SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1884

GALJORKINOVA METODA KONAČNIH ELEMENATA

Hrvoje Radoš

Zagreb, lipanj, 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 1884

Pristupnik:	**Hrvoje Radoš (0036548995)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Tomislav Burić

Zadatak: **Galjorkinova metoda konačnih elemenata**

Opis zadatka:

Galjorkinova metoda konačnih elemenata se često koristi u numeričkom rješavanju (parcijalnih) diferencijalnih jednadžbi te time ima veliki značaj u modeliranju i rješavanju inženjerskih problema. Njezina implementacija se bazira na raznim metodama iz numeričke matematike kao što su rješavanje linearnih sustava i numeričko integriranje. Složenost implementacije proizlazi i iz particioniranja domene na kojoj se problem rješava, a dodatna složenost je u optimizaciji pri čemu se može koristiti višedretveno programiranje. Cilj ovog rada je implementirati Galjorkinovu metodu te analizirati njene performanse na konkretnim primjerima.

Rok za predaju rada: 23. lipnja 2025.

Hvala na kavi...

Sadržaj

1. Uvod

Diferencijalne jednadžbe su jedan od temeljnih matematičkih alata koji se koriste za matematičko modeliranje. Njihova važnost se obično prvo primijeti u fizici gdje se one vrlo intuitivno pojavljuju u područjima kao što su klasična mehanika, elektromagnetizam, termodinamika pa i malo manje intuitivno u područjima kao što su kvantna mehanika. Diferencijalne jednadžbe imaju velik utjecaj i na područja izvan fizike. S njima se mogu modelirati izmjene populacija, širenje zaraza, oblikovanje raznih genetskih obilježja (uzoraka), širenje signala u neuronima, procesiranju slika, raznim modeliranjima tržišta itd.

Diferencijalne jednadžbe uglavnom nije teško tumačiti. Najveći izazov kod rada s njima je svakako traženje rješenja. Rješenje ne mora uvijek postojati, ne mora biti jedinstveno, a ako postoji i jedinstveno je ne mora biti moguće zapisati ga analitički. Tim problemima se između ostaloga bavi numerička matematika. Ovaj rad se bavi linearnim parcijalnim diferencijalnim jednadžbama s konstantnim koeficijentima i rubnim uvjetom odnosno traženjem njihova rješenja koristeći Galjorkinovu metodu konačnih elemenata. Cilj rada je napraviti C++ biblioteku koja služi kao efikasan, ali iznad svega jednostavan alat za rješavanje PDJ-ova.

2. Glavni dio

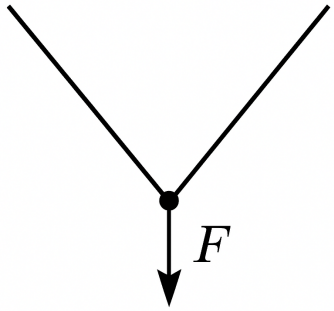
Galjorkinova metoda konačnih elemenata je složen proces koji se sastoji od više dijelova. Na početku ovog poglavlja nudim kratki opis kako ovaj proces izgleda, a u narednim poglavljima ulazim u detalje.

Jednadžba koju rješavam na domeni $\Omega \subset \mathbb{R}^2$ je:

$$C_0\frac{\partial^2 u}{\partial x^2} + C_1\frac{\partial^2 u}{\partial x \partial y} + C_2\frac{\partial^2 u}{\partial y^2} + C_3\frac{\partial u}{\partial x} + C_4\frac{\partial u}{\partial y} + C_5 u = f(x,y) \quad u \in C^2(\Omega) \tag{2.1}$$

Ključna ideja ove metode je particioniranje domene na što sitnije odnosno finije dijelove koje nazivamo elementima. Takvu particiju nazivamo mesh. S meshom potom definiramo bazne funkcije koje čine bazu s kojom ćemo interpolirati rješenje koje tražimo. Dodatnim raspisivanjem ćemo dobiti linearni sustav čijim rješavanjem dobivamo naše traženo rješenje.

