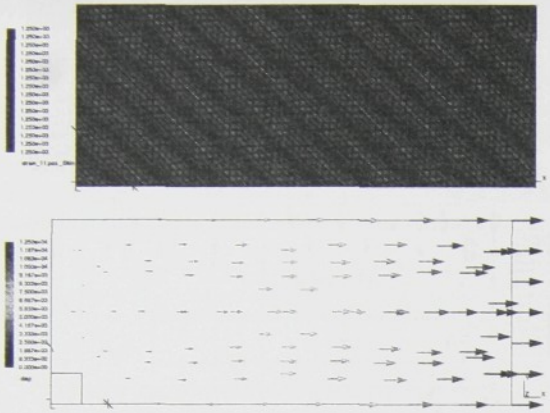
# 5 Aplikace modelu pro výpočet elektrických a elastických polí na doménovém rozhraní feroelektrika

Makroskopické vlastnosti feroelektrických a feroelastických materiálů hrají důležitou roli v jejich aplikacích. V důsledku jejich dielektrických, piezoelektrických a elastických vlastností patří mezi v současné době nejvíce zkoumané materiály. Zatížení vzorků materiálu obsahujících různé doménové stavy elektrickým polem či elastickým napětím vede k rozložení elektrických a elastických polí, která vykazují značnou prostorovou nehomogenitu. Je nanejvýš vhodné uvažovat zásadní roli prostorového rozložení elektrických a elastických polí na jejich materiálové vlastnosti.

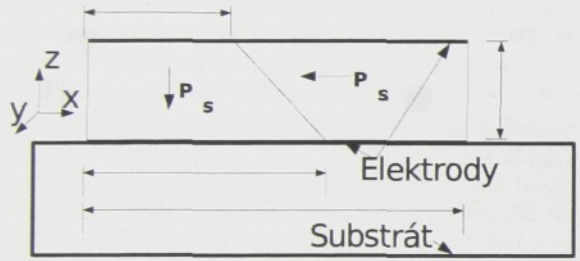
## 5.1 Zadání úlohy

Model je aplikován pro výpočet rozložení elektrických a mechanických veličin v okolí $90^0$ doménové stěny. Schéma modelu je na obr. 4. Na obrázku je znázorněn řez vzorkem. Symbol $P_s$ se šipkou značí směr spontánní polarizace. Jedná se o jednotlivý případ, který se běžně vyskytuje v multidoménových strukturách, $90^0$ doménovou stěnu v tzv. "tail to head" uspořádání spontánní polarizace. Podobný případ je popsán i v literatuře, např. [5], je ale řešen metodou konečných diferencí, nikoliv metodou konečných prvků.

Jak je patrné z obrázku, feroelektrický film je upevněn na tuhém substrátu. Na elektrody, které jsou umístěny na protilehlých stranách vzorku je přivedeno elektrické napětí. Materiál je $BaTiO_3$. Důležité parametry vzorku, naznačené na obr. 4 a materiálové vlastnosti jsou uvedeny v tabulce 6.1 v disertační práci. Materiálové parametry byly převzaty z publikace [18].



Obrázek 3: Rozložení složek $S_{11}$ (nahoře) a $u_1$ (dole) pro první piezoelektrický test.




Obrázek 4: Schéma zadání úlohy.

Úkolem je zjistit rozložení elektrického pole, mechanických posunutí, elastických napětí a deformací v řezu, který je specifikován schématem na obr. 4.

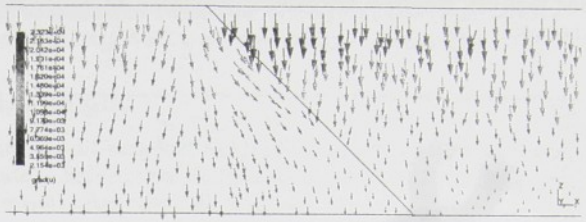
## 5.2 Výsledky a jejich diskuse

Úvodem této části se zaměřme na výsledky rozložení elektrických veličin v okolí doménového rozhraní. Na obr. 5 je znázorněno prostorové rozložení vektoru elektrického pole. Je zobrazen pouze detail v okolí doménové stěny, v okrajových částech vzorku je elektrické pole již homogenní a tedy již méně zajímavé z hlediska jevů, který zkoumáme.

Z grafických výsledků je patrné, že výrazné nehomogenity elektrického pole jsou především v místech styku domén s elektrodou. Zajímavé je též rozložení elektrického pole podél celého doménového rozhraní, kde je výrazná složka elektrického pole kolmá k elektrickému poli působícímu na vzorek.

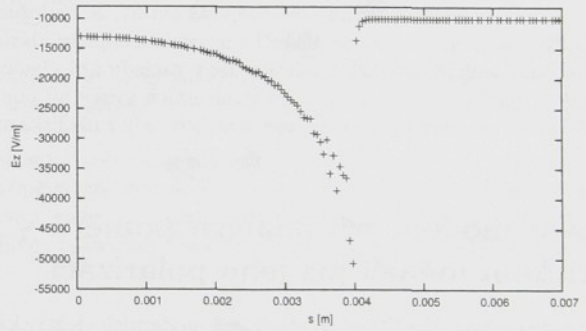
Z obrázku, kde je znázorněn vektor elektrického pole získáváme přehled o oblas-





Obrázek 5: Prostorové rozložení vektoru elektrického pole v okolí $90^0$ doménové stěny.

tech, kde je jeho rozložení z hlediska vlastností struktury nejzajímavější. Nyní těmto oblastem věnujme pozornost. Jedná se především o oblasti s extrémními hodnotami vektoru elektrického pole, tedy oblastem v okolí doménového rozhraní a elektrod. V grafu na obrázku 6 jsou znázorněny složky $E_z$ elektrického pole v okolí horní. Doménové rozhraní probíhá v těchto grafech mezi délkovými souřadnicemi 0,003m - 0,004m.

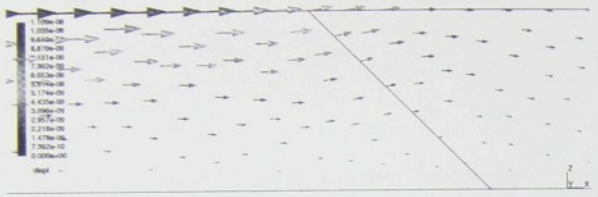



Obrázek 6: Rozložení složek $E_z$ podél horní elektrody v okolí doménové stěny. Doménová stěna je mezi délkovými souřadnicemi $s$ 0,003m - 0,004m.

V grafu názorně vidíme oblasti, kde je překročena velikost koercitivního elektrického pole $E_c$. Důsledkem překročení koercitivního elektrického pole je $180^0$ změna směru spontánní polarizace feroelektrika. Pokud uvážíme přepolarizaci v oblastech s polem $E_z > E_c$, bude na doménové stěně generován elektrický náboj, který vyvolá dodatečné elektrické pole působící na vzorek. Pokud přepolarizaci uvážíme, model ztrácí vypovídací schopnost, protože náboje na stěnách neuvažujeme.

Budeme tedy dále uvažovat, že ke změně polarizace nedojde. Potom můžeme v důsledku sil, které elektrické pole vyvolává očekávat deformaci doménové stěny. Tento předpoklad bude diskutován dále, kde se budeme věnovat prostorovému rozložení mechanických veličin. Uvedené výsledky a závěry jsou publikovány v [28], [29].

Dále ukážeme výsledky rozložení mechanických veličin. Na obr. 7 jsou znázorněny vektory posunutí.



Obrázek 7: Prostorové rozložení vektoru posunutí v okolí $90^0$ doménové stěny.

Vzhledem k zadání okrajových podmínek - nulové posuny zadané na dolní elektrodě - dochází k výraznějším posunům pouze v oblasti horní elektrody. Významný gradient posunutí na rozhraní domén u rozhraní elektrod může mít důsledek v deformaci doménové stěny. Tato hypotéza je v disertační práce dále sledována u rozložení dalších mechanických veličin.

Na základě prezentovaných výsledků můžeme však tvrdit, že rozložení elektrických a mechanických polí má významné důsledky na makroskopické vlastnosti feroelektrických tenkých filmů. Velmi diskutabilní je tedy ustřeďování vlastností podle průměrných hodnot polí, která se v uvedených strukturách vyskytují pouze v nezanedbatelné vzdálenosti od doménového rozhraní a nejsou tedy z hlediska aplikačního zajímavá.

# 6 Aplikace modelu při analýze poměrů v piezoelektrickém měniči při jeho polarizaci

Piezoelektrické měniče jsou důležitými součástkami moderních elektrických a elektronických zařízení. Vyrábějí se z piezoelektrických keramik. Přesto, že je materiál měniče piezoelektrický, každé zrno materiálu je po vyrobení měniče polarizováno jiným směrem. Proto na měniči nelze pozorovat makroskopické piezoelektrické vlastnosti. Piezoelektrické vlastnosti získají měniče polarizací, při které dojde k jednotné orientaci zrn keramiky a tím i ke vzniku makroskopicky pozorovatelných piezoelektrických vlastností měniče.

V důsledku nehomogenního rozložení elektrického pole v piezoelektrických měničích dochází při jejich polarizaci vlivem změny strukturní mřížky materiálu ke vzniku elastických deformací a napětí. Tato elastická napětí jsou vedou k mechanickým defektům v měniči.

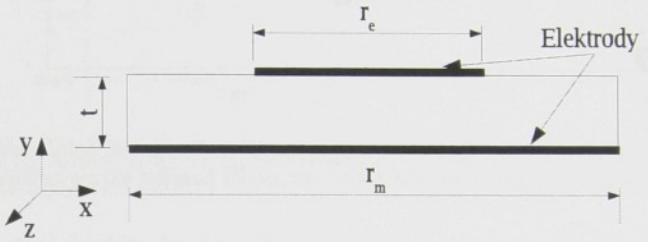
Eliminace nejvýraznějších nehomogenit v rozložení elektrického pole povede k rovnoměrnější polarizaci a tím i k rovnoměrnějšímu rozložení elastických napětí, k menšímu riziku vzniku strukturních defektů. Jedná se především o potlačení složek elektrického pole v nežádoucích směrech polarizace měniče.



Analýza rozložení elektrického pole v měniči při jeho polarizaci je prvním nutným krokem k modelu, který bude poskytovat údaje o elastických poměrech v měniči.

## 6.1 Zadání úlohy

Úkolem je analyzovat rozložení elektrického pole v piezoelektrickém měniči, jehož dolní elektroda pokrývá celou jeho podstavu, průměr horní elektrody je však menší než průměr měniče. Řez měničem je na obr. 8.




Obrázek 8: Řez analyzovaným měničem. Znázorněny jsou nejdůležitější rozměry a umístění elektrod.

Horní elektroda je umístěna symetricky k ose souměrnosti měniče. Významné rozměry parametry měniče a procesu jeho polarizace jsou uvedeny v práci v tab. 7.1.

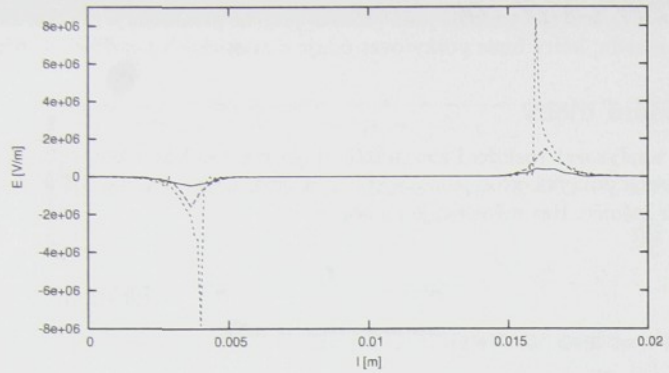
Před započetím polarizace má vzhledem k náhodné polarizaci jednotlivých zrn materiálu měnič téměř izotropní vlastnosti. Ty se polarizací změní na vlastnosti anizotropní. My budeme v našem modelu pro zjednodušení uvažovat vlastnosti izotropní i v průběhu polarizace. Proto má tenzor permitivity všechny diagonální složky shodné.

## 6.2 Výsledky a jejich diskuse

V grafu na obr. 9 je znázorněna složka $E_x$ pole v řezech vedených podél dolní elektrody, v polovině tloušťky měniče a podél horního okraje měniče.

Vzhledem k tomu, že elektrické pole je na každém konečném prvku počítáno v těžišti, nelze (vzhledem k tomu, že prvky jsou čtyřstěny), zobrazit pole přímo na elektrodách či okrajích měniče. Zobrazené elektrické pole v okolí dolní elektrody je zobrazeno ve vzdálenosti $0,1t$ od dolní elektrody, u horního okraje ve vzdálenosti $0,03t$ od horního okraje. Uvedené platí i pro graf na obrázku 10, kde je zobrazena $E_y$ složka pole.

Z grafů je patrné, že u horního okraje v okolí hrany horní elektrody bude pravděpodobně docházet k polarizaci ve vodorovném směru, protože složka $E_x$ má vyšší amplitudu než složka $E_y$. Zároveň je dodržena podmínka velikosti pole vyššího než $E_c$. K polarizaci ve směru působení aplikovaného pole, ve směru svislém dojde v celé oblasti pod horní elektrodou a v úzkém, cca 1mm okolí.




Obrázek 9: Rozložení složky $E_x$ elektrického pole. Plná čára podél dolní elektrody, tečkovaná v polovině tloušťky, dvojitě tečkovaná podél horního okraje měniče.

Pro lepší názornost jsou oblasti polarizace v jednotlivých směrech znázorněny na obrázcích 11 a 12, kde jsou zobrazeny složky elektrického pole pouze v oblastech, kde překračují $E_c > 1 \cdot 10^6 V/m$.

Z prezentovaných výsledků lze vyvodit závěr, že k největším mechanickým napětím a deformacím bude docházet v prstenci, který bude mít průměr cca 12-15 mm v celé tloušťce měniče. Napětí jsou způsobena jak polarizací ve vodorovném směru, tak nízkou polarizací ve svislém směru mimo oblast ležící pod horní elektrodou.

Tato napětí lze dle mého názoru přiblížit k působení tlaku tekutiny na stěnu tlakové nádoby ve tvaru válce. I důsledky tohoto působení budou ekvivalentní. Dojde pravděpodobně k roztržení měniče, které je schématicky znázorněno na obr. 13.

Jde však o pouhý odhad důsledků nehomogenity elektrického pole. K podrobnější a přesnější analýze je nutné model doplnit o část, která bude popisovat mechanické vlastnosti uvažované struktury v důsledku působení elektrického pole.

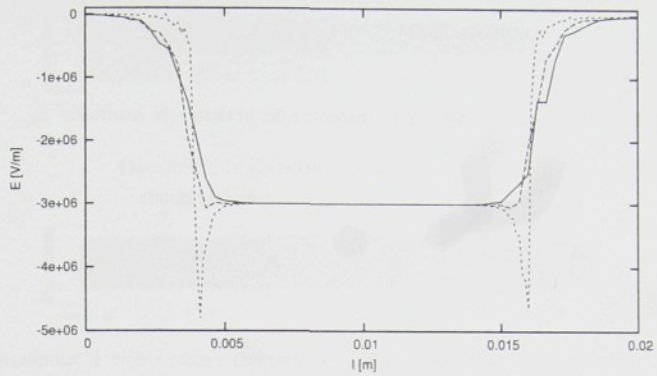
## 6.3 Analýza elastických polí

Jak již bylo uvedeno v předchozí části, při polarizaci piezoelektrického měniče dochází vlivem změny polarizace ke vzniku elastických napětí. Tato elastická napětí vznikají jako důsledek změny směru spontánní deformace, která přímo souvisí se změnou směru spontánní polarizace, viz např. [25].

Změna polarizace, resp. spontánní deformace, je přímo ovlivněna elektrickým polem, které na materiál působí. Čím vyšším elektrickým polem na materiál působíme, tím více jeho zrn se přepolarizuje do směru působícího elektrického pole. V tím větším počtu zrn dojde ke změně směru spontánní polarizace a bude indukováno větší elastické napětí.

Tento jev svými důsledky připomíná jev piezoelektrický, tedy že na základě změny elektrické polarizace dojde ke změně elastického napětí. Vliv tohoto jevu na






Obrázek 10: Rozložení složky $E_y$ elektrického pole. Plná čára podél dolní elektrody, tečkovaná v polovině tloušťky, dvojitě tečkovaná podél horního okraje měniče.

Obrázek 11: $E_x$ složka pole v oblastech kde překračuje hodnotu $E_c$.

elastická napětí v materiálu je však o několik řádů vyšší, než u jevu piezoelektrického, jak je ukázáno např. v [26].

Uvedených faktů, tedy ekvivalence s jevem piezoelektrickým a kvantitativní převahy nad jevem piezoelektrickým, využijeme při konstrukci modelu, který bude popisovat rozložení elastických napětí v závislosti na polarizačním elektrickém poli.

## 6.4 Zadání úlohy

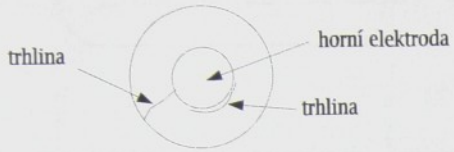
Úkolem je vyšetřit rozložení mechanických napětí a deformací v piezoelektrickém měniči vyrobeném z PZT keramiky. Měnič má před započetím samotné polarizace téměř izotropní vlastnosti. My se tohoto konstatování přidržíme a budeme uvažovat materiál měniče za izotropní a nebudeme uvažovat změny jeho vlastností v průběhu polarizace.

Při procesu polarizace nedochází k jednorázové změně spontánní polarizace v celém objemu materiálu. Procento zpolarizovaného materiálu je úměrné velikosti polarizačního pole, jak je uvedeno v [24].

## 6.5 Výsledky a jejich diskuse

Prvním výsledkem, který budeme diskutovat je rozložení posunutí v měniči, viz obr. 14.

Obrázek 12: $E_y$ složka pole v oblastech kde překračuje hodnotu $E_c$.




Obrázek 13: Schématické naznačení míst, ve kterých může dojít k mechanickým defektům v měniči.

V okolí uchycení měniče jsou posunutí zcela zřejmě nulová. V oblasti, která je mezi oběmi elektrodami, tedy v oblasti, kde je vytvořeno homogenní elektrické pole, jsou posunutí rovnoměrná. V této oblasti dochází k rovnoměrné polarizaci materiálu ve svislém směru, proto nelze výrazné nehomogenity v poli posunutí očekávat. Je zřejmé, že v okolí hrany horní elektrody dojde v důsledku významné složky $E_x$ elektrického pole k ohybu měniče. V krajní části, kde je elektrické pole prakticky nulové k výrazným změnám posunutí nedochází.

Dalšími výsledky jsou prostorová rozložení složek $T_{11}, T_{12}, T_{22}$ tenzoru napětí, viz obr. 15.

Jako první si povšimněme skutečnosti, že složka $T_{12}$ je řádově menší než složky $T_{11}, T_{22}$. To je logickým důsledkem zavedení předpokladu, že elektrické pole nebude indukovat smyková napětí. Existence smykových napětí a deformací je pouze důsledkem vazby hlavních směrů napětí a deformací se smykovými dané tenzorem elastických modulů.

Již díky znalosti rozložení vektoru posunutí jsme mohli odhadnout, že k největším deformacím a napětím dojde v okolí hrany horní elektrody. Toto místo je kritické a ke strukturním defektům bude docházet pravděpodobně právě v okolí hrany horní elektrody.

Na základě získaných výsledků můžeme provést návrh optimalizace umístění elektrod za účelem eliminace napětí, která způsobují praskání měničů při jejich výrobě. Kritériem by tedy byla minimalizace sumy napětí přes celou oblast měniče. Tento úkol ale již přesahuje rámec této práce a proto ho nebudeme v tuto chvíli diskutovat. Přesto však již prezentované výsledky jsou jistým vodítkem k návrhu nového konstrukčního uspořádání měniče.



Obrázek 14: Rozložení posunutí v měniči.

Obrázek 15: Prostorové rozložení složek $T_{11}, T_{12}, T_{22}$ tenzoru napětí.

# 7 Závěr

Disertační práce představuje výsledky výzkumu, jehož hlavními přínosy jsou především implementace kódu primární formulace úlohy metody konečných prvků a jeho testování na základních úlohách a aplikace na vybrané problémy piezoelektrických a feroelektrických struktur.

Přínosem disertační práce je vytvoření funkčního prostředku pro analýzu elektrických a elastických polí v piezoelektrických a feroelektrických strukturách. Tento je založen na primární formulaci metody konečných prvků a implementován v jazyce C. Dalším přínosem je použití tohoto nástroje ve dvou reálných konkrétních aplikacích.

Tyto aplikace ukazují široké možnosti použití modelu, které jsou demonstrovány výpočtem elektrických a elastických polí na doménovém rozhraní feroelektrika a analýzou poměrů v piezoelektrickém měniči při jeho polarizaci.

První aplikace je z oblasti teoretického materiálového výzkumu tenkých vrstev feroelektrik. Výsledky mají i praktické dopady, např. v oblasti konstrukce nových typů paměťových médií. V disertační práci je poukázáno na významnou nehomogenitu elastických a elektrických polí v okolí devadesátistupňové doménové stěny.

Tento jev ovlivňuje makroskopické vlastnosti takových struktur. Lze tedy vyslovit domněnku, že je velmi diskutabilní homogenizace vlastností podle průměrných

hodnot polí, která se v uvedených strukturách vyskytují pouze v nezanedbatelné vzdálenosti od doménového rozhraní a nejsou tedy z hlediska aplikačního zajímavá. Výsledky uvedené v disertační práci byly publikovány např. v [28], [29].

V budoucnu by model mohl být rozšířen pro řešení multidoménových struktur a měl by vhodně doplňovat experimentální výzkum.

Druhá aplikace modelu je zaměřena do oblasti průmyslové výroby piezoelektrických měničů. Dotýká se problému mechanických defektů v měniči, které vznikají v průběhu polarizace. Tyto jsou způsobeny nehomogenním rozložením polarizačního elektrického pole a tím nerovnoměrné polarizace materiálu.

Na konkrétním piezoelektrickém měniči jsou analyzována elektrická a elastická pole a vymezeny oblasti, ve kterých může docházet k mechanickým defektům při polarizaci měniče. Uvedené výsledky se shodují s praktickými zkušenostmi při výrobě piezoelektrických měničů tohoto typu.

