# Objektno orijentirani pristup metodi konačnih elemenata

**Laković, Ivan**

**SVEUČILIŠTE U ZAGREBU**

**PRIRODOSLOVNO–MATEMATIČKI FAKULTET**

**MATEMATIČKI ODSJEK**

Ivan Laković

# OBJEKTNO ORIJENTIRANI PRISTUP METODI KONAČNIH ELEMENATA

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, rujan, 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. _____

*Ovaj rad posvećujem mami, tati i bratu koji su mi neizmjerna podrška svih ovih godina te svim prijateljima bez kojih iskustvo studiranja ne bi bilo potpuno.*

# Sadržaj

# Uvod

Moderno strojarsko inženjerstvo (kao što je zrakoplovno, biomehaničko i automobilsko) obično koristi metodu konačnih elemenata (MKE) u dizajnu i razvoju novih proizvoda. Većina modernog softvera za MKE uključuje specifične komponente za različite analize poput toplinskih, elektromagnetskih, strukturnih a i analize fluida. U strukturnoj simulaciji, MKE uvelike pomaže u proizvodnji, vizualizaciji krutosti i snage, kao i smanjenju težine, materijala i troškova. Osim toga samo testiranje postaje ciljano na najslabije komponente te samim time i jeftinije i brže. [20]

Metoda konačnih elemenata numerička je metoda za rješavanje parcijalnih diferencijalnih jednadžbi koja se temelji na fizičkoj diskretizaciji kontinuuma te se koristi za rješavanje problema u inženjerstvu i fizici. Primjeri domena problema su strukturna analiza, prijenos topline, protok tekućine i slično. Početak razvoj metode konačnih elemenata se procjenjuje na četrdeset godine dvadesetog stoljeća. Za rješavanje problema potrebno je riješiti sustave velikog broja algebarskih jednadžbi što ubrzo postaje nemoguće izvesti bez pomoći računala. Samim time javljaju se prvi softveri poput NASTRAN-a (sponzoriran od strane NASA-e) koji se danas broje u desetinama pa i stotinama. Za razliku od prvih alata za rješavanje problema metodom konačnih elemenata, današnji se većinom baziraju na programskom jeziku C++ koji je nastao 1985. Izbor C++-a je logičan radi njegove brzine i podrške objektno orijentiranom programiranju što omogućava modularan pristup razvoju što se pokazalo kao dobar model razvoja softvera. Osim toga većina modernih programa nudi i grafičko sučelje (Autodesk Simulation), ali i programsko sučelje prema višim programskim jezicima poput Pythona, Rubya, i Julije čime se pokušava postići apstrakcija između softvera za rješavanje i specifičnog problema potrebnog za riješiti. [33]

U ranim 1990-tim objektno orijentirano programiranje (OOP) postaje dominantna programerska paradigma obećavajući mogućnost ponovne upotrebe komponenata i proširenje postojećih putem nasljeđivanja, motivirana rastućoj popularnosti grafičkih korisničkih sučelja. OOP se temelji na konceptu ”objekata” koji najčešće sadrže podatke, te metodama koje izvršavaju neke procedure nad objektima. Samim time gotovo se svi softveri za rješavanje problema metode konačnih elemenata nakon toga razvijaju pomoću objektno orijentirane paradigme koja uvelike olakšava proširenje generalnih značajki programa na specifične zahtjeve ne toliko raširenih domena. To dovodi do daljnjeg rasta u broju softvera koji su

specijalizirani za rješavanje metode konačnih elemenata koji danas obuhvaćaju raznolike probleme i domene.

U ovom radu opisat ćemo OOFEM koji je jedan takav alat, kako ga koristiti pomoću Pythona i komandne linije, te kako mu dodati nove značajke. OOFEM koji je akronim za Object Oriented Finite Element Method je softver za rješavanje problema metode konačnih elemenata razvijen s GNU General Public License licencom što omogućava njegovu slobodnu primjenu i modifikaciju čak i u komercijalne svrhe. Borek Patzak započeo je razvoj OOFEM-a 1993. godine.

