7. naloga
Metoda končnih elementov: Poissonova enačbe

Miha Srdinšek

I. O metodi

Poissonovo enačbo bomo reševali s pomočjo variacijskega principa. V smislu, da bomo enačbo

$$\mathcal{L}v(x) = -\frac{d}{dx}\left(p(x)\frac{dv}{dx}\right) + q(x)v = f(x), \tag{1}$$

skalarno pomnožili z utežno funkcijo w. Zahtevamo potem, da je ta utežna funkcija glede na skalarni produkt pravokotna residualu oblike

$$\langle w, \mathcal{L}v - f\rangle = 0. \tag{2}$$

Želimo da to velja za vsak w, zato u in w razvijemo po baznih funkcijah

$$u = \sum_{j=1}^{J} c_j\phi_j, \qquad w = \sum_{j=1}^{J} d_j\psi_j \tag{3}$$

in torej zahtevo posplošimo za vsak j. Nato enačbo preoblikujemo, tako da izpostavimo

$$\langle w, \mathcal{L}v - f\rangle = \langle w, \mathcal{L}v\rangle - \langle w, f\rangle = 0 = A(w, v) - \langle w, f\rangle \tag{4}$$

in potem po definiciji opazimo kaj je ta A.

$$A(w, v) = \int_a^b \left(w'pv' + wqv\right)dx. \tag{5}$$

Če v vse to vstavimo bazne funkcije dobimo precej enostaven matrični sistem, pod pogojem, da sta bazni funkciji enaki

$$\sum_{k=1}^{J} c_k A(\phi_j, \phi_k) = \langle \phi_j, f\rangle \quad \longrightarrow \quad A\vec{c} = \tilde{f}. \tag{6}$$

Od zdaj naprej si samo še zamislimo, da ustvarimo mrežo točk, v katerih potem izvrednostimo funkcijo. Če si predstavlamo triangulacijo, bomo dobili vse točke povezane tako, da bo vsaka točka oglišče kakšnega od trikotnikov. Potem bazno funkcijo oblikujemo kot $\phi_j(x_k, y_k) = \delta_{jk}$, saj ima vrh nad točko in ničlo v vseh okoliščih točkah, do teh točk pa je linearna funkcija. j-ti indeks pomeni j-to bazno funkcijo, k pa predstavlja inteks oglišč oziroma točk. Vsaki točki torej pripada izbrana funkcija in vse funkcije so identične.

Zdaj lahko zgornjo vsoto preoblikujemo iz vsote po točkah, na vsoto po trikotnikih in v vsakem trikotniku na vsoto po ogljiščih trikotnika. Tako zapišemo

$$\sum_{t=1}^{N_\tau} A^{(t)}(w,u) = \sum_{t=1}^{N_\tau} \langle w, f\rangle^{(t)}, \tag{7}$$

pri čemer t predstavlja indeks trikotnika, N_{tau} pa število trikotnikov - število težišč. Iz tega je očitno da mora biti

$$A^{(t)}(w,u) = \iint_{T_t} \Big(p(\nabla w)^T \nabla u + qwu \Big) dxdy, \tag{8}$$

pri čemer T_t pomeni integral to t-tem trikotniku. To lahko izvrednotimo kar v

$$A^{(t)}_{m,n} = \frac{1}{4|T|} \left(y_{m+1} - y_{m+2}, x_{m+2} - x_{m+1}\right) \begin{pmatrix} y_{n+1} - y_{n+2} \\ x_{n+2} - x_{n+1} \end{pmatrix} \tag{9}$$

in potem

$$\langle w, f\rangle^{(t)}_m = \frac{1}{6} \begin{vmatrix} x_{m+1} - x_m & x_{m+2} - x_m \\ y_{m+1} - y_m & y_{m+2} - y_m \end{vmatrix} f(x_T, y_T), \tag{10}$$

če (x_T, y_T) pomeni težišče trikotnika. Sedaj moramo zgolj še sešteti vse to

$$S_{\tau(t,m),\tau(t,n)} += A^{(t)}_{mn} \quad in \quad g_{\tau(t,m)} += \langle w, f\rangle^{(t)}_m, \tag{11}$$

po vseh trikotnikih in ogliščih teh trikotnikov. $\tau(t,m)$ predstavlja indeks vsake točke. Vsaka točka se namreč nahaja v nekem trikotniku. Tako ta vsota teče po vseh točkah, ampak važno je da tee po vseh trikotnikih, ker se točke pojavljajo v veih trikotnikih in je treba upoštevati vsak prispevek.