Práce též otevřela řadu možných směrů dalšího výzkumu a vývoje modelovacího nástroje. Jedná se především o možnosti upřesnění modelu doplněním jednak nelineárních závislostí mezi veličinami vystupujícími v modelu a tím rozšířit možnosti nasazení tohoto prostředku a dále i napojením na modely, které budou sledovat další přidružené stavy jako je například teplotní pole. To jsou však úkoly, které je třeba řešit vzhledem k jejich rozsahu ve větším kolektivu a tedy přesahují rámec disertační práce. Autorovy publikace [30], [31] ukazují řešení tématiky příbuzné disertační práci.

Pro tyto modely zahrnující nelinearity, respektive neustálené procesy, bude nutné propojit řešení stavových soustav v iteračním procesu. V těchto případech volání externího řešiče může vést k časovému zhroucení výpočtů. Nejinak tomu je i v případě tvarových optimalizací, které matematicky charakterizují geometrické nelinearity a vedou tedy na obdobné numerické potíže, jako nelinearity fyzikální.

Model může být dále rozšířen o nadstavbové moduly, které budou optimalizovat návrh piezoelektrického měniče tak, aby nedocházelo ke strukturním defektům a přitom aby zůstaly zachovány jeho provozní parametry. Jedná se především o optimalizaci tvaru a umístění elektrod.

Dalšími pracemi, které budou na předkládanou disertační práci bezprostředně navazovat, bude srovnání modelu prezentovaného v této práci s modelem na bázi smíšené hybridní formulace metody konečných prvků uvedeným v disertační práci Ing. Jiřiny Královcové.

Každý z modelů má své výhody a nevýhody a jejich vhodnou kombinací lze dosáhnout na řešení úloh, které nejsou samy o sobě dosažitelné pouze jedním typem modelu.

## Reference


[1] G.L. Link,*Motion of c Domain Centers in BaTiO3*, J. Appl. Phys. 32 (1961) 2566.a





[2] M. Omura, H. Adachi, Y. Ishibashi, *Simulation of Polarization Reversals by a Two-Dimensional Lattice Model*, Jpn. J. Appl. Phys. Vol. 31, pp. 3238-3240, September 1992.

[3] V.N. Nesterov, A.V. Shilnikov, *The Computer Analyses of Dynamics of Domain Boundaries in Ferroelectrics - Ferroelastics*, Ferroelectrics, Vol. 265, pp. 153-159, 2002.

[4] D.B.A. Rep, M.W.J. Prins, *Equivalent-circuit modelling of ferroelectric switching devices*, J. Appl. Phys., Vol. 85, pp. 7923-7930, 1999.

[5] Ricinschi D., Ishibashi Y., Okuyama M., *Electrostatic Model for the Dielectric Permitivity of Ferroelectric Films with* $90^0$ *Domain Structures*, Jpn. J. Appl. Phys. Vol. 42, pp. 6183 - 6187 (2003).

[6] Th. Steinkopff, *Micromechanical Modeling of Ferroelasticity and Ferroelectricity and Finite-Element Results for Nonlinear Piezoelectric Applications*, Ferroelectrics, Vol. 222, pp. 126-129, 1999.

[7] S. Kovalev, M. Sakai, *Numerical Modeling of Electro-Elastic Field in Ferroelectric Crystal Containing* $90^0$ *Twin Boundary*, Acta mater., Vol. 46, pp. 3015-3026, 1998.

[8] J. Zelenka, *Piezoelektrické rezonátory a jejich použití*, Academia, Praha 1983.

[9] J. F. Nye, *Physical Properties of Crystals*, Oxford University Press 2003, ISBN 0 19 851165 5.

[10] Z. Horák, F. Krupka, *Fyzika*, SNTL, Praha 1976.

[11] I. Hlaváček, J. Nečas, *Úvod do matematické teorie pružných a pružně plastických těles*, Praha, SNTL 1983.

[12] S. Míka, A. Kufner, *Parciální diferenciální rovnice I*, SNTL Praha 1983.

[13] K. Rektorys *Variační metody v inženýrských problémech a v problémech matematické fyziky*, Academia, Praha 1999.

[14] J. Nečas, *Les methodes directes en theorie des equations elliptiques*, Masson et Cie, Editeurs, Paris, 1967.

[15] H. Allik, T. J. R. Hughes, *Finite Element Method for Piezoelectric Vibrations*, International Journal for Numerical Methods in Engineering, Vol 2, pp. 151-157, 1970.

[16] O.C. Zienkiewicz, R.L. Taylor, *The Finite Element Method - 4th ed.*, McGraw-Hill Book Company (UK) Limited 1989

[17] D.S. Watkins, *Fundamentals of Matrix Computations*, John Wiley & Sons 1991

[18] M. Zgonik, P. Bernasconi, M. Duelli, R. Schlesser, P.Gunter, *Dielectric, elastic, piezoelectric, electro-optic and elasto-optic tensors of $BaTiO_3$ crystals*, Physical Review B, Vol. 50, No. 9, September 1994.

[19] *GMSH - mesh generator home page*, http://www.geuz.org/gmsh.

[20] *LAPACK - Linear Algebra Package*, http://www.netlib.org/lapack.

[21] *GSL - The GNU Scientific Library, Development Page*, http://sources.redhat.com/gsl.

[22] *Gnuplot Homepage*, http://www.gnuplot.info

[23] K. Uchino, *Ferroelectric Devices*, Marcel Dekker Inc., New York 2000, ISBN 0-8247-8133-3.

[24] K. Uchino, *Piezoelectric Actuators and Ultrasonic motors*, Kluwer Academic Publishers, 1996, ISBN 0792398114.

[25] J. Fousek, *Základy fysiky dielektrik a ferroelektrik*, ČSAV, Praha 1961.

[26] F. Kroupa, K. Nejezchleb, I. Saxl, *Anisotropy of Internal Stresses in Poled PZT Ceramics*, Ferroelectrics, 1988, Vol. 88, pp. 123-137.

[27] Okrouhlík M, *Technická mechanika II.*, ČVUT Praha 1984

[28] J. Novák, J. Maryška, J. Fousek, *Modelling of Electric Field in Ferroelectrics*, Proceedings of ECMS 2003, Liberec, Czech Republic, June 2-4, 2003, pp. 427 - 431. ISBN 80-7083-708-X.

[29] J. Novák, J. Fousek, J. Maryška, M. Marvan, *Distributions of Electric and Elastic Fields at Domain Boundaries*, Material Science & Engineering - B, přijato k publikaci, vyjde v roce 2005.

[30] J. Maryška, J. Novák, P. Rálek, *Modelling of the Resonance Characteristics of the Piezoelectric Resonators*, Proceedings of EFTF 02, St. Petetersburg Russia 2002, ISBN 5-8088-0082-X.

[31] J. Maryška, J. Novák, P. Rálek, J. Šembera, *Finite Element Model of Piezoelectric Resonator*, Current Trends in Scientific Computing, Contemporary Mathematics, Vol. 329, pp. 263-270, 2003.

Příloha B:

Soubor matice.c

```
        /*****************************************************************
        ************          FUNKCE PRO OPERACE S MATICEMI
****************
        ************  Copyright (c) 1999 by Josef Novak    ***************
        ******************          All Rights Reserved.  ******************
        *****************************************************************/

#include "big-head.h"

#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>


 /*********************************************************************
        *       nalezeni matice pro vypocet alg. doplnku prvku o indexech i, j    *
        *       dim - dimenze vstupni matice                                      *
        *       mat1   - vstupni matice                                           *
        *       mat2 - matice pro vypocet alg. dopl                               *
        *********************************************************************
*/
void algdopl(double **mat1, double **mat2, int dim, int i, int j)
{
        int k, l, m=0, n=0;
        for (k=0; k<i; k++) {
                for (l=0; l<j; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                for (l=j+1; l<dim; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                n=0;
                m++;
        }
        for (k=i+1; k<dim; k++) {
                for (l=0; l<j; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                for (l=j+1; l<dim; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
```

```
                n=0;
                m++;
        }
}

void algdopl2(double ****mat1, double ****mat2, int dim, int i, int j)
{
        int k, l, m=0, n=0;
        for (k=0; k<i; k++) {
                for (l=0; l<j; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                for (l=j+1; l<dim; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                n=0;
                m++;
        }
        for (k=i+1; k<dim; k++) {
                for (l=0; l<j; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                for (l=j+1; l<dim; l++) {
                        mat2[m][n]=mat1[k][l];
                        n++;
                }
                n=0;
                m++;
        }
}

/*****************************************************************************
        *       nasobeni matic o dimenzich mxn a nxk                        *
        *       m, n, k - pocet radku a sloupcu prvni mat. pocet sloupcu druhe mat*
        *       mat1, mat2 - vstupni matice                                 *
        *       mat3 - vystupni matice                                      *
        *****************************************************************************
*/
void nasobmat(double **mat1, double **mat2, double **mat3, int m, int n, int p)
{
        int i, j, k;

        for (j=0; j<p; j++) {
                for (i=0; i<m; i++) {
                        mat3[i][j]=0;
                        for (k=0; k<n; k++) {
                                mat3[i][j]+=(mat1[i][k]*mat2[k][j]);
```

```
                        }
                }
        }
}

/*****************************************************************
        *       transpozice matice dimenze ixj                  *
        *       i, j, -  pocet radku a sloupcu matice           *
        *       vraci matici dimenze jxi                        *
        *                                                       *
        *****************************************************************
*/
void transpmat(double **mat, double **mattr ,int i, int j)
{
        int k, l;

        for (l=0; l<j; l++)
                for (k=0; k<i; k++)
                        mattr[l][k]=mat[k][l];
}
                                /* determinant z matice 4x4 */
double det4(double **mat)
{
        double det=0;
        det+=(mat[0][0]*(mat[1][1]*mat[2][2]*mat[3][3]+mat[1][2]*mat[2][3]*mat[3][1]+ma
t[1][3]*mat[2][1]*mat[3][2]-mat[1][3]*mat[2][2]*mat[3][1]-mat[2][3]*mat[3][2]*mat[1][1]-
mat[3][3]*mat[1][2]*mat[2][1]));
        det+=(-
mat[0][1]*(mat[1][0]*mat[2][2]*mat[3][3]+mat[1][2]*mat[2][3]*mat[3][0]+mat[1][3]*mat[2]
[0]*mat[3][2]-mat[1][3]*mat[2][2]*mat[3][0]-mat[2][3]*mat[3][2]*mat[1][0]-
mat[3][3]*mat[1][2]*mat[2][0]));
        det+=(mat[0][2]*(mat[1][0]*mat[2][1]*mat[3][3]+mat[1][1]*mat[2][3]*mat[3][0]+ma
t[1][3]*mat[2][0]*mat[3][1]-mat[1][3]*mat[2][1]*mat[3][0]-mat[2][3]*mat[3][1]*mat[1][0]-
mat[3][3]*mat[1][1]*mat[2][0]));
        det+=(-
mat[0][3]*(mat[1][0]*mat[2][1]*mat[3][2]+mat[1][1]*mat[2][2]*mat[3][0]+mat[1][2]*mat[2]
[0]*mat[3][1]-mat[1][2]*mat[2][1]*mat[3][0]-mat[2][2]*mat[3][1]*mat[1][0]-
mat[3][2]*mat[1][1]*mat[2][0]));
        return(det);
}

                                /* determinant z matice 3x3 */
double det3(double **mat)
{
        return(mat[0][0]*mat[1][1]*mat[2][2]+mat[0][1]*mat[1][2]*mat[2][0]+mat[0][2]*mat
[1][0]*mat[2][1]-mat[0][2]*mat[1][1]*mat[2][0]-mat[1][2]*mat[2][1]*mat[0][0]-
mat[2][2]*mat[0][1]*mat[1][0]);
}
                                /* vytvori inverzni matici 4x4 */
int invmat4(double **mat1, double **mat2)
```

```
{
        int i, j;
        double det, **mat3, determ1, determ2;

        if ((mat3=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++) {
                if ((mat3[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        }
        determ1=det4(mat1);
        for (i=0; i<4; i++) {
                for (j=0; j<4; j++) {
                        algdopl(mat1, mat3, 4, j, i);
                        determ2=pow(-1, i+j)*det3(mat3);
                        mat2[i][j]=(1/determ1)*determ2;
                }
        }
return(1);
}
/* Primy chod Gaussovy eliminace
        vstup - vstupni matice
        vystup - vystupni matice
        dim - dimenze matice
        */
int gausspc(double **input, int dim, double **output)
{
        int i, j, k, l;
        double x, **help;
        double prvek;

        if ((help=(double **)malloc(dim*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<dim; j++) {
                if ((help[j]=(double *)malloc(dim*sizeof(double)))==NULL)
                        return(0);
        }
        k=1;        //testovani nenulovosti pivota
        while (input[0][0]==0 && k<dim)  {
                for (i=0; i<dim; i++)  {
                        help[0][i]=input[0][i];
                        input[0][i]=input[k][i];
                        input[k][i]=help[0][i];
                }
                k++;
        }
        k=1;
        while (input[0][0]==0 && k<dim)  {
                for (i=0; i<dim; i++)  {
                        help[0][i]=input[i][0];
```

```
                        input[i][0]=input[i][k];
                        input[i][k]=help[0][i];
                }
                k++;
        }       //v pripade nulove matice se vraci z funkce tez nulova matice
        if (input[0][0]==0)     {
                for (i=0; i<dim; i++)
                        for (j=0; j<dim; j++)
                                output[i][j]=input[i][j];
                        return(1);
        }

        for (i=0; i<dim; i++)
                output[0][i]=input[0][i];
        for (i=0; i<dim; i++)
                for (j=0; j<dim; j++)
                        help[i][j]=input[i][j];

        for (k=0; k<dim-1; k++){
//todle je doplneny

        l=k+1;          //testovani nenulovosti pivota
        while (help[k][k]==0 && l<dim)      {
                for (i=0; i<dim; i++)  {
                        prvek = help[k][i];
                        output[k][i] = help[k][i] = help[l][i];
                        output[l][i] = help[l][i] = prvek;
//                      help[0][i]=input[0][i];
//                      input[0][i]=input[k][i];
//                      input[k][i]=help[0][i];
                }
                l++;
        }
//az potud


                for (j=1+k; j<dim; j++) {
                        if ( help[j][k] != 0.00) //tedle if je tam taky dosmazenej
                        {
                                x=(help[j][k]/help[k][k]);      // help[j][k]/help[k][k]
                                for (i=k; i<dim; i++) {
                                        output[j][i]=(help[j][i]-(help[k][i]*x));  // *x
                                }
                        }
                        else
                        {
                                for (i=0; i<dim; i++) {
                                        output[j][i]=help[j][i];  // *x
                                }
                        }
```

```
                }
                for (i=0; i<dim; i++)
                        for (j=0; j<dim; j++)
                                help[i][j]=output[i][j];
        }
        return(1);
}
/* dopredny chod Gaussovy eliminace pro matici dim x dim+1 tj.
                s pravou stranou */
int gausspcp(double **input, int dim, double **output)
{
        int i, j, k;
        double x, **help;

        if ((help=(double **)malloc((dim+1)*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<(dim+1); j++) {
                if ((help[j]=(double *)malloc((dim+1)*sizeof(double)))==NULL)
                        return(0);
        }

        for (i=0; i<dim+1; i++)
                output[0][i]=input[0][i];
        for (i=0; i<dim; i++)
                for (j=0; j<dim+1; j++)
                        help[i][j]=input[i][j];

        for (k=0; k<dim-1; k++){
                for (j=1+k; j<dim; j++) {
                        x=(help[j][k]/help[k][k]);
                        for (i=k; i<dim+1; i++) {
                                output[j][i]=(help[j][i]-(help[k][i]*x));
                        }
                }
                for (i=0; i<dim; i++)
                        for (j=0; j<dim+1; j++)
                                help[i][j]=output[i][j];
        }
        return(1);
}

/* Vypocet determinantu z matice **mat o dimenzi dim*/
double determ(double **mat, int dim)
{
        double **m1, vysl=1;
        int i, j;

        if ((m1=(double **)malloc(dim*sizeof(double *)))==NULL)
                return(0.0);
        for (j=0; j<dim; j++) {
```

```
                if ((m1[j]=(double *)malloc(dim*sizeof(double)))==NULL)
                        return(0.0);
        }

        gausspc(mat, dim, m1);
        for (i=0; i<dim; i++)
                vysl*=m1[i][i];
        return(vysl);
}
// Zpetny chod Gaussovy eliminace
int gausszcp(double **input, int dim, double *output)
{
        double sum;
        int i, j, k, l;
        dim;
        output[dim-1]=input[dim-1][dim]/input[dim-1][dim-1];
        for (i=0; i<dim-1; i++)         {
                for (l=dim-i-1; l<dim; l++)
                        sum+=input[dim-2-i][l]*output[l];
                output[dim-i-2]=(input[dim-2-i][dim]-sum)/input[dim-2-i][dim-2-i];
                sum=0;
        }
        return(1);
}
// Vypocet inverzni matice dim x dim. matin je vstupni, matout je vystupni
int invmat(double **matin, double **matout, int dim)
{
        int i, j;
        double detc, det, **help;

        if ((help=(double **)malloc(dim*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<dim; j++) {
                if ((help[j]=(double *)malloc(dim*sizeof(double)))==NULL)
                        return(0);
        }

        detc=determ(matin, dim);
        for (i=0; i<dim; i++)
                for (j=0; j<dim; j++)  {
                        algdopl(matin, help, dim, i, j);
                        det=determ(help, dim-1);
                        matout[j][i]=(pow(-1, i+j)*(det/detc));
                }   //tady je to taky dodelany
        for (i=0; i<dim; i++)
                free(help[i]);
        free(help);
        return(1);
}
int make_vektor(double *bod1, double *bod2, double *vektor)
```

```
{
	vektor[0] = bod2[0] - bod1[0];
	vektor[1] = bod2[1] - bod1[1];

	return(1);
}

double vector_norm(double *vektor)
{

	double norm;

	norm = sqrt(vektor[0]*vektor[0] + vektor[1]*vektor[1] + vektor[2]*vektor[2]);

	return(norm);

}

double inner_vector_product(double *vektor_1, double *vektor_2)
{

	double product;

	product = (vektor_1[0]*vektor_2[0] + vektor_1[1]*vektor_2[1] +
vektor_1[2]*vektor_2[2]);

	return(product);

}

double vector_angle(double *vektor_1, double *vektor_2)
{

	double angle;

	angle = acos ( (inner_vector_product(vektor_1, vektor_2)) / (vector_norm(vektor_1) *
vector_norm(vektor_2)) );

	return(angle);
}

int make_vektor_3D(double *bod1, double *bod2, double *vektor)
{
	vektor[0] = bod2[0] - bod1[0];
	vektor[1] = bod2[1] - bod1[1];
	vektor[2] = bod2[2] - bod1[2];

	return(1);
}
```