Rad se sastoji od 3 poglavlja. Nakon Uvoda, u prvom poglavlju predstavljamo metodu konačnih elemenata gdje će se ukratko opisati najvažnija svojstva potrebna za razumijevanje korištenja softvera za njezino rješavanje. Poglavlje 2 opisuje OOFEM. U njemu će biti predstavljeno što program očekuje kao ulaz, kako ga proširiti u slučaju nepostojanja traženih značajki te njegovo upravljanje putem Python programskog jezika. Također, bit će objašnjeno što izlazni podaci znače te kako ih vizualizirati radi jasnije interpretacije rezultata. U trećem poglavlju pokazat ćemo na testnom primjeru kojeg ćemo ručno riješit u prvom poglavlju kako izgleda učitavanje podatak, izračunavanje, te vizualizacija dobivenih rezultata.

# Poglavlje 1

# Metoda konačnih elemenata

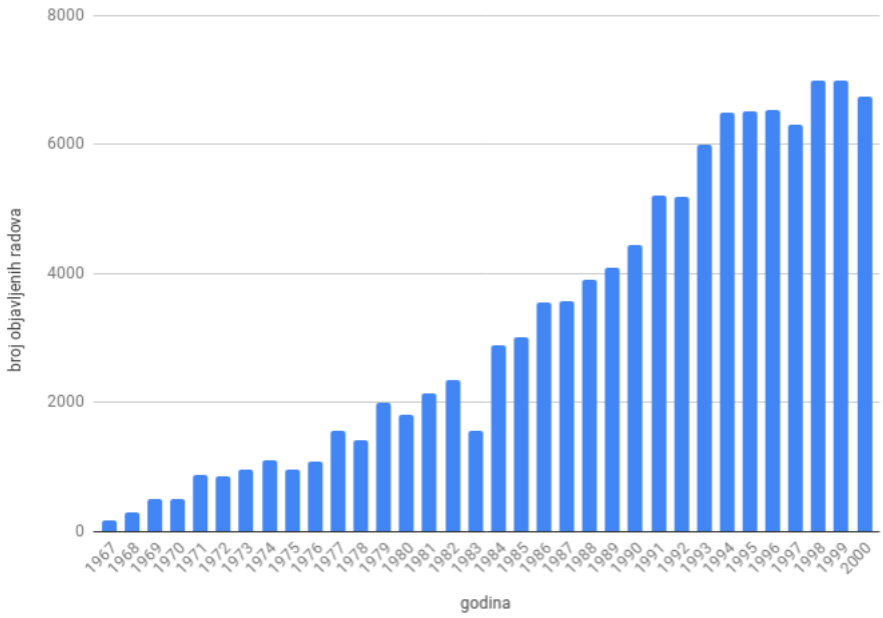
U ovom poglavlju cilj nam je objasniti ključne pojmove vezane uz metodu konačnih elemenata potrebne za razumijevanje računarskih alata za njihovo računanje tako da se njihovo korištenje ne svede samo na "crnu kutiju" kojoj se da ulaz i ona vrati slike. Za početak dat ćemo kratki povijesni uvod. Nakon uvoda radi lakšeg snalaženja opisat ćemo nekoliko konačnih elemenata. Nakon toga, radi lakšeg razumijevanja apstraktnoga opisa metode konačnih elemenata zadajemo jednostavan zadatka koji ćemo riješiti putem metode konačnih elemenata i koji će pokazati što se sve koristi pri rješavanju problema. Za kraj, ukratko ćemo objasniti fizikalne poveznice s matematičkom pozadinom što će nam dati bolju intuiciju o samim podacima. Radi jednostavnosti i povezanošću s temom rada ograničili smo se samo na probleme mehanike deformabilnih tijela u elastičnom području. Ovo poglavlje temelji se na [31], [8] i [20].

## 1.1 Razvoj metode konačnih elemenata

Iako točan datum izuma metode konačnih elemenata nije poznat, postupak je nastao iz potrebe za rješavanjem složenih problema iz područja elastičnosti i strukturne analize u civilnom i aeronautičkom inženjerstvu. A. Hrennikoff je 1941. godine pokušao riješiti problem teorije elastičnosti pri čemu je na temelju intuicije elastični kontinuum podijelio na više jednostavnih međusobno spojenih štapnih elemenata. Na taj je način kontinuirani sustav zamijenio rešetkastom konstrukcijom za koju je bilo moguće naći rješenje pomoću tada već poznatih standardnih metoda. U Kini, u kasnijim pedesetim i početkom šezdesetih godina, na temelju izračuna konstrukcija brane K. Feng predložio je sustavnu numeričku metodu za rješavanje parcijalnih diferencijalnih jednadžbi. Metoda je nazvana metodom konačnih razlika temeljenih na načelu varijacije, što je bio još jedan neovisan izum metode konačnih elemenata. Iako su pristupi različiti, oni dijele jednu bitnu karakteristiku koja je nadalje postala okosnica metode konačnih elemenata, a to je mrežna diskretizacija kontinu-