Robni pogoj upoštevamo enostavno tako, da tej končni matriki zmanjšamo število dimenzij iz dimenzije enake številu vseh točk na število vseh točk, ki niso robne točke. Vsote po robnih točkah ne delamo, a jih moramo upoštevati pri računanju $A^{(t)}_{mn}$. Ko imamo enkrat matriko S in vektor $\vec{g}$ smo se znašli pred enostavno enačbo

$$S\vec{c} = \vec{g}, \tag{12}$$

ki jo moramo rešiti za $\vec{c}$. To je potem že naša rešitev, ki predstavlja vrednost funkcije v točki.

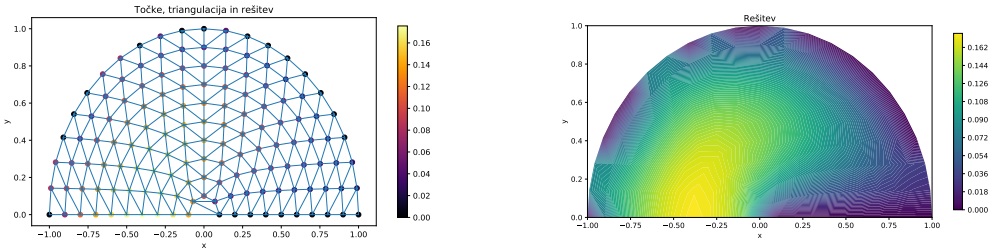
II. Pretok skozi polkrožno cev

Zdaj bi se radi zgolj lotili reševanja problema s zgoraj opisanim algoritmom. To se na videz zdi malček zapleteno, a se izkaže za zelo preprosto, saj uporabimo funkijo, ki sam nsama naredi triangulacijo. Ko imamo enkrat triangulacijo, moramo zgolj prepisati definicije za matrike v ustrazne zanke in smo na konju. Malček se moramo poigrati š z robnimi pogoji. Da bo zgled bolj poučen, bom najprej pokazal, kaj sem naredil narobe.

Najprej sem se rešavnja lotil brez zavedanja, da so robne točke kakrneškoli problem. Enostavno sem napisal točke s pomočjo radija in kota, tako da sem dobil točke in triangulacijo, kot na sliki 1.

Slika 1: *Slika prikazuje kako sem s prva določil točke.*

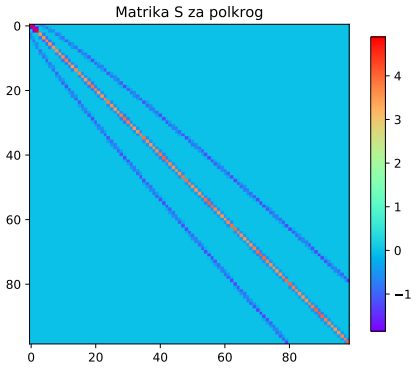
Potem pa sem za robne točke kar na pamet zapisal precej neumen pogoj, ki ven ni pobral vseh robnih točk. Kar znotraj algoritma sem napisal še en if stavek, ki je izložal robne točke. Na koncu s eje zgodilo, da so bile robne točke razmetane povsod po matriki in na koncu sem moral brisati te točke, si shranjevati mesta, da sem ohranil položaje neničelnih točk itd. S to metodo sem najprej dobil rešitev prikazano na slikah 2.



Slika 2: *Sliki prikazujeta rešitvi pri preslabih robnih pogojih.*

Pokazal sem jo, ker je zanimivo kaj se zgodi, če robnega pogoja ni. S tem smo pravzaprav daobili nekakšen prost robni pogoj. Nebi si upal preveč trditi, amapk mislim da bi se dalo na tak način upoštevati robni pogoj z odvodom.

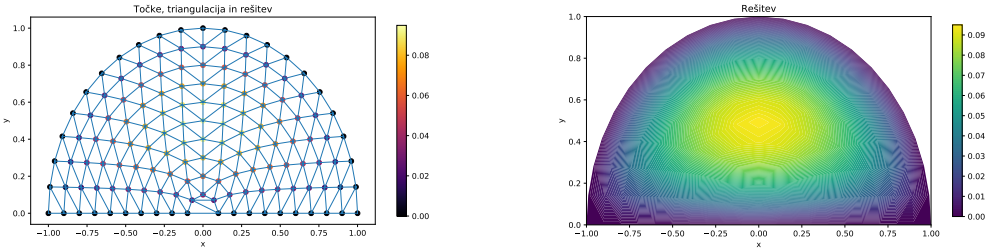
Poglejmo si zdaj raje pravo rešitev. Pri rpavi rešitvi smo vse robne točke umaknili, poiskali rešitev in jih na koncu prlepili nazaj. Tako smo najprej pridelali matriko prikazano na sliki 3.