Soubor mkp.c

```
#include "big-head.h"

#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>


void maticeB(double **matB, double **matfi, int i) // i je index uzlu v simplexu, matfi je
{
                                                    // inv. mat. souradnic uzlu simplexu
        int j, k;

        for (j=0; j<6; j++)
                for (k=0; k<3; k++)
                        matB[j][k]=0;
        i--;
        matB[0][0]=matfi[1][i];
        matB[4][0]=0.5*matfi[3][i];
        matB[5][0]=0.5*matfi[2][i];
        matB[1][1]=matfi[2][i];
        matB[3][1]=0.5*matfi[3][i];
        matB[5][1]=0.5*matfi[1][i];
        matB[2][2]=matfi[3][i];
        matB[3][2]=0.5*matfi[2][i];
        matB[4][2]=0.5*matfi[1][i];
}


int BtxCxB(double **matB1, double **matC, double **matB, double **matv)
{
        double **matBt, **matm;
        int j;

        if ((matBt=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++) {
                if ((matBt[j]=(double *)malloc(6*sizeof(double)))==NULL)
                        return(0);
        }
        if ((matm=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++) {
                if ((matm[j]=(double *)malloc(6*sizeof(double)))==NULL)
                        return(0);
        }
```

```
        transpmat(matB1, matBt, 6, 3);
        nasobmat(matBt, matC, matm, 3, 6, 6);
        nasobmat(matm, matB, matv, 3, 6, 3);
        for (j=0; j<3; j++) {
                free((void *)matBt[j]);
                free((void *)matm[j]);
        }
        free((void **)matBt);
        free((void **)matm);
}

int GrxexGrt(double **matB1, double **matE, double **matB, double **matv)
{
        double **matBt, **matm;
        int j;

        if ((matBt=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<1; j++) {
                if ((matBt[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        }
        if ((matm=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<1; j++) {
                if ((matm[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        }

        transpmat(matB1, matBt, 3, 1);
        nasobmat(matBt, matE, matm, 1, 3, 3);
        nasobmat(matm, matB, matv, 1, 3, 1);
        for (j=0; j<1; j++) {
                free((void *)matBt[j]);
                free((void *)matm[j]);
        }
        free((void **)matBt);
        free((void **)matm);
}

/* nacteni koefic. materialu pro dany element
                jmenostm - soubor *.mst
                jmenomat - soubor s mater. konstantami
                elem - cislo elementu, pro ktery nacitame
                **elasto - matice, do ktere budou nacitany elast. koef.  matice 6x6
                **piezo -
                                                        piezo. koef.  matice 3x6
                **elektro -
                                                elektr. koef. matice 3x3 */
```

```
int readstm(char jmenostm[20], char jmenomat[20], int elem, double **elasto, double
**piezo, double **elektro)
{
	int cislo, mater, id, i, j, d;
	char c;
	FILE *frstm, *frmat;


	mater = 0;

//	if ((frstm=fopen(jmenostm, "r"))==NULL)
//		return(0);
	if ((frmat=fopen(jmenomat, "r"))==NULL)
		return(0);
//	fscanf(frstm, "%d", &cislo);
//	while (cislo!=elem) {	//nalezeni radku s pozad. elementem
//		while ((c=getc(frstm))!='\n') ;
//		fscanf(frstm, "%d", &cislo);
//	}
//	fscanf(frstm, "%d", &mater);
//	fscanf(frstm, "%d", &mater); //zjisteni cisla materialu
//	fclose(frstm);
	fscanf(frmat, "%d", &id); //nalezeni mista v *.mtr, kde zacinaji koef.
//	while(id!=mater) {	//pozad. materialu
//		while ((d=getc(frmat))!='\n') ;
//		fscanf(frmat, "%d", &id);
//	}
//	while ((d=getc(frmat))!='\n') ;
	for (i=0; i<6; i++)
		for (j=0; j<6; j++)
			fscanf(frmat,"%lf", &(elasto[i][j]));
	for (i=0; i<3; i++)
		for (j=0; j<6; j++)
			fscanf(frmat,"%lf", &(piezo[i][j]));
	for (i=0; i<3; i++)
		for (j=0; j<3; j++)
			fscanf(frmat,"%lf", &(elektro[i][j]));

	return(1);
}
/* vypocet lokalni matice elasticke casti
	uzel1,..., uzel4 - souradnice uzlu simplexu
	elem - cislo elementu, jehoz soucasti simplex je
	elast - matice elastickych koeficientu
	objem - objem simplexu
	vystup - prom. typu lokal:

		typedef struct {
```

```
                        int u1, u2, u3, u4;

                        double ****lok;

                } lokal;
                alokace pameti pro prom. typu lokal:
                                                                lokal ml;
                                                                ml.lok=(double
****)malloc(4*sizeof(double ***));
                                                                for (i=0; i<4; i++)

        ml.lok[i]=(double ***)malloc(4*sizeof(double **));
                                                                for (i=0; i<4; i++)
                                                                        for (j=0; j<4;
j++)

        ml.lok[i][j]=(double **)malloc(3*sizeof(double *));
                                                                for (i=0; i<4; i++)
                                                                        for (j=0; j<4;
j++)
                                                                                for
(k=0; k<3; k++)

        ml.lok[i][j][k]=(double *)malloc(3*sizeof(double));
*/

int LokalMate(double *uzel1, double *uzel2, double *uzel3, double *uzel4, int elem, double
**elast,  double objem, lokal vystup)
{
        int i, j, k, l;
        double **uzly, **matf, **mb1, **mb2, **mb3, **mb4;
 double volume;

        if ((uzly=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((uzly[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);
        if ((matf=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((matf[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);
        if ((mb1=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++)
                if ((mb1[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
```

```
        if ((mb2=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++)
                if ((mb2[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);

        if ((mb3=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((mb3[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        }

        if ((mb4=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((mb4[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        }

        for (j=0; j<4; j++)
                uzly[j][0]=1;
        for (j=1; j<4; j++)
                uzly[0][j]=uzel1[j-1];
        for (j=1; j<4; j++)
                uzly[1][j]=uzel2[j-1];
        for (j=1; j<4; j++)
                uzly[2][j]=uzel3[j-1];
        for (j=1; j<4; j++)
                uzly[3][j]=uzel4[j-1];

volume = det4(uzly);
volume /= 6;
volume = fabsl(volume);
        invmat4(uzly, matf);
        maticeB(mb1, matf, 1);
        maticeB(mb2, matf, 2);
        maticeB(mb3, matf, 3);
        maticeB(mb4, matf, 4);
        BtxCxB(mb1, elast, mb1, vystup.lok[0][0]);
        BtxCxB(mb1, elast, mb2, vystup.lok[0][1]);
        BtxCxB(mb1, elast, mb3, vystup.lok[0][2]);
        BtxCxB(mb1, elast, mb4, vystup.lok[0][3]);
        BtxCxB(mb2, elast, mb1, vystup.lok[1][0]);
        BtxCxB(mb2, elast, mb2, vystup.lok[1][1]);
        BtxCxB(mb2, elast, mb3, vystup.lok[1][2]);
        BtxCxB(mb2, elast, mb4, vystup.lok[1][3]);
        BtxCxB(mb3, elast, mb1, vystup.lok[2][0]);
        BtxCxB(mb3, elast, mb2, vystup.lok[2][1]);
```

```
        BtxCxB(mb3, elast, mb3, vystup.lok[2][2]);
        BtxCxB(mb3, elast, mb4, vystup.lok[2][3]);
        BtxCxB(mb4, elast, mb1, vystup.lok[3][0]);
        BtxCxB(mb4, elast, mb2, vystup.lok[3][1]);
        BtxCxB(mb4, elast, mb3, vystup.lok[3][2]);
        BtxCxB(mb4, elast, mb4, vystup.lok[3][3]);

        for (i=0; i<4; i++)
                for (j=0; j<4; j++)
                        for (k=0; k<3; k++)
                                for (l=0; l<3; l++)
                                        vystup.lok[i][j][k][l]*=volume;      //objem;
        for (i=0; i<4; i++)
                free(uzly[i]);
        free(uzly);
}

 int LokalEle(double *uzel1, double *uzel2, double *uzel3, double *uzel4, int elem, double
**elektro,  double objem, lokale vystup)
{
        int i, j;
        double **uzly, **matf, **alfa1, **alfa2, **alfa3, **alfa4;
        double volume;

        if ((uzly=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((uzly[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);
        if ((matf=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((matf[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);
        if ((alfa1=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++)
                if ((alfa1[j]=(double *)malloc(1*sizeof(double)))==NULL)
                        return(0);
        if ((alfa2=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++)
                if ((alfa2[j]=(double *)malloc(1*sizeof(double)))==NULL)
                        return(0);
        if ((alfa3=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++)
                if ((alfa3[j]=(double *)malloc(1*sizeof(double)))==NULL)
                        return(0);
        if ((alfa4=(double **)malloc(3*sizeof(double *)))==NULL)
```

```
            return(0);
        for (j=0; j<3; j++)
            if ((alfa4[j]=(double *)malloc(1*sizeof(double)))==NULL)
                return(0);

        for (j=0; j<4; j++)
            uzly[j][0]=1;
        for (j=1; j<4; j++)
            uzly[0][j]=uzel1[j-1];
        for (j=1; j<4; j++)
            uzly[1][j]=uzel2[j-1];
        for (j=1; j<4; j++)
            uzly[2][j]=uzel3[j-1];
        for (j=1; j<4; j++)
            uzly[3][j]=uzel4[j-1];

        invmat4(uzly, matf);

        for (i=0; i<3; i++)
            alfa1[i][0]=matf[i+1][0];
        for (i=0; i<3; i++)
            alfa2[i][0]=matf[i+1][1];
        for (i=0; i<3; i++)
            alfa3[i][0]=matf[i+1][2];
        for (i=0; i<3; i++)
            alfa4[i][0]=matf[i+1][3];

        GrxexGrt(alfa1, elektro, alfa1, vystup.lok[0][0]);
        GrxexGrt(alfa1, elektro, alfa2, vystup.lok[0][1]);
        GrxexGrt(alfa1, elektro, alfa3, vystup.lok[0][2]);
        GrxexGrt(alfa1, elektro, alfa4, vystup.lok[0][3]);
        GrxexGrt(alfa2, elektro, alfa1, vystup.lok[1][0]);
        GrxexGrt(alfa2, elektro, alfa2, vystup.lok[1][1]);
        GrxexGrt(alfa2, elektro, alfa3, vystup.lok[1][2]);
        GrxexGrt(alfa2, elektro, alfa4, vystup.lok[1][3]);
        GrxexGrt(alfa3, elektro, alfa1, vystup.lok[2][0]);
        GrxexGrt(alfa3, elektro, alfa2, vystup.lok[2][1]);
        GrxexGrt(alfa3, elektro, alfa3, vystup.lok[2][2]);
        GrxexGrt(alfa3, elektro, alfa4, vystup.lok[2][3]);
        GrxexGrt(alfa4, elektro, alfa1, vystup.lok[3][0]);
        GrxexGrt(alfa4, elektro, alfa2, vystup.lok[3][1]);
        GrxexGrt(alfa4, elektro, alfa3, vystup.lok[3][2]);
        GrxexGrt(alfa4, elektro, alfa4, vystup.lok[3][3]);

volume = det4(uzly);
volume /= 6;
volume = fabsl(volume);

        for (i=0; i<4; i++)
            for (j=0; j<4; j++)
```

```
                        vystup.lok[i][j][0][0]*=volume;
}

/* Vypocet gradientu bazove funkce
                uzel - cislo uzlu pro vypocet baz. fce
                matfi - matice bazovych koeficientu
                grad - matice 1x3 gradientu baz. fce */
void gradw(double **matfi, int uzel, double **grad)
{
        uzel--;
        grad[0][0]=matfi[1][uzel];
        grad[0][1]=matfi[2][uzel];
        grad[0][2]=matfi[3][uzel];
}

int GrxDxB(double **grad, double **matD, double **matB, double **matv)
{
        double **meziv;

        if ((meziv=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
        if ((meziv[0]=(double *)malloc(6*sizeof(double)))==NULL)
                        return(0);

        nasobmat(grad, matD, meziv, 1, 3, 6);
        nasobmat(meziv, matB, matv, 1, 6, 3);
}
/* vypocet lokalni matice piezoelektricke casti
        uzel1,.., uzel4 - souradnice uzlu simplexu
        elem - cislo elementu, jehoz soucasti simplex je
        piezo - matice piezoelektrickych koeficientu
        objem - objem simplexu
        vystup - prom. typu lokalp:

                typedef struct {

                        int u1, u2, u3, u4;

                        double ****lokp;

                } lokalp;
                alokace pameti pro prom. typu lokalp:
                                                        lokalp ml;
                                                        ml.lokp=(double
****)malloc(4*sizeof(double ***));
                                                        for (i=0; i<4; i++)

        ml.lokp[i]=(double ***)malloc(4*sizeof(double **));
                                                        for (i=0; i<4; i++)
```

```
                                                                    for (j=0; j<4;
j++)

        ml.lokp[i][j]=(double **)malloc(1*sizeof(double *));
                                                                for (i=0; i<4; i++)
                                                                    for (j=0; j<4;
j++)

        ml.lokp[i][j][0]=(double *)malloc(3*sizeof(double));
*/

int LokalMatp(double *uzel1, double *uzel2, double *uzel3, double *uzel4, int elem, double
**piezo,  double objem, lokalp vystup)
{
        int i, j, k, l;
        double **uzly, **matf, **mb1, **mb2, **mb3, **mb4, **mg1, **mg2, **mg3,
**mg4;
        double volume;

        if ((uzly=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((uzly[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);
        if ((matf=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((matf[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);
        if ((mb1=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++)
                if ((mb1[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        if ((mb2=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++)
                if ((mb2[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        if ((mb3=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((mb3[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        }
        if ((mb4=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((mb4[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
```

```
}

if ((mg1=(double **)malloc(1*sizeof(double *)))==NULL)
        return(0);
if ((mg1[0]=(double *)malloc(3*sizeof(double)))==NULL)
                return(0);
if ((mg2=(double **)malloc(1*sizeof(double *)))==NULL)
        return(0);
if ((mg2[0]=(double *)malloc(3*sizeof(double)))==NULL)
                return(0);
if ((mg3=(double **)malloc(1*sizeof(double *)))==NULL)
        return(0);
if ((mg3[0]=(double *)malloc(3*sizeof(double)))==NULL)
                return(0);
if ((mg4=(double **)malloc(1*sizeof(double *)))==NULL)
        return(0);
if ((mg4[0]=(double *)malloc(3*sizeof(double)))==NULL)
                return(0);


for (j=0; j<4; j++)
        uzly[j][0]=1;
for (j=1; j<4; j++)
        uzly[0][j]=uzel1[j-1];
for (j=1; j<4; j++)
        uzly[1][j]=uzel2[j-1];
for (j=1; j<4; j++)
        uzly[2][j]=uzel3[j-1];
for (j=1; j<4; j++)
        uzly[3][j]=uzel4[j-1];
invmat4(uzly, matf);
gradw(matf, 1, mg1);
gradw(matf, 2, mg2);
gradw(matf, 3, mg3);
gradw(matf, 4, mg4);
maticeB(mb1, matf, 1);
maticeB(mb2, matf, 2);
maticeB(mb3, matf, 3);
maticeB(mb4, matf, 4);
GrxDxB(mg1, piezo, mb1, vystup.lokp[0][0]);
GrxDxB(mg1, piezo, mb2, vystup.lokp[0][1]);
GrxDxB(mg1, piezo, mb3, vystup.lokp[0][2]);
GrxDxB(mg1, piezo, mb4, vystup.lokp[0][3]);
GrxDxB(mg2, piezo, mb1, vystup.lokp[1][0]);
GrxDxB(mg2, piezo, mb2, vystup.lokp[1][1]);
GrxDxB(mg2, piezo, mb3, vystup.lokp[1][2]);
GrxDxB(mg2, piezo, mb4, vystup.lokp[1][3]);
GrxDxB(mg3, piezo, mb1, vystup.lokp[2][0]);
GrxDxB(mg3, piezo, mb2, vystup.lokp[2][1]);
GrxDxB(mg3, piezo, mb3, vystup.lokp[2][2]);
```

```
        GrxDxB(mg3, piezo, mb4, vystup.lokp[2][3]);
        GrxDxB(mg4, piezo, mb1, vystup.lokp[3][0]);
        GrxDxB(mg4, piezo, mb2, vystup.lokp[3][1]);
        GrxDxB(mg4, piezo, mb3, vystup.lokp[3][2]);
        GrxDxB(mg4, piezo, mb4, vystup.lokp[3][3]);

 volume = det4(uzly);
 volume /= 6;
 volume = fabsl(volume);

        for (i=0; i<4; i++)
                for (j=0; j<4; j++)
                        for (l=0; l<3; l++)
                                vystup.lokp[i][j][0][l]*=volume;
}
/* nalezeni matice piezoelektrickych koeficientu
        double **matelm - matice elastickych modulu
        double **matpem - matice piezoelektrickych modulu
        double **matpek - matice piezoelektrickych koeficientu
                !!      bez uvolneni pameti vykricnik  !!
*/
int piezokoef(double **matelm, double **matpem, double **matpek)
{
        double **determin, **doplnek, **matelk, det, detdopl;
        int i,j;

        if ((determin=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((determin[j]=(double *)malloc(6*sizeof(double)))==NULL)
                        return(0);
        }
        if ((doplnek=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((doplnek[j]=(double *)malloc(6*sizeof(double)))==NULL)
                        return(0);
        }
        if ((matelk=(double **)malloc(6*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<6; j++) {
                if ((matelk[j]=(double *)malloc(6*sizeof(double)))==NULL)
                        return(0);
        }

        for (i=0; i<6; i++)
                for (j=0; j<6; j++)
                        determin[i][j]=matelm[i][j];
        for (i=0; i<6; i++)
                for (j=3; j<6; j++)
```

```
                    determin[i][j]=2*determin[i][j];

        det=determ(determin, 6);

        // vytvoreni matice elastickych koeficientu
        for (i=0; i<6; i++) {
                for (j=0; j<6; j++) {
                        algdopl(determin, doplnek, 6, i, j);
                        detdopl=determ(doplnek, 5);
                        if (j>2) {
                                matelk[i][j]=((detdopl/det)/2); }
                        else {
                                matelk[i][j]=(detdopl/det);          }
                }
        }
//  naplneni matice piezoel. koef.
        for (i=0; i<3; i++)
                for (j=0; j<6; j++)    //nulovani matice
                        matpek[i][j]=0;

        for (i=0; i<3; i++) {
                for (j=0; j<3; j++)
                        matpek[i][i]+=(matelk[i][j]*matpem[i][j]);
                for (j=3; j<6; j++)
                        matpek[i][i]+=(2*matelk[i][j]*matpem[i][j]);
        }
        // d122 - d112 pvni radek vyjma d111
        for (j=1; j<6; j++) {
                for (i=0; i<3; i++)
                        matpek[0][j]+=(matpem[0][i]*matelk[i][j]);
                for (i=3; i<6; i++)
                        matpek[0][j]+=2*(matpem[0][i]*matelk[i][j]);
        }
        // d233 - d212 druhy radek vyjma d211 d222
        for (j=2; j<6; j++) {
                for (i=0; i<3; i++)
                        matpek[1][j]+=(matpem[1][i]*matelk[i][j]);
                for (i=3; i<6; i++)
                        matpek[1][j]+=2*(matpem[1][i]*matelk[i][j]);
        }
        // d323 - d312 treti radek vyjma d311 d322 d333
        for (j=3; j<6; j++) {
                for (i=0; i<3; i++)
                        matpek[2][j]+=(matpem[2][i]*matelk[i][j]);
                for (i=3; i<6; i++)
                        matpek[2][j]+=2*(matpem[2][i]*matelk[i][j]);
        }
        // d311, d322
        for (j=0; j<2; j++) {
                for (i=0; i<3; i++)
```

```
                matpek[2][j]+=(matpem[2][i]*matelk[i][j]);
        for (i=3; i<6; i++)
                matpek[2][j]+=2*(matpem[2][i]*matelk[i][j]);
    }
    // d211
    for (i=0; i<3; i++)
        matpek[1][0]+=(matpem[1][i]*matelk[i][0]);
    for (i=3; i<6; i++)
        matpek[1][0]+=2*(matpem[1][i]*matelk[i][0]);

    return(1);
}

double GrxexGrt2D(double **matB1, double **matE, double **matB)
{
    double **matBt, **matm;
    double **help;
    int j;

    if ((matBt=(double **)malloc(1*sizeof(double *)))==NULL)
    return(0.0);
    for (j=0; j<1; j++) {
        if ((matBt[j]=(double *)malloc(2*sizeof(double)))==NULL)
            return(0.0);
    }
    if ((matm=(double **)malloc(1*sizeof(double *)))==NULL)
        return(0.0);
    for (j=0; j<1; j++) {
        if ((matm[j]=(double *)malloc(2*sizeof(double)))==NULL)
            return(0.0);
    }
    if ((help=(double **)malloc(1*sizeof(double *)))==NULL)
        return(0.0);
    for (j=0; j<1; j++) {
        if ((help[j]=(double *)malloc(1*sizeof(double)))==NULL)
            return(0.0);
    }

    transpmat(matB1, matBt, 2, 1);
    nasobmat(matBt, matE, matm, 1, 2, 2);
    nasobmat(matm, matB, help, 1, 2, 1);
    //printf(" %lf \n", matv);
    for (j=0; j<1; j++) {
        free((void *)matBt[j]);
        free((void *)matm[j]);
    }
    free((void **)matBt);
    free((void **)matm);
    return(help[0][0]);
}
```

```
int LokalEle2D(double *uzel1, double *uzel2, double *uzel3, int elem, double **elektro,
double objem, lokale2D vystup)
{
        int i, j;
        double **uzly, **matf, **alfa1, **alfa2, **alfa3, v;
        double surface;

        if ((uzly=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++)
                if ((uzly[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);
        if ((matf=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++)
                if ((matf[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);

        if ((alfa1=(double **)malloc(2*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<2; j++)
                if ((alfa1[j]=(double *)malloc(1*sizeof(double)))==NULL)
                        return(0);

        if ((alfa2=(double **)malloc(2*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<2; j++)
                if ((alfa2[j]=(double *)malloc(1*sizeof(double)))==NULL)
                        return(0);

        if ((alfa3=(double **)malloc(2*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<2; j++)
                if ((alfa3[j]=(double *)malloc(1*sizeof(double)))==NULL)
                        return(0);

        for (j=0; j<3; j++)
                uzly[j][0]=1;
        for (j=1; j<3; j++)
                uzly[0][j]=uzel1[j-1];
        for (j=1; j<3; j++)
                uzly[1][j]=uzel2[j-1];
        for (j=1; j<3; j++)
                uzly[2][j]=uzel3[j-1];

        surface = det3(uzly);
        surface/= 6;
        surface = fabsl(surface);
```

```
        invmat(uzly, matf, 3);

        for (i=0; i<2; i++)
                alfa1[i][0]=matf[i+1][0];
        for (i=0; i<2; i++)
                alfa2[i][0]=matf[i+1][1];
        for (i=0; i<2; i++)
                alfa3[i][0]=matf[i+1][2];

        vystup.lok[0][0]=GrxexGrt2D(alfa1, elektro, alfa1);
        vystup.lok[0][1]=GrxexGrt2D(alfa1, elektro, alfa2);
        vystup.lok[0][2]=GrxexGrt2D(alfa1, elektro, alfa3);
        vystup.lok[1][0]=GrxexGrt2D(alfa2, elektro, alfa1);
        vystup.lok[1][1]=GrxexGrt2D(alfa2, elektro, alfa2);
        vystup.lok[1][2]=GrxexGrt2D(alfa2, elektro, alfa3);
        vystup.lok[2][0]=GrxexGrt2D(alfa3, elektro, alfa1);
        vystup.lok[2][1]=GrxexGrt2D(alfa3, elektro, alfa2);
        vystup.lok[2][2]=GrxexGrt2D(alfa3, elektro, alfa3);


        for (i=0; i<3; i++)
                for (j=0; j<3; j++)
                        vystup.lok[i][j]*=surface;
}
//---------------------------------------------------------------------------
//  vypocet soucinu bazovych funkci
//  **matfi - matice koeficientu baz. fci.
//  i, j - cisla vrcholu v simplexu
//  **wxw - vystup, pole 10 doublu, jsou to koeficienty u jednotlivych
//          clenu nasobeni v poradi
//        x2 y2 z2 xy xz yz x y z konst.
//---------------------------------------------------------------------------

void wixwj(double **matfi, int i, int j, double *wxw)
{
  wxw[0] = matfi[1][i-1] * matfi[1][j-1];      // a1i * a1j
  wxw[1] = matfi[2][i-1] * matfi[2][j-1];      // a2i * a2j
  wxw[2] = matfi[3][i-1] * matfi[3][j-1];      // a3i * a3j

  wxw[3] = (matfi[1][i-1] * matfi[2][j-1]) + (matfi[2][i-1] * matfi[1][j-1]);   // a1i*a2j +
a2i*a1j
  wxw[4] = (matfi[1][i-1] * matfi[3][j-1]) + (matfi[3][i-1] * matfi[1][j-1]);   // a1i*a3j +
a3i*a1j
  wxw[5] = (matfi[2][i-1] * matfi[3][j-1]) + (matfi[3][i-1] * matfi[2][j-1]);   // a2i*a3j +
a3i*a2j

  wxw[6] = (matfi[0][i-1] * matfi[1][j-1]) + (matfi[1][i-1] * matfi[0][j-1]);   // a0i*a1j +
a1i*a0j
  wxw[7] = (matfi[0][i-1] * matfi[2][j-1]) + (matfi[2][i-1] * matfi[0][j-1]);   // a0i*a2j +
a2i*a0j
```

```
  wxw[8] = (matfi[0][i-1] * matfi[3][j-1]) + (matfi[3][i-1] * matfi[0][j-1]);   // a0i*a3j +
a3i*a0j

  wxw[9] = matfi[0][i-1] * matfi[0][j-1];      // a0i * a0j
}

//---------------------------------------------------------------------------
//              Vypocet integralu ze soucinu wi * wj
//
//---------------------------------------------------------------------------
double integr_wixwj(double **souradnice, int i, int j)
{
  double integral = 0.00;
  double **matkoef;
  double *soucin;
  double a[3], b[3], c[3], d[3], alfa, beta, volume;
  double pom;

  int k, l;

  if ((matkoef=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0.00);
        for (k=0; k<4; k++)
                if ((matkoef[k]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0.00);

  soucin = (double *)malloc(10*sizeof(double));

  invmat4(souradnice, matkoef);  // ziskani koeficientu bazovych fci

  wixwj(matkoef, i, j, soucin);

  for (k=0; k<10; k++)
    pom = soucin[k];

  alfa = 0.5854102;
  beta = 0.13819660;

  volume = det4(souradnice);
  volume /= 6;
  volume = fabsl(volume);

  a[0] = alfa*souradnice[0][1] + beta*souradnice[1][1] + beta*souradnice[2][1] +
beta*souradnice[3][1];
  a[1] = alfa*souradnice[0][2] + beta*souradnice[1][2] + beta*souradnice[2][2] +
beta*souradnice[3][2];
  a[2] = alfa*souradnice[0][3] + beta*souradnice[1][3] + beta*souradnice[2][3] +
beta*souradnice[3][3];
```

```
  b[0] = beta*souradnice[0][1] + alfa*souradnice[1][1] + beta*souradnice[2][1] +
beta*souradnice[3][1];
  b[1] = beta*souradnice[0][2] + alfa*souradnice[1][2] + beta*souradnice[2][2] +
beta*souradnice[3][2];
  b[2] = beta*souradnice[0][3] + alfa*souradnice[1][3] + beta*souradnice[2][3] +
beta*souradnice[3][3];

  c[0] = beta*souradnice[0][1] + beta*souradnice[1][1] + alfa*souradnice[2][1] +
beta*souradnice[3][1];
  c[1] = beta*souradnice[0][2] + beta*souradnice[1][2] + alfa*souradnice[2][2] +
beta*souradnice[3][2];
  c[2] = beta*souradnice[0][3] + beta*souradnice[1][3] + alfa*souradnice[2][3] +
beta*souradnice[3][3];

  d[0] = beta*souradnice[0][1] + beta*souradnice[1][1] + beta*souradnice[2][1] +
alfa*souradnice[3][1];
  d[1] = beta*souradnice[0][2] + beta*souradnice[1][2] + beta*souradnice[2][2] +
alfa*souradnice[3][2];
  d[2] = beta*souradnice[0][3] + beta*souradnice[1][3] + beta*souradnice[2][3] +
alfa*souradnice[3][3];

  integral+= soucin[0]*a[0]*a[0] + soucin[1]*a[1]*a[1] + soucin[2]*a[2]*a[2];
  integral+= soucin[3]*a[0]*a[1] + soucin[4]*a[0]*a[2] + soucin[5]*a[1]*a[2];
  integral+= soucin[6]*a[0] + soucin[7]*a[1] + soucin[8]*a[2] + soucin[9];

  integral+= soucin[0]*b[0]*b[0] + soucin[1]*b[1]*b[1] + soucin[2]*b[2]*b[2];
  integral+= soucin[3]*b[0]*b[1] + soucin[4]*b[0]*b[2] + soucin[5]*b[1]*b[2];
  integral+= soucin[6]*b[0] + soucin[7]*b[1] + soucin[8]*b[2] + soucin[9];

  integral+= soucin[0]*c[0]*c[0] + soucin[1]*c[1]*c[1] + soucin[2]*c[2]*c[2];
  integral+= soucin[3]*c[0]*c[1] + soucin[4]*c[0]*c[2] + soucin[5]*c[1]*c[2];
  integral+= soucin[6]*c[0] + soucin[7]*c[1] + soucin[8]*c[2] + soucin[9];

  integral+= soucin[0]*d[0]*d[0] + soucin[1]*d[1]*d[1] + soucin[2]*d[2]*d[2];
  integral+= soucin[3]*d[0]*d[1] + soucin[4]*d[0]*d[2] + soucin[5]*d[1]*d[2];
