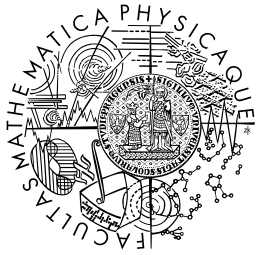


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Radek Vavřička

Aplikace metody hraničních prvků při výpočtu elektrického pole radiofrekvenční iontové pasti

Katedra fyziky povrchů a plazmatu

Vedoucí bakalářské práce:	RNDr. Štěpán Roučka, Ph.D.
Studijní program:	Fyzika
Studijní obor:	Obecná fyzika

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Děkuji své rodině za její neutuchající podporu. Děkuji svým učitelům, za vědomosti, které mi poskytli. A především děkuji doktoru Roučkovi, za trpělivost a cenné rady.

Název práce: Aplikace metody hraničních prvků při výpočtu elektrického pole radiofrekvenční iontové pasti

Autor: Radek Vavřička

Katedra: Katedra fyziky povrchů a plazmatu

Vedoucí bakalářské práce: RNDr. Štěpán Roučka, Ph.D., Katedra fyziky povrchů a plazmatu

Abstrakt: V této práci se věnujeme numerickým metodám řešení Laplaceovy úlohy. Srovnáváme metodu hraničních prvků (BEM), implementovanou pomocí knihovny Bempp, a metodu konečných prvků (FEM), implementovanou pomocí knihovny FEniCS, a to pro ilustrační příklad multipólu i pro specifickou radiofrekvenční iontovou past, jejíž model vytvoříme v programu Gmsh. Abychom určili efektivnost metody, zkoumáme dobu výpočtu, paměťovou náročnost a celkovou kvadratickou odchylku od teoretických hodnot, známe-li je. Získané hodnoty elektrické intenzity následně použijeme k simulaci pohybu částice v pasti, opět srovnáváme BEM a FEM.

Klíčová slova: BEM, FEM, rf iontová past

Title: Application of boundary element method for calculation of electric field in a radiofrequency ion trap

Author: Radek Vavřička

Department: Department of Surface and Plasma Science

Supervisor: RNDr. Štěpán Roučka, Ph.D., Department of Surface and Plasma Science

Abstract: In this thesis we explore numerical methods that could be used to solve Laplace's equation. We compare the boundary element method (BEM), implemented in Bempp library, and the finite element method (FEM), implemented in FEniCS library, by using them to solve Laplace's equation in the cases of a multipole and a rf ion trap, which we create a model of through the use of Gmsh. In order to gauge effectiveness of a method, we measure the time of evaluation, memory usage and the squared deviation from theoretical values (if we know them) summed over the points of evaluation. In the end, we use the known values of electric field to simulate a charged particle moving within the trap, comparing evolution of its position and velocity for BEM and FEM.

Keywords: BEM, FEM, rf ion trap

Obsah

Úvod

Počítačové simulace se prokázaly jako neocenitelná pomůcka při moderním studiu radiofrekvenčních iontových pastí. Umožňují nám s dostatečnou přesností numericky určit elektromagnetická pole pasti, pro které je nelze vypočítat analyticky.

Doposud široce používaná metoda hraničních prvků (FEM), spočívající v diskretizaci domény, se potýká s přílišnou výpočetní náročností, chceme-li dosáhnout přesnosti požadované zkoumaným experimentálním uspořádáním. Jako perspektivní alternativa se jeví metoda hraničních prvků (BEM), spočívající v diskretizaci hranice, a její implementace v knihovně Bempp.

V následujících kapitolách nastíníme výhody a nevýhody BEM v porovnání s FEM, a to pomocí ilustračního příkladu i výpočtu eletrického pole zkoumané pasti. Rovněž stručně uvedeme teoretické podklady BEM a radiofrekvenčních iontových pastí.

1. Metoda hraničních prvků (BEM) a její implementace v knihovně Bempp

1.1 Motivace

Jisté parciální diferenciální rovnice, například pro nás důležité rovnice elektrostatického pole, lze, hledáme-li řešení u na definiční množině $\Omega \subset \mathbb{R}^d, d \in \{2, 3\}$, převést na integrální rovnice definované na hranici $\Gamma = \partial\Omega$.

Na rozdíl od metody konečných prvků (FEM), kdy provádíme diskretizaci definiční množiny Ω, při řešení dané parciální diferenciální rovnice metodou hraničních prvků (BEM) postačí diskretizace hranice Γ, což řádově snižuje počet neznámých.

V následujícím stručném rozboru metody hraničních prvků (BEM) a knihovny Bempp se zaměříme na řešení rovnice elektrostatického pole, vycházíme přitom z Śmigaj a kol. (2015).

1.2 Rovnice elektrostatického pole

Řešíme Laplaceovu rovnici

$$-\Delta u(\boldsymbol{x}) = 0 \tag{1.1}$$

na množině $\Omega \subset \mathbb{R}^d, d \in \{2, 3\}$, s po částech spojitou Lipschitzovskou hranicí Γ. Díky Greenově větě o reprezentaci lze řešení u určit jako

$$u(\boldsymbol{x}) = \int_\Gamma g(\boldsymbol{x}, \boldsymbol{y}) \frac{\partial}{\partial n(\boldsymbol{y})} u(\boldsymbol{y}) \mathrm{d}\Gamma(\boldsymbol{y}) - \int_\Gamma \frac{\partial}{\partial n(\boldsymbol{y})} g(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \mathrm{d}\Gamma(\boldsymbol{y}), \tag{1.2}$$

kde $\boldsymbol{n}$ značí jednotkový normálový vektor Γ orientovaný z Ω a $g(\boldsymbol{x}, \boldsymbol{y})$ značí Greenovu funkci definovanou jako

$$g(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} -\frac{1}{2\pi} \log |\boldsymbol{x} - \boldsymbol{y}|, & d = 2, \\ \frac{1}{4\pi |\boldsymbol{x} - \boldsymbol{y}|}, & d = 3. \end{cases} \tag{1.3}$$

Tedy známe-li u, resp. $\partial_n u$ na Γ, lze $\partial_n u$, resp. u získat, řešíme-li rovnici (1.2) na Γ.

1.3 Srovnání s jinými metodami

Mezi výhody formulace parciální diferenciání rovnice ve tvaru hraničního integrálu patří fakt, že nám postačuje pouze $O(N^{d-1})$ neznámých pro diskretizaci hranice Γ, zatímco pro metody řešící parciální diferenciální rovnice diskretizací definiční množiny Ω potřebujeme neznámých $O(N^d)$, kde N značí lineární rozměr mříže.

Existují však rovněž podstatné nevýhody:

- Výpočet maticových hodnot vyžaduje vyhodnocení singulárních integrálů, které jsou 4-rozměrné v případě Galerkinovské formulace 3-rozměrného problému.
- Operátory jsou nelokální, tedy se vyjadřují jako husté matice, pro které se násobení matice vektorem ve 3 rozměrech chová jako $O(N^4)$, kdežto při použití metody konečných prvků (FEM) s operátory vyjádřenými jako řídké matice jde o $O(N^3)$.

1.4 Integrální operátory a Calderónova projekce

Budeme se zabývat použitím integrálních rovnic na hranici k řešení Laplaceovské úlohy (1.1), vycházime přitom z Steinbach (2008).

$v := \gamma_0^{\text{int}} u$ značí Dirichletovu stopu $u(\boldsymbol{x})$ na hranici Γ, $t := \gamma_1^{\text{int}} u$ značí konormální derivaci, v rámci Laplaceových úloh se jedná o normálovou derivaci, tedy Neumannovu stopu $u(\boldsymbol{x})$ na hranici Γ. Předpokládáme, že normálové vektory na Γ směřují z Ω. $H^s(\Omega)$ značí Sobolevovy prostory řádu $s \in \mathbb{R}$ na Ω.

Zavedeme jednovrstvý potenciálový operátor $\mathcal{V} : H^{-\frac{1}{2}}(\Gamma) \to H^1(\Omega)$ a dvouvrstvý potenciálový operátor $\mathcal{K} : H^{\frac{1}{2}}(\Gamma) \to H^1(\Omega)$ pro $\boldsymbol{x} \in \Omega$ jako

$$\begin{aligned} [\mathcal{V}\psi](x) &= \int_\Gamma g(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{y})\mathrm{d}\Gamma(\boldsymbol{y}), \\ [\mathcal{K}\phi](x) &= \int_\Gamma \gamma_{1,\boldsymbol{y}}^{\text{int}} g(\boldsymbol{x},\boldsymbol{y})\phi(\boldsymbol{y})\mathrm{d}\Gamma(\boldsymbol{y}). \end{aligned} \tag{1.4}$$

S využitím Greenovy věty o reprezentaci (1.2) lze řešení napsat jako

$$u = \mathcal{V}t - \mathcal{K}v. \tag{1.5}$$

Vezmeme-li stopy obou stran rovnice (1.5), získáme

$$v = (\frac{1}{2}I - K)v + Vt, \tag{1.6}$$

kde $V : H^{-\frac{1}{2}}(\Gamma) \to H^{\frac{1}{2}}(\Gamma)$ je jednovrstvý potenciálový operátor na hranici a $K : H^{\frac{1}{2}}(\Gamma) \to H^{\frac{1}{2}}(\Gamma)$ dvouvrstvý potenciálový operátor na hranici, definované jako

$$\begin{aligned} [V\psi](x) &= \int_\Gamma g(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{y})\mathrm{d}\Gamma(\boldsymbol{y}), \\ [K\phi](x) &= \int_\Gamma \gamma_{1,\boldsymbol{y}}^{\text{int}} g(\boldsymbol{x},\boldsymbol{y})\phi(\boldsymbol{y})\mathrm{d}\Gamma(\boldsymbol{y}). \end{aligned} \tag{1.7}$$

Operátor identity v rovnici (1.6) značí skokovou změnu dvouvrstvého potenciálu.

Vezmeme-li konormálovou derivaci obou stran rovnice (1.5), získáme

$$t = Dv + (\frac{1}{2}I + T)t, \tag{1.8}$$

kde $D : H^{\frac{1}{2}}(\Gamma) \to H^{-\frac{1}{2}}(\Gamma)$ je hypersingulární operátor a $T : H^{-\frac{1}{2}}(\Gamma) \to H^{-\frac{1}{2}}(\Gamma)$ sdružený dvouvrstvý potenciálový operátor na hranici, definované jako

$$\begin{aligned} [T\psi](x) &= \int_\Gamma \gamma_{1,\boldsymbol{y}}^{\text{int}} g(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{y})\mathrm{d}\Gamma(\boldsymbol{y}), \\ [D\phi](x) &= -\gamma_{1,\boldsymbol{y}}^{\text{int}} \left[\int_\Gamma \gamma_{1,\boldsymbol{y}}^{\text{int}} g(\boldsymbol{x},\boldsymbol{y})\phi(\boldsymbol{y})\mathrm{d}\Gamma(\boldsymbol{y})\right]. \end{aligned} \tag{1.9}$$

Z výrazů (1.6) a (1.8) získáme Calderónovu projekci

$$\begin{bmatrix} v \\ t \end{bmatrix} = \begin{bmatrix} \frac{1}{2}I - K & V \\ D & \frac{1}{2}I + T \end{bmatrix} \begin{bmatrix} v \\ t \end{bmatrix}. \tag{1.10}$$

Známe-li v, resp. t, lze z (1.10) získat t, resp. v. Dvojice funkcí $(v,t) \in H^{\frac{1}{2}}(\Gamma) \times H^{-\frac{1}{2}}(\Gamma)$ značí hraniční a konormální stopu řešení Laplaceovy úlohy (1.1) právě tehdy, platí-li (1.10).

1.5 Implementace v Bempp

Níže uvádíme příklad ze stránek BemppProject. Pracujeme v jazyce Python, konkrétně v prostředí IPython.

Pro ilustraci hledáme řešení úlohy (1.1) na jednotkové kouli při hraničních podmínkách, pro které známe analytické řešení.

$$u = \frac{1}{4\pi|\boldsymbol{x} - \boldsymbol{s}|}, \boldsymbol{s} = (0.9, 0, 0) \tag{1.11}$$

Nejprve importujeme Bempp (verze 3.3.2) a NumPy.

```
import bempp.api
import numpy as np
```

Následně definujeme síť - lze ji importovat ze souboru, či použít předem připravený tvar.

```
grid = bempp.api.shapes.sphere(h=0.1)
```

Nyní definujeme funkční prostory na meshi hranice Γ. Jedná se o prostor po částech konstantních funkcí, na kterém reprezentujeme Neumannova data, a o prostor spojitých, po částech lineárních funkcí, na kterém reprezentujeme Dirichletova data.

```
dp0_space = bempp.api.function_space(grid, "DP", 0)
p1_space = bempp.api.function_space(grid, "P", 1)
```

Dále definujeme hraniční operátory - operátor identity, dvouvrstvý operátor K a jednovrstvý operátor V.

```
identity = bempp.api.operators.boundary.sparse.identity(
    p1_space, p1_space, dp0_space)
dlp = bempp.api.operators.boundary.laplace.double_layer(
    p1_space, p1_space, dp0_space)
slp = bempp.api.operators.boundary.laplace.single_layer(
    dp0_space, p1_space, dp0_space)
```

Definujeme Dirichletovu funkci pomocí známých hodnot hledaného řešení na hranici Γ.

```
def dirichlet_data(x, n, domain_index, result):
```

```
    result[0] = 1./(4 * np.pi * ((x[0] - .9)**2 + x[1]**2 +
        x[2]**2)**(0.5))

dirichlet_fun = bempp.api.GridFunction(p1_space, fun=dirichlet_data)
```

Poté napíšeme pravou stranu rovnice

$$Vt = (\frac{1}{2}I + K)v. \tag{1.12}$$

```
rhs = (.5 * identity + dlp) * dirichlet_fun
```

Nyní vyřešíme rovnici (1.12) metodou sdružených gradientů.

```
neumann_fun, info = bempp.api.linalg.cg(slp, rhs, tol=1E-3)
```

Definujeme body v rovině $(\boldsymbol{x}, \boldsymbol{y})$.

```
n_grid_points = 150
plot_grid = np.mgrid[-1:1:n_grid_points*1j, -1:1:n_grid_points*1j]
points = np.vstack((plot_grid[0].ravel(),
                    plot_grid[1].ravel(),
                    np.zeros(plot_grid[0].size)))
```

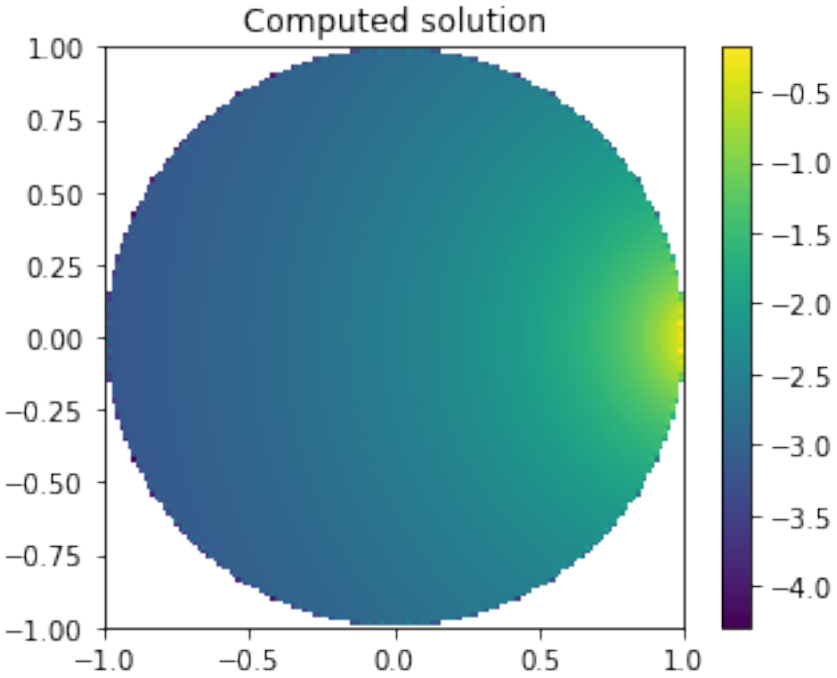
Na těchto bodech lze u vyhodnotit pomocí věty (1.2), definujeme-li jednovrstvý a dvouvrstvý potenciálový operátor.

```
slp_pot = bempp.api.operators.potential.laplace.single_layer(
    dp0_space, points)
dlp_pot = bempp.api.operators.potential.laplace.double_layer(
    p1_space, points)
u_evaluated = slp_pot * neumann_fun - dlp_pot * dirichlet_funLXXXII
```

Hodnoty u na řezu Ω poté logaritmicky zobrazíme pomocí knihovny Matplotlib.

```
%matplotlib inline

u_evaluated = u_evaluated.reshape((n_grid_points,n_grid_points))
radius = np.sqrt(plot_grid[0]**2 + plot_grid[1]**2)
u_evaluated[radius>1] = np.nan

import matplotlib
matplotlib.rcParams['figure.figsize'] = (5.0, 4.0)

from matplotlib import pylab as plt

plt.imshow(np.log(np.abs(u_evaluated.T)), extent=(-1,1,-1,1))
plt.title('Computed solution')
plt.colorbar()
```

V podstatě identický postup lze požít k řešení (1.1) na libovolné množně Ω s libovolnou hraniční podmínkou.