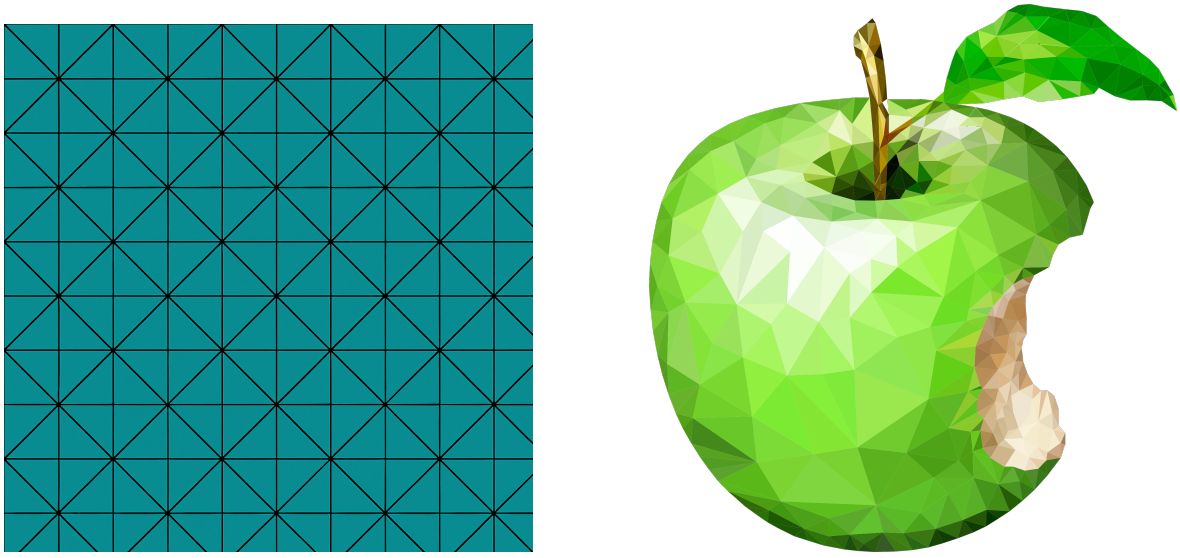
Postoji par napomena koje valja primijetiti. Ako obratimo pažnju na jednadžbu (2.1), možemo primijetiti da smo se ograničili na rješenja iz prostora $C^2(D)$ dodatnom analizom možemo zaključiti i da funkcija f mora biti neprekinuta. Ovo se na prvu iz perspektive primjene ne čini kao velika restrikcija jer nekako očekujemo da će fizikalne veličine u stvarnom životu uvijek biti neprekinute, ali ne samo da to nije uvijek slučaj to jako često nije slučaj. Kao primjer navodim primjer djelovanja sile (npr. ovješen uteg) na nit. Za modeliranje ovakvog sustava je potrebno opisati djelovanje te sile duž cijelu nit, ali takav opis bi zahtijevao uporabu Diracove delta funkcije koja nikako nije neprekinuta, štoviše ona nije ni funkcija nego distribucija.



Slika 2.1. Djelovanje točkaste sile na nit

2.1. Mesh

Mesh je particija domene na sitne dijelove s kojima ju možemo aproksimirati. U ovoj implementaciji ti dijelovi će biti trokuti (jer radimo s 2D domenom), pa ću proces stvaranja mesha često zvati i triangulacija. Ti sitni dijelovi koji sačinjavaju mesh se zovu elementi. Odavde i naziv "metoda konačnih elemenata". Točke koje čine jedan element (npr. vrhovi trokuta koji predstavlja jedan elemente) ću u ovom radu zvati čvorovima.