  integral+= soucin[6]*d[0] + soucin[7]*d[1] + soucin[8]*d[2] + soucin[9];

  integral *= 0.25;
  integral *= volume;

  return(integral);
}
//---------------------------------------------------------------------------

void LokalMass(double *uzel1, double *uzel2, double *uzel3, double *uzel4, double rho, int
elem, lokal vystup)
{
  double **uzly, integral;
  int k, l, j, m;
```

```
        uzly=(double **)malloc(4*sizeof(double *));
        for (j=0; j<4; j++)
         uzly[j]=(double *)malloc(4*sizeof(double));

        for (j=0; j<4; j++)
                uzly[j][0]=1;
        for (j=1; j<4; j++)
                uzly[0][j]=uzel1[j-1];
        for (j=1; j<4; j++)
                uzly[1][j]=uzel2[j-1];
        for (j=1; j<4; j++)
                uzly[2][j]=uzel3[j-1];
        for (j=1; j<4; j++)
                uzly[3][j]=uzel4[j-1];

 for ( k=0; k<4; k++)
 {
  for ( l=0; l<4; l++)
  {
   integral = integr_wixwj(uzly, k+1, l+1);
   for (m=0; m<3; m++)
    vystup.lok[k][l][m][m] = integral*rho;
  }
 }
}

void A_w2M(double *uzel1, double *uzel2, double *uzel3, double *uzel4, int elem, double
**elast, double hustota, double frekvence, lokal vystup)
{
 lokal elasticky, hmotnostni;
 int i, j, k, l;
 double volume = 0, omega;

 elasticky.lok=(double ****)malloc(4*sizeof(double ***));
        for (i=0; i<4; i++)
                elasticky.lok[i]=(double ***)malloc(4*sizeof(double **));
               for (i=0; i<4; i++)
               for (j=0; j<4; j++)
                       elasticky.lok[i][j]=(double **)malloc(3*sizeof(double *));
                       for (i=0; i<4; i++)
                       for (j=0; j<4; j++)
                          for (k=0; k<3; k++)
                                  elasticky.lok[i][j][k]=(double
*)malloc(3*sizeof(double));

 hmotnostni.lok=(double ****)malloc(4*sizeof(double ***));
        for (i=0; i<4; i++)
                hmotnostni.lok[i]=(double ***)malloc(4*sizeof(double **));
               for (i=0; i<4; i++)
               for (j=0; j<4; j++)
```

```
                        hmotnostni.lok[i][j]=(double **)malloc(3*sizeof(double *));
                        for (i=0; i<4; i++)
                        for (j=0; j<4; j++)
                           for (k=0; k<3; k++)
                                        hmotnostni.lok[i][j][k]=(double
*)malloc(3*sizeof(double));
 // se zadanym volume se stejnak nepocita
 LokalMate(uzel1, uzel2, uzel3, uzel4, 1, elast, volume, elasticky);
 LokalMass(uzel1, uzel2, uzel3, uzel4, hustota, 1, hmotnostni);

 omega = 6.283185 * frekvence;

 for (i=0; i<4; i++)
  for (j=0; j<4; j++)
   for (k=0; k<3; k++)
    elasticky.lok[i][j][k][k]-=(omega * omega * hmotnostni.lok[i][j][k][k]);

 for (i=0; i<4; i++)
  for (j=0; j<4; j++)
   for (k=0; k<3; k++)
    for (l=0; l<3; l++)
     vystup.lok[i][j][k][l] = elasticky.lok[i][j][k][l];

}

int readuzl(char jmenouzl[20], double **uzly)
{
        int cislo, i, pocuzl;
        FILE *fruzl;


        if ((fruzl=fopen(jmenouzl, "r"))==NULL)
                return(0);

        fscanf(fruzl, "%d", &pocuzl);

 for (i=0; i<pocuzl; i++)
 {
  fscanf(fruzl, "%d %lf %lf %lf", &cislo, &uzly[i][0], &uzly[i][1], &uzly[i][2]);
 }

 fclose(fruzl);
 return(pocuzl);
}

int readspl(char jmenospl[20], int **simplexy)
{
        int cislo, i, pocspl;
        FILE *frspl;
```

```
        if ((frspl=fopen(jmenospl, "r"))==NULL)
                return(0);

        fscanf(frspl, "%d", &pocspl);

 for (i=0; i<pocspl; i++)
 {
  fscanf(frspl, "%d %d %d %d %d", &cislo, &simplexy[i][0], &simplexy[i][1],
&simplexy[i][2], &simplexy[i][3]);
 }

 fclose(frspl);
 return(pocspl);
}

/*---------------------------------------------------------------------------
 * Funkce vypocte koeficienty bi, ci a determinant z matice souradnic nodu
 * vstupem jsou pole se souradnicemi nodu
 * vystupem pole [0]-[2] s koeficienty bi resp. ci a [3] s det(S)
---------------------------------------------------------------------------*/
int vypocti_bi_ci_detS_2D (double *uzel1, double *uzel2, double *uzel3, double *b, double
*c, double *detS)
{
        int i, j;
        double **uzly;

        if ((uzly=(double **)malloc(3*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<3; j++)
                if ((uzly[j]=(double *)malloc(3*sizeof(double)))==NULL)
                        return(0);

        for (j=0; j<3; j++)
                uzly[j][0]=1;
        for (j=1; j<3; j++)
                uzly[0][j]=uzel1[j-1];
        for (j=1; j<3; j++)
                uzly[1][j]=uzel2[j-1];
        for (j=1; j<3; j++)
                uzly[2][j]=uzel3[j-1];

// pocita se bi = yj - yk ; bj = yk - yi ; bk = yi - yj
//           ci = xk - xj ; cj = xi - xk ; ck = xj - xi

        b[0] = uzel2[1] - uzel3[1];
        b[1] = uzel3[1] - uzel1[1];
        b[2] = uzel1[1] - uzel2[1];

        c[0] = uzel3[0] - uzel2[0];
```

```
        c[1] = uzel1[0] - uzel3[0];
        c[2] = uzel2[0] - uzel1[0];

        *detS =  det3(uzly);

        return(1);
}

/*---------------------------------------------------------------------------
 * Vypocte teziste elementu
 *  vstupem souradnice nodu, vystupem souradnice teziste
---------------------------------------------------------------------------*/
int teziste_elementu_2D(double *uzel1, double *uzel2, double *uzel3, double *teziste)
{
        teziste[0] = (uzel1[0] + uzel2[0] + uzel3[0])/3;
        teziste[1] = (uzel1[1] + uzel2[1] + uzel3[1])/3;

        return(1);
}

/*---------------------------------------------------------------------------
 * Vypocte teziste elementu
 *  vstupem souradnice nodu, vystupem souradnice teziste
 ---------------------------------------------------------------------------*/
int teziste_elementu_3D(double *uzel1, double *uzel2, double *uzel3, double *uzel4, double
*teziste)
{
  teziste[0] = (uzel1[0] + uzel2[0] + uzel3[0] + uzel4[0])/4;
  teziste[1] = (uzel1[1] + uzel2[1] + uzel3[1] + uzel4[1])/4;
  teziste[2] = (uzel1[2] + uzel2[2] + uzel3[2] + uzel4[2])/4;

  return(1);
}

/* Vypocte grad(u) na elementu
 *
*/
int grad_u_2D(double *uzel1, double *uzel2, double *uzel3, double pot1, double pot2, double
pot3, double *gradient)
{
        double *be_koef, *ce_koef;
        double det_S;

        be_koef = (double *) malloc(3*sizeof(double));
        ce_koef = (double *) malloc(3*sizeof(double));

  vypocti_bi_ci_detS_2D (uzel1, uzel2, uzel3, be_koef, ce_koef, &det_S);

        gradient[0] = (be_koef[0]*pot1 + be_koef[1]*pot2 + be_koef[2]*pot3) / det_S;
        gradient[1] = (ce_koef[0]*pot1 + ce_koef[1]*pot2 + ce_koef[2]*pot3) / det_S;
```

```
	return(1);
}


/*---------------------------------------------------------------------------
 *  Zjisti hodnoty vektorove veliciny podel usecky zadane bod1, bod2
 *  vstupem souradnice bodu, souradnice lokace veliciny, hodnoty veliciny,
 *  pocet bodu ve kterych je velicina, rozliseni z intervalu (0, oo)
 *  vystup delkova souradnice podel usecky, velicina podel usecky
---------------------------------------------------------------------------*/
int vektor_on_line_2D(double *bod1, double *bod2, double **in_sour, double **in_velic, int
poc_bodu_velic, int resolution, double **out_sour, double **out_velic)
{
	int i, j;
	double *smer_vektor;
	double *bod_aktual;
	double x_delka, y_delka, celk_delka;
	double delta;


	smer_vektor = (double *) malloc (2 * sizeof(double));
	bod_aktual = (double *) malloc (2 * sizeof(double));


	x_delka = fabs(bod2[0] - bod1[0]);
	y_delka = fabs(bod2[1] - bod1[1]);
	celk_delka = sqrt(x_delka*x_delka + y_delka*y_delka);

	delta = 1.0/(2 * (double) resolution) * celk_delka;

	make_vektor(bod1, bod2, smer_vektor);

	for ( i = 0; i <= resolution; i++)
                                {
		bod_aktual[0] = bod1[0] + ((double) i/resolution)*smer_vektor[0];
		bod_aktual[1] = bod1[1] + ((double) i/resolution)*smer_vektor[1];

		out_velic[i][0] = out_velic[i][1] = 0.00;

		out_sour[i][0] = celk_delka * ((double) i / (double) resolution);

//		out_sour[i][0] = 0;
//		out_sour[i][1] = 0;

		for ( j = 0; j < poc_bodu_velic; j++)
		{
			if ( in_sour[j][0] > bod_aktual[0] - delta  &&
                                        in_sour[j][0] < bod_aktual[0] + delta  &&
                                        in_sour[j][1] > bod_aktual[1] - delta  &&
```

```
                                        in_sour[j][1] < bod_aktual[1] + delta  )
                        {
                        out_velic[i][0] = in_velic[j][0];
                        out_velic[i][1] = in_velic[j][1];


//              out_sour[i][0] = in_sour[j][0];
//              out_sour[i][1] = in_sour[j][1];
                                break;
                        }
                }
        }

        return(1);
}

int plumb_comp_on_line_2D(double *bod1, double *bod2, double **in_velic, int
poc_bodu_velic, double **out_velic)
{
        double *normala, *vektor, *jednot_vektor_smer, *jednot_vektor_norm, norma;
        int i;

        normala = (double *) malloc (2 * sizeof(double));
        vektor = (double *) malloc (2 * sizeof(double));
        jednot_vektor_smer = (double *) malloc (2 * sizeof(double));
        jednot_vektor_norm = (double *) malloc (2 * sizeof(double));

        make_vektor(bod1, bod2, vektor);

        norma = sqrt(vektor[0]*vektor[0] + vektor[1]*vektor[1]);

        for ( i = 0 ; i < 2 ; i++ )
                jednot_vektor_smer[i] = vektor[i] / norma;

        jednot_vektor_norm[0] = - jednot_vektor_smer[1];
        jednot_vektor_norm[1] = jednot_vektor_smer[0];


        for ( i = 0 ; i < poc_bodu_velic ; i++ )
        {
                out_velic[i][0] = in_velic[i][0] * jednot_vektor_smer[0] + in_velic[i][1] *
jednot_vektor_smer[1];
                out_velic[i][1] = in_velic[i][0] * jednot_vektor_norm[0] + in_velic[i][1] *
jednot_vektor_norm[1];
        }

        return(1);

}
```

```
/*************************************************************************
**
 * Pocita POUZE NORMALOVOU komponentu vstupni veliciny
 * ************************************************************************/

int plumb_comp_on_line_3D(double *bod1, double *bod2, double *bod3, double **in_velic,
int poc_bodu_velic, double **out_velic)
{
        double *normala, *vektor_1, *vektor_2, *jednot_vektor_norm, norma;
        int i;

        normala = (double *) malloc (3 * sizeof(double));
        vektor_1 = (double *) malloc (3 * sizeof(double));
        vektor_2 = (double *) malloc (3 * sizeof(double));
        jednot_vektor_norm = (double *) malloc (3 * sizeof(double));

        make_vektor_3D(bod1, bod2, vektor_1);
        make_vektor_3D(bod1, bod3, vektor_2);

        vector_product(vektor_1, vektor_2, jednot_vektor_norm);

        norma = vector_norm(jednot_vektor_norm);

        for ( i = 0 ; i < 3 ; i++ )
                jednot_vektor_norm[i] /= norma;

        for ( i = 0 ; i < poc_bodu_velic + 1 ; i++ )
        {
                out_velic[i][0] = in_velic[i][0] * jednot_vektor_norm[0] + in_velic[i][1] *
jednot_vektor_norm[1] + in_velic[i][2] * jednot_vektor_norm[2];
        }

        return(1);

}

int source_ele_3d (double *uzel1, double *uzel2, double *uzel3, double *uzel4, double
source, double *vystup)
{

        int i, j;
        double **uzly, **matf, *teziste, *integral;
        double volume;
        double **help;
        double **result;

        if ((uzly=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
```

```
for (j=0; j<4; j++)
  if ((uzly[j]=(double *)malloc(4*sizeof(double)))==NULL)
          return(0);

if ((matf=(double **)malloc(4*sizeof(double *)))==NULL)
         return(0);
         for (j=0; j<4; j++)
          if ((matf[j]=(double *)malloc(4*sizeof(double)))==NULL)
            return(0);

        if ((help=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
for (j=0; j<1; j++)
  if ((help[j]=(double *)malloc(4*sizeof(double)))==NULL)
          return(0);

        if ((result=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
for (j=0; j<1; j++)
  if ((result[j]=(double *)malloc(4*sizeof(double)))==NULL)
          return(0);

        teziste = (double *) malloc(3*sizeof(double));
        integral = (double *) malloc(4*sizeof(double));

        for (j=0; j<4; j++)
                uzly[j][0]=1;
for (j=1; j<4; j++)
          uzly[0][j]=uzel1[j-1];
        for (j=1; j<4; j++)
          uzly[1][j]=uzel2[j-1];
        for (j=1; j<4; j++)
          uzly[2][j]=uzel3[j-1];
        for (j=1; j<4; j++)
          uzly[3][j]=uzel4[j-1];

        invmat4(uzly, matf);

        teziste_elementu_3D (uzel1, uzel2, uzel3, uzel4, teziste);

volume = det4(uzly);
volume /= 6;
volume = fabsl(volume);

        integral[0] = source * volume;

        for ( i = 1 ; i < 4 ; i++ )
                integral[i] = teziste[i-1] * source * volume;

        for ( i = 0 ; i < 4 ; i++ )
```

```
                help[0][i] = integral[i];

        nasobmat(help, matf, result, 1, 4, 4);

        for ( i = 0 ; i < 4 ; i++ )
                vystup[i] = result[0][i];

}

int source_ele_1d (double *uzel1, double *uzel2, double source, double *vystup)
{

        int i, j;
        double **uzly, **matf, *teziste, *integral;
        double volume;
        double **help;
        double **result;

        if ((uzly=(double **)malloc(2*sizeof(double *)))==NULL)
                return(0);
 for (j=0; j<2; j++)
  if ((uzly[j]=(double *)malloc(2*sizeof(double)))==NULL)
           return(0);

 if ((matf=(double **)malloc(2*sizeof(double *)))==NULL)
          return(0);
         for (j=0; j<2; j++)
          if ((matf[j]=(double *)malloc(2*sizeof(double)))==NULL)
            return(0);

        if ((help=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
 for (j=0; j<1; j++)
  if ((help[j]=(double *)malloc(4*sizeof(double)))==NULL)
           return(0);

        if ((result=(double **)malloc(1*sizeof(double *)))==NULL)
                return(0);
 for (j=0; j<1; j++)
  if ((result[j]=(double *)malloc(4*sizeof(double)))==NULL)
           return(0);

        teziste = (double *) malloc(3*sizeof(double));
        integral = (double *) malloc(4*sizeof(double));

        for (j=0; j<2; j++)
                uzly[j][0]=1;
 for (j=1; j<2; j++)
          uzly[0][j]=uzel1[j-1];
        for (j=1; j<2; j++)
```

```
        uzly[1][j]=uzel2[j-1];

        invmat(uzly, matf, 2);

        teziste[0] = ( uzel1[0] + uzel2[0] ) / 2 ;

 volume = sqrt((uzel2[0] - uzel1[0])*(uzel2[0] - uzel1[0]) + (uzel2[1] - uzel1[1])*(uzel2[1] -
uzel1[1]));

        integral[0] = source * volume;

        for ( i = 1 ; i < 2 ; i++ )
                integral[i] = teziste[i-1] * source * volume;

        for ( i = 0 ; i < 2 ; i++ )
                help[0][i] = integral[i];

        nasobmat(help, matf, result, 1, 2, 2);

        for ( i = 0 ; i < 2 ; i++ )
                vystup[i] = result[0][i];

}


int neumann_ele_2d (double *uzel1, double *uzel2, double source, double tloustka, double
*vystup)
{
        double delka;
        int i;

 delka = sqrt((uzel2[0] - uzel1[0])*(uzel2[0] - uzel1[0]) + (uzel2[1] - uzel1[1])*(uzel2[1] -
uzel1[1]));

        for ( i = 0 ; i < 2 ; i++ )
                vystup[i] = 0.5 * delka * tloustka * source;

}

int grad_u_3D(double *uzel1, double *uzel2, double *uzel3, double *uzel4, double pot1,
double pot2, double pot3, double pot4, double *gradient)
{
        double **uzly, **matf;
        double determinant;

        int j;

        if ((uzly=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
 for (j=0; j<4; j++)
```

```
                if ((uzly[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);

        if ((matf=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((matf[j]=(double *)malloc(4*sizeof(double)))==NULL)
                        return(0);



        for (j=0; j<4; j++)
                uzly[j][0]=1;
        for (j=1; j<4; j++)
          uzly[0][j]=uzel1[j-1];
        for (j=1; j<4; j++)
          uzly[1][j]=uzel2[j-1];
        for (j=1; j<4; j++)
           uzly[2][j]=uzel3[j-1];
        for (j=1; j<4; j++)
          uzly[3][j]=uzel4[j-1];

        determinant = det4(uzly);
 determinant = fabsl(determinant);
 invmat4(uzly, matf);



        gradient[0] = (matf[1][0]*pot1 + matf[1][1]*pot2 + matf[1][2]*pot3 +
matf[1][3]*pot4);
        gradient[1] = (matf[2][0]*pot1 + matf[2][1]*pot2 + matf[2][2]*pot3 +
matf[2][3]*pot4);
        gradient[2] = (matf[3][0]*pot1 + matf[3][1]*pot2 + matf[3][2]*pot3 +
matf[3][3]*pot4);

        return(1);
}

double objem_tetrahedron(double *uzel1, double *uzel2, double *uzel3, double *uzel4)
{

        double **uzly;
        double volume;
        int j;


 if ((uzly=(double **)malloc(4*sizeof(double *)))==NULL)
                return(0);
        for (j=0; j<4; j++)
                if ((uzly[j]=(double *)malloc(4*sizeof(double)))==NULL)
```

```
                return(0);


        for (j=0; j<4; j++)
                uzly[j][0]=1;
        for (j=1; j<4; j++)
                uzly[0][j]=uzel1[j-1];
        for (j=1; j<4; j++)
                uzly[1][j]=uzel2[j-1];
        for (j=1; j<4; j++)
                uzly[2][j]=uzel3[j-1];
        for (j=1; j<4; j++)
                uzly[3][j]=uzel4[j-1];

 volume = det4(uzly);
 volume /= 6;
 volume = fabsl(volume);

        return(volume);
}

/*---------------------------------------------------------------------------
 *  Zjisti hodnoty vektorove veliciny podel usecky zadane bod1, bod2
 *  vstupem souradnice bodu, souradnice lokace veliciny, hodnoty veliciny,
 *  pocet bodu ve kterych je velicina, rozliseni z intervalu (0, oo)
 *  vystup delkova souradnice podel usecky, velicina podel usecky
---------------------------------------------------------------------------*/
int vektor_on_line_3D(double *bod1, double *bod2, double **in_sour, double **in_velic, int
poc_bodu_velic, int resolution, double **out_sour, double **out_velic)
{
        int i, j;
        double *smer_vektor;
        double *bod_aktual;
        double x_delka, y_delka, z_delka, celk_delka;
        double delta;


        smer_vektor = (double *) malloc (3 * sizeof(double));
        bod_aktual = (double *) malloc (3 * sizeof(double));


        x_delka = fabs(bod2[0] - bod1[0]);
        y_delka = fabs(bod2[1] - bod1[1]);
        z_delka = fabs(bod2[2] - bod1[2]);
        celk_delka = sqrt(x_delka*x_delka + y_delka*y_delka + z_delka*z_delka);

        delta = 1.0/(2 * (double) resolution) * celk_delka;

        make_vektor_3D(bod1, bod2, smer_vektor);
```

```
        for ( i = 0; i <= resolution; i++)
                                        {
                bod_aktual[0] = bod1[0] + ((double) i/resolution)*smer_vektor[0];
                bod_aktual[1] = bod1[1] + ((double) i/resolution)*smer_vektor[1];
                bod_aktual[2] = bod1[2] + ((double) i/resolution)*smer_vektor[2];

                out_velic[i][0] = out_velic[i][1] = 0.00;

                out_sour[i][0] = celk_delka * ((double) i / (double) resolution);

//              out_sour[i][0] = 0;
//              out_sour[i][1] = 0;

                for ( j = 0; j < poc_bodu_velic; j++)
                {
                        if ( in_sour[j][0] > bod_aktual[0] - delta  &&
                                        in_sour[j][0] < bod_aktual[0] + delta  &&
                                        in_sour[j][1] > bod_aktual[1] - delta  &&
                                        in_sour[j][1] < bod_aktual[1] + delta  &&
                                        in_sour[j][2] > bod_aktual[2] - delta  &&
                                        in_sour[j][2] < bod_aktual[2] + delta  )
                        {
                        out_velic[i][0] = in_velic[j][0];
                        out_velic[i][1] = in_velic[j][1];
                        out_velic[i][2] = in_velic[j][2];


                                break;
                        }
                }
        }

        return(1);
}

/*-------------------------------------------------------------------------
 * Vypocita deformaci na tetrahedronu.
 *
 * poradi deformace je S11, S12, S13, S22, S23, S33
 * -------------------------------------------------------------------------*/

int strain_tetrahedron (double *uzel1, double *uzel2, double *uzel3, double *uzel4, double
*posun1, double *posun2, double *posun3, double *posun4, double *strain)
{

        double **parc_u_x;
        int i, j;

        if ((parc_u_x = (double **)malloc(3*sizeof(double *))) == NULL)
                return(0);
```

```
        for (j = 0 ; j < 3 ; j++)
                if ((parc_u_x[j] = (double *)malloc(3*sizeof(double))) == NULL)
                        return(0);

/*---------------------------------------------
 * parc_u_x[i][j] = du_i / dx_j
 * ---------------------------------------------*/

        for ( i = 0 ; i < 3 ; i++ )
        {
                grad_u_3D(uzel1, uzel2, uzel3, uzel4, posun1[i], posun2[i], posun3[i],
posun4[i], parc_u_x[i]);
        }

        for ( i = 0 ; i < 1 ; i++ )
                for ( j = i ; j < 3 ; j++ )
                {
                        strain[i+j] =  parc_u_x[i][j] + parc_u_x[j][i];
                        strain[i+j] *= 0.5;
                }

        for ( i = 1 ; i < 3 ; i++ )
                for ( j = i ; j < 3 ; j++ )
                {
                        strain[i+j+1] =  parc_u_x[i][j] + parc_u_x[j][i];
                        strain[i+j+1] *= 0.5;
                }

        return(1);
}


/*-------------------------------------------------------------------------
 * Vypocita mechanicke napeti z deformaci na tetrahedronu.
 *
 * poradi deformace a napeti je S11, S12, S13, S22, S23, S33
 * -------------------------------------------------------------------------*/

int stress_tetrahedron (double *strain, double **elast_moduly, double *stress)
{

        int i, j;
        double *strain_lokal;

        strain_lokal = (double *)malloc(6 * sizeof(double));


/*-------------------------------------------------------------------
 * Protoze S jsou v poli v poradi 11, 12, 13, 22 ... je nutne je preusporadat
 * tak, aby se dalo nasobit pres indexy i, j. Tj. do poradi
```

```
 * 11, 22, 33, 23, 13, 12
 * ---------------------------------------------------------------------*/

        strain_lokal[0] = strain[0];
        strain_lokal[1] = strain[3];
        strain_lokal[2] = strain[5];
        strain_lokal[3] = strain[4];
        strain_lokal[4] = strain[2];
        strain_lokal[5] = strain[1];


/*---------------------------------------------------------------------
 * Protoze se pouziva zkraceneho indexoveho oznaceni, slozky deformace
 * s indexy i != j se ve zkracenem oznaceni pocitaji 2x - tedy
 * S_k = 2*S_ij
 * ---------------------------------------------------------------------*/

        for ( i = 0 ; i < 6 ; i++ )
                for ( j = 0 ; j < 6 ; j++ )
                {

                        if ( j == 1 || j == 2 || j == 4 )
                                stress[i] += 2 * elast_moduly[i][j] * strain[j];
                        else
                                stress[i] += elast_moduly[i][j] * strain[j];
                }

        return(1);
}


/*---------------------------------------------------------------------
 *  Zjisti hodnoty tensoru napeti ci deformace podel usecky zadane bod1, bod2
 *  vstupem souradnice bodu, souradnice lokace veliciny, hodnoty veliciny,
 *  pocet bodu ve kterych je velicina, rozliseni z intervalu (0, oo)
 *  vystup delkova souradnice podel usecky, velicina podel usecky
 ---------------------------------------------------------------------*/
int tensor_on_line_3D(double *bod1, double *bod2, double **in_sour, double **in_velic, int
poc_bodu_velic, int resolution, double **out_sour, double **out_velic)
{
        int i, j;
        double *smer_vektor;
        double *bod_aktual;
        double x_delka, y_delka, z_delka, celk_delka;
        double delta;


        smer_vektor = (double *) malloc (3 * sizeof(double));
        bod_aktual = (double *) malloc (3 * sizeof(double));
```

```
        x_delka = fabs(bod2[0] - bod1[0]);
        y_delka = fabs(bod2[1] - bod1[1]);
        z_delka = fabs(bod2[2] - bod1[2]);
        celk_delka = sqrt(x_delka*x_delka + y_delka*y_delka + z_delka*z_delka);

        delta = 1.0/(2 * (double) resolution) * celk_delka;

        make_vektor_3D(bod1, bod2, smer_vektor);

        for ( i = 0; i <= resolution; i++)
                                        {
                bod_aktual[0] = bod1[0] + ((double) i/resolution)*smer_vektor[0];
                bod_aktual[1] = bod1[1] + ((double) i/resolution)*smer_vektor[1];
                bod_aktual[2] = bod1[2] + ((double) i/resolution)*smer_vektor[2];

                out_velic[i][0] = out_velic[i][1] = 0.00;

                out_sour[i][0] = celk_delka * ((double) i / (double) resolution);

//              out_sour[i][0] = 0;
//              out_sour[i][1] = 0;

                for ( j = 0; j < poc_bodu_velic; j++)
                {
                        if ( in_sour[j][0] > bod_aktual[0] - delta  &&
                                        in_sour[j][0] < bod_aktual[0] + delta  &&
                                        in_sour[j][1] > bod_aktual[1] - delta  &&
                                        in_sour[j][1] < bod_aktual[1] + delta  &&
                                        in_sour[j][2] > bod_aktual[2] - delta  &&
                                        in_sour[j][2] < bod_aktual[2] + delta  )
                        {
                        out_velic[i][0] = in_velic[j][0];
                        out_velic[i][1] = in_velic[j][1];
                        out_velic[i][2] = in_velic[j][2];
                        out_velic[i][3] = in_velic[j][3];
                        out_velic[i][4] = in_velic[j][4];
                        out_velic[i][5] = in_velic[j][5];


                                break;
                        }
                }
        }

        return(1);
}

int stress_components_on_surface (double *bod_1, double *bod_2, double *bod_3, int
resolution, double **stress_in, double **stress_out)
```

```
{

        int i, j;

        double *unit_vector;
        double **stress_matrix;
        double **unit_vector_matrix;
        double **force;

        unit_vector = (double *) malloc(3 * sizeof(double));

        stress_matrix = (double **)malloc(3*sizeof(double *));
 for (j = 0 ; j < 3 ; j++)
                stress_matrix[j] = (double *)malloc(3*sizeof(double));

        force = (double **)malloc(3*sizeof(double *));
 for (j = 0 ; j < 3 ; j++)
                force[j] = (double *)malloc(1*sizeof(double));

        unit_vector_matrix = (double **)malloc(3*sizeof(double *));
 for (j = 0 ; j < 3 ; j++)
                unit_vector_matrix[j] = (double *)malloc(1*sizeof(double));

        normal_unit_vector_on_surface(bod_1, bod_2, bod_3, unit_vector);

        for ( i = 0 ; i < 3 ; i++ )
                unit_vector_matrix[i][0] = unit_vector[i];

        for ( j = 0 ; j < resolution+1 ; j++ )
        {
                for ( i = 0 ; i < 3 ; i++ )
                        stress_matrix[0][i] = stress_in[j][i];

                stress_matrix[1][0] = stress_matrix[0][1];
                stress_matrix[1][1] = stress_in[j][3];
                stress_matrix[1][2] = stress_in[j][4];
                stress_matrix[2][0] = stress_matrix[0][2];
                stress_matrix[2][1] = stress_matrix[1][2];
                stress_matrix[2][2] = stress_in[j][5];

                nasobmat(stress_matrix, unit_vector_matrix, force, 3, 3, 1);


                for ( i = 0 ; i < 3 ; i++ )
                        stress_out[j][i] = force[0][i];
        }
        return(1);

}
```