Obrázek 1.1: Ilustrační řešení (1.1) na jednotkové kouli.

2. Iontové pasti - stručný teoretický úvod

2.1 Motivace

Studium reakcí mezi ionty a neutrálními částicemi se v posledních desetiletích rozvinulo v plnohodnotný fyzikální obor, těšící se pozornosti vědecké komunity. V přirozeném důsledku došlo k rozvoji teorie a technologie pro vytváření, záchyt a detekci nabitých částic. V této kapitole se budeme věnovat základním teoretickým principům iontových pastí.

Elektrostatická pole nelze sama o sobě použít ke konstrukci iontových pastí, jelikož nedokáží vytvořit v oblasti záchytu potenciálová minima, ale jak vyplývá z beznábojové Laplaceovy rovnice, pouze sedlové body. Tento problém lze obejít, použijeme-li *efektivní potenciál*, vytvořený oscilujícím elektrickým nebo elektromagnetickým polem, který umožňuje vznik potenciálových minim.

V následujícím teoretickém rozboru vycházíme primárně z Gerlich (1992).

2.2 Pohyb nabité částice v rychle oscilujícím elmag. poli

Uvažujeme-li nabitou částici pohybující se v elektromagnetickém poli $\boldsymbol{E}(\boldsymbol{r},t)$ a $\boldsymbol{B}(\boldsymbol{r},t)$, nerelativistická rovnice pohybu je

$$m\ddot{\boldsymbol{r}} = q\boldsymbol{E}(\boldsymbol{r},t) + q\dot{\boldsymbol{r}} \times \boldsymbol{B}(\boldsymbol{r},t). \tag{2.1}$$

Tuto rovnici lze řešit analyticky pouze pro jistá elektromagnetická pole, tedy pro elektromagnetické pole zadané iontové pasti rovnici zjednodušíme a pokusíme se nalézt přibližné řešení numericky. Víme, že pracujeme se slabými elektrickými poli a těžkými částicemi s nízkou pohybovou energií. Rychlost $\dot{\boldsymbol{r}}$ je nízká v porovnání s rychlostí světla, tedy sílu způsobenou interakcí částice s magnetickým polem lze zanedbat. Kvazistacionární magnetická pole neuvažujeme.

Dále předpokládáme, že elektrické pole $\boldsymbol{E}(\boldsymbol{r},t)$ je lineární kombinace stacionárního pole $\boldsymbol{E}_S(\boldsymbol{r})$ a časově závislého pole $\boldsymbol{E}_0(\boldsymbol{r})\cos(\omega t + \varphi_0)$, kde ω je konstantní úhlová frekvence a ϕ_0 počáteční fáze. Pohyb částice lze poté popsat rovnicí

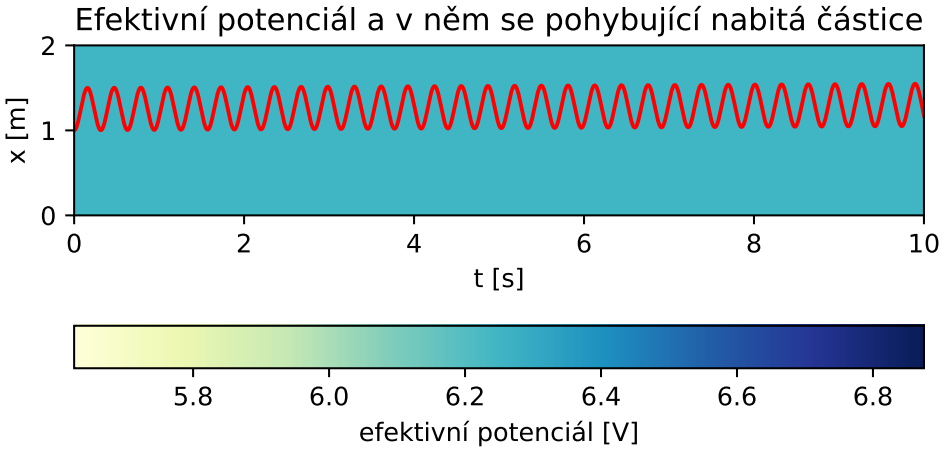
$$m\ddot{\boldsymbol{r}} = q\boldsymbol{E}_0(\boldsymbol{r})\cos(\omega t + \varphi_0) + q\boldsymbol{E}_S(\boldsymbol{r}). \tag{2.2}$$

Pro ilustraci se nejprve zabývejme výše zmíněnou rovnicí při prostorově homogenním elektrickém poli $\boldsymbol{E}_0(\boldsymbol{r}) = \boldsymbol{E}_0$, $\boldsymbol{E}_S(\boldsymbol{r}) = 0$, která popisuje například nabitou částici mezi rovnoběžnými deskami kapacitoru, a kterou lze analyticky vyřešit jako

$$\boldsymbol{r}(t) = \boldsymbol{r}(0) + \boldsymbol{a}\cos\varphi_0 + (\dot{\boldsymbol{r}}(0) - \boldsymbol{a}\sin\varphi_0)t - \underbrace{\frac{q\boldsymbol{E}_0}{m\omega^2}}_{\boldsymbol{a}}\cos(\omega t + \varphi_0) \tag{2.3}$$

Pro řešení ilustračních případů jsme napsali Ipythonový sešit A.2, zdrojový kód přikládáme elektronicky jako *EffPot.ipynb*.

Závislost polohy na čase je pro $\dot{\boldsymbol{r}}(0) = 0$ znázorněna (redukujeme-li prostorové rozměry na 1) na obrázku 2.1. Částice osciluje v okolí počáteční polohy $x(0)$ s amplitudou $\boldsymbol{a}$.



Obrázek 2.1: Pohyb částice s nulovou počáteční rychlostí v prostorově homogenním, s časem oscilujícím el. poli.

Jiná situace nastane, je-li elektrické pole nehomogenní. Tento případ demonstrujeme na příkladu jednorozměrného potenciálu $V(x) \sim x^3$, jemu odpovídajícího elektrického pole $E(x) \sim x^2$, a efektivního potenciálu určeného jako (2.11). Pohyb částice v tomto potenciálu zobrazujeme na obrázku 2.2, Specifika tohoto pohybu diskutujeme níže.

2.3 Efektivní potenciál

Pro popis pohybu nabité částice v nehomogenním radiofrekvenčním poli předpokládáme, že elektrická intenzita je hladkou funkcí prostorových souřadnic $\boldsymbol{r}$, a že úhlová rychlost ω je vysoká natolik, aby cosinová amplituda $\boldsymbol{a}$ z rovnice (2.3) byla dostatečně malá. Dále předpokládáme, že amplituda $\boldsymbol{a}(t)$ a oscilační fáze se s časem mění pozvolna. Tedy hledáme řešení rovnice (2.2) jako složený pohyb pozvolného posuvu $\boldsymbol{R}_0(t)$ a prudce oscilujícího pohybu $\boldsymbol{R}_1(t)$

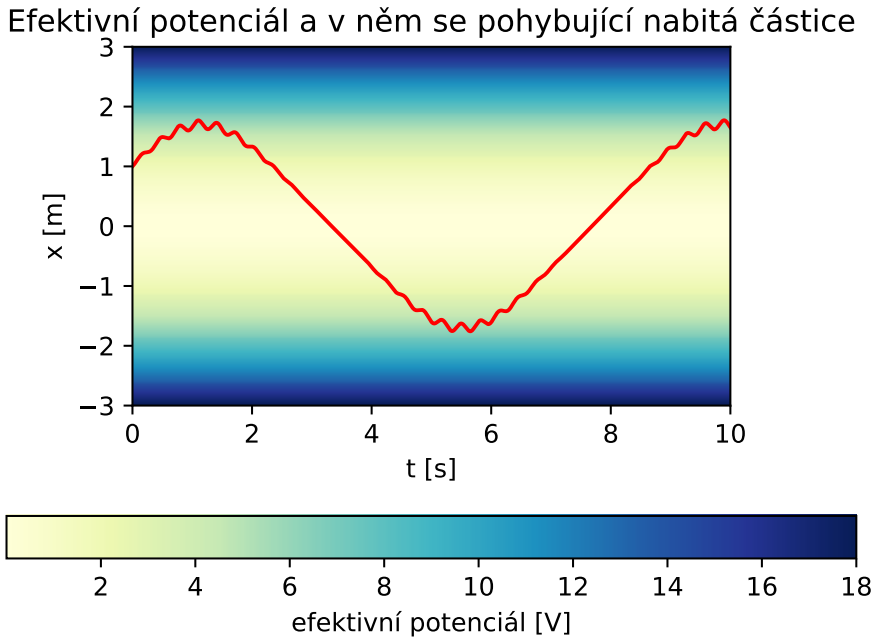
$$\boldsymbol{r}(t) = \boldsymbol{R}_0(t) + \boldsymbol{R}_1(t), \tag{2.4}$$

kde

$$\boldsymbol{R}_1(t) = -\boldsymbol{a}\cos\omega t. \tag{2.5}$$

Předpokládáme, že $\boldsymbol{E}_0$ se v prostoru mění dostatečně pozvolna na to, abychom zbylé členy rozvoje než první dva mohli zanedbat

$$\boldsymbol{E}_0(\boldsymbol{R}_0 - \boldsymbol{a}\cos\omega t) = \boldsymbol{E}_0(\boldsymbol{R}_0) - (\boldsymbol{a}\cdot\nabla)\boldsymbol{E}_0(\boldsymbol{R}_0)\cos\omega t + ..., \tag{2.6}$$



Obrázek 2.2: Pohyb částice v prostorově nehomogenním, s časem oscilujícím el. poli.

dále z pomalé změny $\boldsymbol{a}$ a $\boldsymbol{R}_0$ s časem vyplývá $\dot{\boldsymbol{a}} << \omega \boldsymbol{a}$ a $\ddot{\boldsymbol{R}}_0 << \omega \dot{\boldsymbol{R}}_0$. Dosadíme-li (2.4) a (2.6) do (2.2), získáme

$$m\ddot{\boldsymbol{R}}_0 + m\omega^2 \boldsymbol{a}(t)\cos\omega t = q\boldsymbol{E}_0(\boldsymbol{R}_0)\cos\omega t + q\left[\boldsymbol{a}(t)\cdot\nabla\right]\boldsymbol{E}_0(\boldsymbol{R}_0)\cos^2\omega t. \tag{2.7}$$

Předpokládáme-li, že se $\boldsymbol{a}$ mění s časem pouze v důsledku pohybu podél $\boldsymbol{R}_0$, lze $\boldsymbol{a}(t)$ napsat jako $\boldsymbol{a}(\boldsymbol{R}_0)$ a $\cos\omega t$ na obou stranách rovnice se navzájem vyruší.

Nahradíme-li $\cos^2\omega t$ časově vystředovanou hodnotou $1/2$, lze diferenciální rovnici pro sekulární pohyb vyjádřit jako

$$m\ddot{\boldsymbol{R}}_0 = -\frac{q^2}{4m\omega^2}\nabla E_0^2 \tag{2.8}$$

Tedy na částici působí síla vyvolaná nehomogennitou elektrického pole $\boldsymbol{E}_0$, přímo úměrná q^2, a proto nezávisí na polaritě náboje.

Uvažujeme-li rovněž i statické elektrické pole $\boldsymbol{E}_S$, které působí na částici silou $q\boldsymbol{E}_S = -q\nabla\Phi_S$, lze celkovou sílu působící na částici popsat pomocí *efektivního potenciálu*

$$V_{\text{eff}} = \frac{q^2 E_0^2}{4m\omega^2} + q\Phi_S \tag{2.9}$$

jako

$$m\ddot{\boldsymbol{R}}_0 = -\nabla V_{\text{eff}}. \tag{2.10}$$

Abychom však efektivní potenciál uváděli ve stejných jednotkách jako běžný potenciál, tedy ve voltech, budeme dále efektivní potenciál psát jako

$$U_{\text{eff}} = \frac{qE_0^2}{4m\omega^2} + \Phi_S. \tag{2.11}$$

Z rovnice 2.5 dále můžeme určit rovnici oscilačního pohybu

$$\boldsymbol{R}_1(t) = -\frac{q\boldsymbol{E}_0(\boldsymbol{R}_0)}{m\omega^2}\cos\omega t. \tag{2.12}$$

Integrací rovnice (2.10) získáme vztah

$$\frac{1}{2}\dot{\boldsymbol{R}}_0^2 + \frac{q^2E_0^2}{4m\omega^2} + q\Phi_S = E_m, \tag{2.13}$$

kde celková energie E_m je adiabatickou konstantou pohybu. Druhý člen rovnice (2.13) je shodný s časově vystředovanou kinetickou energií oscilačního pohybu podle rovnice (2.12)

$$\langle\frac{1}{2}m\dot{\boldsymbol{R}}_1^2\rangle = \frac{q^2E_0^2}{4m\omega^2}. \tag{2.14}$$

Tedy efektivněpotenciální energie se ve skutečnosti ukládá jako kinetická energie rychle oscilujícího pohybu. Během pohybu se proto energie E_m vyměňuje mezi 3 složkami

- kinetickou energií $\frac{1}{2}m\dot{\boldsymbol{R}}_0^2$,
- kinetickou energií $\frac{1}{2}m\dot{\boldsymbol{R}}_1^2$,
- potenciální energií $q\Phi_S$.

2.4 Adiabacita

Celková energie systému se obecně zachovává pouze pro pomalou změnu $\boldsymbol{a}$ a $\boldsymbol{R}_0$, jak uvádíme výše.

Nakolik přesně lze aplikovat efektivněpotenciální aproximaci lze určit pomocí parametru adiabacity

$$\eta(\boldsymbol{r}) = \frac{2q|\nabla E_0(\boldsymbol{r})|}{m\omega^2}. \tag{2.15}$$

Obecně požadujeme $|\eta(\boldsymbol{r})| < 1$ na trajektorii částice, maximální možný parametr η, při němž se systém chová adiabaticky, se však liší pro různé geometrie iontové pasti.

2.5 Konstrukce pasti

V následujících kapitolách budeme podrobně zkoumat lineární iontovou past, tvořenou 22 rovnoběžnými tyčemi, na které je přivedeno střídavé napětí vytvářející efektivní potenciál, a uzavřenou na obou stranách endcapy, na které je přivedeno stejnosměrné napětí. Podrobný popis pasti uvádíme v kapitole 4.

3. Srovnání BEM a FEM pomocí úlohy, pro kterou známe analytické řešení

Pro Laplaceovu rovnici (1.1) známe řadu analytických řešení, mimo jiné potenciál elektrického multipólu

$$u(r, \varphi, \theta) = \Re(Y_n^m(\varphi, \theta))r^n \tag{3.1}$$

Sférické harmonické funkce lze importovat do prostředí IPython díky knihovně SciPy.

```
from scipy.special import sph_harm
def mlpl (x, m, n):
   r = np.sqrt(x[0]**2 + x[1]**2 + x[2]**2)
   theta = np.arccos(x[2]/r)
   phi = np.arctan2(x[1], x[0])+np.pi
   return sph_harm (m, n, phi, theta) . real * r ** (n)
```

Konkrétně se budeme zabývat funkcí Y_{11}^5.

```
def mlp (x):
   return mlpl (x, 5, 11)
```

Řešení se však pro srovnání pokusíme získat i numericky, s použitím BEM a FEM, na jednotkové kouli Ω přičemž jako hraniční podmínku použijeme hodnoty Y_{11}^5 na hranici $\Gamma = \partial\Omega$.

Zdrojový kód k této kapitole uvádíme v příloze na stránce A.3, ve formátu *Multipole_BEM_FEM.ipynb* jej přikládáme elektronicky.

3.1 Implementace v Bempp

Při výpočtu numerických hodnot u postupujeme obdobně jako v kapitole 1.