Slika 3: *Slika prikazuje matriko S za polkrožni presek.*

Ta matrika izgleda zelo enostavna, zato jo lahko prevedemo na redke matrike in s tem pospešimo proces. Tukaj to sploh ni bilo potrebno, saj ej bil sistem precej majhen in sem ga

reševal kar z normalnimi postopki. Ko sem na koncu rešil sistem enačb sem dobil rešitev priakzano na slikah 4. Ti sliki prikazujeta to kar bi načeloma pričakovali, zato se tu več ne zamujamo. Naloga je rešena, probil smo pokaali. Lahko bi opazovali kako se natančnost spreminja s številom točk, a že zdaj si ogledujemo primer pri zelo malem številu točk. Število točk bi zagotovo lahko še zmanjšali, kar bomo bolje opazovali pri naslednji nalogi.

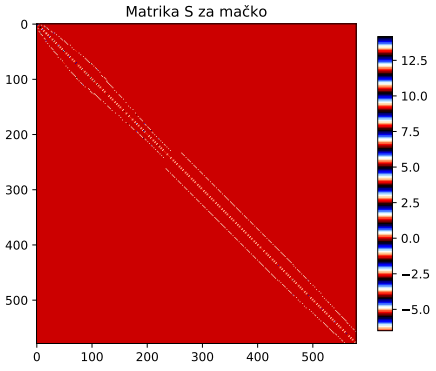


Slika 4: *Sliki prikazujeta rešitvi pri pravih robnih pogojih.*

III. Presek iz 5.naloge - Mačka

V prejšnji nalogi sem presek napisal zelo površno prilagojen za primer $N = 100$ in ne kot funkcijo. Zato sem prejšnjič lahko izrisal zgolj določene primere. Profil sem lahko povečeval a zelo težko zmanjšal. Tokrat sem profil zapisal kot funkcijo N in si zato omogočil poigravanje s številom točk. Da bo zgled lepši bom prikazal najprej primer z malim N, kljub temu da je moje reševanje potekalo v obratnem vrstnem redu. Na povsem očiten način sem napisal točke na kvadratni mreži, ki je kot nalašč za tak profil določen z diagonalami, potem pa sem robne točke umaknil in jih tokrat nanesel šele na koncu. To je bila velika izboljšava od prvotnega algoritma, ki sem ga uporabil pri prvi nalogi. Tam sem robne točke razmetal po celi matriki, tu pa sem robne točke dodal šele po tem, ko sem dodal vse ostale točke. S tem sem močno poenostavil dodajanje in odstranjevanje teh točk.

Tako sem z zelo malo napora prišel do matrike S, prikazane na sliki 5. Na zgornji sliki

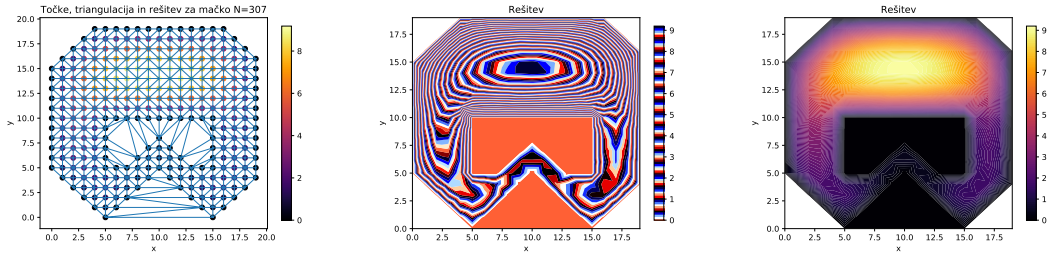


Slika 5: *Slika prikazuje matriko S za profil cevi v obliki mačke.*

sem uporabil neprimerno barvno preslikavo, ker so vrednosti tako majhne, da jih drugač nebi morali opaziti. Povdarek je torej na obliki matrike, ne na vrednosti elementov metrike.