Soubor gmsh.c

```
        /*********************************************************************
        ********** FUNKCE PRO IMPORT DAT ZE SYSTEMU GMSH
**********
        **********  Copyright (c) 2001 by Josef Novak  **********
        ******************        All Rights Reserved.  **********************
        *********************************************************************/

#include "big-head.h"

#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>
//#include"gmsh.h"


/*-------------------------------------------------------------------------
int pocet_nodu(char *);
int * indexy_nodu(char *);
double ** souradnice_nodu(char *);
double ** serad_nody(double **, int *, int);
int pocet_elementu(char *);
int ** nody_elementu(char *);
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
        Funkce pro zjištění počtu nodu v *.msh
 jméno souboru i s příponou je parametrem funkce a je typu char *
 vrací se počet nodu v souboru - typ int
-------------------------------------------------------------------------*/
int pocet_nodu(char *soubor)
{
 FILE *fr;
 int pocet, c;

        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 c = getc(fr);
 while ( c != '\n')  // jde na konec radku s identif. $NOD
        c = getc(fr);

        fscanf(fr, "%d", &pocet);   // nacte pocet nodu
 fclose(fr);
 return(pocet);
}

/*-------------------------------------------------------------------------
```

```
        Funkce pro zjištění cisel nodu, ze kterych je tvorena sit tetrahedru *.msh
 jméno souboru i s příponou je parametrem funkce a je typu char *
 vrací se pole cisel nodu v souboru v poradi v jakem jsou uvedeny v *.msh
 je typu int * - pole se alokuje ve funkci - pri vstupu se zadava jen pointer,
 pole ma delku rovnou poctu nodu site
-------------------------------------------------------------------------------*/
int * indexy_nodu(char *soubor)
{
        FILE *fr;
 int *indexy;
 int i, pocet, c;

        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 c = getc(fr);
 while ( c != '\n') // jde na konec radku s identif. $NOD
        c = getc(fr);

        fscanf(fr, "%d", &pocet);  // cte pocet nodu

        indexy = (int *) malloc(pocet * sizeof(int)); // alokace pole pro cisla nodu

 for ( i=0; i<pocet; i++)
 {
        fscanf(fr, "%d", &indexy[i]);  //cteni cisel nodu
  c = getc(fr);
          while ( c != '\n')
                        c = getc(fr);
 }

 fclose(fr);
 return(indexy);
}

/*-------------------------------------------------------------------------------
        Funkce pro nacteni souradnic nodu, ze kterych je tvorena sit tetrahedru *.msh
 jméno souboru i s příponou je parametrem funkce a je typu char *
 vrací se pole double ** kde prvni index je poradi nodu v souboru - indexovano od 0;
 druhy index je souradnice - [0] = x, [1] = y, [2] = z;
 pole se alokuje ve funkci - pri vstupu se zadava jen pointer
-------------------------------------------------------------------------------*/
double ** souradnice_nodu(char *soubor)
{
        FILE *fr;
 double **souradnice;
 int i, j, k, pocet, c;
 char *retezec;

 retezec = (char *) malloc(50 * sizeof(char));
```

```
        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 c = getc(fr);
 while ( c != '\n')   // jde na konec radku s identif $NOD
        c = getc(fr);

        fscanf(fr, "%d", &pocet);  // cte pocet nodu

 //alokace pole pro souradnice nodu
        souradnice = (double **) malloc(pocet * sizeof(double*));
 for ( i=0; i<pocet; i++)
        souradnice[i] = (double *) malloc(3 * sizeof(double));

 /*      Problemem je to, ze nektere souradnice se tvari jako cisla int, a pak
    je fscanf jako double nenacte - preskoci je a cte prvni double na ktery narazi
    Proto se kazde cislo nacte jako retezec charu a ten se pak konvertuje na cislo.
    Vyuziva se toho ze cisla jsou od sebe oddelena mezerou ci odradkovanim
 */

 for ( i=0; i<pocet; i++)
 {
  fscanf(fr, "%d", &c); // preskakuje se cislo elementu

  fscanf(fr, "%lf", &souradnice[i][0]);
  fscanf(fr, "%lf", &souradnice[i][1]);
  fscanf(fr, "%lf", &souradnice[i][2]);

/*   c = getc(fr);
   for (j=0; j<3; j++)  // souradnice kazdeho nodu jsou 3 - x, y, z
   {
          k=0;
         retezec[k] = getc(fr);
    // identifikace oddeleni cisel mezerou ci odradkovanim
        while (retezec[k] != ' ' && retezec[k] != '\n')
          {
               k++;
         retezec[k] = getc(fr);
         }
        retezec[k] = '\0'; // ukonceni retezce

    souradnice[i][j] = atof(retezec); // prevod retezce na cislo
*/

//                }

 }

 fclose(fr);
```

```
 return(souradnice);
}

/*---------------------------------------------------------------------------
        Funkce vraci pole nodu, ve kterem jsou nody usporadany podle jejich cisel,
 tj. prvni index vraceneho pole znaci cislo nodu uvedene v souboru *.msh.
 Druhy index je souradnice - x, y, z
 pole se alokuje ve funkci - pri vstupu se zadava jen pointer
 Nody jsou v *.msh cislovany od jednicky, takze prvni polozka pole je nenaplnena
 a pole ma delku maximalniho cisla nodu +1
 parametry jsou double **, ktery vraci funkce souradnice_nodu();
                                                    int *, ktery vraci funkce indexy_nodu();
            int, kteruy vraci funkce pocet_nodu();
---------------------------------------------------------------------------*/
double ** serad_nody(double **neserazene, int *indexy, int pocet)
{
        double **serazene;
 int i;

 serazene = (double **) malloc((indexy[pocet-1] + 1) * sizeof(double *));
 for ( i=0; i<(indexy[pocet-1] + 1); i++)
        serazene[i] = (double *) malloc(3 * sizeof(double));

 for ( i = (pocet - 1); i >= 0; i--)
 {
        serazene[indexy[i]][0] = neserazene[i][0];
  serazene[indexy[i]][1] = neserazene[i][1];
  serazene[indexy[i]][2] = neserazene[i][2];
 }
 return(serazene);
}

/*---------------------------------------------------------------------------
        Funkce vraci int, pocet tetrahedrovych elementu v souboru *.msh
 jméno souboru i s příponou je parametrem funkce a je typu char *
---------------------------------------------------------------------------*/
int pocet_elementu(char *soubor)
{
        FILE *fr;
 int pocet = 0, cislo, typ, c;

        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 while ( getc(fr) != 'L'); // dostane se az na radek kde je $ELM

 while ( getc(fr) != '\n');  //jde na konec radku $ELM
 while ( getc(fr) != '\n');  // jde na konec radku s celkovym poctem elementu

 /* tetrahedry jsou typem elementu, ktery je v *.msh oznacen cislem 4
```

```
        a typ elementu je vzdy druhym cislem na radku. Prvnim cislem je cislo
   elementu.
 */
 while ( (c = (getc(fr))) != '$')  // zjistuje se, zda uz neni $ENDELM
 {
        ungetc (c, fr);  //
                fscanf(fr, " %d %d ", &cislo, &typ);  //

  if ( typ == 4)
        pocet++;

                while ( getc(fr) != '\n'); // postup na dalsi radku*/
        }
 fclose(fr);
 return(pocet);
}

/*---------------------------------------------------------------------------
        Funkce vraci int **, pole obsahujici nody nody tetrahedrovych elementu
 druhy index ma 5 polozek - [0] - [3] = cisla nodu, [4] = cislo tetrahedr.
 elementu v souboru *.msh
        [5] - cislo fyz. entity, kam patri
 jméno souboru i s příponou je parametrem funkce a je typu char *
---------------------------------------------------------------------------*/
int ** nody_elementu(char *soubor)
{
        FILE *fr;
 int **nody, pocet_el, typ, i;


 pocet_el = pocet_elementu(soubor);  // zjisti pocet elementu v souboru

 // alokace pole pro cisla elementu
 nody = (int **) malloc(pocet_el * sizeof(int *));
 for ( i=0; i<pocet_el; i++)
        nody[i] = (int *) malloc(6 * sizeof(int));

 if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 while ( getc(fr) != 'L');  // dojde az k $ELM
 while ( getc(fr) != '\n'); // preskoci radek s $ELM
 while ( getc(fr) != '\n'); // preskoci radek s celkovym poctem elementu

 fscanf(fr, " %d %d ", &nody[0][4], &typ);
 while ( typ != 4 )   // zjistuje zacatek vypisu tetrahedru
        {
                while ( getc(fr) != '\n');
                fscanf(fr, " %d %d ", &nody[0][4], &typ);
        }
```

```
 // zapise prvni z tetrahedru
        fscanf(fr, " %d %d %d %d %d %d %d ", &nody[0][5], &typ, &typ, &nody[0][0],
&nody[0][1], &nody[0][2], &nody[0][3]);

 //dale cte a zapisuje dle znameho poctu tetrahedru v souboru a poctu promennych na radce
 for ( i=1; i<pocet_el; i++)
        fscanf(fr, " %d %d %d %d %d %d %d %d %d", &nody[i][4], &typ, &nody[i][5],
&typ, &typ, &nody[i][0], &nody[i][1], &nody[i][2], &nody[i][3]);

 fclose(fr);
 return(nody);
}

/*---------------------------------------------------------------------
 Funkce pro zjisteni nodu nalezajicich se v urcite geom. entite
---------------------------------------------------------------------*/
int nody_v_entite (char *soubor, int entita, int *seznam)
{
 int pocet_vsech_nodu, pocet_vsech_elementu, region, cislo, typ;
 int *node_numbers, nb_nodes, unused, poc_v_seznamu, empty;
        int i, j, k, nalezeno, c;
 FILE *fr;


 node_numbers = (int *) malloc (8 * sizeof(int));
 poc_v_seznamu = 0 ;
 nb_nodes = 0;
        nalezeno = 0;

 //nejdriv se zjisti pocet vsech nodu v siti a alokuje se pole pro seznam
//  pocet_vsech_nodu = pocet_nodu(soubor);
//  seznam = (int *) malloc(pocet_vsech_nodu * sizeof(int));

 if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 while ( getc(fr) != 'L');  // dojde az k $ELM
 while ( getc(fr) != '\n'); // preskoci radek s $ELM
//  while ( getc(fr) != '\n'); // preskoci radek s celkovym poctem elementu

 fscanf(fr, " %d ", &pocet_vsech_elementu);

        empty = 1;

 for ( i = 0 ; i < pocet_vsech_elementu ; i++ )
 {
                //precte se cislo elementu, jeho typ a region
                fscanf(fr, "%d %d %d ", &cislo, &typ, &region);

                //jestlize elem. patri do hledane entity
```

```
                if ( region == entita )
                {
                        //podle typu elementu - kolik nodu ma
                        switch (typ)
                        {
                                case 1 :
                                {
                                        fscanf (fr, " %d %d %d %d ", &unused, &nb_nodes,
&node_numbers[0], &node_numbers[1]);
                                } break;
                                case 2 :
                                {
                                        fscanf (fr, " %d %d %d %d %d ", &unused, &nb_nodes,
&node_numbers[0], &node_numbers[1], &node_numbers[2]);
                                }       break;
                                case 3 :
                                {
                                        fscanf (fr, " %d %d %d %d %d %d ", &unused,
&nb_nodes, &node_numbers[0], &node_numbers[1], &node_numbers[2],
&node_numbers[3]);
                                }       break;
                                case 4 :
                                {
                                        fscanf (fr, " %d %d %d %d %d %d ", &unused,
&nb_nodes, &node_numbers[0], &node_numbers[1], &node_numbers[2],
&node_numbers[3]);
                                }       break;
                                default :
                                  break;
                }
        }
        else
                while((c=getc(fr)) !='\n'); //dojde na konec radku

                if (empty == 1 && nb_nodes != 0)
                {
                        for ( j = 0 ; j < nb_nodes ; j++ )
                        {
                                seznam[j] = node_numbers[j];
                                poc_v_seznamu ++;
                        }
                        empty = 0;
                }

                for ( j = 0 ; j < nb_nodes ; j++ )
                {
                        for ( k = 0 ; k < poc_v_seznamu ; k++ )

                        {
                                if ( node_numbers[j] == seznam[k] )
```

```
                                {
                                        nalezeno = 1;
                                        break;
                                }
//                              else
//                                      nalezeno = 0;
                        }
                        if ( nalezeno == 0 )
                        {
                                seznam[poc_v_seznamu] = node_numbers[j];
                                poc_v_seznamu ++;
                        }
                        nalezeno = 0;
                }
        }

        return(poc_v_seznamu);
}

/*-------------------------------------------------------------------------
        Funkce pro nacteni souradnic nodu, ze kterych je tvorena sit trianglu *.msh
 jméno souboru i s příponou je parametrem funkce a je typu char *
 vrací se pole double ** kde prvni index je poradi nodu v souboru - indexovano od 0;
 druhy index je souradnice - [0] = x, [1] = y;
 pole se alokuje ve funkci - pri vstupu se zadava jen pointer
-------------------------------------------------------------------------*/
double ** souradnice_nodu_2D(char *soubor)
{
        FILE *fr;
 double **souradnice;
 int i, j, k, pocet, c;
 char *retezec;

 retezec = (char *) malloc(50 * sizeof(char));

        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 c = getc(fr);
 while ( c != '\n')   // jde na konec radku s identif $NOD
        c = getc(fr);

        fscanf(fr, "%d", &pocet);  // cte pocet nodu

 //alokace pole pro souradnice nodu
        souradnice = (double **) malloc(pocet * sizeof(double*));
 for ( i=0; i<pocet; i++)
        souradnice[i] = (double *) malloc(3 * sizeof(double));

 /*     Problemem je to, ze nektere souradnice se tvari jako cisla int, a pak
```

```
    je fscanf jako double nenacte - preskoci je a cte prvni double na ktery narazi
    Proto se kazde cislo nacte jako retezec charu a ten se pak konvertuje na cislo.
    Vyuziva se toho ze cisla jsou od sebe oddelena mezerou ci odradkovanim
 */

 for ( i=0; i<pocet; i++)
 {
  fscanf(fr, "%d", &c); // preskakuje se cislo elementu

  fscanf(fr, "%lf", &souradnice[i][0]);
  fscanf(fr, "%lf", &souradnice[i][1]);
//          while ( getc(fr) != '\n');   // jde na konec radku
  fscanf(fr, "%lf", &souradnice[i][2]);

/*   c = getc(fr);
  for (j=0; j<3; j++)  // souradnice kazdeho nodu jsou 3 - x, y, z
  {
           k=0;
         retezec[k] = getc(fr);
    // identifikace oddeleni cisel mezerou ci odradkovanim
        while (retezec[k] != ' ' && retezec[k] != '\n')
          {
               k++;
         retezec[k] = getc(fr);
         }
        retezec[k] = '\0'; // ukonceni retezce

    souradnice[i][j] = atof(retezec); // prevod retezce na cislo
*/

//                }

 }

 fclose(fr);
 return(souradnice);
}

/*-------------------------------------------------------------------------
        Funkce vraci pole nodu, ve kterem jsou nody usporadany podle jejich cisel,
 tj. prvni index vraceneho pole znaci cislo nodu uvedene v souboru *.msh.
 Druhy index je souradnice - x, y, z
 pole se alokuje ve funkci - pri vstupu se zadava jen pointer
 Nody jsou v *.msh cislovany od jednicky, takze prvni polozka pole je nenaplnena
 a pole ma delku maximalniho cisla nodu +1
 parametry jsou double **, ktery vraci funkce souradnice_nodu();
                                                    int *, ktery vraci funkce indexy_nodu();
           int, kteruy vraci funkce pocet_nodu();
-------------------------------------------------------------------------*/
double ** serad_nody_2D(double **neserazene, int *indexy, int pocet)
```

```
{
        double **serazene;
 int i;

 serazene = (double **) malloc((indexy[pocet-1] + 1) * sizeof(double *));
 for ( i=0; i<(indexy[pocet-1] + 1); i++)
        serazene[i] = (double *) malloc(3 * sizeof(double));

 for ( i = (pocet - 1); i >= 0; i--)
 {
        serazene[indexy[i]][0] = neserazene[i][0];
   serazene[indexy[i]][1] = neserazene[i][1];
//   serazene[indexy[i]][2] = neserazene[i][2];
 }
 return(serazene);
}

/*-------------------------------------------------------------------------
        Funkce vraci int, pocet trianglovych elementu v souboru *.msh
 jméno souboru i s příponou je parametrem funkce a je typu char *
-------------------------------------------------------------------------*/
int pocet_elementu_triangle(char *soubor)
{
        FILE *fr;
 int pocet = 0, cislo, typ, c;

        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 while ( getc(fr) != 'L'); // dostane se az na radek kde je $ELM

 while ( getc(fr) != '\n');  //jde na konec radku $ELM
 while ( getc(fr) != '\n');  // jde na konec radku s celkovym poctem elementu

 /* tetrahedry jsou typem elementu, ktery je v *.msh oznacen cislem 4
        a typ elementu je vzdy druhym cislem na radku. Prvnim cislem je cislo
   elementu.
 */
 while ( (c = (getc(fr))) != '$')  // zjistuje se, zda uz neni $ENDELM
 {
        ungetc (c, fr);  //
                fscanf(fr, " %d %d ", &cislo, &typ);  //

  if ( typ == 2)
        pocet++;

                while ( getc(fr) != '\n'); // postup na dalsi radku*/
        }
 fclose(fr);
 return(pocet);
```

```
}

/*---------------------------------------------------------------------------
        Funkce vraci int **, pole obsahujici nody nody triangloovych elementu
  druhy index ma 5 polozek - [0] - [2] = cisla nodu, [3] = cislo triangl.
  elementu v souboru *.msh, [4] - physical surface, ve kterem se nachazi
  jméno souboru i s příponou je parametrem funkce a je typu char *
---------------------------------------------------------------------------*/
int ** nody_elementu_triangle(char *soubor)
{
        FILE *fr;
  int **nody, pocet_el, typ, i;


  pocet_el = pocet_elementu_triangle(soubor);  // zjisti pocet elementu v souboru

  // alokace pole pro cisla elementu
  nody = (int **) malloc(pocet_el * sizeof(int *));
  for ( i=0; i<pocet_el; i++)
        nody[i] = (int *) malloc(5 * sizeof(int));

  if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

  while ( getc(fr) != 'L');  // dojde az k $ELM
  while ( getc(fr) != '\n'); // preskoci radek s $ELM
  while ( getc(fr) != '\n'); // preskoci radek s celkovym poctem elementu

  fscanf(fr, " %d %d ", &nody[0][3], &typ);
  while ( typ != 2 )   // zjistuje zacatek vypisu trianglu
  {
                while ( getc(fr) != '\n');
                fscanf(fr, " %d %d ", &nody[0][3], &typ);
        }
  // zapise prvni z trianglu
        fscanf(fr, " %d %d %d %d %d %d ", &nody[0][4], &typ, &typ, &nody[0][0],
&nody[0][1], &nody[0][2]);

  //dale cte a zapisuje dle znameho poctu trianglu v souboru a poctu promennych na radce
  for ( i=1; i<pocet_el; i++)
        fscanf(fr, " %d %d %d %d %d %d %d %d ", &nody[i][3], &typ, &nody[i][4], &typ,
&typ, &nody[i][0], &nody[i][1], &nody[i][2]);

 fclose(fr);
 return(nody);
}

/*---------------------------------------------------------------------------
        Funkce vraci int, pocet line elementu v souboru *.msh
  jméno souboru i s příponou je parametrem funkce a je typu char *
```

```
-------------------------------------------------------------------------*/
int pocet_elementu_line(char *soubor)
{
        FILE *fr;
 int pocet = 0, cislo, typ, c;

        if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 while ( getc(fr) != 'L'); // dostane se az na radek kde je $ELM

 while ( getc(fr) != '\n');  //jde na konec radku $ELM
 while ( getc(fr) != '\n');  // jde na konec radku s celkovym poctem elementu

 /* line jsou typem elementu, ktery je v *.msh oznacen cislem 1
        a typ elementu je vzdy druhym cislem na radku. Prvnim cislem je cislo
   elementu.
 */
 while ( (c = (getc(fr))) != '$')  // zjistuje se, zda uz neni $ENDELM
 {
        ungetc (c, fr);  //
                fscanf(fr, " %d %d ", &cislo, &typ);  //

  if ( typ == 1)
        pocet++;

                while ( getc(fr) != '\n'); // postup na dalsi radku*/
        }
 fclose(fr);
 return(pocet);
}

/*-------------------------------------------------------------------------
        Funkce vraci int **, pole obsahujici nody nody line elementu
 druhy index ma 5 polozek - [0] - [1] = cisla nodu, [2] = cislo line
 elementu v souboru *.msh, [3] - physical line, ve kterem se nachazi
 jméno souboru i s příponou je parametrem funkce a je typu char *
-------------------------------------------------------------------------*/
int ** nody_elementu_line(char *soubor)
{
        FILE *fr;
 int **nody, pocet_el, typ, i;


 pocet_el = pocet_elementu_line(soubor);  // zjisti pocet elementu v souboru

 // alokace pole pro cisla elementu
 nody = (int **) malloc(pocet_el * sizeof(int *));
 for ( i=0; i<pocet_el; i++)
        nody[i] = (int *) malloc(4 * sizeof(int));
```

```
 if ((fr = fopen(soubor, "r")) == NULL)
        return(0);

 while ( getc(fr) != 'L');  // dojde az k $ELM
 while ( getc(fr) != '\n'); // preskoci radek s $ELM
 while ( getc(fr) != '\n'); // preskoci radek s celkovym poctem elementu

 fscanf(fr, " %d %d ", &nody[0][2], &typ);
 while ( typ != 1 )   // zjistuje zacatek vypisu line elementu
 {
                while ( getc(fr) != '\n');
                fscanf(fr, " %d %d ", &nody[0][2], &typ);
        }
 // zapise prvni z trianglu
        fscanf(fr, " %d %d %d %d %d ", &nody[0][3], &typ, &typ, &nody[0][0],
&nody[0][1]);

 //dale cte a zapisuje dle znameho poctu trianglu v souboru a poctu promennych na radce
 for ( i=1; i<pocet_el; i++)
        fscanf(fr, " %d %d %d %d %d %d %d ", &nody[i][2], &typ, &nody[i][3], &typ, &typ,
&nody[i][0], &nody[i][1]);

 fclose(fr);
 return(nody);
}
```