Potenciál na zadaných bodech evaluace určíme pro libovolnou jemnost gridu.

```
def aMlpBEM (u_eval, p, h):
   grid = bempp.api.shapes.sphere (h = h)

   SpBop = eSpBop (grid)

   NDf = eNDf (SpBop, dirichlet_data)

   Pop = ePop (SpBop, p)

   u_eval [:] = uBEM (Pop, NDf)
```

Jemnost gridu zavádíme jako posloupnost délek diskretizačního elementu.

```
def hfBEM (i):
   return 1.0 / float (1 + i)
```

Vytvoříme funkci, která vyhodnotí čas výpočtu, celkovou kvadratickou odchylku (viz rovnice (3.2)) a paměťovou náročnost za dobu běhu programu (maximum, minimum, průměr, střední kvadratickou odchylku) v závislosti na hustotě meshe.

$$s = \sum_{i=0}^{n-1} (\bar{u} - u_i)^2, \{u_i\}_{i=0}^{n} \text{ značí hodnoty potenciálu v bodech evaluace.} \quad (3.2)$$

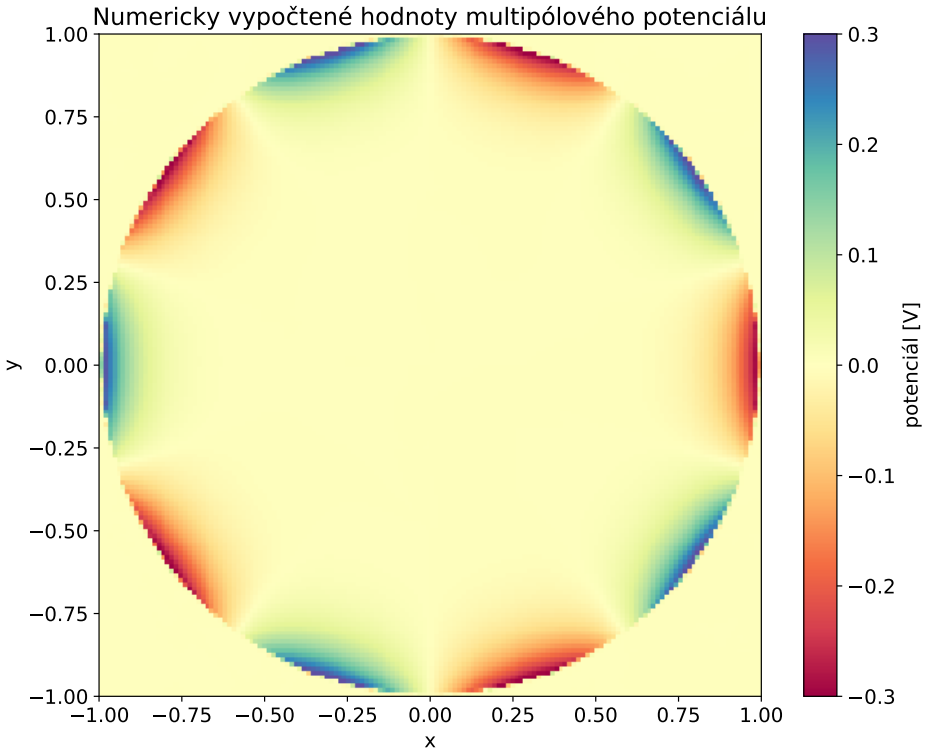
```
def memt (Pl, Ins, asgn, j, p, b, dt = 0.1):
   n = p [0]. size
   u_eval = np.empty ((1, n))
   teor = np.empty ((1, n))
   mem = np.array (memory_usage ((asgn, (u_eval, p, b), {}), interval =
       dt))
   s = 0.0
   for i in range (0, n - 1):
   if (Ins (p [:, i])):
      teor [0, i] = mlp (p [:, i])
      s = s + (u_eval [0, i] - teor [0, i]) ** 2
   else:
      teor [0, i] = 0

   if (j <= 0):
      Pl (u_eval, teor)
   t = mem.size
   mx = max (mem [:])
   mn = min (mem [:])
   ms = sum (mem [:])
   ma = ms / float (t)
   me = np.sqrt (sum ((mem [:] - ma) ** 2) / float (t))
   return np.array ([t * dt, s, mx, mn, ma, me])
```

Následně můžeme pro hustoty meshe v rozmezí 1 až l definovat funkci, která jim přiřadí odpovídající výše jmenované veličiny. Mezi každým voláním výše zmíněné funkce program posečká $10s$, za účelem snížení možného vlivu zahřátí počítačových komponent na výpočetní výkonnost.

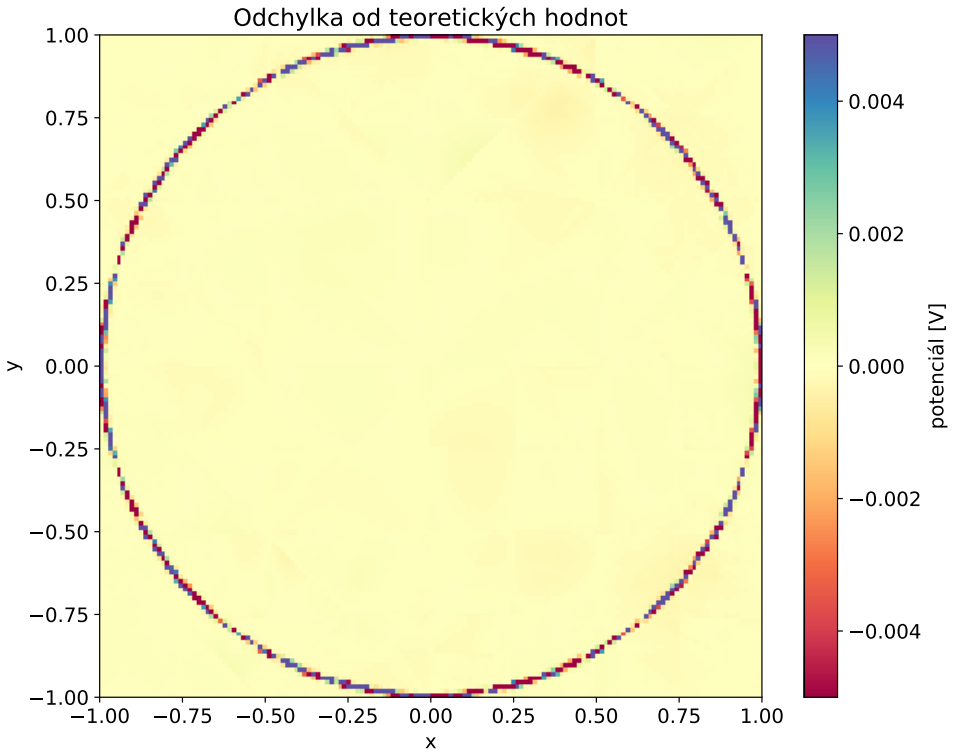
```
def arr (Pl, Ins, asgn, l, p, hf):
   z = np.empty ((7, l))
   for i in range (1, l + 1):
      z [0, i - 1] = i
      time.sleep (10)
      z [1:7, i - 1] = memt (Pl, Ins, asgn, l - i, p, hf (i))
   return z
```

Vypočtené hodnoty potenciálu a jeho odchylky od teoretických hodnot znázorníme na obrázcích 3.1 a 3.2.

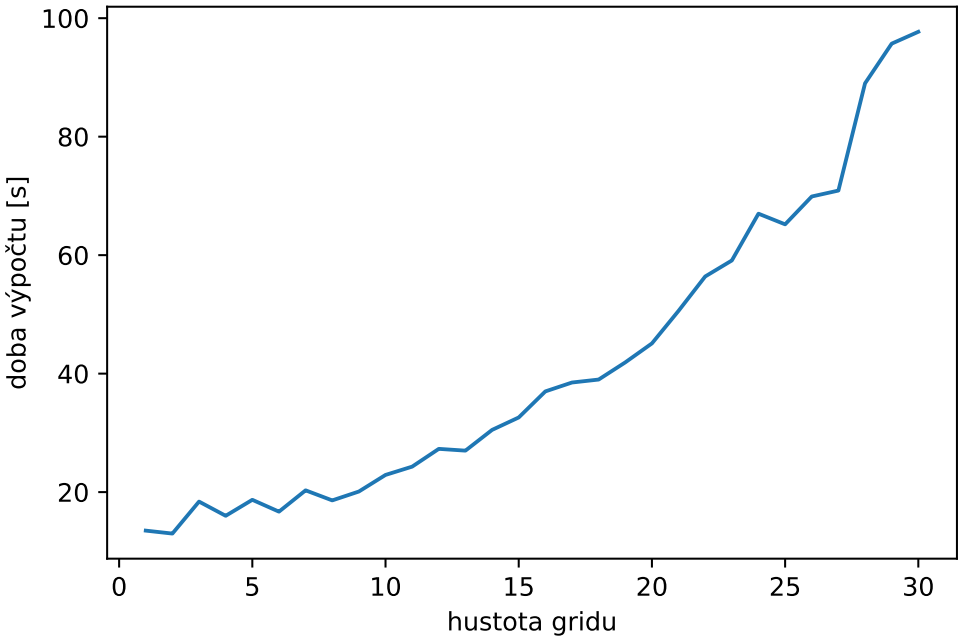
Nyní získané hodnoty znazorníme na obrázcích 3.3, 3.4, 3.5 a 3.6.



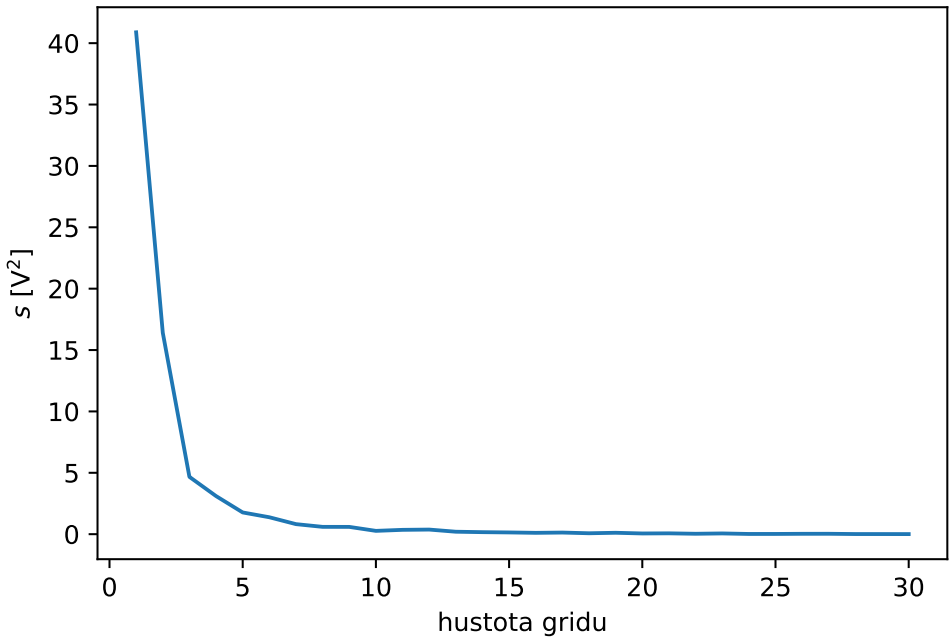
Obrázek 3.1: El. potenciál multipólu, určený pomocí BEM.



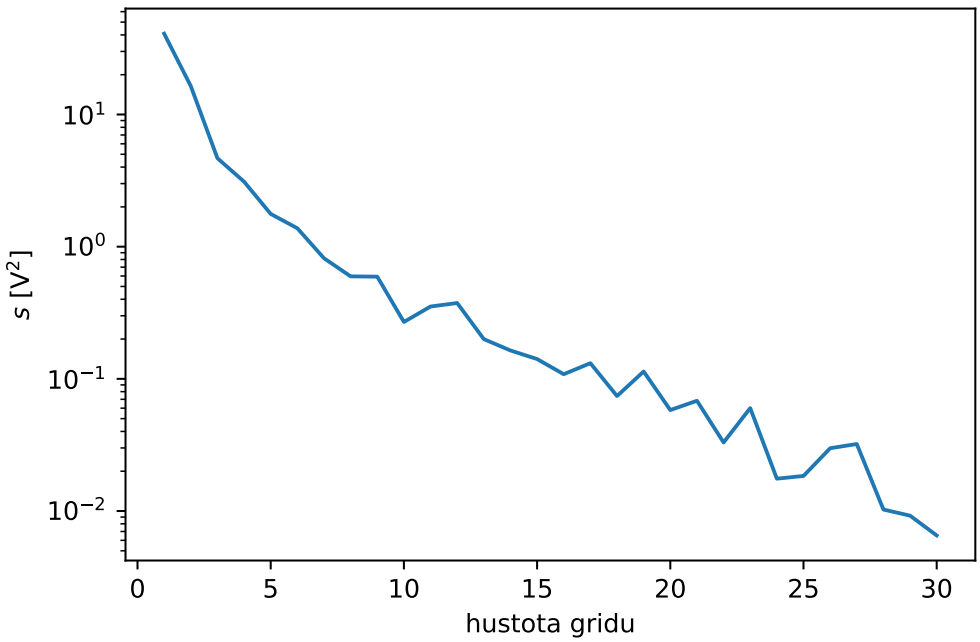
Obrázek 3.2: Odchylka 3.1 od teoretických hodnot.



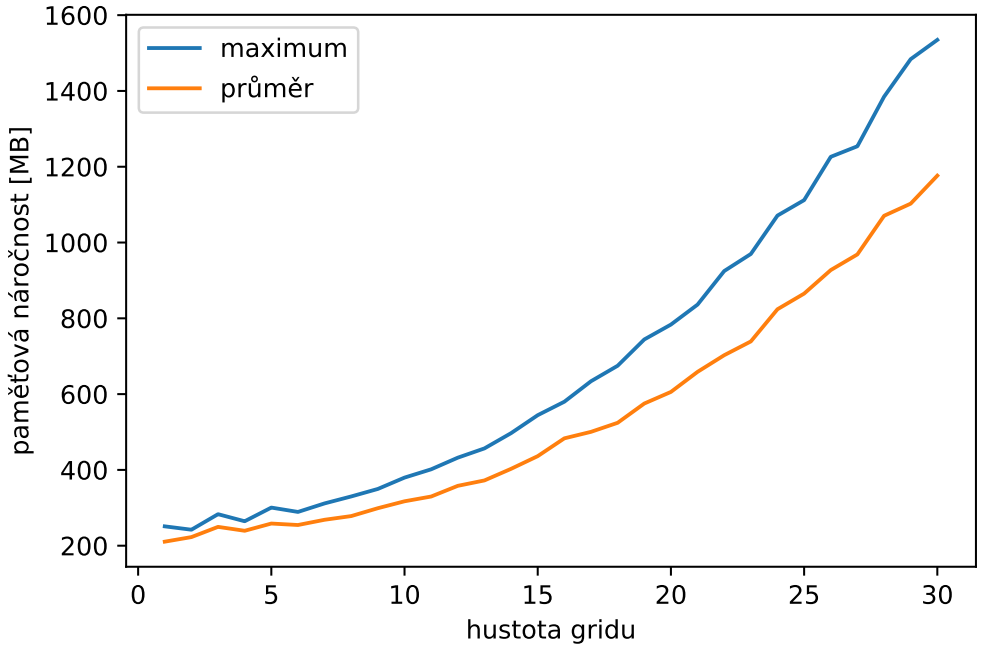
Obrázek 3.3: Závislost času výpočtu na hustotě gridu.



Obrázek 3.4: Závislost celkové kvadratické odchylky od teoretických hodnot na hustotě gridu.



Obrázek 3.5: Závislost celkové kvadratické odchylky od teoretických hodnot (logaritmická škála) na hustotě gridu.



Obrázek 3.6: Závislost paměťové náročnosti výpočtu na hustotě gridu.

3.2 Implementace ve FEniCS

FEniCS nám umožňuje přímo určit prostorovou funkci potenciálu.

```
class MyExpression0 (Expression):
   def eval (self, value, x):
      value [0] = mlp (x)
g0 = MyExpression0 (degree = 2)

def uFEM (bc, V):
   u = TrialFunction (V)
   v = TestFunction (V)
   a = inner (grad(u), grad(v)) * dx
   f = Constant (0.0)
   g = Constant (0.0)
   L = f * v * dx + g * v * ds

   u = Function(V)

   # solve(a == L, u, bc)
   solve(a == L, u, bc, solver_parameters = {'linear_solver':'gmres'})
   return u
```

Potenciál následně vyhodnotíme na zadaných bodech evaluace pro hustotu meshe b.

```
def aMlpFEM (u_eval, p, b):
   domain = mshr.Sphere(Point(0.0, 0.0, 0.0), 1)
```

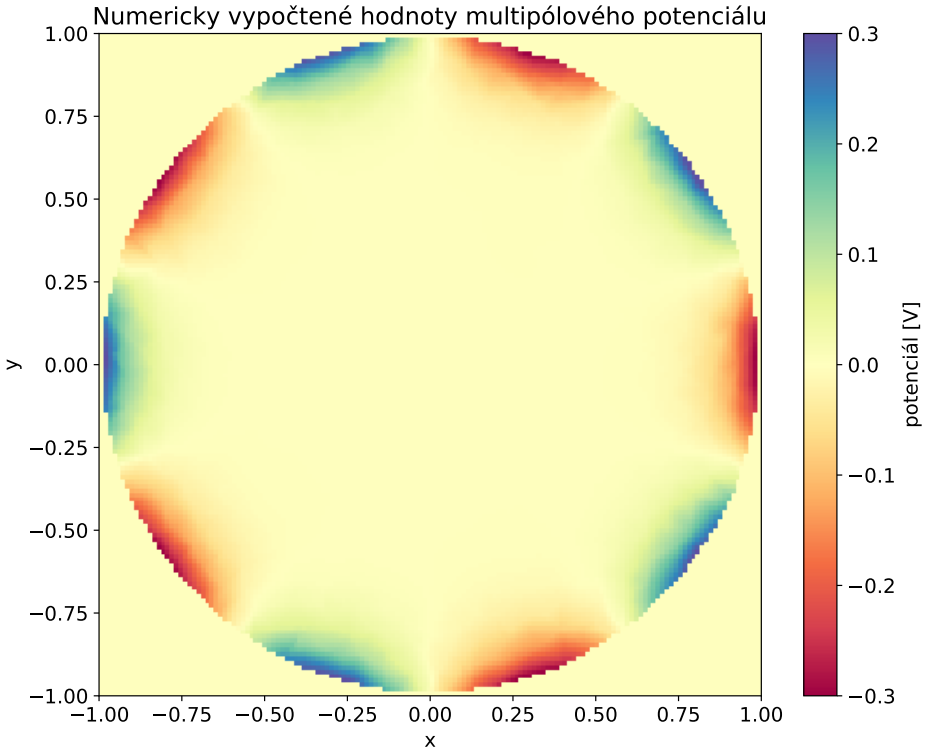
```
mesh = mshr.generate_mesh(domain, b)
V = FunctionSpace(mesh, "Lagrange", 1)
def boundary(x, on_boundary): return on_boundary
bc = DirichletBC (V, g0, boundary)
u = uFEM (bc, V)
n = p [0].size
for i in range (0, n):
   try:
      u_eval [0, i] = u (p [0, i], p [1, i], p [2, i])
   except:
      pass
```