irane domene u skupinu diskretnih poddomena, obično nazvanih elemenata. A. Hrennikoff diskretizira domenu pomoću analogije rešetke, dok R. Courant uzima pristup tako da dijeli domenu u konačne trokutaste podregije za rješavanje eliptičnih parcijalnih diferencijalnih jednadžbi (PDE) drugog reda koji proizlaze iz problema torzije cilindra. Courantov je doprinos bio evolucijski, na temelju velikog broja prethodnih rezultata za PDE koje su razvili Rayleigh, Ritz i Galerkin. [19] [7]

Podjela elastičnog kontinuuma na više elemenata, koji čine manje dijelove kontinuuma koji se spajaju u zajedničkim točkama, prvi se put pojavljuje u radovima J. H. Argzrisa 1954. goidne i M. J. Turnera, R. W. Clouga, H.C. Martina i L.J. Toppa 1956. godine. Metoda konačnih elemenata dobila je pravi poticaj 1960-ih i 1970-ih godina. Matričnim zapisom u metodama analize konstrukcija omogućava se primjena na računalima što daje dodatni poticaj za daljnji razvoj. To je najbolje vidljivo iz broja publikacija na temu metode konačnih elemenata koji naglo raste. Prema [35], broj radova je s 10 i 15 1961. godine i 1962. godine narastao na 162 rada 1967. godine te na 303 rada 1968. godine. Rast popularnosti metode konačnih elemenata dobro se vidi iz idućeg grafa s brojem publikacija po godini. Paralelno s time rastao je i broj dostupnih softverskih programa. Tako su

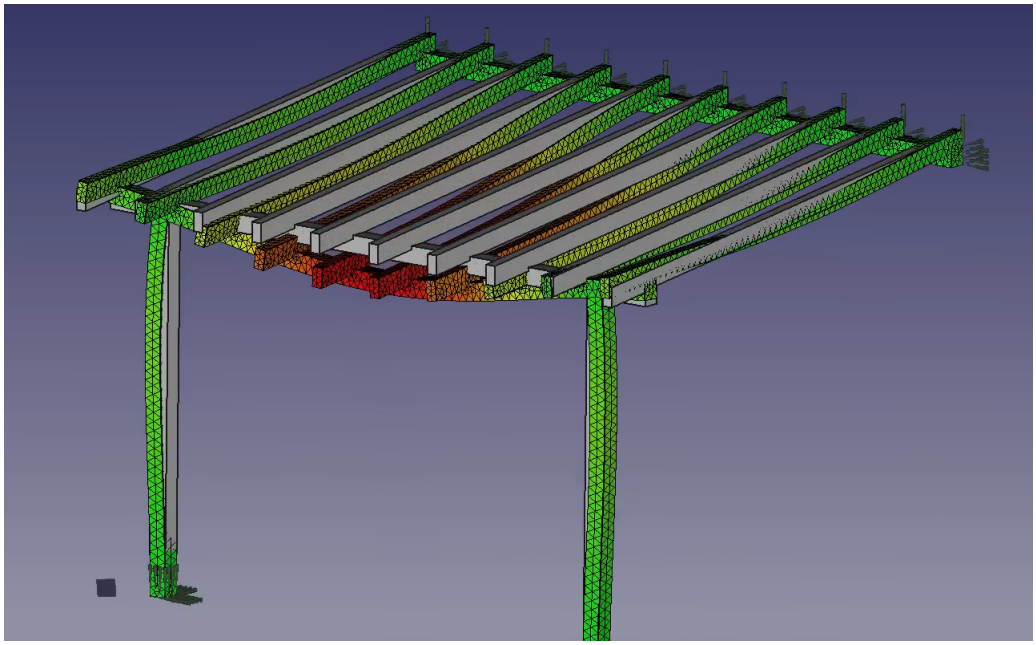



Slika 1.1: Pregled objavljene literature iz metode konačnih elemenata za razdoblje 1967.-2000. [21]

Abaqus, Adina, Ansys, i drugi softverski paketi nastali već 1970-tih godina od kojih se mnogi i dan danas koriste. Još jedan od ključnih razloga za strmoglavi rast popularnosti je

to što se metoda konačnih elemenata počela primjenjivati ne samo u strukturnoj analizi već i u analizi toka fluida, prijenosu topline, elektrostatičkim problemima i još brojim drugim područjima.