Zdrojový kód programu pro výpočet vektoru posunutí a elektrických potenciálů:

```
#define MAIN
#include "big-head.h"

#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
#include<gsl/gsl_cblas.h>

/* Parametry programu
 *
 * jmeno souboru site
 *
*/

main(int argc, char* argv[])
{

        if (argc != 8)
        {
                fprintf(stderr," \n Chybne argumenty spusteni!\n");
                fprintf(stderr," \n jmeno souboru site  \n soubor s materialem 1 \n soubor s
materialem 2 \n lokace materialu 1 \n lokace materialu 2 \n velikost potencialu \n velikost sily
\n \n");
                return(0);
        }



 int poc_nod, poc_elem;
 int i, j, k, l, m, n, r, s;
 int *indexove, **skladba_elementu, fix[10];
 int *okp_locate, *pocet_node_okp, **seznam_node_okp, *surface_locate;
 double **pom, **sour_nody;
 double *nod_0, *nod_1, *nod_2, *nod_3;
 FILE *fw;

 double **ela1, **pie1, **ele1;
 double **ela2, **pie2, **ele2;
 double hustota;

        double *right_side_elast, *right_side_elektro;


 int ITYPE = 1;
 unsigned char JOBZ = 'V';
```

```
        unsigned char UPLO = 'U';
        double *A;
        double *b;
        int LDA;
        double *C;
        int LDB;
        int LDC;
        int *ipiv;
        double *W;
        double *WORK;
        int LWORK;
        int INFO;

        int M, N, K;
        double ALPHA, BETA;

        int NRHS = 1;
  unsigned char TRANS = 'T';
  unsigned char TRANSA, TRANSB;

        double *We, *Wd, *Wp;

        double force;
        double potential;

//-------------------------------------------------------------------------
//      ALOKACE PAMETI PRO LOKALNI PROMENNE
//-------------------------------------------------------------------------

  nod_0 = (double *) malloc(3 * sizeof(double));
  nod_1 = (double *) malloc(3 * sizeof(double));
  nod_2 = (double *) malloc(3 * sizeof(double));
  nod_3 = (double *) malloc(3 * sizeof(double));

  pocet_node_okp  = (int *) malloc(4 * sizeof(int));
  okp_locate  = (int *) malloc(4 * sizeof(int));
  surface_locate  = (int *) malloc(2 * sizeof(int));

//-------------------------------------------------------------------------
//      ALOKACE PAMETI PRO GLOBALNI PROMENNE
//-------------------------------------------------------------------------

  ela1=(double **)malloc(6*sizeof(double *));
        for (j=0; j<6; j++)
                        ela1[j]=(double *)malloc(6*sizeof(double));

        pie1=(double **)malloc(3*sizeof(double *));
                for (j=0; j<3; j++)
                        pie1[j]=(double *)malloc(6*sizeof(double));
```

```
        ele1=(double **)malloc(3*sizeof(double *));
                for (j=0; j<3; j++)
                        ele1[j]=(double *)malloc(3*sizeof(double));

    ela2=(double **)malloc(6*sizeof(double *));
        for (j=0; j<6; j++)
                        ela2[j]=(double *)malloc(6*sizeof(double));

        pie2=(double **)malloc(3*sizeof(double *));
                for (j=0; j<3; j++)
                        pie2[j]=(double *)malloc(6*sizeof(double));

        ele2=(double **)malloc(3*sizeof(double *));
                for (j=0; j<3; j++)
                        ele2[j]=(double *)malloc(3*sizeof(double));


        lok_A.lok=(double ****)malloc(4*sizeof(double ***));
        for (i=0; i<4; i++)
                lok_A.lok[i]=(double ***)malloc(4*sizeof(double **));
                for (i=0; i<4; i++)
                        for (j=0; j<4; j++)
                                lok_A.lok[i][j]=(double **)malloc(3*sizeof(double *));
                                for (i=0; i<4; i++)
                                        for (j=0; j<4; j++)
                                                for (k=0; k<3; k++)
        {
                                                        lok_A.lok[i][j][k]=(double
*)malloc(3*sizeof(double));
                                        for (l=0; l<3; l++)
                                    lok_A.lok[i][j][k][l]=0;
                                }

        lok_E.lok=(double ****)malloc(4*sizeof(double ***));
        for (i=0; i<4; i++)
                lok_E.lok[i]=(double ***)malloc(4*sizeof(double **));
        for ( i = 0 ; i < 4 ; i++ )
                for ( j = 0 ; j < 4 ; j++ )
                        lok_E.lok[i][j] = (double **)malloc(1*sizeof (double *));
        for ( i = 0 ; i < 4 ; i++ )
                for ( j = 0 ; j < 4 ; j++ )
                        for ( k = 0 ; k < 1 ; k++ )
                                lok_E.lok[i][j][k] = (double *)malloc(1*sizeof (double));


    lok_P.lokp=(double ****)malloc(4*sizeof(double ***));
        for (i=0; i<4; i++)
          lok_P.lokp[i]=(double ***)malloc(4*sizeof(double **));
          for (i=0; i<4; i++)
            for (j=0; j<4; j++)
```

```
            lok_P.lokp[i][j]=(double **)malloc(1*sizeof(double *));
           for (i=0; i<4; i++)
            for (j=0; j<4; j++)
             lok_P.lokp[i][j][0]=(double *)malloc(3*sizeof(double));

 We = (double *) malloc(1 * sizeof(double));
 Wd = (double *) malloc(1 * sizeof(double));
 Wp = (double *) malloc(1 * sizeof(double));



//---------------------------------------------------------------------------
//      NACTENI SITE
//---------------------------------------------------------------------------

 if ( (indexove = indexy_nodu(argv[1])) == 0)
 {
         printf(" \n   Chybne zadane jmeno souboru site! \n");
  return(0);
 }
 pom = souradnice_nodu(argv[1]);
 //pocet nodu
        poc_nod = pocet_nodu(argv[1]);

 //souradnice nodu podle cisel v *.msh
 sour_nody = serad_nody(pom, indexove, poc_nod);
 //pocet elementu site
 poc_elem = pocet_elementu(argv[1]);
 // nody tvorici elementy
 skladba_elementu = nody_elementu(argv[1]);

 printf ("\n\n   Sit nactena ... \n");

//---------------------------------------------------------------------------
//      ALOKACE PAMETI PRO GLOBALNI PROMENNE
//---------------------------------------------------------------------------

        global_A.lok=(double ****)malloc(poc_nod*sizeof(double ***));
        for (i=0; i<poc_nod; i++)
        global_A.lok[i]=(double ***)malloc(poc_nod*sizeof(double **));
                for (i=0; i<poc_nod; i++)
                for (j=0; j<poc_nod; j++)
                                global_A.lok[i][j]=(double **)malloc(3*sizeof(double *));
                                for (i=0; i<poc_nod; i++)
                                        for (j=0; j<poc_nod; j++)
                                                for (k=0; k<3; k++)
        {
                                                        global_A.lok[i][j][k]=(double
*)malloc(3*sizeof(double));
```

```
                        for (l=0; l<3; l++)
                                global_A.lok[i][j][k][l]=0;
                }


        global_E.lok=(double ****)malloc(poc_nod*sizeof(double ***));
        for (i=0; i<poc_nod; i++)
        global_E.lok[i]=(double ***)malloc(poc_nod*sizeof(double **));
        for ( i = 0 ; i < poc_nod ; i++ )
                for ( j = 0 ; j < poc_nod ; j++ )
                        global_E.lok[i][j] = (double **)malloc(1*sizeof (double *));
        for ( i = 0 ; i < poc_nod ; i++ )
                for ( j = 0 ; j < poc_nod ; j++ )
                        for ( k = 0 ; k < 1 ; k++ )
                                global_E.lok[i][j][k] = (double *)malloc(1*sizeof (double));

        for ( i = 0 ; i < poc_nod ; i++ )
                for ( j = 0 ; j < poc_nod ; j++ )
                                global_E.lok[i][j][0][0] = 0.0;


global_P.lokp=(double ****)malloc(poc_nod*sizeof(double ***));
        for (i=0; i < poc_nod ; i++)
          global_P.lokp[i]=(double ***)malloc(poc_nod*sizeof(double **));
          for (i=0; i < poc_nod; i++)
            for (j=0; j < poc_nod ; j++)
              global_P.lokp[i][j]=(double **)malloc(1*sizeof(double *));
            for (i=0; i < poc_nod ; i++)
              for (j=0; j < poc_nod ; j++)
                global_P.lokp[i][j][0]=(double *)malloc(3*sizeof(double));

global_Pt.lokp=(double ****)malloc(poc_nod*sizeof(double ***));
        for (i=0; i < poc_nod ; i++)
          global_Pt.lokp[i]=(double ***)malloc(poc_nod*sizeof(double **));
          for (i=0; i < poc_nod; i++)
            for (j=0; j < poc_nod ; j++)
              global_Pt.lokp[i][j]=(double **)malloc(3*sizeof(double *));
            for (i=0; i < poc_nod ; i++)
              for (j=0; j < poc_nod ; j++)
                  for (k=0; k < 3 ; k++)
                      global_Pt.lokp[i][j][k]=(double *)malloc(1*sizeof(double));


        seznam_node_okp = (int **) malloc(4*sizeof(int *));
        for ( i = 0 ; i < 4 ; i++)
                seznam_node_okp[i] = (int *) malloc(poc_nod*sizeof(int));

 right_side_elast = (double *) malloc(3*poc_nod * sizeof(double));
 right_side_elektro = (double *) malloc(poc_nod * sizeof(double));
```

```
        A=(double *)malloc(poc_nod*poc_nod*16*sizeof(double));

        b=(double *)malloc(4*poc_nod*sizeof(double ));


        C=(double *)malloc(3*poc_nod*sizeof(double));

        W=(double *)malloc(poc_nod*4*sizeof(double));

        WORK=(double *)malloc((poc_nod*16-1)*sizeof(double));

        ipiv=(int *)malloc(4*poc_nod*sizeof(int ));

//-------------------------------------------------------------------------
//      NACTENI MATERIALU
//-------------------------------------------------------------------------

        if ( (readstm("kanal.stm", argv[2], 0, ela1, pie1, ele1)) == 0)
 {
          printf(" \n   Chybne zadane jmeno souboru materialovych parametru! \n");
  return(0);
 }

        if ( (readstm("kanal.stm", argv[3], 0, ela2, pie2, ele2)) == 0)
 {
          printf(" \n   Chybne zadane jmeno souboru materialovych parametru! \n");
  return(0);
 }


 hustota = 2649;       //2649 - quartz , 7850 - ocel  3570 - GaPo4


        printf ("\n   Materialove parametry nacteny ... \n");

//-------------------------------------------------------------------------
//      OKRAJOVE PODMINKY
//-------------------------------------------------------------------------

        printf(" Zadej entitu s nulovymi posunutimi: ");
 scanf("%d", &okp_locate[0]);
/*
        printf(" Zadej entitu s nulovymi posunutimi v osach x, z: ");
 scanf("%d", &okp_locate[1]);
*/
        printf(" Zadej entitu se silou: ");
 scanf("%d", &okp_locate[1]);

        printf(" Zadej entitu s nulovym potencialem: ");
 scanf("%d", &okp_locate[2]);
```

```
        printf(" Zadej entitu s nenulovym potencialem: ");
  scanf("%d", &okp_locate[3]);

        pocet_node_okp[0] = nody_v_entite (argv[1], okp_locate[0], seznam_node_okp[0]);

        pocet_node_okp[1] = nody_v_entite (argv[1], okp_locate[1], seznam_node_okp[1]);

        pocet_node_okp[2] = nody_v_entite (argv[1], okp_locate[2], seznam_node_okp[2]);

        pocet_node_okp[3] = nody_v_entite (argv[1], okp_locate[3], seznam_node_okp[3]);



//-------------------------------------------------------------------------
//      LOKACE MATERIALU
//-------------------------------------------------------------------------

        surface_locate[0] = atoi(argv[4]);
        surface_locate[1] = atoi(argv[5]);

//-------------------------------------------------------------------------
//      NAPLNENI GLOBALNICH MATIC
//-------------------------------------------------------------------------

        for ( i=0; i<poc_elem; i++ )
  {

        for ( j = 0; j<3; j++)
    {
        nod_0[j] = sour_nody[skladba_elementu[i][0]][j];
           nod_1[j] = sour_nody[skladba_elementu[i][1]][j];
         nod_2[j] = sour_nody[skladba_elementu[i][2]][j];
        nod_3[j] = sour_nody[skladba_elementu[i][3]][j];
                }

  if (skladba_elementu[i][5] == surface_locate[0] )
        {
        LokalMate(nod_0, nod_1, nod_2, nod_3, 0, ela1, 1, lok_A);
        LokalEle(nod_0, nod_1, nod_2, nod_3, 0, ele1, 1, lok_E);
                LokalMatp(nod_0, nod_1, nod_2, nod_3, 0, pie1, 1, lok_P);
        }
        else
        {
                LokalMate(nod_0, nod_1, nod_2, nod_3, 0, ela2, 1, lok_A);
        LokalEle(nod_0, nod_1, nod_2, nod_3, 0, ele2, 1, lok_E);
                LokalMatp(nod_0, nod_1, nod_2, nod_3, 0, pie2, 1, lok_P);
        }

    for (m=0; m<4; m++)
```

```
    for (n=0; n<4; n++)
      for (k=0; k<3; k++)
        for(l=0; l<3; l++)
        {

         r = 0;
         while ( skladba_elementu[i][m] != indexove[r] )
                r++;

         s = 0;
         while ( skladba_elementu[i][n] != indexove[s] )
                s++;

         global_A.lok[r][s][k][l] += lok_A.lok[m][n][k][l];
//                                          global_E.lok[r][s][0][0] +=
lok_E.lok[m][n][0][0];
//                                          global_P.lokp[r][s][0][l] +=
lok_P.lokp[m][n][0][l];
        }

                        for (m=0; m<4; m++)
            for (n=0; n<4; n++)
                            for(l=0; l<3; l++)
                            {
                                    r = 0;
                                    while ( skladba_elementu[i][m] != indexove[r] )
                                            r++;

                                            s = 0;
                                            while ( skladba_elementu[i][n] != indexove[s] )
                                                    s++;

                                            global_P.lokp[r][s][0][l] +=
lok_P.lokp[m][n][0][l];

                                    }

            for (m=0; m<4; m++)
              for (n=0; n<4; n++)
              {
                  r = 0;
                                    while ( skladba_elementu[i][m] != indexove[r] )
                                            r++;

                                    s = 0;
                                    while ( skladba_elementu[i][n] != indexove[s] )
                                            s++;

                                    global_E.lok[r][s][0][0] += lok_E.lok[m][n][0][0];
                            }
```

```
 }

 printf ("\n   Globalni matice vytvorena ... \n");


//---------------------------------------------------------------------------
//      TESTOVACI VYPIS
//---------------------------------------------------------------------------
/*
        fw=fopen("Piezo", "w");

        for ( i = 0 ; i < poc_nod ; i++ )
        {
                for ( j = 0 ; j < poc_nod ; j ++ )
                        for ( k = 0 ; k < 3 ; k++ )
                                fprintf(fw, " %e", global_P.lokp[i][j][0][k]);
                fprintf(fw, "\n");
        }
 fclose(fw);



*/
//---------------------------------------------------------------------------
//      PREPIS P -> Pt
//---------------------------------------------------------------------------

        for ( i = 0 ; i < poc_nod ; i++ )
                for ( j = 0 ; j < poc_nod ; j++ )
                        for ( k = 0 ; k < 1 ; k++ )
                                for ( l = 0 ; l < 3 ; l++)
                                        global_Pt.lokp[j][i][l][k] = global_P.lokp[i][j][k][l];


//---------------------------------------------------------------------------
//      ZADANI OKRAJOVYCH PODMINEK
//---------------------------------------------------------------------------

                // UCHYCENI PEVNE

        for (i = 0 ; i < pocet_node_okp[0] ; i++ )
        {
                        for ( k = 0 ; k < poc_nod ; k++ )
                        {
                                if ( seznam_node_okp[0][i] == indexove[k] )
                                        break;
```

```
                }


        for ( j = 0 ; j < poc_nod ; j++ )
        {
                for ( l = 0 ; l < 3 ; l++ )
                        for ( m = 0 ; m < 3 ; m++ )
                        {
                                global_A.lok[k][j][l][m] = 0;
                        }
        }

        for ( j = 0 ; j < poc_nod ; j++ )
        {
                for ( l = 0 ; l < 3 ; l++ )
                        for ( m = 0 ; m < 1 ; m++ )
                        {
                                global_Pt.lokp[k][j][l][m] = 0;
                        }
        }

        for ( l = 0 ; l < 3 ; l++ )
        {
                global_A.lok[k][k][l][l] = 1.00;
        }

        for ( j = 0 ; j < 3 ; j ++ )
                right_side_elast[k*3 + j] = 0;
}

        // SILA

force = atof(argv[7]);

for (i = 0 ; i < pocet_node_okp[1] ; i++ )
{
                for ( k = 0 ; k < poc_nod ; k++ )
                {
                        if ( seznam_node_okp[1][i] == indexove[k] )
                                break;
                }

                right_side_elast[k*3 + 0] = force;  //zadani velikosti sily v ose x
}

        // UCHYCENI POHYBLIVE
/*
        for (i = 0 ; i < pocet_node_okp[1] ; i++ )
        {
```

```
                for ( k = 0 ; k < poc_nod ; k++ )
                {
                        if ( seznam_node_okp[1][i] == indexove[k] )
                                break;
                }



        for ( j = 0 ; j < poc_nod ; j++ )
        {
                for ( l = 0 ; l < 3 ; l++ )
                        for ( m = 0 ; m < 3 ; m++ )
                        {
                                global_A.lok[k][j][l][m] = 0;
                        }
        }

        for ( j = 0 ; j < poc_nod ; j++ )
        {
                for ( l = 0 ; l < 3 ; l++ )
                        for ( m = 0 ; m < 1 ; m++ )
                        {
                                global_Pt.lokp[k][j][l][m] = 0;
                        }
        }

        for ( l = 0 ; l < 3 ; l++ )
        {
                global_A.lok[k][k][l][l] = 1.00;
        }

        for ( j = 0 ; j < 3 ; j ++ )
                right_side_elast[k*3 + j] = 0;
}

*/
                // POTENCIAL NULA

        for (i = 0 ; i < pocet_node_okp[2] ; i++ )
{
                for ( k = 0 ; k < poc_nod ; k++ )
                {
                        if ( seznam_node_okp[2][i] == indexove[k] )
                                break;
                }

        for ( j = 0 ; j < poc_nod ; j++ )
        {
                global_E.lok[k][j][0][0] = 0;
                for ( l = 0 ; l < 3 ; l++ )
```

```
                        global_P.lokp[k][j][0][l] = 0;
                }
                global_E.lok[k][k][0][0] = 1.00;

                right_side_elektro[k] = 0;              //zadani velikosti potencialu

        }

                // POTENCIAL NENULA

        potential = atof(argv[6]);

                for (i = 0 ; i < pocet_node_okp[3] ; i++ )
        {
                        for ( k = 0 ; k < poc_nod ; k++ )
                        {
                                if ( seznam_node_okp[3][i] == indexove[k] )
                                        break;
                        }

                for ( j = 0 ; j < poc_nod ; j++ )
                {
                        global_E.lok[k][j][0][0] = 0;
                        for ( l = 0 ; l < 3 ; l++ )
                                global_P.lokp[k][j][0][l] = 0;
                }
                global_E.lok[k][k][0][0] = 1.00;

                right_side_elektro[k] = potential;  //zadani velikosti potencialu

        }

        printf ("\n   Okrajove podminky zadany ... \n");

//-------------------------------------------------------------------------
//      ZAPIS DO *.m SOUBORU
//-------------------------------------------------------------------------


//-------------------------------------------------------------------------
//      NASOBENI PtEP, SECTENI K + PtEP
//-------------------------------------------------------------------------



        for (i=0; i<poc_nod; i++)
                for (j=0; j<3; j++)
                        for (k=0; k<poc_nod; k++)
                                for (l=0; l<3; l++)
                                {
```

```
        A[(12*i + 4*j)*poc_nod+ 3*k + l] = global_A.lok[i][k][j][l];
                                }



         for (i=0; i<poc_nod; i++)
                for (j=0; j<3; j++)
                        for (k=0; k<poc_nod; k++)
                                for (l=0; l<1; l++)
                                {
        A[(12*i + 4*j + 3) * poc_nod + k + l] = global_Pt.lokp[i][k][j][l];
                                }

         for (i=0; i<poc_nod; i++)
                for (j=0; j<1; j++)
                        for (k=0; k<poc_nod; k++)
                                for (l=0; l<3; l++)
                                {
                                        A[12 * poc_nod * poc_nod + 4 * i * poc_nod + 3 * k + l]
= global_P.lokp[i][k][j][l];
                                }


         for (i=0; i<poc_nod; i++)
                for (j=0; j<1; j++)
                        for (k=0; k<poc_nod; k++)
                                for (l=0; l<1; l++)
                                {
        A[3*poc_nod*3*poc_nod + 3*poc_nod*poc_nod + 3* poc_nod*(i+1) + i*poc_nod +
k] = global_E.lok[i][k][j][l];
                                }

         for ( k = 0 ; k < 3*poc_nod; k++ )
                b[k] = right_side_elast[k];

         for ( k = 0 ; k < poc_nod; k++ )
                b[3*poc_nod + k] = right_side_elektro[k];

//---------------------------------------------------------------------------
//      ZAPIS C DO SOUBORU C.m
//---------------------------------------------------------------------------
/*

         fw=fopen("C.m", "w");
         for (i=0; i< 4 * poc_nod; i++)
                for (j=0; j< 4 * poc_nod; j++)
                        fprintf(fw, "C(%d,%d)= %e ;\n", i+1, j+1, A[i* 4 * poc_nod + j]);

         for (i=0; i< 3 * poc_nod; i++)
                for (j=0; j< 3 * poc_nod; j++)
```

```
                        fprintf(fw, "A(%d,%d)= %e ;\n", i+1, j+1, A[i* 4 * poc_nod + j]);

  fclose(fw);

*/
//---------------------------------------------------------------------------


        LWORK = 9 * poc_nod;
        LDA = 4*poc_nod;


  dgetrf_( &LDA, &LDA, A, &LDA, ipiv, &INFO );

  printf("\n INFO=%d \n", INFO);

        dgetrs_( &TRANS, &LDA, &NRHS, A, &LDA, ipiv, b, &LDA, &INFO);

        printf("\n INFO=%d \n", INFO);


//---------------------------------------------------------------------------
//                              ZAPIS VYSLEDKU
//---------------------------------------------------------------------------

        fw=fopen("posuny", "w");

        for ( k = 0 ; k < 3*poc_nod ; k++ )
                fprintf (fw, "%e \n", b[k]);

        fclose(fw);

        fw=fopen("potencialy", "w");

        for ( k = 3*poc_nod ; k < 4*poc_nod ; k++ )
                fprintf (fw, "%e \n", b[k]);

        fclose(fw);


}
```