(a) Primjer 2D mesha pravokutnika **(b)** Složeni 3D mesh jabuke

Slika 2.2. Primjeri mesheva

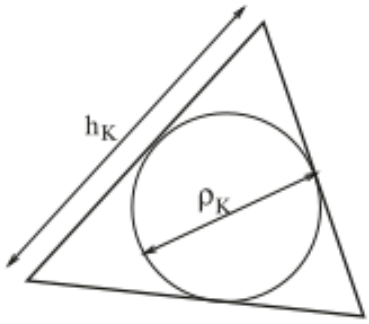
Bitno je napomenuti kako izbor mesha ne smije biti proizvoljan. Postoje kvalitetni i manje kvalitetni meshevi. U literaturi ne postoji nekakav jedinstveni kriterij koji mesh mora zadovoljavati, tako da je generiranje mesha u primijeni uvijek heuristički algoritam. Očito je za zaključiti da bi mesh trebao pravilno popločavati domenu tj. da ne bi

smio sadržavati rupe koje u potpunosti pripadaju domeni i da bi trebao što bolje aproksimirati domenu kako su elementi manji odnosno kako je mesh finiji.

Svakako valja napomenuti da ne želimo imati previše izdužene elemente, stoga uvodimo sljedeći kriterij.

$$\frac{h_k}{\rho_k} \leq C$$

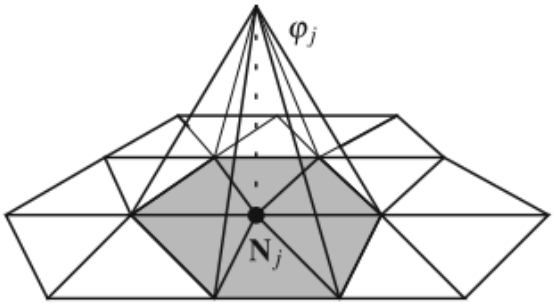
Gdje je h_k duljina najdulje stranice trokuta, a ρ je promjer upisane kružnice promatranog elementa. Razlog zašto ovakav kriterij smanjuje izduženost jednog elementa se lako vidi na sljedećoj slici.



Slika 2.3. najdulja stranica trokuta h_k i promjer upisane kružnice ρ_k

2.1.1. Bazne funkcije

Nakon uspješno generiranog mesha je potrebno definirati bazne funkcije. Bazne fukcije ćemo definirati tako da na svakom elementu definiramo polinom prvog stupnja koji iznosi 1 iznad jednog vrha, a 0 iznad ostalih 2.4. Ovo ćemo napraviti za svaki čvor.



Slika 2.4. bazna fukcija za neki čvor j

Kada budemo radili račun kao što su numerička integracija i slično nad elementima bit

će od velike koristi transformirati koordinate tako da element koji trenutačno računamo postane oblika pravokutnog trokuta s vrhovima u $(0,0)$,$(1,0)$,$(0,1)$. Takav trokut se u literaturi uobičajeno zove referentni element. Neka su (X_0, Y_0),(X_1, Y_1),(X_2, Y_2) vrhovi elementa kojeg želimo preslikati. Supstitucija $(x, y) \to (\hat{x}, \hat{y})$ je dana s:

$$x = (X_1 - X_0)\hat{x} + (X_2 - X_0)\hat{y} + X_0 \tag{2.2}$$

$$y = (Y_1 - Y_0)\hat{x} + (Y_2 - Y_0)\hat{y} + Y_0 \tag{2.3}$$

Potrebno je odrediti kako točno glase bazne funkcije nad referentnim elementom. Indeksirajmo čvor $(0,0)$ s 0, čvor $(1,0)$ s 1a i čvor $(0,1)$ s 2. Imamo:

$$\varphi_0(x, y) = 1 - x - y \tag{2.4}$$

$$\varphi_1(x, y) = x \tag{2.5}$$

$$\varphi_2(x, y) = y \tag{2.6}$$

2.2. Slaba formulacija problema

Radi daljnje analize potrebno je navesti jedan bitan teorem.

Teorem 1 (Pravilo za divergenciju produkta) *Neka je $\Omega \subset \mathbb{R}^n$ omeđen otvoreni skup s komadno glatkom granicom $\Gamma = \partial\Omega$, te neka su $u : \Omega \to \mathbb{R}$ i $\mathbf{V} : \Omega \to \mathbb{R}^n$ dovoljne regularnosti odnosno $\partial\Omega$ je Lipschitz neprekinuta, a $u, v \in H^1(\Omega)$. Tada vrijedi:*

$$\int_\Gamma u\mathbf{V} \cdot \hat{\mathbf{n}}\, d\Gamma = \int_\Omega u\, \nabla \cdot \mathbf{V}\, d\Omega + \int_\Omega \nabla u \cdot \mathbf{V}\, d\Omega.$$

Napomena: $H^1(\Omega)$ označava Soboljev prostor.