Hustotu meshe určíme jako posloupnost počtu diskretizačních elementů.

```
def bFEM (i):
   return 10 + i
```

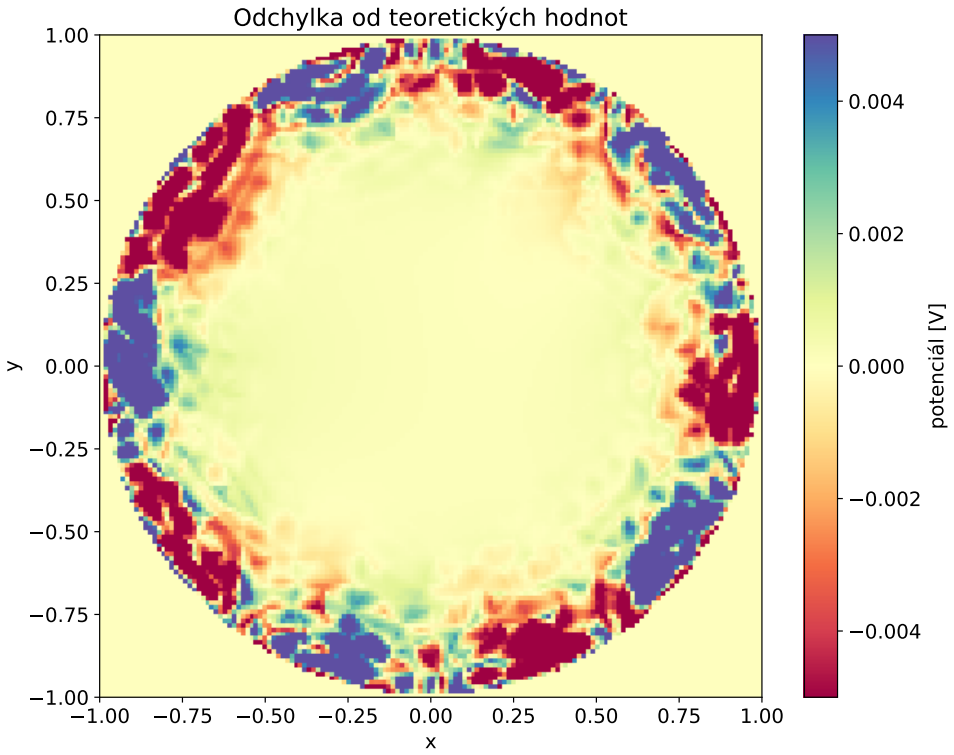
Použitím funkcí *memt* a *arr* získáme obdobné výsledky jako pro BEM.

Hodnoty potenciálu a příslušné odchylky od teoretických hodnot znázorníme na obrázcích 3.7 a 3.8.

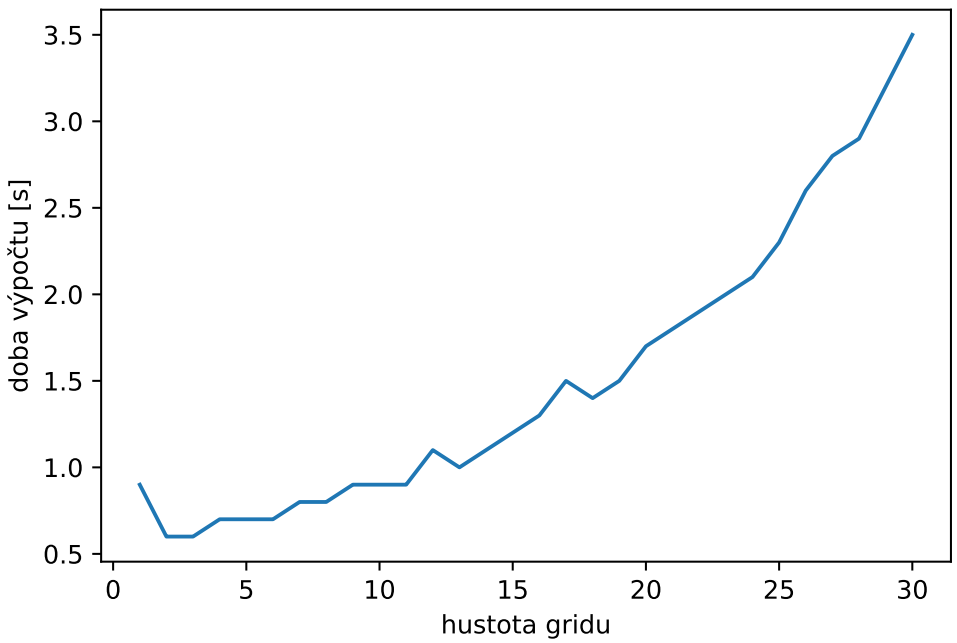


Obrázek 3.7: El. potenciál multipólu, určený pomocí FEM.

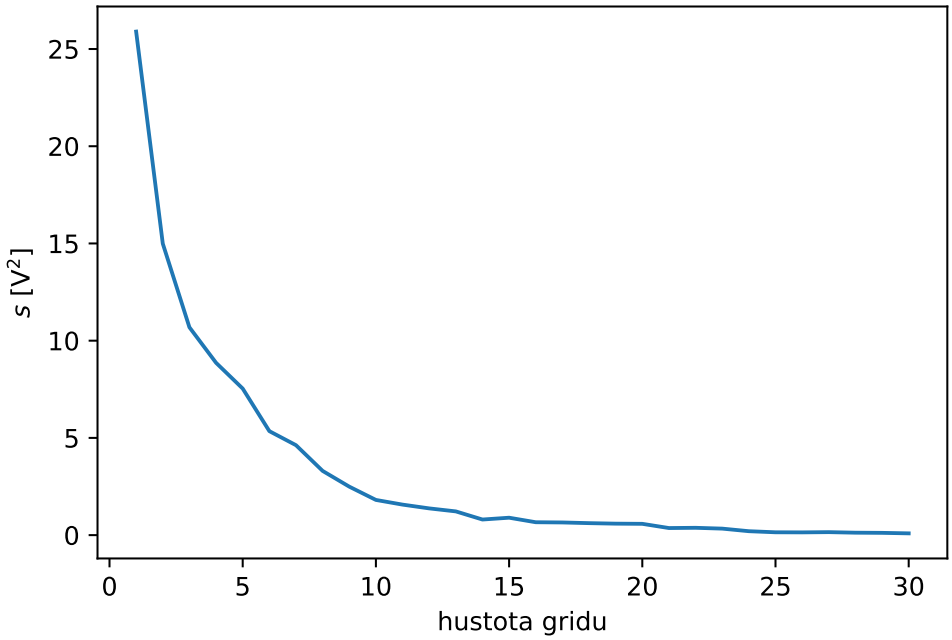
Závislosti doby výpočtu, celkové kvadratické odchylky a paměťové náročnosti na hustotě meshe ukazujeme na obrázcích 3.9, 3.10, 3.11 a 3.12.



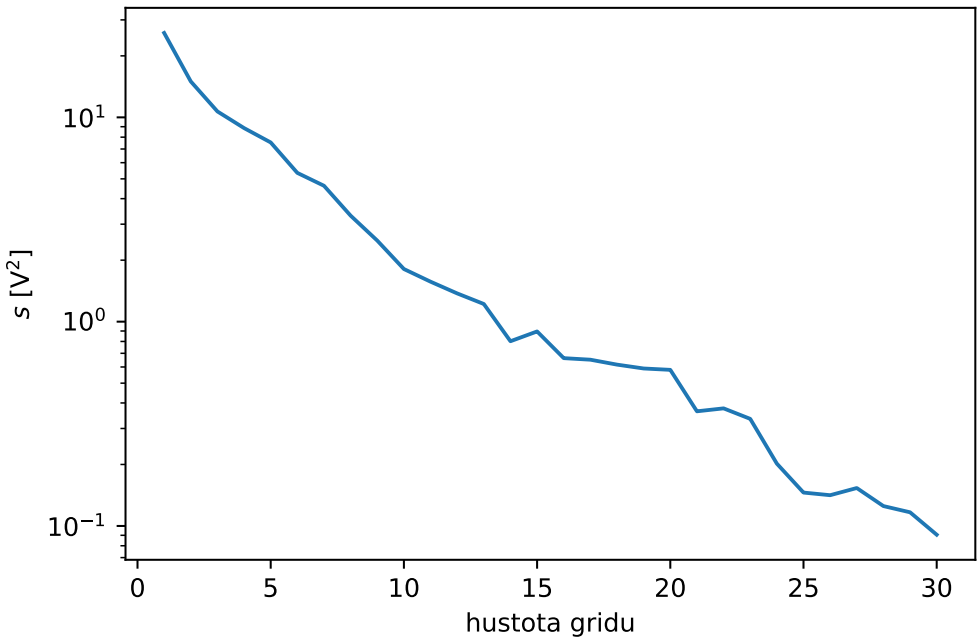
Obrázek 3.8: Odchylka 3.7 od teoretických hodnot.



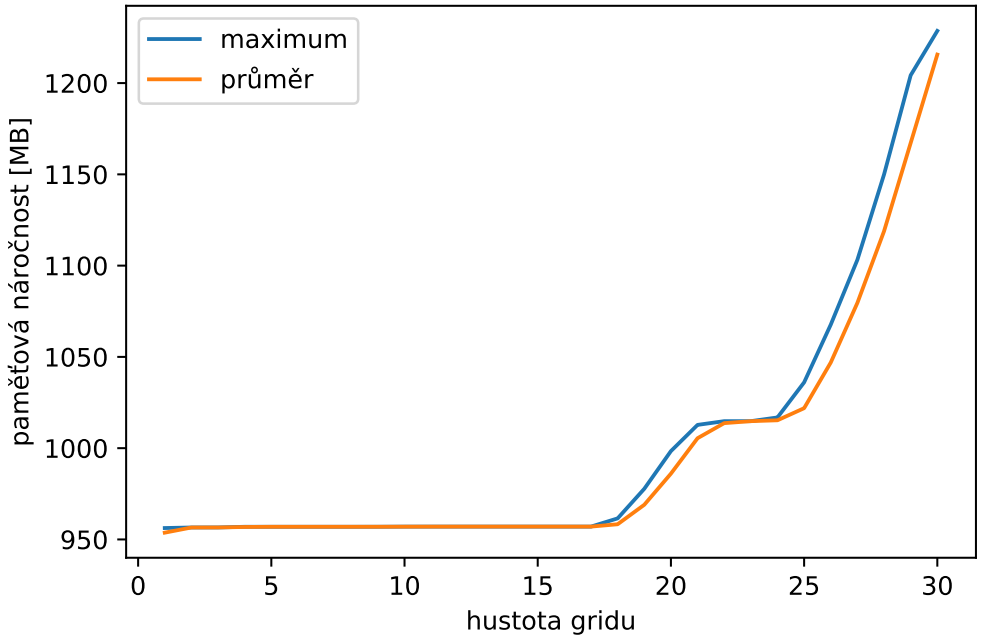
Obrázek 3.9: Závislost času výpočtu na hustotě gridu.



Obrázek 3.10: Závislost s na hustotě gridu.



Obrázek 3.11: Závislost s (logaritmická škála) na hustotě gridu.

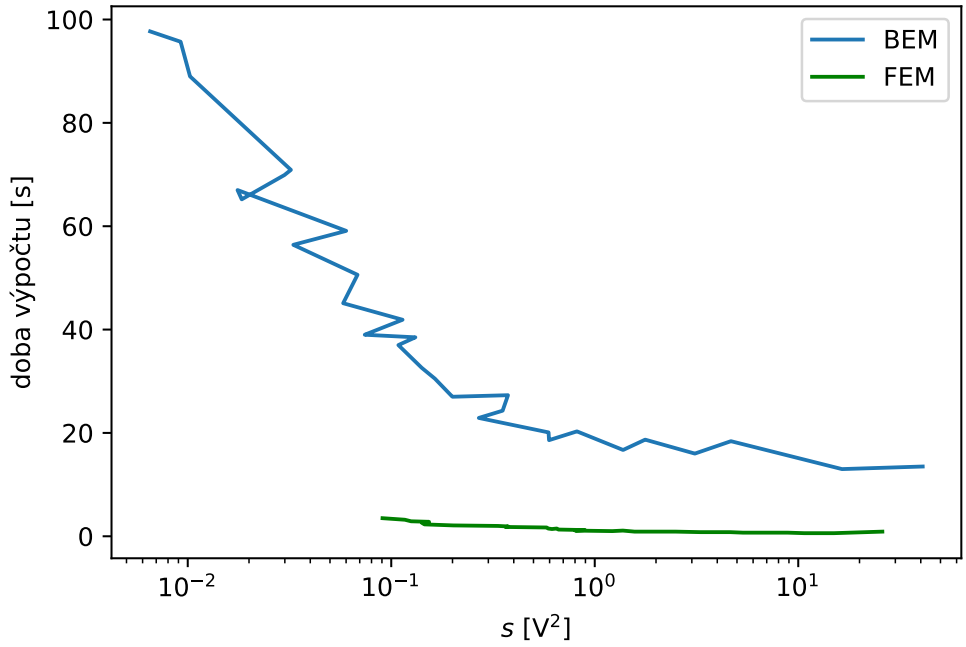


Obrázek 3.12: Závislost paměťové náročnosti výpočtu na hustotě gridu.

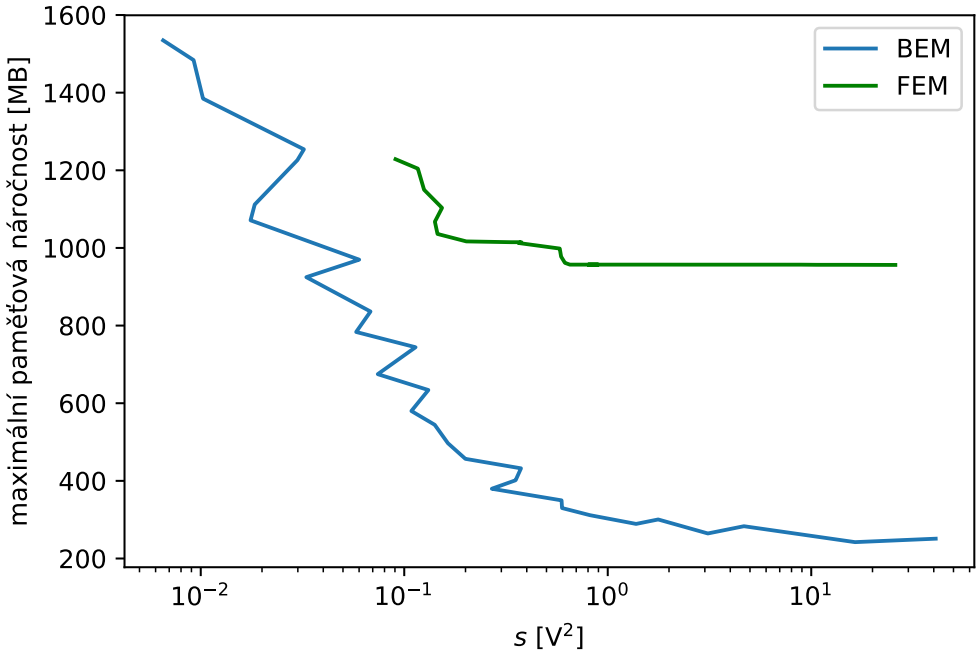
3.3 Srovnání

Jelikož jemnost diskretizace zavádíme pro BEM a FEM různě, ke srovnání obou metod použijeme závislosti na s, které znázorníme na obrázcích 3.13 a 3.14 (lomenná čára spojuje body seřazené podle jemnosti diskretizace).

Jednoznačně pozorujeme, že pro odpovídající hodnoty s má BEM vyšší dobu výpočtu, ale nižší paměťovou náročnost. Vzhledem k prudkému nárůstu paměťové náročnosti FEM pro rostoucí jemnost meshe se nám však nepodařilo minimalizovat s na hodnoty získané pomocí BEM, aniž bychom překročili alokovanou paměť. Při použitém výpočetním uspořádání tedy s BEM dosahujeme řádově nižších hodnot s než s FEM.



Obrázek 3.13: Srovnání BEM a FEM pro závislost doby výpočtu na s.



Obrázek 3.14: Srovnání BEM a FEM pro závislost maxima paměťové náročnosti výpočtu na s.

4. Modelace iontové pasti v programu Gmsh

4.1 Geometrické zavedení pasti

Vytvoříme model studované iontove pasti pomocí geometrických primitiv

- $k = 22$ souosých válců $\{V_i\}_1^k$ o poloměru $r = 0.5$, polovýšce $v = 18$, se středy ve vrcholech pravidelného k-úhelníku se středem $\boldsymbol{s}$, s opsaným poloměrem $q = 5.5$, ležící v rovinně α, jehož osa $\boldsymbol{z}$ je rovnoběžná s osami válců,
- 2 souosá meziválcí ("Endcaps") $\{M_j\}_1^2$ o vnějším poloměru $r_2 = 3.5$ a vnitřním poloměru $r_1 = 3.2$, výšce $d = 9$, vzájemné vzdálenosti $s = 32$, se středy na ose $\boldsymbol{z}$, osami rovnoběžnými s osou $\boldsymbol{z}$,
- "Bounding Box", krychle K se středem v $\boldsymbol{s}$, rovnoběžná s rovinou α a osou $\boldsymbol{z}$, o délce strany $a =$.

