SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Marko Dominković

RAČUNANJE REZONANTNIH MODOVA U SUSATAVU BEM++

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Luka Grubišić

Zagreb, studenoga, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U ovom radu ćemo implementirat metodu za rješavanje problema računanja rezonatnih modova u problemu akustičkog ili elektromagnetskog raspršenja. Za rješavanje danih problema koristit ćemo sustav BEM++.

Poglavlje 1

Rezonance i modovi

U ovom poglavlju treba difinrati navedene pojmove

Poglavlje 2

BEM++

2.1 O BEM++

[1] Bem++ je sustav za rješavanje akustičnih, elektrostatičkih i elektromagnetnih problema pomoću metode rubnih elemenata (boundary element method). Moguće ga je koristiti kroz sučelja programskih jezika Python i C++.

2.2 Primjena BEM++

Rešetke i funkcijski prostori

Za prikaz oblika nekog tijela u BEM++ koristimo se rešetkama. Konstruirajmo jednu jednostavnu rešetku, kuglu radijusa 1.

```
import bempp.api
import numpy as np

grid = bempp.api.shapes.regular_sphere(3)
```

Dobivena kugla se satoji od $8 * 4^n$ elemenata, gdje je n argument funkcije koji označava stupanj uglađenosti kugle. Drugi način konstruiranja rešetke je pomoću polja vrhova i elemenata koji opisuju kako su vrhovi povezani. Sa `vertices` označimo polje svih vrhova. `evaluation_points` opisuje način na koji su ovrhovi povezani.

```
vertices = np.array([[0,1,2,1],
                     [0,2,0,1],
```

```
                                [0,0,0,2]])

evaluation_points = np.array([[0,0,0,1],
                              [1,1,2,2],
                              [2,3,3,3]])

grid = bempp.api.grid_from_element_data(vertice, evaluation_points)
```

Prvi element povezuje vrhove 0, 1 i 2. Drugi povezuje 0, 1 i 3 itd. Rešetke možemo učitati iz .msh datoteka. BEM++ podržava Gmsh v2.2 msh format

```
grid = bempp.api.import_grid('my_grid.msh')
```

Važnu ulogu u BEM++ imaju funkcijski prostori. Za inicijalizaciju funkcijskog prostora potrebna nam je rešetka.

```
space = bempp.api.function_space(grid, "P", 0)
```

Prvi argument funkcije je rešetka. Drugi argument je tip prosotra, u ovom slučaju `"P"` označava prostor polinoma.

Operatori

Operator A definiramo kao

$$A : \mathcal{D} \to \mathcal{R}$$

preslikavanje s domene D na kodomenu R, gdje su D i R definirane na površini dane rešetke. BEM++ ne koristi s direktno rubnim operatorom A, već slabijom formom

$$a(u, v) := \int_\Gamma [Au](\mathbf{y})\overline{v(\mathbf{y})}d\mathbf{y}, \quad u \in \mathcal{D}, v \in \mathcal{V} \tag{2.1}$$

gdje je $\mathcal{V}$ dualni prostor prostora $\mathcal{R}$. Rubni operatori se nalaze u paketu `bempp.api.operators.boundary`. Neka je $g(x, y)$ Greenova funkcija. Definiramo jednoslojni $\mathcal{V}$ i dvoslojni $\mathcal{K}$ potencijalni operator sa:

$$[\mathcal{V}\Phi](\mathbf{x}) = \int_\Gamma g(\mathbf{x}, \mathbf{y})\Phi(\mathbf{y})ds(\mathbf{y}), \quad \mathbf{x} \in \mathbb{R}^3 \backslash \Gamma \tag{2.2}$$

$$[\mathcal{K}\Phi](\mathbf{x}) = \int_\Gamma \frac{\partial g(\mathbf{x}, \mathbf{y})}{\partial v(\mathbf{y})} g(\mathbf{x}, \mathbf{y})\Phi(\mathbf{y})ds(\mathbf{y}), \quad \mathbf{x} \in \mathbb{R}^3 \backslash \Gamma \tag{2.3}$$

Sada iz njih izvodimo sljedeće granične operatore:

Operator	Formula
Jednoslojni rubni operator	$[V\phi](x) = \int_\Gamma g(\mathbf{x},\mathbf{y})\phi(\mathbf{y})d\mathbf{y}, \quad \mathbf{x} \in \Gamma$
Dvoslojni rubni operator	$[K\phi](\mathbf{x}) = \int_\Gamma \frac{\partial g(\mathbf{x},\mathbf{y})}{\partial v(\mathbf{y})}\phi(\mathbf{y})d\mathbf{y}, \quad \mathbf{x} \in \Gamma$
Adjungirani dvoslojni rubni operator	$[K'\phi](\mathbf{x}) = \int_\Gamma \frac{\partial g(\mathbf{x},\mathbf{y})}{\partial v(\mathbf{y})}\phi(\mathbf{y})d\mathbf{y}, \quad \mathbf{x} \in \Gamma$
Hipersingularni rubni operatori	$[D\phi](x) = -\frac{\partial}{\partial v(\mathbf{x})}\int_\Gamma \frac{\partial g(\mathbf{x},\mathbf{y})}{\partial v(\mathbf{y})}\phi(\mathbf{y})d\mathbf{y}, \quad \mathbf{x} \in \Gamma$

Dostupni operator s obzirom na odabir Greenove funkcije su:

PDJ	Greenova funkcija	Modul
Laplace $-\Delta u = 0$	$g(x,y) = \frac{1}{4\pi\lvert x-y\rvert}$	`bempp.api.operators.boundary,laplace`
Helmholtz $\Delta u + k^2 u = 0$	$g(x,y) = \frac{e^{ik\lvert x-y\rvert}}{4\pi\lvert x-y\rvert}$	`bempp.api.operators.boundary.helmholtz`
Modified Helmholtz $-\Delta u + w^2 u = 0$	$g(x,y) = \frac{e^{-w\lvert x-y\rvert}}{4\pi\lvert x-y\rvert}$	`bempp.api.operators.boundary.modified_helmholtz`

Svi rubni operatori su definirani sa domenom, kodomenom i dualni prostorom kodomene. U BEM++ se implementiraju na sljedeći način:

```
grid = bempp.api.shapes.regular_sphere(3)
space = bempp.api.function_space(grid, "DP", 0)
slp = bempp.api.operators.boundary.laplace.
      single_layer(space, space, space)
```

Rješavanje linearnih jednadžbi

Promotrimo sljedeći primjer

$$A\phi = f$$

gdje su f i ϕ funkcijske mreže. Zadanu jednadžbu možemo zapisati u njezinoj slabijoj formi:

$$< Au, v >=< f, v >$$

za sve v u dualnom prostoru. Uobičajni način rješavanje ovakvog sustava u Bempp je sljedeći:

1. Izračunati slabu formu od A pomoću

```
A_discrete = A.weak_form()
```

2. Računanje projekcije f na dualni prostor

```
p = f.projections(A.)
```

3. Pomoću funkcije `gmres`, iz SciPy paketa, riješimo sustav

```
x, info = gmres(discrete_op, p)
```

4. Iz rezultata `x` konstruirajmo funkcijsku mrežu sa

```
phi = bempp.api.GridFunction(A.domain, coefficients=x)
```

Poglavlje 3

Alogritmi za računanje rezonatnih modova

3.1 Računanje najmanje svojstvene vrijednosti

Neka je $A \in M_n$ kvadratna matrica reda n, $n \in \mathbb{N}$. Najmanja svojstvena vrijednost matrice A ujedno je i najveća svojstvena vrijednost matrice A^{-1}. Najveća svojstvena vrijednost može se izračunati iterativnom metodom. Opišimo iterativnu metodu za dani matricu A. Za početak odaberimo vektor $b_0 \in \mathbb{R}^n$. Definirajmo niz (b_n) na sljedeći način:

$$b_{k+1} = \frac{Ab_k}{||Ab_k||}, \quad k \in \mathbb{N}$$

Navedeni niz konvergira prema svojstvenom vektoru $x \in \mathbb{R}^n$ čija je odogovarajuća svojstvena vrijednost λ najveća svojstvena vrijednost matrice A. Za računanje najmanje svojstvene vrijednosti treba izračunati inverz matrice A, A^{-1}, s pretpostavkom da je matrica A regularna. Pritom primjenimo iterativnu metodu sa A^{-1}. Naravno, time se složenost samog algortima povećava. Druga mogučnost je rješavanje sustava

$$Ax = b \tag{3.1}$$

u svakoj od iteraciji, gdje je $A \in M_n$, a $b \in \mathbb{R}^n$. Broj iteracija ovisi o izboru početnog vektora i točnosti koju želimo postići. Svojstvena vrijednost jednaka je inverzu norme svojstvenog vektora x, $\lambda = 1/||x||$. Opišimo algoritme za računanje najmanje singularne vrijednosti matrice A.