Vratimo se sada na 2.1 Cilj nam je dobiti njezinu takozvanu slabu formulaciju. Prvo ćemo reformulirati PDJ. Primijetimo da vrijedi:

$$\nabla \cdot (A\nabla u) = A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2}$$

Gdje:

$$M = \begin{pmatrix} C_0 & C_1 \\ 0 & C_2 \end{pmatrix}$$

Također vrijedi:

$$\nabla u \cdot a = C_3 \frac{\partial u}{\partial x} + C_4 \frac{\partial u}{\partial y}$$

Gdje:

$$a = \begin{pmatrix} C_3 \\ C_4 \end{pmatrix}$$

Time dobivamo sljedeću jednadžbu:

$$\nabla \cdot (M \nabla u) + \nabla u \cdot a + C_5 u = f(x, y) \tag{2.7}$$

Do slabe formulacije dolazimo tako da obje strane 2.7 pomnožimo s baznom funkcijom φ_i, integriramo po Ω i onda primijenimo 1 imamo:

$$\int_\Omega \nabla \cdot (M \nabla u) \varphi_i \, d\Omega + \int_\Omega \nabla u \cdot a \varphi_i \, d\Omega + C_5 \int_\Omega u \varphi_i \, d\Omega = \int_\Omega f(x, y) \varphi_i \, d\Omega \tag{2.8}$$

Primjenom teorema 1 dobije se:

$$\int_{\partial\Omega} (M \nabla u) \varphi_i \, d\Gamma - \int_\Omega (M \nabla u) \cdot \nabla \varphi_i \, d\Omega + \int_\Omega \nabla u \cdot a \varphi_i \, d\Omega + C_5 \int_\Omega u \varphi_i \, d\Omega = \int_\Omega f(x, y) \varphi_i \, d\Omega \tag{2.9}$$

s obzirom na to da su funkcije φ_i po definiciji na rubu jednake 0, jednadžba 2.9 se svede na:

$$- \int_\Omega (M \nabla u) \cdot \nabla \varphi_i \, d\Omega + \int_\Omega \nabla u \cdot a \varphi_i \, d\Omega + C_5 \int_\Omega u \varphi_i \, d\Omega = \int_\Omega f(x, y) \varphi_i \, d\Omega \tag{2.10}$$

Primijetimo kako u ovakvom obliku funkcija $f(x, y)$ na sebi ima puno slabije restrikcije (Sjetimo se 2.)

Sada interpolacijom:

$$u = \sum_{j=0}^{N} \alpha_j \varphi_j$$

dobivamo sustav:

$$Ax = b \tag{2.11}$$

Gdje:

$$(A)_{i,j} = \int_\Omega (M\nabla\varphi_j)\cdot\nabla\varphi_i \, d\Omega + \int_\Omega (\nabla\varphi_j \cdot a)\varphi_i \, d\Omega + C_5 \int_\Omega \varphi_j\varphi_i \, d\Omega \tag{2.12}$$

$$b_i = \int_\Omega f(x,y)\varphi_i \, d\Omega$$

Naime ovaj postupak vrijedi samo s Dirichletovim rubnim uvjetima. Neka su čvorovi mesha na rubu indeksirani od $N+1$ do N_b. Kada bi se rješenje na rubu ponašalo kao $R(x,y)$ gdje $R(x,y) = \sum_{j=N+1}^{N_b} g_j\varphi_j$, onda bismo morali napraviti sljedeću supstituciju:

$$u = \mathring{u} + R$$

Sustav 2.11 postaje:

$$Ax = b - r$$

gdje je:

$$r_i = -\int_\Omega (M\nabla R)\nabla\varphi_i \, d\Omega + \int_\Omega (\nabla R \cdot a)\varphi_i \, d\Omega + C_5 \int_\Omega R\varphi_i \, d\Omega$$

Time smo problem rješavanja PDJ sveli na problem rješavanja linearnog sustava.