Zdrojový kód programu pro výpočet elektrického pole a indukce:

```
#define MAIN
#include "big-head.h"

#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>

main(int argc, char* argv[])
{

        if (argc != 5)
        {
                fprintf(stderr," \n Chybne argumenty spusteni!\n");
                fprintf(stderr," \n jmeno souboru site  \n soubor s materialem 1 \n soubor s
materialem 2 \n jmeno souboru potencialu \n \n");
   return(0);
        }



 int poc_nod, poc_elem;
 int *okp_locate, *pocet_node_okp, **seznam_node_okp, *surface_locate;
 int i, j, k, l, m, n, r, s, p, q;
 int *indexove, **skladba_elementu, fix[10];
 double **pom, **sour_nody, *potencial_okp;
 double *nod_0, *nod_1, *nod_2, *nod_3, *right_side;
        double *volumes;
 FILE *fw, *f_pot;

        double **ela, **pie, **ele1, **ele2;
        double **grad_lok, **flux_lok;


        double *potencial;
        double **gradient_u, **flux_density, **teziste_elem;

        double energy_sum;

 double *bod_1, *bod_2;
 int rozliseni;
 double **sour_line;
 double **gradient_line;

//-------------------------------------------------------------------------
//      ALOKACE PAMETI PRO LOKALNI PROMENNE
```

```
//---------------------------------------------------------------------------

  nod_0 = (double *) malloc(3 * sizeof(double));
  nod_1 = (double *) malloc(3 * sizeof(double));
  nod_2 = (double *) malloc(3 * sizeof(double));
  nod_3 = (double *) malloc(3 * sizeof(double));

  potencial_okp = (double *) malloc(2 * sizeof(double));
  okp_locate  = (int *) malloc(2 * sizeof(int));
  surface_locate  = (int *) malloc(2 * sizeof(int));
  pocet_node_okp  = (int *) malloc(2 * sizeof(int));

        surface_locate  = (int *) malloc(2 * sizeof(int));

        grad_lok = (double **)malloc(3*sizeof(double *));
        for (j=0; j<3; j++)
          grad_lok[j]=(double *)malloc(1*sizeof(double));

        flux_lok = (double **)malloc(3*sizeof(double *));
        for (j=0; j<3; j++)
          flux_lok[j]=(double *)malloc(1*sizeof(double));

  bod_1 = (double *) malloc(3 * sizeof(double));
        bod_2 = (double *) malloc(3 * sizeof(double));



//---------------------------------------------------------------------------
//      ALOKACE PAMETI PRO GLOBALNI PROMENNE
//---------------------------------------------------------------------------

  ela=(double **)malloc(6*sizeof(double *));
  for (j=0; j<6; j++)
    ela[j]=(double *)malloc(6*sizeof(double));

        pie=(double **)malloc(3*sizeof(double *));
        for (j=0; j<3; j++)
          pie[j]=(double *)malloc(6*sizeof(double));

        ele1=(double **)malloc(3*sizeof(double *));
        for (j=0; j<3; j++)
          ele1[j]=(double *)malloc(3*sizeof(double));

        ele2=(double **)malloc(3*sizeof(double *));
        for (j=0; j<3; j++)
          ele2[j]=(double *)malloc(3*sizeof(double));


//---------------------------------------------------------------------------
//      NACTENI MATERIALU
```

```
//---------------------------------------------------------------------------

 if ( (readstm("kanal.stm", argv[2], 0, ela, pie, ele1)) == 0)
        {
                printf(" \n   Chybne zadane jmeno souboru materialovych parametru! \n");
                return(0);
        }

        if ( (readstm("kanal.stm", argv[3], 0, ela, pie, ele2)) == 0)
        {
                printf(" \n   Chybne zadane jmeno souboru materialovych parametru! \n");
                return(0);
        }

/*
        for ( i = 0 ; i < 3 ; i ++ )
                for ( j = 0 ; j < 3 ; j ++ )
                {
                        ele1[i][j] *= 8.859e-12;
                        ele2[i][j] *= 8.859e-12;
                }
*/

//---------------------------------------------------------------------------
//      NACTENI SITE
//---------------------------------------------------------------------------


 if ( (indexove = indexy_nodu(argv[1])) == 0)
 {
          printf(" \n   Chybne zadane jmeno souboru site! \n");
  return(0);
 }
 pom = souradnice_nodu(argv[1]);
 //pocet nodu
        poc_nod = pocet_nodu(argv[1]);

 //souradnice nodu podle cisel v *.msh
 sour_nody = serad_nody(pom, indexove, poc_nod);
 //pocet elementu site
 poc_elem = pocet_elementu(argv[1]);
 // nody tvorici elementy
 skladba_elementu = nody_elementu(argv[1]);

 printf ("\n\n   Sit nactena ... \n");

//---------------------------------------------------------------------------
//      NACTENI BODU MEZI KTERYMI SE BUDE ZJISTOVAT
//---------------------------------------------------------------------------
```

```
  printf("\n Zadej x-ovou souradnici prvniho bodu: ");
  scanf("%lf", &bod_1[0]);

  printf("\n Zadej y-ovou souradnici prvniho bodu: ");
  scanf("%lf", &bod_1[1]);

  printf("\n Zadej z-ovou souradnici prvniho bodu: ");
  scanf("%lf", &bod_1[2]);

  printf("\n Zadej x-ovou souradnici druheho bodu: ");
  scanf("%lf", &bod_2[0]);

  printf("\n Zadej y-ovou souradnici druheho bodu: ");
  scanf("%lf", &bod_2[1]);

  printf("\n Zadej z-ovou souradnici prvniho bodu: ");
  scanf("%lf", &bod_2[2]);

  printf("\n Zadej rozliseni: ");
  scanf("%d", &rozliseni);


//---------------------------------------------------------------------------
//      LOKACE MATERIALU
//---------------------------------------------------------------------------

  printf(" Zadej entitu s materialem 1: ");
  scanf("%d", &surface_locate[0]);

  printf(" Zadej entitu s materialem 2: ");
  scanf("%d", &surface_locate[1]);


//---------------------------------------------------------------------------
//      ALOKACE PAMETI PRO GLOBALNI PROMENNE
//---------------------------------------------------------------------------

        seznam_node_okp = (int **) malloc(2*sizeof(int *));
        for ( i = 0 ; i < 2 ; i++)
                seznam_node_okp[i] = (int *) malloc(poc_nod*sizeof(int));

        gradient_u = (double **) malloc(poc_elem * sizeof(double *));
        for ( i = 0 ; i < poc_elem ; i++ )
                gradient_u[i] = (double *) malloc(3 * sizeof(double));

        flux_density = (double **) malloc(poc_elem * sizeof(double *));
        for ( i = 0 ; i < poc_elem ; i++ )
                flux_density[i] = (double *) malloc(3 * sizeof(double));

        teziste_elem = (double **) malloc(poc_elem * sizeof(double *));
```

```
        for ( i = 0 ; i < poc_elem ; i++ )
                teziste_elem[i] = (double *) malloc(3 * sizeof(double));

        potencial = (double *) malloc(poc_nod * sizeof(double));

        volumes = (double *) malloc(poc_elem * sizeof(double));


 sour_line = (double **) malloc((rozliseni+1) * sizeof(double *));
 for ( i = 0 ; i < rozliseni+1; i++ )
        sour_line[i] = (double *) malloc(3 * sizeof(double));

        gradient_line = (double **) malloc((rozliseni+1) * sizeof(double *));
        for ( i = 0 ; i < rozliseni+1; i++ )
                gradient_line[i] = (double *) malloc(3 * sizeof(double));


//-------------------------------------------------------------------------
//      NACTENI POTENCIALU ZE SOUBORU
//-------------------------------------------------------------------------


        if ((f_pot = fopen(argv[4], "r")) == NULL)
        {
                printf("Neplatne jmeno souboru potencialu \n");
                return(0);
        }

        for ( i = 0 ; i < poc_nod ; i++)
                fscanf(f_pot, "%lf", &potencial[i]);

 printf ("\n\n   Potencial nacten ... \n");

//-------------------------------------------------------------------------
//      VYPOCET EL. POLE
//-------------------------------------------------------------------------
        for ( i=0; i<poc_elem; i++ )
 {

        for ( j = 0; j<3; j++)
  {
        nod_0[j] = sour_nody[skladba_elementu[i][0]][j];
           nod_1[j] = sour_nody[skladba_elementu[i][1]][j];
         nod_2[j] = sour_nody[skladba_elementu[i][2]][j];
         nod_3[j] = sour_nody[skladba_elementu[i][3]][j];
                }

        r = 0;
 while ( skladba_elementu[i][0] != indexove[r] )
        r++;
```

```
 s = 0;
 while ( skladba_elementu[i][1] != indexove[s] )
        s++;

 p = 0;
 while ( skladba_elementu[i][2] != indexove[p] )
        p++;

        q = 0;
 while ( skladba_elementu[i][3] != indexove[q] )
        q++;

        grad_u_3D (nod_0, nod_1, nod_2, nod_3, potencial[r], potencial[s], potencial[p],
potencial[q], gradient_u[i]);

        teziste_elementu_3D(nod_0, nod_1, nod_2, nod_3, teziste_elem[i]);

        volumes[i] = objem_tetrahedron(nod_0, nod_1, nod_2, nod_3);

        for ( k = 0 ; k < 3 ; k ++ )
                grad_lok[k][0] = -gradient_u[i][k];

        if (skladba_elementu[i][5] == surface_locate[0] )
                nasobmat(ele1, grad_lok, flux_lok, 3, 3, 1);

        if (skladba_elementu[i][5] == surface_locate[1] )
                nasobmat(ele2, grad_lok, flux_lok, 3, 3, 1);

        for ( k = 0 ; k < 3 ; k ++ )
                flux_density[i][k] = flux_lok[k][0];

        }

 printf ("\n   Globalni matice vytvorena ... \n");


//-------------------------------------------------------------------------
//      VYPOCET ENERGII
//-------------------------------------------------------------------------

        energy_sum = 0;

        for ( i = 0 ; i < poc_elem ; i++ )
                energy_sum += 0.5 * vector_norm(gradient_u[i]) *
vector_norm(flux_density[i]) * volumes[i];

//-------------------------------------------------------------------------
//      VYPOCET PODEL USECKY
//-------------------------------------------------------------------------
```

```
  vektor_on_line_3D(bod_1, bod_2, teziste_elem, gradient_u, poc_elem, rozliseni,sour_line,
gradient_line);

                                                                    //-----------------------------------------
----------------------------------  //      ZAPIS DO *.dat SOUBORU PRO GNUPLOT
//-------------------------------------------------------------------------------
                                                                      fw = fopen("inten_line.dat", "w");
fprintf(fw, "# soubor site:\t%s\n", argv[1]);




                                                                                fprintf(fw,
"# soubor potencialu:\t%s\n", argv[4]);


                                                                                fprintf(fw,
"# rozliseni:\t%d\n", rozliseni);


                                                                                fprintf(fw,
"# x_1 = %lf ; y_1 = %lf ; z_1 = %lf \n", bod_1[0], bod_1[1], bod_1[2]);


                                                                                fprintf(fw,
"# x_2 = %lf ; y_2 = %lf ; z_2 = %lf \n\n", bod_2[0], bod_2[1], bod_2[2]);


                                                                                for ( i = 0 ; i
<= rozliseni ; i++ )


                                                                                fprintf(fw,
"%e\t%e\t%e\t%e\n", sour_line[i][0], gradient_line[i][0], gradient_line[i][1],
gradient_line[i][2]);


                                                                                fclose(fw);
```

```
//------------------------------------------------------------------------
//      ZAPIS DO *.pos SOUBORU
//------------------------------------------------------------------------

/*      printf (" %d \n", poc_nod);
*/

 fw=fopen("inten_x.pos", "w");
        fprintf(fw, " View \" grad(ux) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
  fprintf(fw, "VP(%lf, %lf, %lf){%lf, 0, 0};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        -gradient_u[i][0]);
        fprintf(fw, "};");
        fclose(fw);

 fw=fopen("inten_y.pos", "w");
        fprintf(fw, " View \" grad(uy) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
                fprintf(fw, "VP(%lf, %lf, %lf){0, %lf, 0};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        -gradient_u[i][1]);
        fprintf(fw, "};");
 fclose(fw);

 fw=fopen("inten_z.pos", "w");
        fprintf(fw, " View \" grad(uz) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
                fprintf(fw, "VP(%lf, %lf, %lf){0, O, %lf};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        -gradient_u[i][2]);
        fprintf(fw, "};");
 fclose(fw);


        fw=fopen("gradient.pos", "w");
        fprintf(fw, " View \" grad(u) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
                fprintf(fw, "VP(%lf, %lf, %lf){%lf, %lf, %lf};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        -gradient_u[i][0], -gradient_u[i][1],
-gradient_u[i][2]);
        fprintf(fw, "};");
 fclose(fw);
```

```
/*
        fw=fopen("grad_abs.pos", "w");
        fprintf(fw, " View \" abs(grad(u)) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
                fprintf(fw, "ST(%lf, %lf, 0,  %lf, %lf, 0,  %lf, %lf, 0){%lf, %lf, %lf};\n",

        sour_nody[skladba_elementu[i][0]][0],

        sour_nody[skladba_elementu[i][0]][1],

        sour_nody[skladba_elementu[i][1]][0],

        sour_nody[skladba_elementu[i][1]][1],

        sour_nody[skladba_elementu[i][2]][0],

        sour_nody[skladba_elementu[i][2]][1],

        sqrt(gradient_u[i][0]*gradient_u[i][0]+gradient_u[i][1]*gradient_u[i][1]),

        sqrt(gradient_u[i][0]*gradient_u[i][0]+gradient_u[i][1]*gradient_u[i][1]),

        sqrt(gradient_u[i][0]*gradient_u[i][0]+gradient_u[i][1]*gradient_u[i][1]));
        fprintf(fw, "};");
 fclose(fw);
*/

 fw=fopen("flux_x.pos", "w");
        fprintf(fw, " View \" flux(Dx) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
  fprintf(fw, "VP(%lf, %lf, %lf){%lf, 0, 0};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        flux_density[i][0]);
        fprintf(fw, "};");
        fclose(fw);

 fw=fopen("flux_y.pos", "w");
        fprintf(fw, " View \" flux(Dy) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
  fprintf(fw, "VP(%lf, %lf, %lf){0, %lf, 0};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        flux_density[i][1]);
        fprintf(fw, "};");
        fclose(fw);

 fw=fopen("flux_z.pos", "w");
        fprintf(fw, " View \" flux(Dz) \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
```

```
   fprintf(fw, "VP(%lf, %lf, %lf){0, 0, %lf};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        flux_density[i][2]);
        fprintf(fw, "};");
        fclose(fw);



        fw=fopen("flux.pos", "w");
        fprintf(fw, " View \" flux D \" { \n");
        for ( i = 0 ; i < poc_elem; i++ )
                fprintf(fw, "VP(%lf, %lf, %lf){%e, %e, %e};\n",
                                                        teziste_elem[i][0],
teziste_elem[i][1], teziste_elem[i][2],
                                                        flux_density[i][0],
flux_density[i][1], flux_density[i][2]);
        fprintf(fw, "};");
  fclose(fw);

        printf ("\n   Soubory *.pos vytvoreny ... \n");


        printf ("\n   Energie je %e \n", energy_sum);


        printf( " \n " );


}
```