$\{V\}_1^k$, $\{M\}_1^2$ a K jsou omezené množiny v R^3 s nenulovou objemovou mírou. Hranici iontové pasti zavedeme jako

$$\Gamma = \bigcup_{i=1}^{k} \partial V_i \cup \partial M_1 \cup \partial M_2 \cup \partial K. \tag{4.1}$$

Prostorový objem iontové pasti zavedeme jako

$$\Omega = K \setminus (\bigcup_{i=1}^{k} V_i \cup M_1 \cup M_2). \tag{4.2}$$

Dodržujeme konvenci, že plochy jsou orientované tak, aby normálový vektor mířil směrem z Ω.

4.2 Konstrukce v programu Gmsh

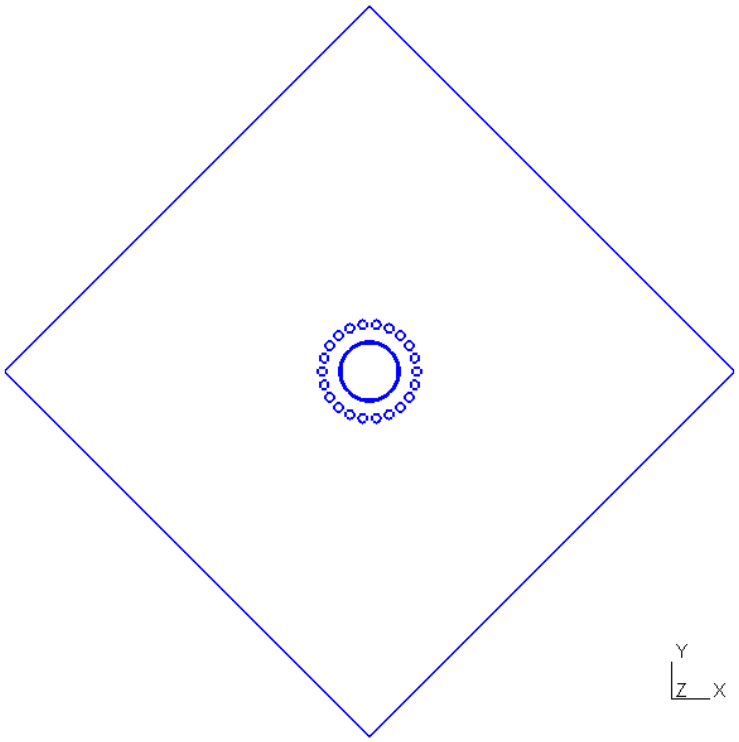
Zdrojový kód k této kapitole uvádíme na straně A.1, ve formátu *Past1.geo* jej přikládáme elektronicky.

Gmsh obsahuje interní programovací jazyk (viz Geuzaine a Remacle), který umožňuje definovat

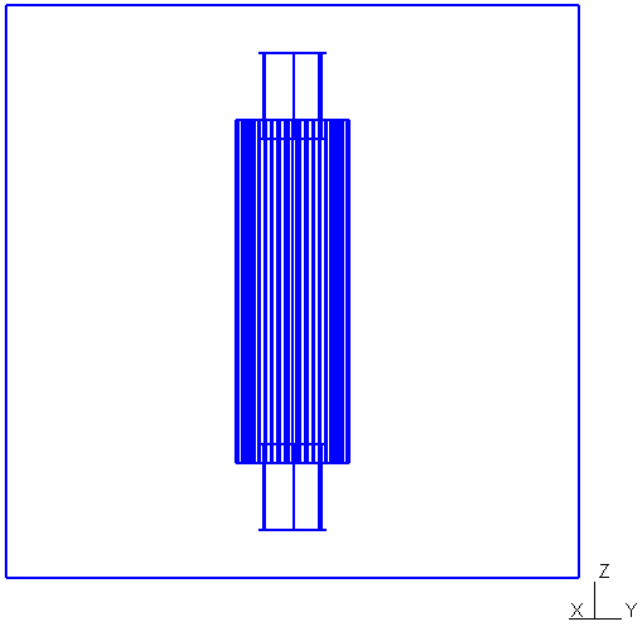
- bod,
- úsečku mezi 2 body,
- kružnicové oblouky mezi 2 body a se středem ve třetím bodě,
- křivkové smyčky z n úseček či kružnicových oblouků,
- rovinné plochy s „děrami“ pomocí křivkových smyček,
- válcové plochy pomocí křivkové smyčky,

- fyzické plochy, vzniklé sloučením několika ploch,
- plochové smyčky z n ploch,
- objemy s „děrami“ pomocí plochových smyček,
- fyzické objemy.

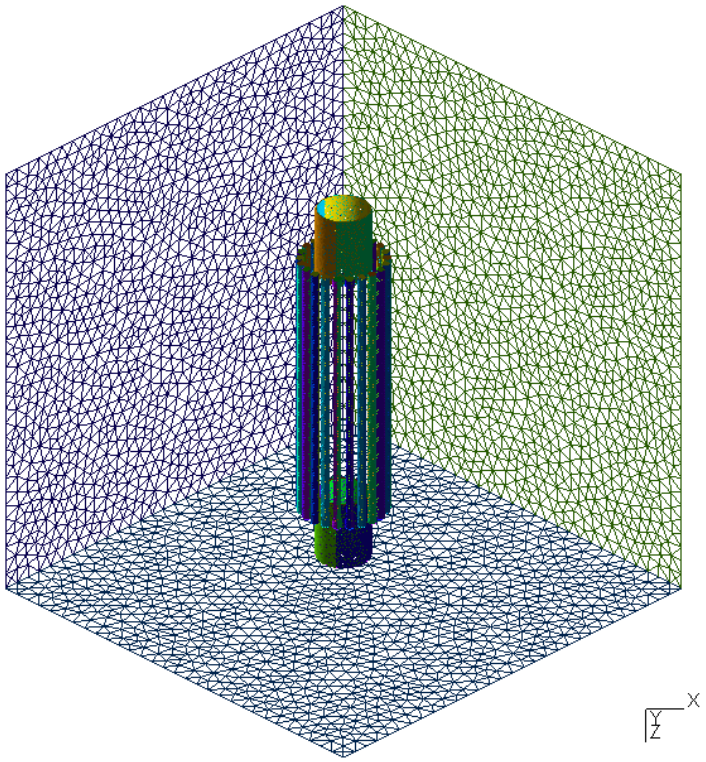
Rovinné plochy pasti lze přirozeně triangulizovat, válcové plochy je třeba rozdělit na $m = 3$ shodné válcové oblouky (Gmsh nedovoluje triangulizovat válcové oblouky o středovém úhlu větším než π).



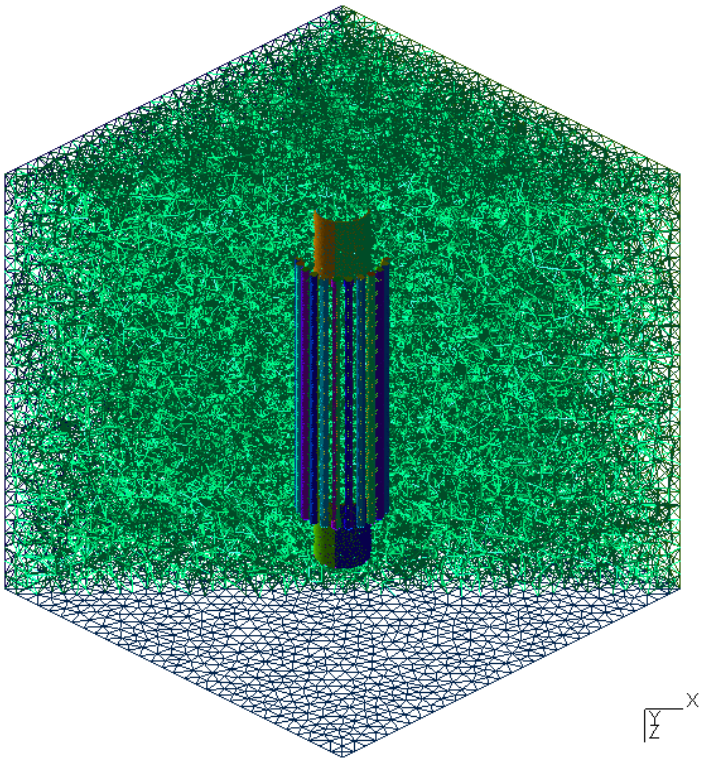
Obrázek 4.1: 1D diskretizace pasti, pohled ve směru $-\boldsymbol{z}$



Obrázek 4.2: 1D diskretizace pasti, pohled ve směru $(-1, -1, 0)$



Obrázek 4.3: 2D diskretizace pasti



Obrázek 4.4: 3D diskretizace pasti, šikmý řez

5. Výpočet elektrického potenciálu a elektrické intenzity iontové pasti

Konečně přistoupíme k samotnému výpočtu elektrického potenciálu a pole iontové pasti. Importujeme mesh vytvořený v kapitole 4 a použijeme obdobné postupy v Bempp a FEniCS jako v kapitole 3.

5.1 Výpočet pomocí BEM, za použití knihovny Bempp

Příslušný zdrojový kód ve formátu *Ion_Trap_BEM.ipynb* přikládáme elektronicky.

Elektrický potenciál radiofrekvenční inotové pasti lze rozložit na 2 lineárně nezávislé složky

- alternující potenciál, který získáme, požijeme-li hraniční podmínku $u = (-1)^n 40V$ na válcích V_n, $u = 0$ na endcapech a $u = 0$ na bounding boxu,
- stálý potenciál, který získáme, použijeme-li hraniční podmínku $u = 0$ na válcích V_n, $u = 1$ na endcapech a $u = 0$ na bounding boxu, viz obrázek 5.1.

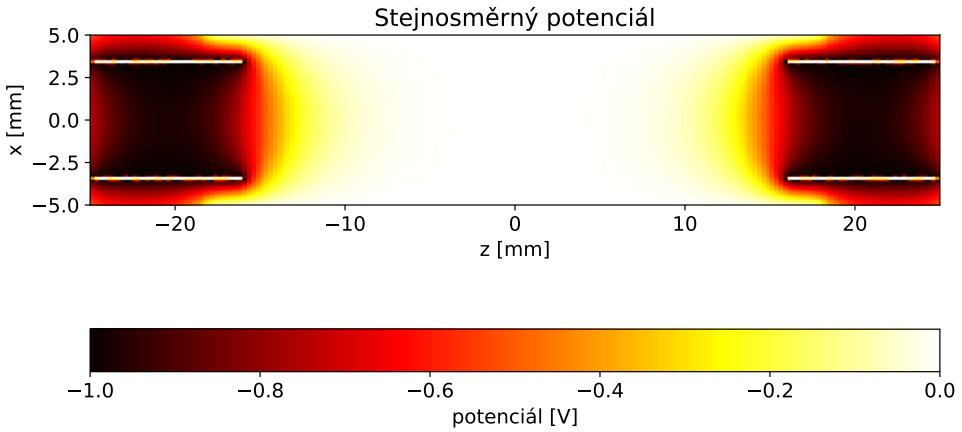
Kromě výpočtu elektrického potenciálu (znázorněného na obrázku 5.2) v této kapitole určujeme elektrickou intenzitu (rovněž rozloženou na 2 lineárně nezávislé složky). Bempp ve verzi 3.3.2 nám ji umožňuje určit pomocí operátorů *gradientu* hledané funkce (v rámci Bempp fungují obdobně jako operátory potenciálu, pouze vracejí vektorové hodnoty namísto skalárních).

```
slp_gradPot =
    bempp.api.operators.potential.laplace.single_layer_gradient(
    dp0_space, points)
dlp_gradPot =
    bempp.api.operators.potential.laplace.double_layer_gradient(
    p1_space, points)
gradU_evaluated = slp_pot * neumann_fun - dlp_pot * dirichlet_fun
```

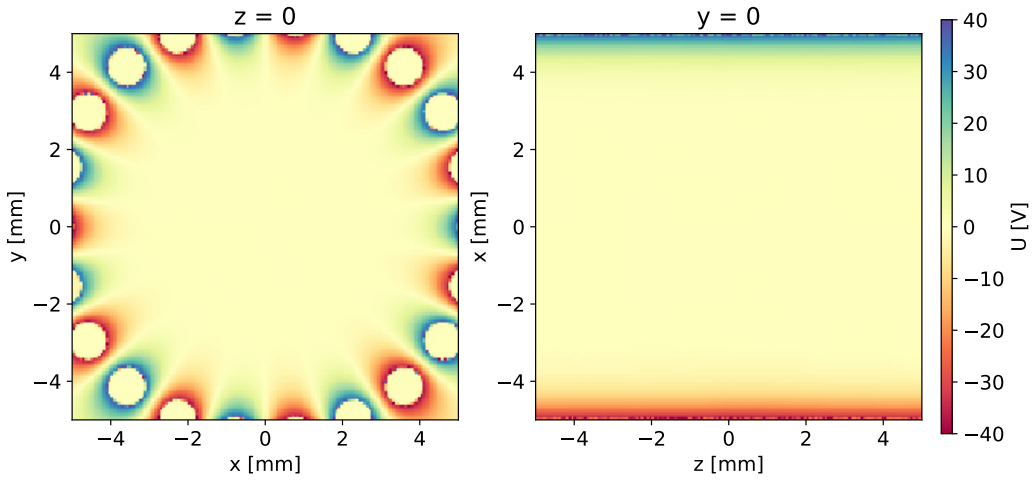
Přičemž využijeme vztahu

$$\boldsymbol{E} = -\nabla u \tag{5.1}$$

Znalost elektrické intenzity později použijeme v kapitole 6.



Obrázek 5.1: Elektrický potenciál endcapů, řez rovinou $y = 0$.



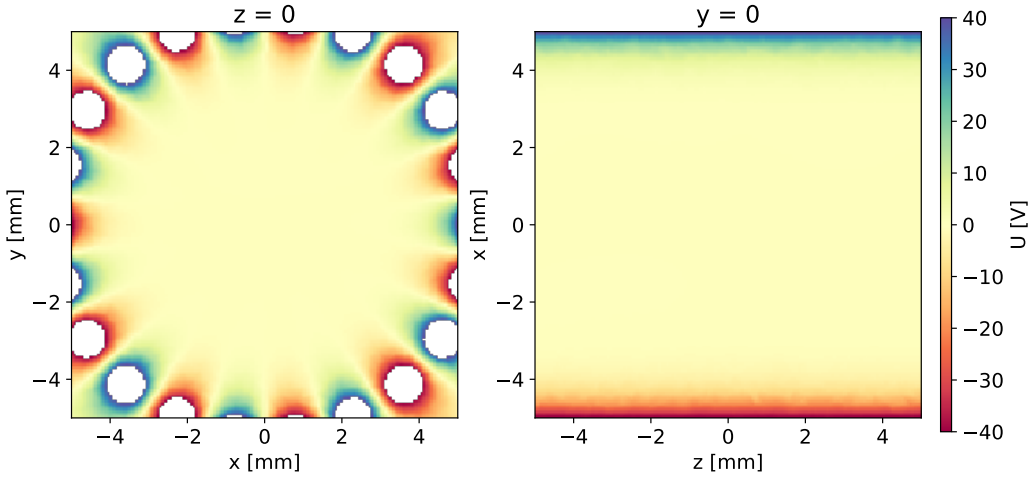
Obrázek 5.2: Elektrický potenciál iontové pasti (BEM), řezy rovinami $z = 0$ a $y = 0$

5.2 Výpočet pomocí FEM, za použití knihovny FEniCS

Příslušný zdrojový kód ve formátu *Ion_Trap_FEM.ipynb* přikládáme elektronicky.

Při výpočtu elektrického pole pomocí FEM postupujeme obdobně jako v předchozích příkladech - získáme funkci $u(\boldsymbol{x})$, která vyhodnotí elektrický potenciál v libovolném bodě Ω. Chceme rovněž získat i funkci elektrické intenzity $\boldsymbol{E}(\boldsymbol{x})$ v libovolném bodě Ω - FEniCS nám ji umožňuje nalézt pomocí gradientu $u(\boldsymbol{x})$.

Výsledky znázorníme na obrázku 5.3.



Obrázek 5.3: Elektrický potenciál iontové pasti (FEM), řezy rovinami $z = 0$ a $y = 0$

Pouhým okem pozorujeme u efektivního potenciálu určeného pomocí FEM podstatně nižší hladkost, než u toho určeného pomocí BEM.

6. Simulace pohybu nabité částice v iontové pasti za použití BEM a FEM, srovnání výsledků

Na Ω lze řešit Newtonovu pohybovou rovnici

$$m\ddot{\boldsymbol{r}} = q\boldsymbol{E}(\boldsymbol{r}, t), \tag{6.1}$$

známe-li $\boldsymbol{E}(\boldsymbol{r}, t)$ pro každý bod Ω.