2.3. Postojanje rješenja

Iako se možda na prvi pogled čini da smo gotovi, potrebno je vidjeti da rješenje uopće postoji, stoga uvodim Lax-Milgramov teorem.

Teorem 2 *Neka je V Hilbertov prostor i neka je $a : V \times V \to \mathbb{R}$ bilinearna forma koja zadovoljava:*

- ***Kontinuitet:*** *postoji konstanta $C > 0$ takva da*

$$|a(u,v)| \leq C\|u\|_V\|v\|_V \quad \textit{za sve } u, v \in V.$$

- ***Koercitivnost (eliptičnost):*** *postoji konstanta* $\alpha > 0$ *takva da*

$$a(v, v) \geq \alpha \|v\|_V^2 \quad za\ sve\ v \in V.$$

Tada za svaki linearni funkcional $f \in V'$ *postoji jedinstveno* $u \in V$ *takvo da:*

$$a(u, v) = f(v) \quad za\ sve\ v \in V.$$

Lako se vidi da se naš sustav 2.11 može svesti na problem s linearnim funkcionalima iz 2 Iz toga se lako zaključuje da postojanje rješenja možemo garantirati samo za ispravan mesh (to garantira implementacija) i za eliptičke PDJ što se lako provjeri programski.

2.4. Implementacijski detalji

Sada kada je jasna matematička pozadina Galjorkinove metode konačnih elemenata potrebno je obratiti pažnju na implementaciju i implementacijske detalje. s obzirom na to da je implementacija izvedena u C++-u postoji dosta različitih optimizacija koje se mogu implementirati, stoga ih navodim u ovom poglavlju.

2.4.1. Generiranje mesha

S obzirom na to da je jednostavnost korištenja ključno svojstvo ove implementacije odlučio sam se za sljedeći način unosa mesha u računalo.

Korisnik definira funkciju koja prima koordinate, a vraća vrijednost "true" ako je točka unutar domene ili "false" ako je točka izvan domene

Time smo implicitno definirali domenu PDJ-a. Ovakav pristup omogućava da se domena definira na sličan način kako bi to radili na papiru iz čega proizlazi jednostavnost. Druge C++ biblioteke za unos mesha uglavnom koriste .msh datoteke. Ovakav pristup je dobar za kompliciranе domene koje se ne mogu lako definirati s implicitnim funkcijama, ali je presložen za neke jednostavnije primjene. Valja napomenuti da neke biblioteke daju mogućnost da se domena definira preko ključnih riječi kao što su "circle" ili "rectangle". U usporedbi s tim rješenjima moj pristup daje veću fleksibilnost.

Jednostavan način korištenja povlači i težinu implementacije. Naime, kvalitetan mesh može značajno smanjiti numeričku grešku rješenja. Naravno greška se uvijek može smanjiti tako da učinimo mesh finijim, ali to ne možemo raditi u nedogled. Ova implementacija mesh generira na poprilično jednostavan način. Za ulaz ona traži pravokutnik koji omeđuje domenu nad kojom rješavamo PDJ. Taj pravokutnik se potom triangulira i za mesh se uzimaju samo elementi čija su sva tri vrha unutar domene. Ovakav pristup daje nekvalitetan mesh na rubu. Istraživajući bolje pristupe generiranju mesha otkrio sam razne knjige koje se bave ovom problematikom. Došao sam do zaključka da je ovaj problem previše složen za ovaj rad i stoga smatram da je za sada bitno napraviti implementaciju koja se može naknadno lako nadograditi.
Iako jednostavan pristup ovom problemu funkcionira mislim da je bitno naglasiti gdje ova implementacija ima mjesta za napredak.