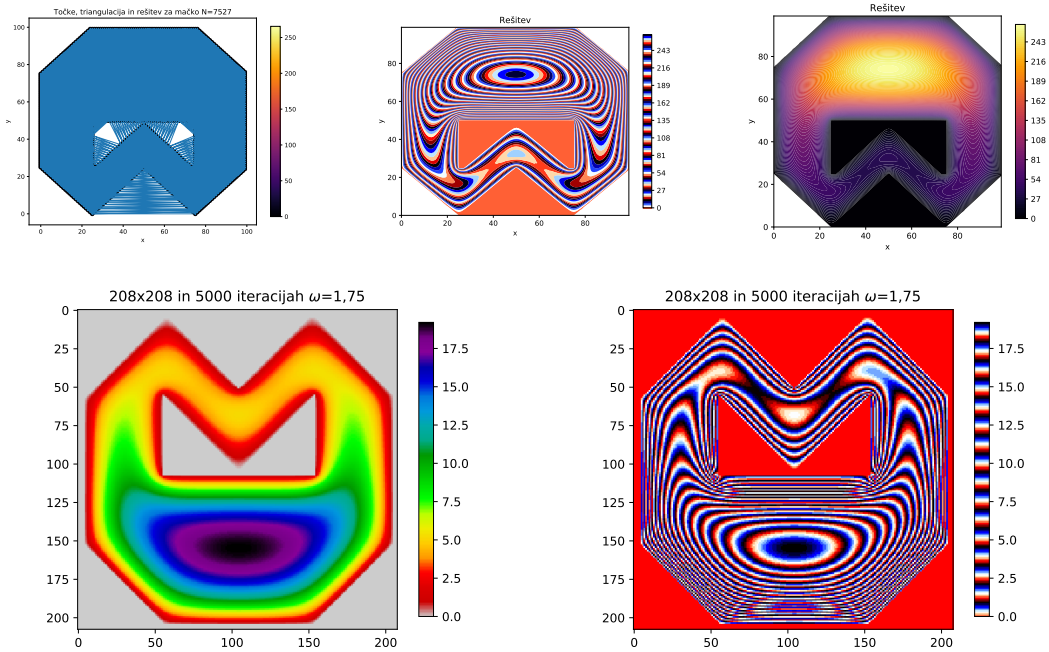
Za majhno število točk, sem potem dobil rezultat prikazan na slikah 6. Vidimo da je pri malem številu točk oblika profila že močno popačena, in kanalček spodaj je izredno majhen. A

kljub temu, lahko že tu opazimo na sredinski sliki, kako lepa je rešitev, kako lepo definirani so nivoji z enako vrednstjo. Druga zanimivost je opazovati levo sliko, kjer vidimo, da je triangulacija povezala kar točke, ki so zeo daleč narazen in s tem zapolnila prostor. To je dovoljeno, ker take povezave med seboj povezujejo robne točke. Robne točke pa tako ali tako k ničemur ne prispevajo in so te povezave čisto nepomembne. Če bi povezale robne točke iz ene strani z odprtimi točkami na drugi strani pa bi verjetno lahko imeli probleme. Recimo da bi na desnem uhlu znotraj imeli prost robni pogoj, desno pa robno točko. V takem primeru bi lahko imeli probleme.