K numerickému řešení použijeme Runge-Kuttovu metodu 4. řádu.

```
def RK4 (a, x0, v0, t0, dt):
   k0 = dt * v0
   l0 = dt * a (x0, t0)
   k1 = dt * (v0 + 0.5 * l0)
   l1 = dt * a (x0 + 0.5 * k0, t0 + 0.5 * dt)
   k2 = dt * (v0 + 0.5 * l1)
   l2 = dt * a (x0 + 0.5 * k1, t0 + 0.5 * dt)
   k3 = dt * (v0 + l2)
   l3 = dt * a (x0 + k2, t0 + dt)
   x = x0 + (k0 + 2 * k1 + 2 * k2 + k3) / 6.0
   v = v0 + (l0 + 2 * l1 + 2 * l2 + l3) / 6.0
   t = t0 + dt
   return (x, v)
```

Při řešení rovnice (6.1) se věnujme konkrétnímu teoretickému uspořádání:

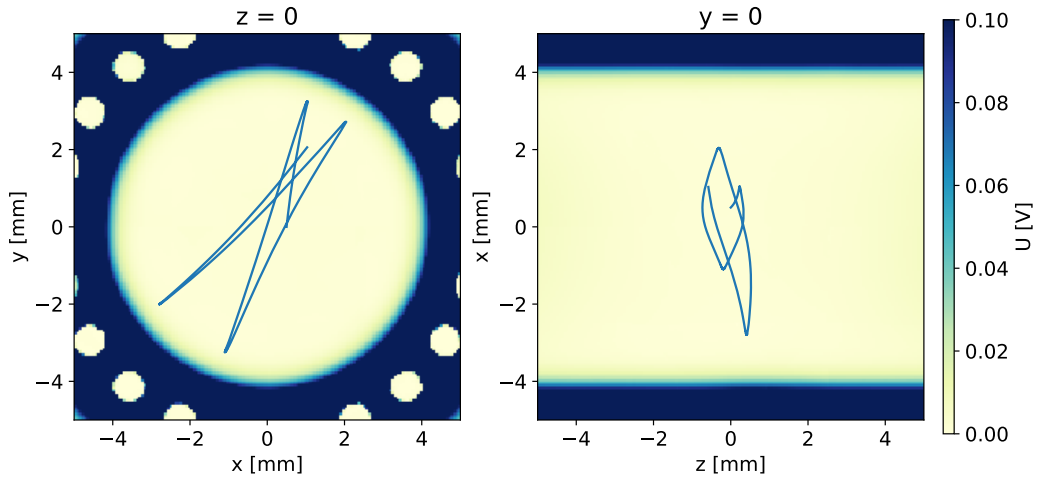
- na $k = 22$ válců pasti zavedeme střídavé napětí o amplitudě 40 V na sudých a -40 V na lichých válcích, frekvenci $f = 27$ MHz,
- na endcapy zavedeme stejnosměrné napětí -1 V,
- bounding box uzemníme,
- simulujeme pohyb iontu D^- o rychlosti odpovídající teplotě 10 K.

6.1 Pohyb částice pro BEM

Používáme stejný zdrojový kód jako v předchozí kapitole.

Z předešlé kapitoly známe hodnoty elektrické intenzity (za použití BEM) na 3-rozměrné bodové síti, veškeré další hodnoty pro body uvniř kvádru sítě lze získat interpolací, kterou lze snadno implementovat pomocí knihovny SciPy.

Známe-li i hodnoty elektrické intenzity, lze v bodech vyhodnovací mříže určit efektivní potenciál podle rovnice (2.11). Pohyb částice v efektivním potenciálu zobrazíme na obrázku 6.1.



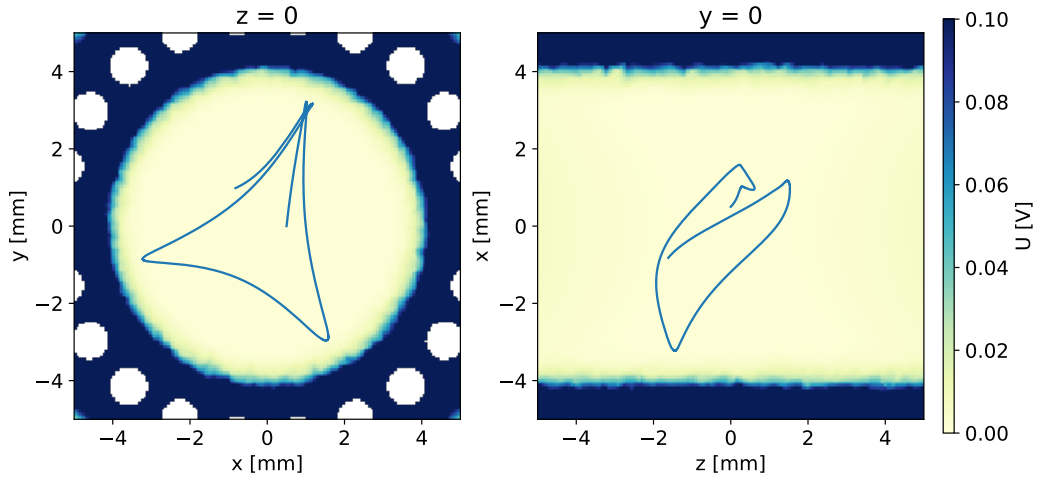
Obrázek 6.1: Efektivní potenciál iontové pasti (BEM), řezy rovinami $z = 0$ a $y = 0$, a trajektorie iontu

6.2 Pohyb částice pro FEM

Používáme stejný zdrojový kód jako v předchozí kapitole.

Za použití FEM zjistíme hodnotu elektrické intenzity v každém bodě objemu pasti Ω numerickou derivací potenciálu u (viz kapitola5).

Pohyb částice v efektivníme potenciálu popisuje obrázek 6.2.

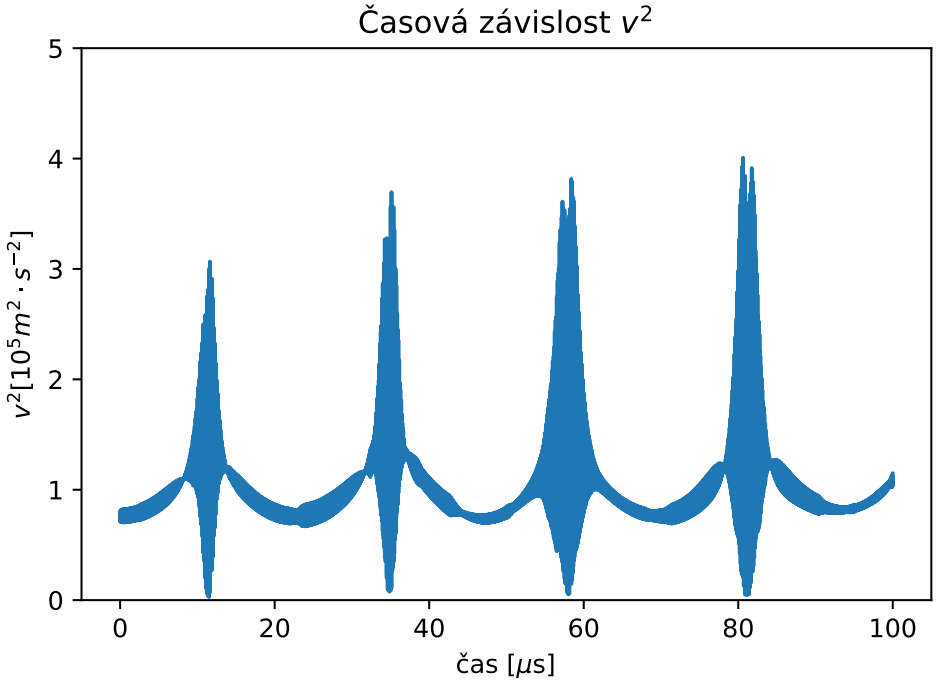


Obrázek 6.2: Efektivní potenciál iontové pasti (FEM), řezy rovinami $z = 0$ a $y = 0$, a trajektorie iontu

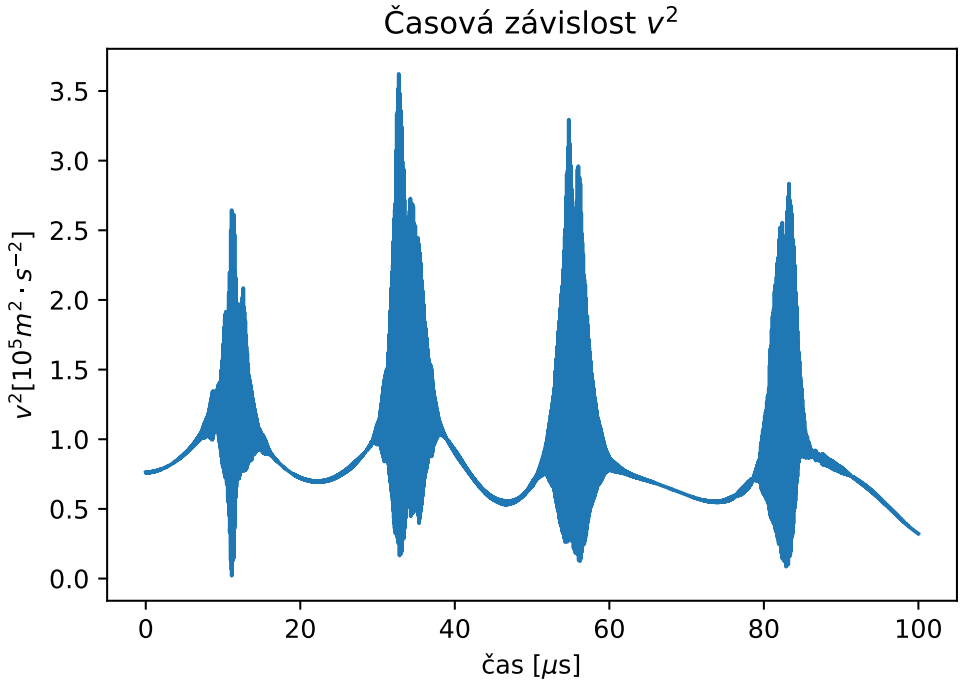
6.3 Srovnání BEM a FEM

Pouhým okem pozorujeme nižší hladkost efektivního potenciálu v blízkosti tyčí pro FEM, jak jsme očekávali z kapitoly 3 a což se projeví níže.

Pro srovnání přesnosti numerického výpočtu elektrické intenzity uvnitř pasti studujeme, jak se s časem vyvíjí kvadrát rychlosti $|\boldsymbol{v}|^2$ pohybující se částice, jemuž je úměrná kinetická energie, viz obrázky 6.3 a 6.4.



Obrázek 6.3: Časový vývoj v^2 pro BEM.



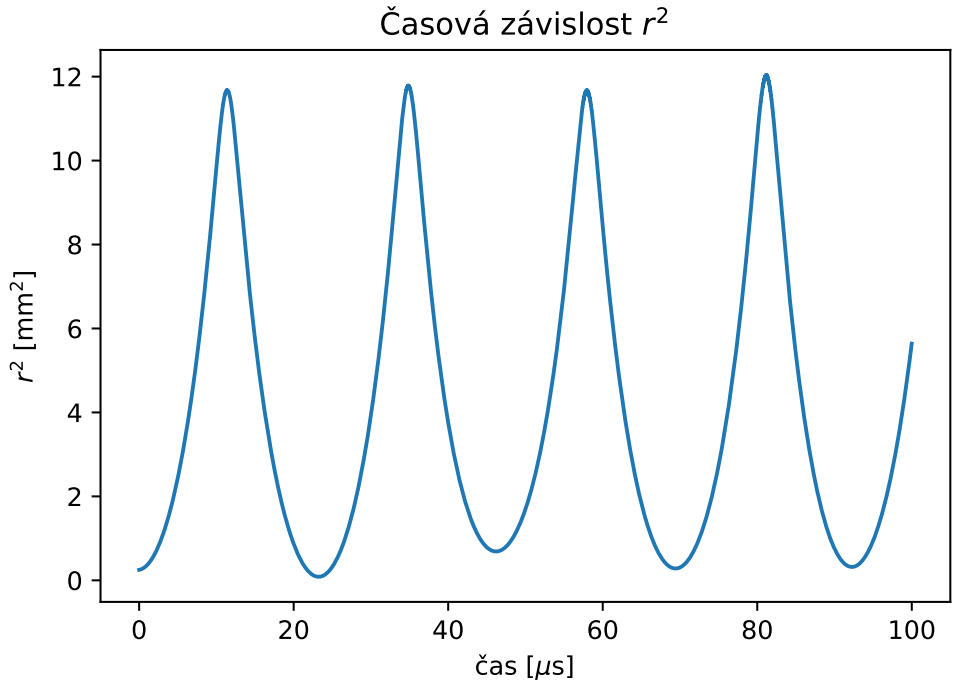
Obrázek 6.4: Časový vývoj v^2 pro FEM.

Vidíme, že pro BEM se částice průběžně navrací do okolí původní polohy, kdežto pro FEM nikoliv.

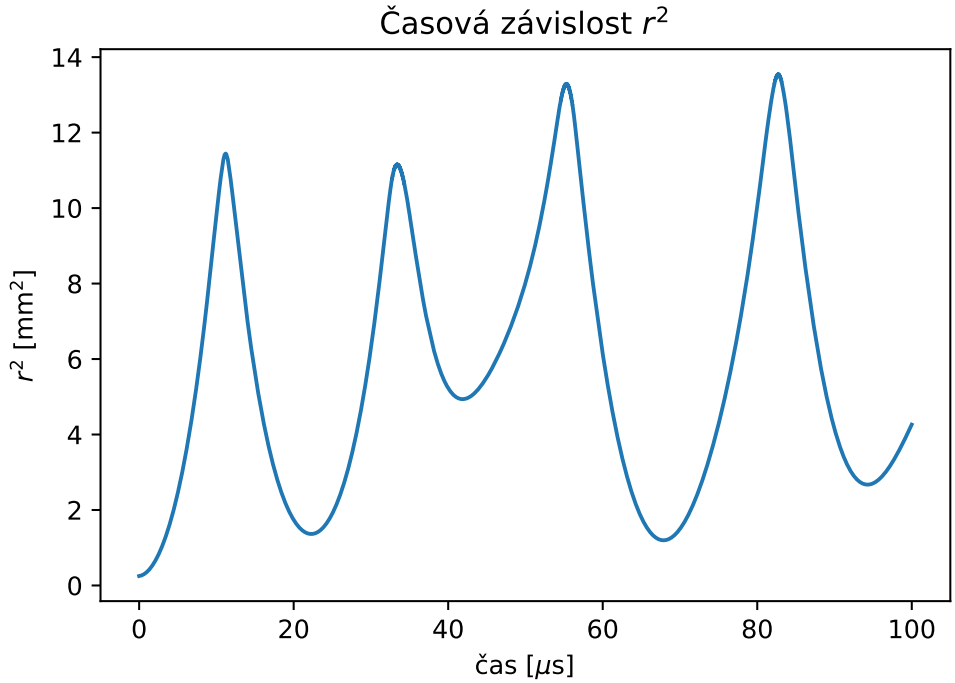
Výsledky odpovídají našim očekáváním, která jsme předložili v kapitole2. V oblasti nízkého efektivního potenciálu se částice pohybuje zhruba s konstantní rychlostí, s rostoucím efektivním potenciálem prudce narůstají rovněž rychlostní oscilace, než se částice odrazí a navrací se zpět do oblasti nízkého efektivního potenciálu. V případě adiabatického pohybu by se kinetická energie před a po odrazu měla zachovávat, což podrobněji diskutujeme níže.

Pro BEM pozorujeme oscilace i v oblasti nízkého efektivního potenciálu, ne však pro FEM. To odpovídá výsledkům z kapitoly 3, kde na obrázcích 3.2 a 3.8 vidíme, že ačkoliv pro BEM je celková kvadratická odchylka nižší, odchylka od teoretických hodnot je zhruba stejná na celé množině evaluace. Oproti tomu pro FEM odchylka od teoretických hodnot klesá a narůstá s klesajícím a rostoucím efektivním potenciálem.

Rovněž srovnáme, jak se s časem vyvíjí kvadrát polohového vektoru $|\boldsymbol{r}|^2$, což znázorníme na obrázcích 6.5 a 6.6.

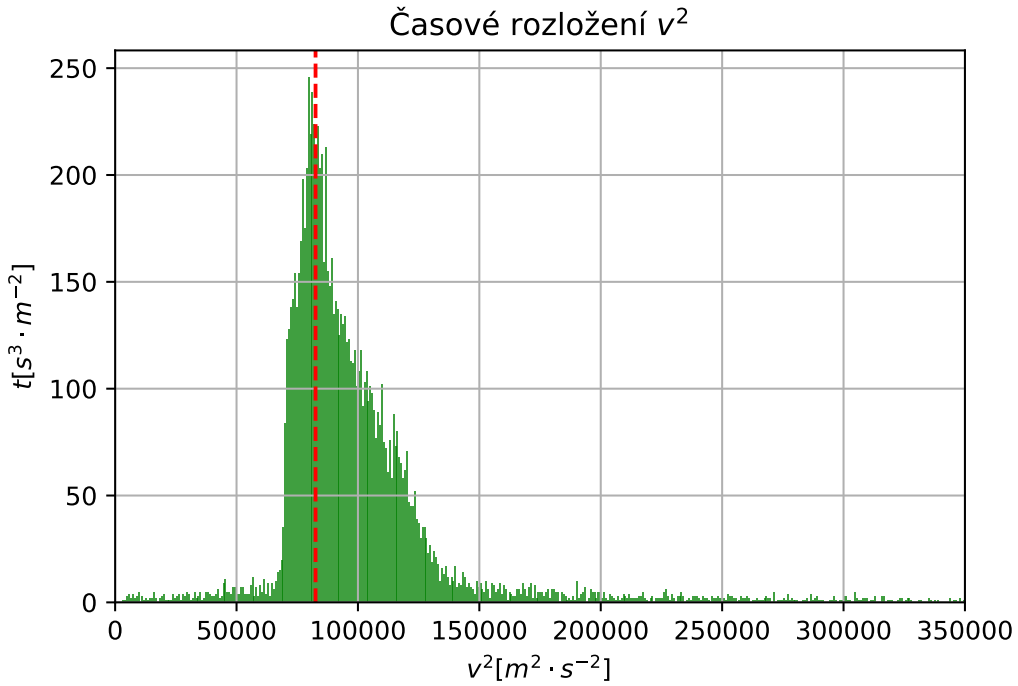


Obrázek 6.5: Časový vývoj r^2 pro BEM.