2.4.2. CSR reprezentacija matrice

Matrica A u 2.11 često sadrži puno nul elemenata. To se vrlo lako može vidjeti iz načina na koji se ona računa. Naime, $(A)_{i,j}$ neće biti jednak 0 samo kada su odgovarajući čvorovi mesha (i i j) susjedni. S obzirom na to da je zbog veće preciznosti poželjno da matrice sustava budu što veće uvodi se CSR (eng. compact sparse row) reprezentacija matrice. Ovaj pojam se može lako objasniti na sljedećem primjeru.

$$M = \begin{pmatrix} a & 0 & f & 0 & g \\ 0 & b & k & m & 0 \\ h & l & c & 0 & r \\ 0 & n & 0 & d & p \\ i & 0 & s & q & e \end{pmatrix}$$

Na slici 2.5. se vidi matrica M zapisana preko CSR. Vidimo da se ova struktura podataka sastoji od 3 vektora. Vektori A i C su jednake veličine kao broj ne nul elemenata, a svaki element R odgovara jednom retku. Matrica A se stvara tako da čitajući redak po redak s lijeva na desno umećemo elemente matrice M koji nisu jednaki 0. Vrijednost C_i odgovara brojčanoj vrijednosti stupca u kojem se element A_i nalazi a element R_i govori od kojeg indeksa kreće $i - ti$ redak matrice M u vektoru C odnosno A.
Dohvat elementa i, j ovakve matrice se vrlo lako implementira.

```
inline ld get(ull i, ull j) {
  ull it = R[i];
  while (it < R[i + 1] && C[it] < j) it++;
  if (C[it] == j) {
    return A[it];
  }
  return 0;
}
```

Pažljivi čitatelj će primijeti da se pri traženju odgovarajućeg stupca (While petlja) radi linearno pretraživanje. Ovaj dio se može ubrzati binarnim pretraživanjem, ali s obzirom na izuzetno malen broj ne nul elementa u jednom stupcu (maksimalno 6) u trenutačnoj implementaciji mesha, upitno je koliko veliko ubrzanje bi se ostvarilo takvom implementacijom.
Slično se implementira i mijenjanje nekog ne nul elementa.

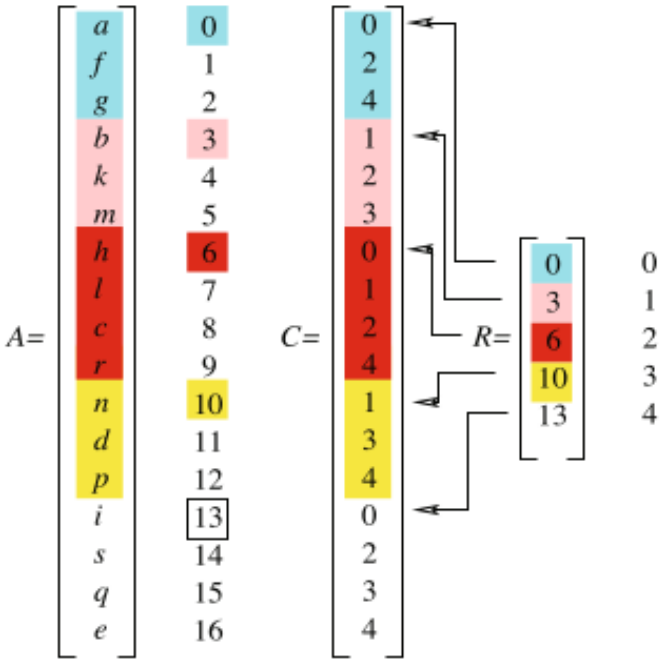
```
inline void inc(ull i, ull j, ld val) {
  ull it = R[i];
  while (it < R[i + 1] && C[it] < j) it++;
  if (C[it] == j) {
    A[it] += val;
  }
}
```

2.4.3. Numerička integracija

Za numeričku integraciju je korištena metoda Gaussovih čvorova odnosno za referentni element $\hat{K}$ integral po njemu se može zapisati kao.

$$\int_{\hat{K}} f(x,y)\, d\hat{K} = \sum_{i=0}^{n} w_i f(x_i, y_i)$$

Gdje su trojke (x_i, y_i, w_i) izračunate unaprijed. Moja implementacija koristi sljedeće Gaussove čvorove: $(0,0,1/3), (1,0,1/3), (0,1,1/3)$



Slika 2.5. grafički prikaz CSR reprezentacije matrice A

2.4.4. Rješavanje sustava

Računski najsloženiji dio je svakako rješavanje linearnog sustava. Na raspolaganju je puno različitih algoritama različitih složenosti, pa tako su neki prilagođeniji nekoj specifičnoj PDJ. Naprimjer Choleskyjeva faktorizacija bi mogla biti korištena kada bih mogao garantirati da će matrica A biti simetrična pozitivno definitna, ali to se iz 2.12 lako vidi da nije slučaj.