Prvi aloritam je modificirani oblik iterativne metode u kojem imamo z

Algorithm 1: Najmanja singularna vrijednost

Ulaz: Regularna kvadratna matrica A veličine $n \times n$
Izlaz: Najmanja singularna vrijednost

b = slučajni vektor ;
$e = 0.0001$ (točnost);
$razlika = 1$;
while $razlika \geq e$ **do**
    riješi jednandžbu $Ax = b$;
    $x = x/||x||$;
    $razlika = ||b - x||$;
    $b = x$;
end
return $1/||x||$

U drugom algoritmu najprije izračunamo inverz matrice A pa potom provodimo klasični iterativni algoritam za traženje najveće svojstvene vrijednosti.

Algorithm 2: Najmanja singularna vrijednost: inverz matrice

Ulaz: Regularna kvadratna matrica A veličine $n \times n$
Izlaz: Najmanja singularna vrijednost

b = slučajni vektor, $e = 0.0001$ (točnost), $razlika = 1$;
$A_i = A^{-1}$;
while $razlika \geq e$ **do**
    $x = A_i b$;
    $x = x/||x||$;
    $razlika = ||b - x||$;
    $b = x$;
end
return $1/||x||$

Koji od navedenih algoritama je bolji? Naime, složenost rješavanja sustava $Ax = b$, gdje je A matrica veličine $n \times n$, je $O(n^3)$. Isto tako, računanje inverza matrice A je složenosti $O(n^3)$. Prednost drugog načina je ta da se inverz matrice izračuna samo jednom u odnosu na prvi alogritam, gdje se u svakoj iteraciji rješava sustav 3.1. S druge strane, zbog načina pohrane matrica u BEM++, javlja se problem memorije prilikom inicijalizace operatora u matričnom obliku. Da bi korisitli funkciju `linalg.inv()` iz paketa Scipy potrebno je operator prikazati u matričnom obliku, što se ostvaruje pozivom funkcije `bempp.api.as_matrix()`.

Usporedimo algoritme na konkretnom primjeru. Konstruirajmo kugle sa stupnjevima uglađenosti $n = 3, 4, 5$. Za svaku od kugle incijalizirajmo funkcijski prostor i na njima definirajmo operatore, točnije Helmholtzove operatore. Za broj valova uzet ćemo $k = 5$

```
k = 5
grids = []
for i in range(3,6):
    print(i)
    grids.append(bempp.api.shapes.regular_sphere(i))

spaces = [];
for grid in grids:
    spaces.append(bempp.api.function_space(grid,"DP",0))

helmholtz = []
for space in spaces:
    helmholtz.append(bempp.api.operators.boundary.helmholtz.
    single_layer(space,space,space,k))
```

Implentacije algoritama, `smallest_eigenvalue` i `smallest_eigenvalue_inverse`, za argument uzimaju operator iz susatva BEM++, a vraćaju vrijednost najmanje svojstvene vrijednosti i broj provedenih iteracija. U tablici 3.1 prikazani su rezultati.

Tablica 3.1: Usporedba algoritama

Algoritam / n	`smallest_eigenvalue`	`smallest_eigenvalue_inverse`
$3 < 512 \times 512 >$	λ = 0.000213838 broj_iteracija=61	λ = 0.000213846 broj_iteracija=60
$4 < 2048 \times 2048 >$	λ = 0.000025730978 broj_iteracija=106	λ = 0.000025730902 broj_iteracija=109
$5 < 8192 \times 8192 >$	λ = 0.000003153575 broj_iteracija=181	NaN

3.2 Akustično raspršenje

Bibliografija

[1] BEM++, https://bempp.com/.

Sažetak

Summary

Životopis