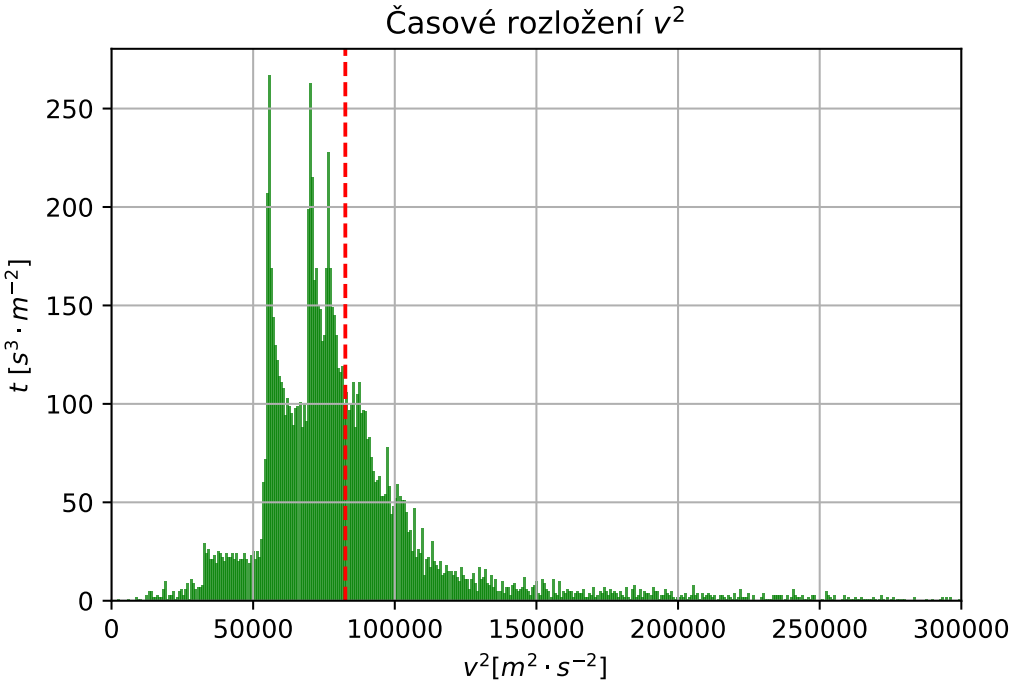


Obrázek 6.6: Časový vývoj r^2 pro FEM.

Dále pro BEM i FEM znázorníme histogram časového rozložení kvadrátu rychlosti $|\boldsymbol{v}|^2$ na obrázcích 6.7 a 6.8.



Obrázek 6.7: Časový histogram v^2 pro BEM s vyznačenou počáteční rychlostí



Obrázek 6.8: Časový histogram v^2 pro FEM s vyznačenou počáteční rychlostí

Pro BEM histogram odpovídá teoretickému tvaru, jak je uveden v Gerlich (1992) - 1 peak, odpovídající nejpravděpodobnější rychlosti. V případě adiabatického pohybu se jedná o počáteční rychlost, což odpovídá našemu případu.

Pro FEM je situace poněkud odlišná. Pozorujeme i další peaky, odpovídající nižším rychlostem. Tyto peaky jsou patrně způsobeny zpomalením částice při odrazu na efektivním potenciálu vlivem neadiabacity pohybu. To odpovídá nižší přesnosti výpočtu elektrického pole pomocí FEM, než pomocí BEM, jak diskutujeme v kapitolách 4 a 5.

Závěr

Podrobně jsme prozkoumali možnosti BEM a FEM, implementované v knihovnách Bempp a FEniCS, pro řešení Laplaceovy úlohy (1.1), na jednotkové kouli i v okolí radiofrekvenční iontové pasti. Jednoznačně jsme potvrdili, že na použitém systému dosahuje BEM řádově vetší přesnosti než FEM, byť za cenu zvýšené doby výpočtu.

Veškeré výpočty jsme prováděli na běžném stolním počítači, na kterém byl ve Virtual Boxu spuštěn operační systém Ubuntu 16.04, kterému jsme dedikovali 11264 MB paměti a 4 jádra procesoru. Předpokládáme, že přesnějších výsledků a v kratším čase bychom dosáhli na vhodnějším zařízení, to však je již nad rámec této práce.

Seznam použité literatury

BemppProject. Řešení laplaceovy rovnice pomocí bempp, ukázkový příklad. `http://nbviewer.jupyter.org/urls/bitbucket.org/bemppsolutions/bempp-tutorials/raw/master/notebooks/laplace_interior_dirichlet.ipynb`. Online; navštíveno 6. května 2018.

Gerlich, D. (1992). Inhomogeneous electrical radio frequency fields: A versatile tool for the study of processes with slow ions. *Adv. Chem. Phys. Ser.*, **LXXXII**.

Geuzaine, C. a Remacle, J.-F. Dokumentace k programu gmsh. `http://gmsh.info/doc/texinfo/gmsh.html`. Online; navštíveno 6. května 2018.

Steinbach, O. (2008). *Numerical Approximation Methods for Elliptic Boundary Value Problems*. 1. Springer-Verlag, New York. ISBN 978-0-387-68805-3.

Śmigaj, W., Arridge, S., Betcke, T., Phillips, J. a Schweiger, M. (2015). Solving boundary integral problems with bem++. *ACM Trans. Math. Software*, **41**.

Seznam obrázků

Seznam použitých zkratek

- BEM - boundary element method, metoda hraničních prvků,
- FEM - finite element method, metoda konečných prvků,
- Ω - množina, na které hledáme řešení,
- Γ - hranice množiny Ω,
- D^- - záporně nabitý ion deuteria,
- ω - úhlová rychlost rf pole.

A. Přílohy

A.1 Zdrojový kód Gmsh

Na následujících stránkách uvádíme zdrojový kód modelace iontové pasti v interním programovacím jazyce programu Gmsh. Zdrojový kód ve formátu *past1.geo* přikládáme elektronicky.

```
// Gmsh project created on Tue Apr 19 15:48:08 2016
// Dimensions in millimeters

// Parametry pasti

q = 5.5 ; // polomer iontove pasti

r = 0.5 ; // polomer valce
v = 18 ;  // polovyska valce
b = 0.4 ; // jemnost meshe

k = 22 ;  // pocet valcu

x0 = 0 ;  // pocatecni hodnota x
y0 = 0 ;  // pocatecni hodnota y

// Endcaps

s = 32 ;    // vzdalenost endcapu
d = 9 ;     // vyska endcapu
t1 = 3.3 ; // vnitrni polomer
t2 = 3.5 ; // vnejsi polomer

// Parametry geometrie

fi = 0 ;    // uhel
A = 0 ;     // pocatecni uhel
h = v ;     // vyska (na ose z)

// x, y - polohove indexy
// f - bodovy index stredu prave vytvarene kruznice

m = 3 ;     // pocet kruznicovych dilu
n = 0 ;     // bodovy index
l = 0 ;     // krivkovy index
o = 0 ;     // index krivkovych smycek
p = 0 ;     // povrchovy index
di = 4 ;    // domenovy index
sl = 4 ;    // index povrchove smycky,
```

```
// 1 pro bounding box, 2 a 3 pro end caps,
// 4 az 25 pro valce

// Bounding box

xbb = 30 ;  // x−ova polosirka bounding boxu
ybb = 30 ;  // y−ova polosirka bounding boxu
zbb = 30 ;  // z−ova polosirka bounding boxu

// Vytvori m bodu na kruznici se stredem (x, y, h),
// osou rovnobeznou s osou z, polomerem r, pocatecnim
// uhlem A.

Macro ndil
   For i In {1 : m}
      Point (n + i) = {x + r * Cos (A + 2 * Pi * (i − 1) / m),
                       y + r * Sin (A + 2 * Pi * (i − 1) / m), h, b} ;
   EndFor
   n = n + m ;
Return

// Vytvori m kruznicovych vyseci se stredem f,
// osou rovnobeznou s osou z, polomerem r,
// pocatecnim uhlem A.

Macro okruh
   For i In {1 : m − 1}
      Circle (l + i) = {n − m + i, f, n − m + i + 1} ;
   EndFor
   Circle (l + m) = {n, f, n − m + 1} ;
   l = l + m ;
Return

// Vytvori m usecek mezi 2 kruznicemi

Macro paprsky
   For i In {1 : m}
      Line (l + i) = {n − m + i, n − 2 * m − 1 + i} ;
   EndFor
   l = l + m ;
Return

// Vytvori krivkove smycky na obvodu valce.

Macro obvodO
   For i In {1 : m − 1}
      Line Loop (o + i) = {l − 2 * m + i, l − m + i + 1,
                           − (l − 3 * m + i), − (l − m + i)} ;
```

```
    EndFor
    Line Loop (o + m) = {l − 1 ∗ m, l − m + 1, − (l − 2 ∗ m), − l} ;
    o = o + m ;
Return

// Vytvori povrchy na obvodu valce.

Macro obvodP
    For i In {1 : m}
        Ruled Surface (p + i) = {− (o − m + i)} ;
    EndFor
    p = p + m ;
Return

// Vytvori podstavu valce.

Macro podstava
    o = o + 1 ;
    p = p + 1 ;

    n = n + 1 ;
    f = n ;

    Point (f) = {x, y, h, b} ;

    Call ndil ;
    Call okruh ;

    Line Loop (o) = {l − m + 1 : l} ;
Return

// Vytvori povrch valce.

Macro valec
    h = v ;
    Call podstava ;
    Plane Surface (p) = {− o} ;

    h = − v ;
    Call podstava ;
    Plane Surface (p) = {o} ;

    Call paprsky ;
    Call obvodO ;
    Call obvodP ;

    Physical Surface (di) = {p − m − 1 : p} ;
    Surface Loop (sl) = {p − m − 1 : p};
```

```
    sl = sl + 1;
    p = p + 1 ;
Return

// Vytvori podstavu Endcaps.

Macro podstavaEC
    n = n + 1;
    f = n;
    Point (f) = {x, y, h, b};

    Call ndil;
    Call okruh;

    o = o + 1;
    Line Loop (o) = {l - m + 1 : l};
Return

// Vytvori opacne orientovane povrchy na obvodu valce.

Macro obvod_P
    For i In {1 : m}
        Ruled Surface (p + i) = {(o - m + i)} ;
    EndFor
    p = p + m ;
Return

// Vytvori endcap.

Macro EndCap
    r = t2;

    Call podstavaEC;

    h = h - d;

    Call podstavaEC;

    Call paprsky;
    Call obvodO;
    Call obvodP;

    h = h + d;
    r = t1;

    Call podstavaEC;

    h = h - d;
```

```
    Call podstavaEC;

    Call paprsky;
    Call obvodO;
    Call obvod_P;

    p = p + 1;
    Plane Surface (p) = {- (o - 2 * m - 3), (o - 1 * m - 1)};
    p = p + 1;
    Plane Surface (p) = {(o - 2 * m - 2), - (o - 1 * m - 0)};
Return

// Vytvori endcaps.

Macro EndCaps
    x = x0;
    y = y0;
    h = 0.5 * s + d;

    Call EndCap;

    h = - 0.5 * s;

    Call EndCap;

    Physical Surface (2) = {p - 4 * (m + 1) + 1 : p};

    Surface Loop (2) = {p - 4 * (m + 1) + 1 : p - 2 * (m + 1)};
    Surface Loop (3) = {p - 2 * (m + 1) + 1 : p - 0 * (m + 1)};
Return

// Vytvori bounding box.

Macro okruhBB
    For i In {1 : m - 1}
        Line (l + i) = {n - m + i, n - m + i + 1} ;
    EndFor
    Line (l + m) = {n, n - m + 1} ;
    l = l + m ;
Return

Macro podstavaBB
    o = o + 1 ;
    p = p + 1 ;

    n = n + 1 ;
    f = n ;
```

```
    Point (f) = {x, y, h, b} ;

    Call ndil ;
    Call okruhBB ;

    Line Loop (o) = {l - m + 1 : l} ;
Return

Macro obvodPbb
    For i In {1 : m}
        Plane Surface (p + i) = {(o - m + i)} ;
    EndFor
    p = p + m ;
Return

Macro valecBB
    h = v;
    Call podstavaBB ;
    Plane Surface (p) = {o} ;

    h = - v;
    Call podstavaBB ;
    Plane Surface (p) = {- o} ;

    Call paprsky ;
    Call obvodO ;
    Call obvodPbb ;

    Physical Surface (1) = {p - m - 1 : p} ;
    Surface Loop (1) = {p - m - 1 : p};
    Volume (1) = {1, 2 : k + 3};
    Physical Volume (1) = {1};
Return

// Vytvarime mesh valcu.

For j In {1 : k}
    A = 2 * Pi * (j - 1) / k + fi ;

    x = x0 + q * Cos (A) ;
    y = y0 + q * Sin (A) ;

    Call valec ;

    di = di + 1 ;
EndFor
```

// Vytvarime mesh end caps.

b = 0.5 ;

Call EndCaps;

// Vytvarime mesh bounding boxu.

A = 0 ;
fi = 0 ;

m = 4 ;
x = x0 ;
y = y0 ;
v = zbb ;
r = Sqrt (xbb * xbb + ybb * ybb) ;
b = 2.0 ;

Call valecBB ;

A.2 Zdrojový kód ke kapitole 2

```
In [1]: import numpy as np
        import logging
        import time
        import memory_profiler
        import matplotlib
        from functools import partial

        from memory_profiler import memory_usage
        from matplotlib import pyplot as plt
        from matplotlib.colors import LogNorm
        from IPython.display import Image

        matplotlib.rcParams.update({'font.size': 14})
```

```
In [2]: # Eulerova metoda
        def smE (a, x0, v0, t, dt = 0.01):

            v = v0 + a (x0, t) * dt
            x = x0 + v * dt

            return (x, v)
```

```
In [3]: # Numerická integrace metodou Mt,
        # trajektorie o n bodech,
        # n * k vyhodnocovacích bodů
```

```
        def intg1 (a, Mt, k, n, x, v, t, dt):

            x1 = x [0]
            v1 = v [0]
            t1 = t [0]

            for j in range (1, n):
                for i in range (0, k):
                    xv = Mt (a, x1, v1, t1, dt)
                    x1 = xv [0]
                    v1 = xv [1]
                    t1 = t1 + dt

                x [j] = x1
                v [j] = v1
                t [j] = t1
```

```
In [4]: # Konstantní el. pole
        def E1 (b): # b [V/m]
            def E (x):
                return b
            return E
```

```
In [5]: # Násobení skaláru a funkce
        def sxf (s, f):
            def h (x, t):
                return s * f (x, t)
            return h
```

```
In [6]: # Násobení cosinu a funkce
        def cxf (w, f):
            def h (x, t):
                return np.cos (w * t) * f (x)
            return h
```

```
In [7]: # Efektivní potenciál
        def effV (q, m, w, E):
            def V (x):
                return q / (4.0 * m * w ** 2) * (E (x)) ** 2
            return V
```

```
In [8]: # Pohyb částice
        def motn (q, m, E, Mt, n, x0, v0, t0, dt = 0.001):

            x = np.empty (n)
            v = np.empty (n)
            t = np.empty (n)

            x [0] = x0
```

```
            v [0] = v0
            t [0] = t0

            intg1 (sxf (q/m, E), Mt,
                   10, n, x, v, t, dt)

            return (t, x, v)
```

```
In [9]: rv = motn (1, 1, cxf (20, E1 (100)), smE,
                   100000, 1.0, 0.0, 0.0, 0.0001)
```

```
In [10]: # 2D bodová mříž
         def pnt (k, y0, y1):
             plot_grid = np.mgrid [0:10:k*1j, y0:y1:k*1j]
             return np.vstack((plot_grid[0].ravel(),
                               plot_grid[1].ravel(),
                               np.zeros (plot_grid[0].size)))
```

```
In [11]: # Vyhodnocení funkce na bodech mříže
         def ePnt (f_eval, f, p):
             for i in range (0, p [0].size):
                 f_eval [0, i] = f (p [:, i])
```

```
In [12]: p = pnt (150, 0, 2)

         V_eval = np.empty ((1, 150*150))

         ePnt (V_eval, effV (1, 1, 20, E1 (100)), p)
```

```
In [13]: def Pl1 (j, rv, V_eval, y1, y2, n):
             plt.plot (rv [0] [:], rv [1] [:], color = 'red')
             plt.ylabel ('x [m]')
             plt.xlabel ('t [s]')

             V_nn = V_eval.reshape ((n, n))

             plt.imshow (V_nn.T, extent=(0, 10, y1, y2),
                         origin='lower', cmap = 'YlGnBu')

             cb1 = plt.colorbar (orientation = 'horizontal',
                                 pad = 0.2)
             cb1.set_label (u'efektivní potenciál [V]')
             plt.title (u'Efektivní potenciál a v něm' +
                        u' se pohybující nabitá částice')

             plt.savefig ('effPot' + str (j) + '.pdf')
```

```
In [14]: Pl1 (0, rv, V_eval, 0, 2, 150)
```

(Viz 2.1.)

```
In [15]: rv2 = motn (1, 1, cxf (20, E1 (100)), smE,
                     100000, 1.0, 0.2, 0.0, 0.0001)

In [16]: Pl1 (1, rv2, V_eval, 0, 5, 150)

In [17]: # Nehomogenní el. pole
         def Enh (b):
             def E (x):
                 return b * x ** 2
             return E

In [18]: rvNh = motn (1, 1, cxf (20, Enh (10)), smE,
                      100000, 1.0, 1.0, 0.0, 0.0001)

In [19]: Vnh_eval = np.empty ((1, 150*150))

         ePnt (Vnh_eval, lambda x: 2 * x [1] ** 2, pnt (150, -3, 3))

In [20]: Pl1 (2, rvNh, Vnh_eval, - 3, 3, 150)
```

(Viz 2.2.)