Daljnjom analizom se može zaključiti da direktne metode nisu dobar izbor jer uz vremenski skupo plaćanje faktorizacija matrice (QR, LU ...), takve matrice mogu dovesti do pojavljivanja dodatnih ne nul elemenata koji bi povećali zahtjeve za memorijom računala.

Preostaju nam iterativne metode i one se uobičajeno i koriste kod rada s rijetko popunjenim matricama. Neke od mogućih metoda su Jacobijeva, Gauss-Seidel, SOR, konjugirani gradijenti, GMRES.

Radi jednostavnosti ova implementacija koristi Gauss-Seidel-ovu metodu koja neće nužno konvergirati za svaku moguću PDJ koju će korisnik unijeti, ali će biti dovoljno dobra za standardne probleme nad 2D meshevima. U nastavku dajem pseudokod Gauss-Seidela:

```
##rješavanje sustava Ax = b Gauss-Seidelom
Uzimamo neki vektor x duljine n
za svaku iteraciju k od brojIteracija
  za svaki redak i matrice A
    ##računamo novi xi
    sumaManjih = 0
    za j od 0 do i
      sumaManjih += Aij* xj
    sumaVecih = 0
    za j od i + 1 do n
      sumaVecih += Aij * xj
    xi = (bi - sumaManjih - sumaVecih) / Aii
```

Dodatno Gauss-Seidel se može ubrzati tako da iskoristimo rijetku popunjenost matrice. Naime najdublje ugniježdene petlje se ne moraju iterirati po svim elementima nego samo po onima koji nisu jednaki 0. Pa tako možemo iskoristiti CSR kako bi dobili i vremensku efikasnost. Petlja po iteracijama Gauss-Seidela u C++ glasi:

```
for (u k = 1; k <= numOfIterations; k++) {
  for (u i = 0; i < M.numOfRows; i++) {
    ld sumOfLessThan_i = 0;
    u j;
    for (j = M.R[i];
        j < M.R[i + 1] && M.C[j] < i;
        j++) {
      sumOfLessThan_i += M.A[j] * x[M.C[j]];
    }
    ld sumOfGreaterThan_i = 0;
    ld Mii = 0;
    if (M.C[j] == i) {
      Mii = M.A[j];
      j++;
    }
```

```
    for (; j < M.R[i + 1]; j++) {
      sumOfGreaterThan_i += M.A[j] * x[M.C[j]];
    }
    x[i] =
      (b[i] - sumOfLessThan_i - sumOfGreaterThan_i) / Mii;
  }
```

Ovom optimizacijom smo sveli vremensku složenost s $\mathcal{O}(kn^2)$ na $\mathcal{O}(k \cdot 6n)$ odnosno na $\mathcal{O}(kn)$.

Kao i prije navodim gdje se implementacija može poboljšati. Istraživanjem sam dobio dojam da je GMRES metoda bolji izbor. Također bi bila potencijalno dobra ideja da se provjerava simetričnost i pozitivna definitnost matrice kako bi se primijenila brža metoda koja ovisi o tim svojstvima matrice.

2.4.5. Prevođenje koda

Ovdje ću malo prokomentirati koje sam točno zastavice i opcije koristio pri prevođenju moje implementacije.

3. Rezultati i rasprava

4. Zaključak

Literatura

Sažetak

Galjorkinova metoda konačnih elemenata

Hrvoje Radoš

Unesite sažetak na hrvatskom.

Ključne riječi: prva ključna riječ; druga ključna riječ; treća ključna riječ

Abstract

Galjerkin finite element method

Hrvoje Radoš

Enter the abstract in English.

Keywords: the first keyword; the second keyword; the third keyword

Privitak A: The Code