Zdrojový kód programu pro výpočet elastických napětí a deformací:

```
#define MAIN
#include "big-head.h"

#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>


main(int argc, char* argv[])
{

        if (argc != 9)
        {
                fprintf(stderr," \n Chybne argumenty spusteni!\n");
                fprintf(stderr," \n jmeno souboru site  \n jmeno souboru vektoru \n poradi
vektoru \n popisek .pos souboru \n soubor s materialem 1 \n soubor s materialem 2 \n lokace
materialu 1 \n lokace materialu 2 \n \n");
                return(0);
        }


        int *indexove;
        double **pom;
        int poc_nod;
        double **sour_nody;
        int poc_elem;
        int **skladba_elementu;
        int i, j, k, l, r, s, p, q;
        int poradi;
        double **displacements;
        double **strain_element;
        double **stress_element;
        double **force_components;
        double **force_normal;
        double **elast_1, **elast_2, **ele, **pie;
 double *nod_0, *nod_1, *nod_2, *nod_3;
        double **teziste_elem;

        int *surface_locate;

 char *soubor_strain;
 char *soubor_stress;

 double *bod_1, *bod_2, *bod_3;
 int rozliseni;
 double **sour_line, **stress_line, **strain_line;
```

```
        FILE *fvekt, *fstrain;

//-------------------------------------------------------------------------
//      ALOKACE PAMETI
//-------------------------------------------------------------------------

  nod_0 = (double *) malloc(3 * sizeof(double));
  nod_1 = (double *) malloc(3 * sizeof(double));
  nod_2 = (double *) malloc(3 * sizeof(double));
  nod_3 = (double *) malloc(3 * sizeof(double));

  surface_locate = (int *) malloc(2 * sizeof(int));

  soubor_strain = (char *) malloc(100 * sizeof(char));
  soubor_stress = (char *) malloc(100 * sizeof(char));

  elast_1 = (double **)malloc(6*sizeof(double *));
        for (j = 0 ; j < 6 ; j++)
                elast_1[j] = (double *)malloc(6*sizeof(double));

  elast_2 = (double **)malloc(6*sizeof(double *));
        for (j = 0 ; j < 6 ; j++)
                elast_2[j] = (double *)malloc(6*sizeof(double));

  ele = (double **)malloc(3*sizeof(double *));
        for (j = 0 ; j < 3 ; j++)
                ele[j] = (double *)malloc(3*sizeof(double));

  pie = (double **)malloc(3*sizeof(double *));
        for (j = 0 ; j < 3 ; j++)
                pie[j] = (double *)malloc(6*sizeof(double));

  bod_1 = (double *) malloc(3 * sizeof(double));
        bod_2 = (double *) malloc(3 * sizeof(double));
        bod_3 = (double *) malloc(3 * sizeof(double));


//-------------------------------------------------------------------------
//      NACTENI MATERIALU
//-------------------------------------------------------------------------

        if ( (readstm(argv[5], argv[5], 0, elast_1, pie, ele)) == 0)
        {
                printf(" \n   Chybne zadane jmeno souboru materialovych parametru! \n");
                return(0);
        }

  if ( (readstm(argv[6], argv[6], 0, elast_2, pie, ele)) == 0)
        {
```

```
            printf(" \n   Chybne zadane jmeno souboru materialovych parametru! \n");
            return(0);
        }


//---------------------------------------------------------------------------
//      NACTENI SITE
//---------------------------------------------------------------------------


 if ( (indexove = indexy_nodu(argv[1])) == 0)
 {
         printf(" \n   Chybne zadane jmeno souboru site! \n");
  return(0);
 }
 pom = souradnice_nodu(argv[1]);
 //pocet nodu
 poc_nod = pocet_nodu(argv[1]);

        //souradnice nodu podle cisel v *.msh
 sour_nody = serad_nody(pom, indexove, poc_nod);
 //pocet elementu site
 poc_elem = pocet_elementu(argv[1]);
 // nody tvorici elementy
 skladba_elementu = nody_elementu(argv[1]);

 printf ("\n\n   Sit nactena ... \n");

//---------------------------------------------------------------------------
//      NACTENI BODU MEZI KTERYMI SE BUDE ZJISTOVAT
//---------------------------------------------------------------------------

 printf("\n Zadej x-ovou souradnici prvniho bodu: ");
        scanf("%lf", &bod_1[0]);

        printf("\n Zadej y-ovou souradnici prvniho bodu: ");
        scanf("%lf", &bod_1[1]);

        printf("\n Zadej z-ovou souradnici prvniho bodu: ");
        scanf("%lf", &bod_1[2]);

        printf("\n Zadej x-ovou souradnici druheho bodu: ");
        scanf("%lf", &bod_2[0]);

        printf("\n Zadej y-ovou souradnici druheho bodu: ");
        scanf("%lf", &bod_2[1]);

        printf("\n Zadej z-ovou souradnici druheho bodu: ");
        scanf("%lf", &bod_2[2]);
```

```
        printf("\n Zadej x-ovou souradnici tretiho bodu: ");
        scanf("%lf", &bod_3[0]);

        printf("\n Zadej y-ovou souradnici tretiho bodu: ");
        scanf("%lf", &bod_3[1]);

        printf("\n Zadej z-ovou souradnici tretiho bodu: ");
        scanf("%lf", &bod_3[2]);



        printf("\n Zadej rozliseni: ");
        scanf("%d", &rozliseni);



//---------------------------------------------------------------------------
//      ALOKACE PAMETI
//---------------------------------------------------------------------------

        displacements = (double **) malloc(poc_nod * sizeof(double *));
  for (i = 0; i < poc_nod; i++)
                displacements[i] = (double *) malloc( 3 * sizeof(double));

        strain_element = (double **) malloc(poc_elem * sizeof(double *));
  for (i = 0; i < poc_elem; i++)
                strain_element[i] = (double *) malloc( 6 * sizeof(double));

        stress_element = (double **) malloc(poc_elem * sizeof(double *));
  for (i = 0; i < poc_elem; i++)
                stress_element[i] = (double *) malloc( 6 * sizeof(double));

  teziste_elem = (double **) malloc(poc_elem * sizeof(double *));
  for ( i = 0 ; i < poc_elem ; i++ )
          teziste_elem[i] = (double *) malloc(3 * sizeof(double));

  sour_line = (double **) malloc((rozliseni+1) * sizeof(double *));
  for ( i = 0 ; i < rozliseni+1; i++ )
   sour_line[i] = (double *) malloc(3 * sizeof(double));

  stress_line = (double **) malloc((rozliseni+1) * sizeof(double *));
  for ( i = 0 ; i < rozliseni+1; i++ )
          stress_line[i] = (double *) malloc(6 * sizeof(double));

        strain_line = (double **) malloc((rozliseni+1) * sizeof(double *));
  for ( i = 0 ; i < rozliseni+1; i++ )
          strain_line[i] = (double *) malloc(6 * sizeof(double));
```

```
 force_components = (double **) malloc((rozliseni+1) * sizeof(double *));
 for ( i = 0 ; i < rozliseni+1; i++ )
          force_components[i] = (double *) malloc(3 * sizeof(double));

        force_normal = (double **) malloc((rozliseni+1) * sizeof(double *));
 for ( i = 0 ; i < rozliseni+1; i++ )
          force_normal[i] = (double *) malloc(3 * sizeof(double));

//---------------------------------------------------------------------------
//      NACTENI PORADI
//---------------------------------------------------------------------------

        poradi = atoi(argv[3]);

//---------------------------------------------------------------------------
//      NACTENI LOKACE MATERIALU
//---------------------------------------------------------------------------

        surface_locate[0] = atoi(argv[7]);
        surface_locate[1] = atoi(argv[8]);


//---------------------------------------------------------------------------
//      NACTENI VEKTORU POSUNU
//---------------------------------------------------------------------------

 if ( (fvekt=fopen(argv[2], "r")) == NULL )
        {
                printf(" \n   Chybne zadane jmeno souboru kmitu! \n");
                return(0);
        }

        for ( k = 0 ; k < poradi+1 ; k++)
                for ( i = 0 ; i < poc_nod ; i++)
                for ( j = 0 ; j < 3 ; j++)
                fscanf(fvekt, "%lf", &displacements[i][j]);

 printf ("\n\n   Posuny nacteny ... \n");

//---------------------------------------------------------------------------
//      VYPOCET STRAINU
//---------------------------------------------------------------------------

 for ( i=0; i<poc_elem; i++ )
 {
        for ( j = 0; j<3; j++)
                {
                        nod_0[j] = sour_nody[skladba_elementu[i][0]][j];
                        nod_1[j] = sour_nody[skladba_elementu[i][1]][j];
                        nod_2[j] = sour_nody[skladba_elementu[i][2]][j];
```

```
                nod_3[j] = sour_nody[skladba_elementu[i][3]][j];
        }

        r = 0;
        while ( skladba_elementu[i][0] != indexove[r] )
          r++;

        s = 0;
        while ( skladba_elementu[i][1] != indexove[s] )
          s++;

        p = 0;
        while ( skladba_elementu[i][2] != indexove[p] )
          p++;

        q = 0;
        while ( skladba_elementu[i][3] != indexove[q] )
          q++;


    strain_tetrahedron (nod_0, nod_1, nod_2, nod_3, displacements[r], displacements[s],
displacements[p], displacements[q], strain_element[i]);

  teziste_elementu_3D(nod_0, nod_1, nod_2, nod_3, teziste_elem[i]);

    }


//-------------------------------------------------------------------------
//      VYPOCET  STRESSU
//-------------------------------------------------------------------------

    for ( i = 0 ; i < poc_elem ; i++ )
    {
        if ( skladba_elementu[i][5] == surface_locate[0] )
            stress_tetrahedron(strain_element[i], elast_1, stress_element[i]);

        if ( skladba_elementu[i][5] == surface_locate[1] )
            stress_tetrahedron(strain_element[i], elast_2, stress_element[i]);
    }


//-------------------------------------------------------------------------
//      VYPOCET  STRESSU PODEL USECKY
//-------------------------------------------------------------------------


    tensor_on_line_3D(bod_1, bod_2, teziste_elem, stress_element, poc_elem, rozliseni,
sour_line, stress_line);
```

```
//---------------------------------------------------------------------------
//        VYPOCET  STRAINU PODEL USECKY
//---------------------------------------------------------------------------


        tensor_on_line_3D(bod_1, bod_2, teziste_elem, strain_element, poc_elem, rozliseni,
sour_line, strain_line);



//---------------------------------------------------------------------------
//        VYPOCET  KOMPONENT STRESSU NA PLOCHU
//---------------------------------------------------------------------------

        stress_components_on_surface(bod_1, bod_2, bod_3, rozliseni, stress_line,
force_components);

//---------------------------------------------------------------------------
//        VYPOCET  NORMALOVE SILY NA PLOCHU
//---------------------------------------------------------------------------


        plumb_comp_on_line_3D(bod_1, bod_2, bod_3, force_components, rozliseni,
force_normal);



//---------------------------------------------------------------------------
//        ZAPIS STRAINU DO SOUBORU
//---------------------------------------------------------------------------


        for ( j = 0 ; j < 6 ; j++ )
        {

        strcpy(soubor_strain, "strain_");

        switch (j)
        {
                case 0: strcat(soubor_strain, "11");   break;
                case 1: strcat(soubor_strain, "12");   break;
                case 2: strcat(soubor_strain, "13");   break;
                case 3: strcat(soubor_strain, "22");   break;
                case 4: strcat(soubor_strain, "23");   break;
                case 5: strcat(soubor_strain, "33");   break;
        }

        strcat(soubor_strain, ".pos");
```

```
 fstrain=fopen(soubor_strain, "w");

 fprintf(fstrain, " View \" %s \" { \n", soubor_strain);

        for ( i = 0 ; i < poc_elem; i++ )
        fprintf(fstrain, "SS(%lf, %lf, %lf,  %lf, %lf, %lf,  %lf, %lf, %lf,  %lf, %lf, %lf){%e,
%e, %e, %e};\n",
                sour_nody[skladba_elementu[i][0]][0],
                        sour_nody[skladba_elementu[i][0]][1],
                        sour_nody[skladba_elementu[i][0]][2],
                        sour_nody[skladba_elementu[i][1]][0],
                        sour_nody[skladba_elementu[i][1]][1],
                        sour_nody[skladba_elementu[i][1]][2],
                        sour_nody[skladba_elementu[i][2]][0],
        sour_nody[skladba_elementu[i][2]][1],
        sour_nody[skladba_elementu[i][2]][2],
                        sour_nody[skladba_elementu[i][3]][0],
        sour_nody[skladba_elementu[i][3]][1],
        sour_nody[skladba_elementu[i][3]][2],
        strain_element[i][j],
        strain_element[i][j],
        strain_element[i][j],
        strain_element[i][j]);
        fprintf(fstrain, "};\n");
        fprintf(fstrain, "Plugin(Skin).Run;\n");
        fprintf(fstrain, "View.Visible=0;");
                fclose(fstrain);

        }


 fstrain = fopen("strain", "w");

        for ( i = 0 ; i < poc_elem ; i++ )
                for ( j = 0 ; j < 6 ; j++ )
                        fprintf(fstrain, "%e \n", strain_element[i][j]);

        fclose(fstrain);



        fstrain = fopen("strain_line.dat", "w");

        fprintf(fstrain, "# soubor site:\t%s\n", argv[1]);
        fprintf(fstrain, "# soubor vektoru:\t%s\n", argv[2]);
        fprintf(fstrain, "# rozliseni:\t%d\n", rozliseni);
        fprintf(fstrain, "# souradnice\t 11-slozka\t 12-slozka\t 13-slozka\t 22-slozka slozka\t
23-slozka\t 33-slozka \n");
        fprintf(fstrain, "# x_1 = %lf ; y_1 = %lf ; z_1 = %lf \n", bod_1[0], bod_1[1],
bod_1[2]);
```

```
        fprintf(fstrain, "# x_2 = %lf ; y_2 = %lf ; z_2 = %lf \n", bod_2[0], bod_2[1],
bod_2[2]);

        for ( i = 0 ; i <= rozliseni ; i++ )
        {
                fprintf(fstrain, "%e\t%e\t%e\t%e\t%e\t%e\t%e\n", sour_line[i][0],
strain_line[i][0], strain_line[i][1], strain_line[i][2], strain_line[i][3], strain_line[i][4],
strain_line[i][5]);
        }

        fclose(fstrain);



//---------------------------------------------------------------------------
//      ZAPIS STRESSU DO SOUBORU
//---------------------------------------------------------------------------


        for ( j = 0 ; j < 6 ; j++ )
        {

        strcpy(soubor_stress, "stress_");

        switch (j)
        {
                case 0: strcat(soubor_stress, "11");    break;
                case 1: strcat(soubor_stress, "12");    break;
                case 2: strcat(soubor_stress, "13");    break;
                case 3: strcat(soubor_stress, "22");    break;
                case 4: strcat(soubor_stress, "23");    break;
                case 5: strcat(soubor_stress, "33");    break;
        }

        strcat(soubor_stress, ".pos");

 fstrain=fopen(soubor_stress, "w");

 fprintf(fstrain, " View \" %s \" { \n", soubor_stress);

        for ( i = 0 ; i < poc_elem; i++ )
        fprintf(fstrain, "SS(%lf, %lf, %lf,  %lf, %lf, %lf,  %lf, %lf, %lf,  %lf, %lf, %lf){%e,
%e, %e, %e};\n",
                sour_nody[skladba_elementu[i][0]][0],
                        sour_nody[skladba_elementu[i][0]][1],
                        sour_nody[skladba_elementu[i][0]][2],
                        sour_nody[skladba_elementu[i][1]][0],
                        sour_nody[skladba_elementu[i][1]][1],
                        sour_nody[skladba_elementu[i][1]][2],
```

```
                        sour_nody[skladba_elementu[i][2]][0],
        sour_nody[skladba_elementu[i][2]][1],
        sour_nody[skladba_elementu[i][2]][2],
                        sour_nody[skladba_elementu[i][3]][0],
        sour_nody[skladba_elementu[i][3]][1],
        sour_nody[skladba_elementu[i][3]][2],
        stress_element[i][j],
        stress_element[i][j],
        stress_element[i][j],
        stress_element[i][j]);
        fprintf(fstrain, "};\n");
        fprintf(fstrain, "Plugin(Skin).Run;\n");
        fprintf(fstrain, "View.Visible=0;");
                fclose(fstrain);

        }


 fstrain = fopen("stress", "w");

        for ( i = 0 ; i < poc_elem ; i++ )
                for ( j = 0 ; j < 6 ; j++ )
                        fprintf(fstrain, "%e \n", stress_element[i][j]);

        fclose(fstrain);


//---------------------------------------------------------------------------
//      ZAPIS SILY DO SOUBORU
//---------------------------------------------------------------------------

        fstrain= fopen ("force.dat", "w");

        fprintf(fstrain, "# soubor site:\t%s\n", argv[1]);
        fprintf(fstrain, "# soubor vektoru:\t%s\n", argv[2]);
        fprintf(fstrain, "# rozliseni:\t%d\n", rozliseni);
        fprintf(fstrain, "# souradnice\t x-slozka\t y-slozka\t z-slozka\t normalova slozka\n");
        fprintf(fstrain, "# x_1 = %lf ; y_1 = %lf ; z_1 = %lf \n", bod_1[0], bod_1[1],
bod_1[2]);
        fprintf(fstrain, "# x_2 = %lf ; y_2 = %lf ; z_2 = %lf \n", bod_2[0], bod_2[1],
bod_2[2]);
        fprintf(fstrain, "# x_3 = %lf ; y_3 = %lf ; z_3 = %lf \n\n", bod_3[0], bod_3[1],
bod_3[2]);

        for ( i = 0 ; i <= rozliseni ; i++ )
        {
                fprintf(fstrain, "%e\t%e\t%e\t%e\t%e\n", sour_line[i][0],
force_components[i][0], force_components[i][1], force_components[i][2],
force_normal[i][0]);
        }
```

```
        fclose(fstrain);

        return(1);
}
```