A.3 Zdrojový kód ke kapitole 3

```
In [1]: import bempp.api
        from dolfin import (Expression,FunctionSpace,
                            TrialFunction,TestFunction,
                            Function, inner, grad, dx,
                            ds,Constant, solve,
                            DirichletBC, Point)
        import mshr
        import numpy as np
        import logging
        import time
        import memory_profiler
        import matplotlib
        from functools import partial

        from memory_profiler import memory_usage
        from matplotlib import pyplot as plt
        from matplotlib.colors import LogNorm
        from IPython.display import Image

        matplotlib.rcParams.update({'font.size': 12})

        logging.getLogger('BEMPP').setLevel(logging.WARNING)

        # grid - mesh studovaného objektu v Bempp
        # mesh - mesh studovaného objektu ve FEniCS
        # points, p - body (2D nebo 3D mříž) na kterých
        # provádíme evaluaci.
```

```
In [2]: # Potenciál multipólu
        from scipy.special import sph_harm
        def mlpl (x, m, n):
            r = np.sqrt(x[0]**2 + x[1]**2 + x[2]**2)
            theta = np.arccos(x[2]/r)
            phi = np.arctan2(x[1], x[0])+np.pi
            return sph_harm (m, n, phi, theta) . real * r ** (n)

        # Konkrétní multipól.
        def mlp (x):
            return mlpl (x, 5, 11)

In [3]: # Známé hodnoty potenciálu na hranici gridu
        def dirichlet_data(x, k, domain_index, result):
            result [0] = mlp (x)

In [4]: # Funkce, která vytvoří potřebné funkční prostory
        # a hraniční operátory na gridu
        def eSpBop (grid):
            # Nespojitý polynomiální prostor řádu 0
            kp = bempp.api.function_space (grid, "DP", 0)
            # Spojitý polynomiální prostor řádu 1
            lp = bempp.api.function_space (grid, "P", 1)

            # Hraniční operátor identity.
            identity = bempp.api.operators.boundary.sparse.identity(
                lp, lp, kp)
            # Hraniční dvouvrstvý potenciálový operátor.
            dlp = bempp.api.operators.boundary.laplace.double_layer(
                lp, lp, kp)
            # Hraniční jednovrstvý potenciálový operátor.
            slp = bempp.api.operators.boundary.laplace.single_layer(
                kp, lp, kp)

            return (kp, lp, identity, dlp, slp)

In [5]: # Funkce, která vytvoří potřebnou Neumannovu a Dirichletovu funkci
        def eNDf (SpBop, DD):
            # Dirichletova stopa na hranici.
            dirichlet_fun = bempp.api.GridFunction (SpBop [1],
                                                    fun = DD)

            rhs = (0.5 * SpBop [2] + SpBop [3]) * dirichlet_fun

            #Neumannova stopa na hranici.
            neumann_fun, info = bempp.api.linalg.cg(SpBop [4],
                                                    rhs, tol=1E-3)

            return (neumann_fun, dirichlet_fun)
```

```
In [6]: # Funkce, která vytvoří potřebné potenciálové operátory
        def ePop (SpBop, p):
            # Jednovrstvý potenciálový operátor.
            slp_pot=bempp.api.operators.potential.laplace.single_layer(
                SpBop [0], p)
            # Dvouvrstvý potenciálový operátor
            dlp_pot=bempp.api.operators.potential.laplace.double_layer(
                SpBop [1], p)

            return (slp_pot, dlp_pot)

In [7]: # Funkce, která provede evaluaci na zadaných bodech
        def uBEM (op, NDf):
            return op [0] * NDf [0] - op [1] * NDf [1]

In [8]: # Funkce, která vyplní pole u_eval hodnotami u v bodech p
        def aMlpBEM (u_eval, p, h):
            # Vytvoří grid s požadovanou jemností
            grid = bempp.api.shapes.sphere (h = h)

            SpBop = eSpBop (grid)

            NDf = eNDf (SpBop, dirichlet_data)

            Pop = ePop (SpBop, p)

            u_eval [:] = uBEM (Pop, NDf)

In [9]: # Pl   - vytvoří výkres pro j <= 0
        # Ins  - určí, zda se provede výpočet teoretické hodnoty
        # asgn - vyplní pole u_eval hodnotami u v bodech p

        def memt (u_eval, teor, Ins, asgn, p, b, dt = 0.1):

            # Určí výpočetní a časovou náročnost
            mem = np.array (memory_usage ((asgn, (u_eval, p, b), {}),
                                          interval = dt))

            # Celková kvadratická odchylka od teoretických hodnot
            s = 0.0

            for i in range (0, p [0]. size):
                if (Ins (p [:, i])):
                    s = s + (u_eval [0, i] - teor [0, i]) ** 2
                else:
                    teor [0, i] = 0

            t = mem.size
            mx = max (mem [:])
```

```
            mn = min (mem [:])
            ms = sum (mem [:])
            ma = ms / float (t)
            me = np.sqrt (sum ((mem [:] - ma) ** 2) / float (t))

            return np.array ([t * dt, s, mx, mn, ma, me])
```

```
In [10]: # Určí časovou a výpočetní náročnost pro různou
         # jemnost gridu
         def arr (u_eval, teor, Ins, asgn, l, p, hf):

             z = np.empty ((7, l))

             for i in range (0, p [0]. size):
                 if (Ins (p [:, i])):
                     teor [0, i] = mlp (p [:, i])
                 else:
                     teor [0, i] = 0

             for i in range (1, l + 1):
                 z [0, i - 1] = i

                 #Minimalizuje dopad zahřátí komponent na měření
                 time.sleep (10)

                 z [1:7, i - 1] = memt (u_eval, teor, Ins, asgn, p, hf (i))

             return z
```

```
In [11]: # Vykreslí vypočtené hodnoty potenciálu a ochylku
         # od teoretických hodnot
         def mlpPl (j, u_eval, teor, n = 150):

             u_nn = u_eval.reshape ((n, n))
             tr = teor.reshape ((n, n))

             fig = plt.figure (figsize = (10, 10))
             plt.imshow (u_nn.T, extent=(- 1, 1, - 1, 1),
                         origin='lower', cmap = "Spectral")
             plt.xlabel (u'x')
             plt.ylabel (u'y')
             plt.clim (- 0.3, 0.3)
             cb1 = plt.colorbar ()
             cb1.set_label (u'potenciál [V]')
             plt.title (u'Numericky vypočtené hodnoty' +
                        u' multipólového potenciálu')
             plt.savefig ("Mlp" + str (j) + ".pdf")

             fig = plt.figure (figsize = (10, 10))
```

```
            plt.imshow ((u_nn - tr).T, extent=(- 1, 1, - 1, 1),
                        origin='lower', cmap = "Spectral")
            plt.xlabel (u'x')
            plt.ylabel (u'y')
            plt.clim (- 0.005, 0.005)
            cb2 = plt.colorbar ()
            cb2.set_label (u'potenciál [V]')
            plt.title (u'Odchylka od teoretických hodnot potenciálu')

            plt.savefig ("Mlp_Err" + str (j) + ".pdf")
```

```
In [12]: def Ins (x):
             return sum (x [:] ** 2) < 0.99
```

```
In [13]: # 2D bodová mříž
         ng = 150

         def pnt (n):
             plot_grid = np.mgrid [-1:1:n*1j, -1:1:n*1j]
             return np.vstack((plot_grid[0].ravel(),
                               plot_grid[1].ravel(),
                               np.zeros (plot_grid[0].size)))
```

```
In [14]: # Koeficient jemnosti gridu
         def hfBEM (i):
             return 1.0 / float (1 + i)
```

```
In [15]: u_eval = np.empty ((1, ng*ng))
         teor = np.empty ((1, ng*ng))

         z = arr (u_eval, teor, Ins, aMlpBEM,
                  30, pnt (ng), hfBEM)
```

```
In [16]: mlpPl (1, u_eval, teor, ng)
```

(Viz 3.1 a 3.2.)

```
In [17]: def t_gPl (j, z):
             plt.plot (z [0, :], z [1, :])
             plt.xlabel ("hustota gridu")
             plt.ylabel (u"doba výpočtu [s]")
             plt.savefig ("time_grid" + str (j) + ".pdf")
```

```
In [18]: def s_gPl (j, z):
             plt.plot (z [0, :], z [2, :])
             plt.xlabel ("hustota gridu")
             plt.ylabel (u"ochylka od teoretických hodnot [V]")
             plt.savefig ("err_grid" + str (j) + ".pdf")
```

```
In [19]: def sl_gPl (j, z):
             plt.plot (z [0, :], z [2, :])
             plt.yscale('log')
             plt.xlabel ("hustota gridu")
             plt.ylabel (u"ochylka od teoretických hodnot [V]")
             plt.savefig ("errl_grid" + str (j) + ".pdf")

In [20]: def m_gPl (j, z):
             plt.plot (z [0, :], z [3, :], label = u"maximum")
             plt.plot (z [0, :], z [5, :], label = u"průměr")
             plt.legend ()
             plt.xlabel ("hustota gridu")
             plt.ylabel (u"paměťová náročnost [MB]")
             plt.savefig ("mem_grid" + str (j) + ".pdf")

In [21]: t_gPl (1,z)
```

(Viz 3.3.)

```
In [22]: s_gPl (1,z)
```

(Viz 3.4.)

```
In [23]: sl_gPl (1, z)
```

(Viz 3.5.)

```
In [24]: m_gPl (1, z)
```

(Viz 3.6.)

```
In [25]: class MyExpression0 (Expression):
             def eval (self, value, x):
                 value [0] = mlp (x)
         g0 = MyExpression0 (degree = 2)

In [26]: def uFEM (bc, V):

             u = TrialFunction (V)
             v = TestFunction (V)
             a = inner (grad(u), grad(v)) * dx
             f = Constant (0.0)
             g = Constant (0.0)
             L = f * v * dx + g * v * ds

             u = Function(V)

             # solve(a == L, u, bc)
             solve(a == L, u, bc,
                   solver_parameters = {
                       'linear_solver':'gmres'})

             return u
```

```
In [27]: def aMlpFEM (u_eval, p, b):

             # Vytvoří mesh a definuje funkční prostor
             domain = mshr.Sphere(Point(0.0, 0.0, 0.0), 1)
             mesh = mshr.generate_mesh(domain, b)
             V = FunctionSpace(mesh, "Lagrange", 1)

             def boundary(x, on_boundary): return on_boundary
             bc = DirichletBC (V, g0, boundary)

             u = uFEM (bc, V)

             n = p [0].size

             for i in range (0, n):
                 try:
                     u_eval [0, i] = u (p [0, i], p [1, i],
                                        p [2, i])
                 except:
                     pass

In [28]: def bFEM (i):
             return 10 + i

In [29]: u_evalFEM = np.empty ((1, ng*ng))
         teorFEM = np.empty ((1, ng*ng))

         zFEM = arr (u_evalFEM, teorFEM, Ins, aMlpFEM,
                     30, pnt (ng), bFEM)

In [30]: mlpPl (2, u_evalFEM, teorFEM, ng)
```

(Viz 3.7 a 3.8.)

```
In [31]: t_gPl (2, zFEM)
```

(Viz 3.9.)

```
In [32]: s_gPl (2, zFEM)
```

(Viz 3.10.)

```
In [33]: sl_gPl (2, zFEM)
```

(Viz 3.11.)

```
In [34]: m_gPl (2, zFEM)
```

(Viz 3.12.)

```
In [35]: plt.plot (z [2, :], z [1, :], label = "BEM")
         plt.plot (zFEM [2, :], zFEM [1, :], label = "FEM")
         plt.legend ()
         plt.ylabel (u"doba výpočtu [s]")
         plt.xscale ("log")
         plt.xlabel (u"odchylka od teoretických hodnot [V]")
         plt.savefig ("Srovnani1.pdf")
```

(Viz 3.13.)

```
In [36]: plt.plot (z [2, :], z [3, :], label = "BEM")
         plt.plot (zFEM [2, :], zFEM [3, :], label = "FEM")
         plt.legend ()
         plt.ylabel (u"maximální paměťová náročnost [MB]")
         plt.xscale ("log")
         plt.xlabel (u"odchylka od teoretických hodnot [V]")
         plt.savefig ("Srovnani2.pdf")
```

(Viz 3.14.)

Erratum, 17. května 2018

Opravy se vztahují na sekce textu, vzorce a ukázky zdrojových kódů na daných stránkách.

Kapitola 3, strany 15 a 16

Větu

> Řešení se však pro srovnání pokusíme získat i numericky, s použitím BEM a FEM, na jednotkové kouli Ω přičemž jako hraniční podmínku použijeme hodnoty Y_{11}^5 na hranici $\Gamma = \partial\Omega$.

nahradit větou

> Řešení se však pro srovnání pokusíme získat i numericky, s použitím BEM a FEM, na jednotkové kouli Ω přičemž jako hraniční podmínku použijeme hodnoty $\Re(Y_{11}^5(\varphi, \theta))$ na hranici $\Gamma = \partial\Omega$.

Definici

$s = \sum_{i=0}^{n-1}(\bar{u} - u_i)^2, \{u_i\}_{i=0}^{n}$ značí hodnoty potenciálu v bodech evaluace.

nahradit definicí

$$s = \sum_{i=0}^{n-1}(u_{t,i} - u_i)^2,$$

> kde $\{u_i\}_{i=0}^{n-1}$ značí numericky určené hodnoty potenciálu v bodech evaluace, $\{u_{t,i}\}_{i=0}^{n-1}$ značí teoretické hodnoty potenciálu v bodech evaluace.

Funkci *memt* nahradit funkcí

```
def memt (u_eval, teor, Ins, asgn, j, p, b, dt = 0.1):

    mem = np.array (memory_usage ((asgn, (u_eval, p, b), {}), interval = dt))

    s = 0.0

    for i in range (0, p [0]. size):
        if (Ins (p [:, i])):
            s = s + (u_eval [0, i] - teor [0, i]) ** 2
        else:
            teor [0, i] = 0

    t = mem.size
    mx = max (mem [:])
    mn = min (mem [:])
    ms = sum (mem [:])
    ma = ms / float (t)
    me = np.sqrt (sum ((mem [:] - ma) ** 2) / float (t))
    return np.array ([t * dt, s, mx, mn, ma, me])
```

Funkci *arr* nahradit funkcí

```
def arr (u_eval, teor, Ins, asgn, l, p, hf):

    for i in range (0, p [0]. size):
        if (Ins (p [:, i])):
            teor [0, i] = mlp (p [:, i])
        else:
            teor [0, i] = 0

    z = np.empty ((7, l))

    for i in range (1, l + 1):
        z [0, i - 1] = i
        time.sleep (10)
        z [1:7, i - 1] = memt (u_eval, teor, Ins, asgn, p, hf (i))
    return z
```

Kapitola 4, strana 27

První odstavec nahradit jako

> Vytvoříme model studované iontove pasti pomocí geometrických primitiv (délkové rozměry uvádíme v milimetrech)
>
> - $k = 22$ souosých válců $\{V_i\}_1^k$ o poloměru $r = 0.5$, polovýšce $v = 18$, se středy ve vrcholech pravidelného k-úhelníku se středem $\boldsymbol{s}$, s opsaným poloměrem $q = 5.5$, ležící v rovinně α, jehož osa $\boldsymbol{z}$ je rovnoběžná s osami válců,
> - 2 souosá meziválcí („endcapy“) $\{M_j\}_1^2$ o vnějším poloměru $r_2 = 3.5$ a vnitřním poloměru $r_1 = 3.2$, výšce $d = 9$, vzdálené od sebe $s = 32$, se středy na ose $\boldsymbol{z}$, osami rovnoběžnými s osou $\boldsymbol{z}$,
> - „bounding box“, krychle K se středem v $\boldsymbol{s}$, rovnoběžná s rovinou α a osou $\boldsymbol{z}$, o délce strany $a = 60$.

Doplnili jsme jednotky délkových rozměrů a délku strany bounding boxu.