Slika 6: *Slike prikazujejo rešitev za malo število točk.*

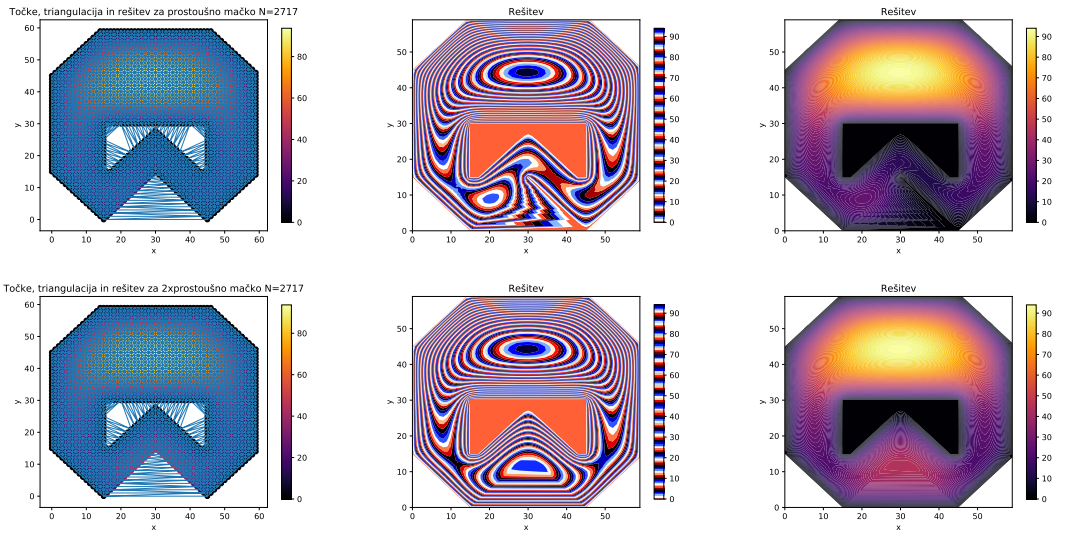
Poglejmo si sedaj še rešitev pri večjem številu točk, ki nam bo pomenila veliko več. To vidimo prikazano na slikah 7. Predvsem sredinska slika nas presune s svojo natančnostjo. Desne slike pa se ne vidi tako kot bi se moralo, ker je preveč skrčena. Prva stvar, ki jo lahko opazimo je recimo to, da je v tem primeru takšno število točk že mnogo preveč, med tem ko je bilo za SOR algoritem še celo premalo. Glavna razlika je predvsem v robovih, kjer se je nova metoda odrezala dosti boljše. Pri tem lahko bralec opazi, da smo za tako natančnost pri *SOR* algoritmu porabili enkrat več točk in tudi precej več časa.



Slika 7: *Slike zgoraj prikazujejo reštev za veliko število točk, sliki spodaj pa rešitev z algoritmom SOR.*

A vseeno nas je rahlo presenetilo, kako je smo izvedli triangulacijo in kar po praznem

prostoru napeljali trikotnike. Zanima nas torej kaj se zgodi, če tam kar pustimo prost robni pogoj. Predvidevamo namreč, da je do sedaj metoda delovala, ker smo imeli na obeh orbovih ničle. Kaj pa se zgodi če imamo ničle samo na enem robu? V tem primeru bi pričakovali da bo rešitev praktično enaka, samo razpotegnjena čez ta prazen prostor, amapk robni pogoj pa bo isti. Zanimivo po pogledati kaj se zgodi, če na teh dveh ušesih ni nobenega robnega pogoja.



Slika 8: *Slike prikazujejo rešitev če se poigravamo z robnimi pogoji. Zgoraj se znebimo enega roba, spodaj pa dveh.*

Rezultate vidimo na slikah zgoraj 8. Vidimo, da so bile naše napovedi napačne. Dejansko naredi novo rešitev in zapolni tudi prostor vmes ne samo z isto a razpotegnjeno rešitvijo, temveč s povsem novo rešitvijo. Pravzaprav dobimo rešitev za primer, ko bi notraj cevi bil še nek tak kratek prekat.

Kako bi upošteval ostale oblike robnih pogojev mi ni povsem jasno. Različne vrednosti na robih bi verjetno upošteval tako, da bi pri računanju $\vec{g}$ upošteval različne vrednosti funkcije v težeščih, če bi bila katera točka na robu. A bolj mi niso jasni robni pogoji z odvodom. Glede na moj prejšnji razmislek in glede na članke, ki sem si jih ogledal, bi sklepal, da je Neumannov robni pogoj še bolj enostaven, saj je naša metoda narejane s tem pogojem že vgrajenim. Tako kot smo videli prej prosti robni pogoj, tako je Neumannov robni pogoj le to da algoritem ne spregleda robnih točk, samo spremenimo vrednost funkcije f v težiščih trikotnikov, ki vsebujejo "*robne*" točke. Na tak način smo zdaj že izpolnili neumannov pogoj v prvi nalogi, samo smo rekli, da je izvor isti kot povsod drugje. Če bi tam spremenili vrednost bi dobili kar želimo.

Tako vidimo, da je upotštevati tudi bolj zapletene robne pogoje zelo enostavno, če moj razmislek drži. Zgolj spremenimo $\vec{g}$ in robe točke umaknemo, če imamo von neumannov robni pogoj, oziroma jih pustimo, če imamo dirichletov. Ker to ni del te naloge, se v to nisem preveč spuščal in sem prikazal le prost robni pogoj.