Danas su strojarstvo, brodogradnja, zrakoplovstvo, automobilska industrija i građevina nezamislivi bez metode konačnih elemenata koja im pomaže u različitim pogledima čineći krajnji proizvod pouzdanijim i jeftiniji. Rastom računalne snage računala te modernog računanja na grafičkim karticama metoda konačnih elemenata može lagano rukovati s vrlo složenim geometrijama što je izdvaja u usporedbi s ostalim metodama ostalih metoda. Važne značajke su i to što može obraditi složena ograničenja na rubne uvjete i opterećenja poput pritiska, temperature i inicijalnih sila.



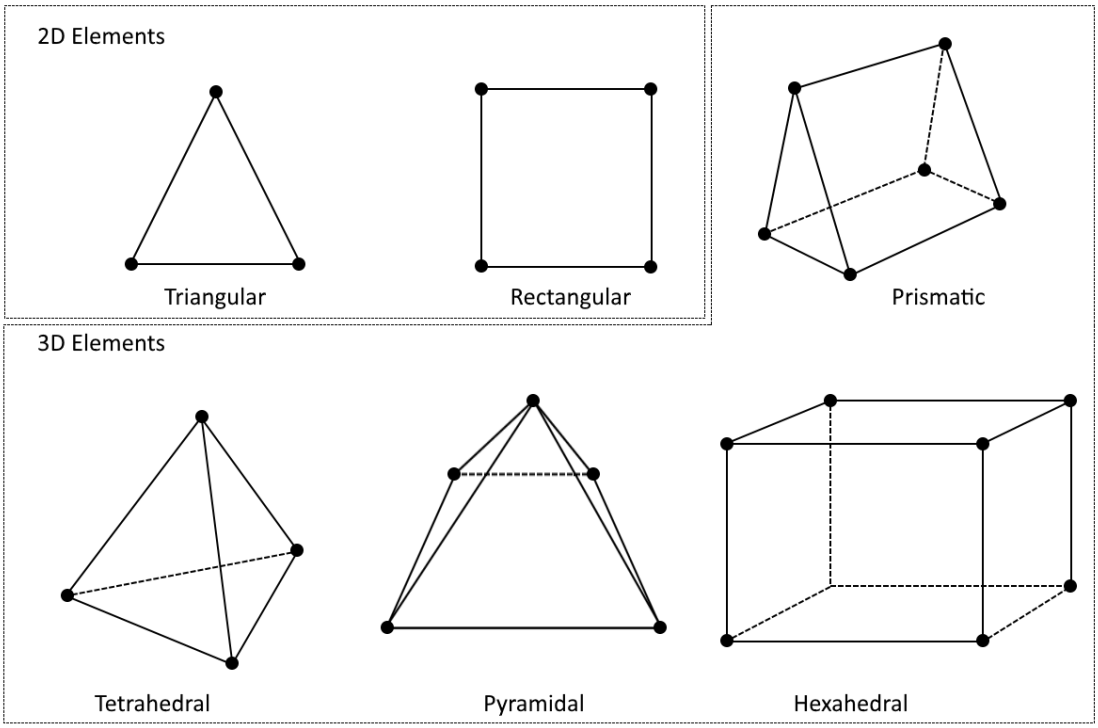
Slika 1.2: Primjer analize problema nadstrešnice u softverskom alatu FreeCAD [1]

## 1.2 Osnovni konačni elementi i njihova primjena

Kako je metoda konačnih elemenata numerička metoda koja kontinuum s beskonačno stupnjeva slobode gibanja zamjenjuje s diskretnim modelom međusobno povezanih elemenata postavlja se pitanje kakvi su ti elementi te kakvi sve mogu biti. Ovisno o obliku i nepoznatim parametrima u čvorovima izvedeni su različiti tipovi konačnih elemenata. Veći broj nepoznanica zahtjeva složeniju interpolacijsku funkciju u području elemenata. Jed-

nostavniji konačni elementi koji se najčešće primjenjuju u mehanici deformabilnih tijela prikazani su na idućoj slici. Nepoznati parametri u čvorovima, koji u metodi pomaka u mehanici deformabilnih tijela opisuju pomake i derivacije pomaka, stupnjevi su slobode elemenata.

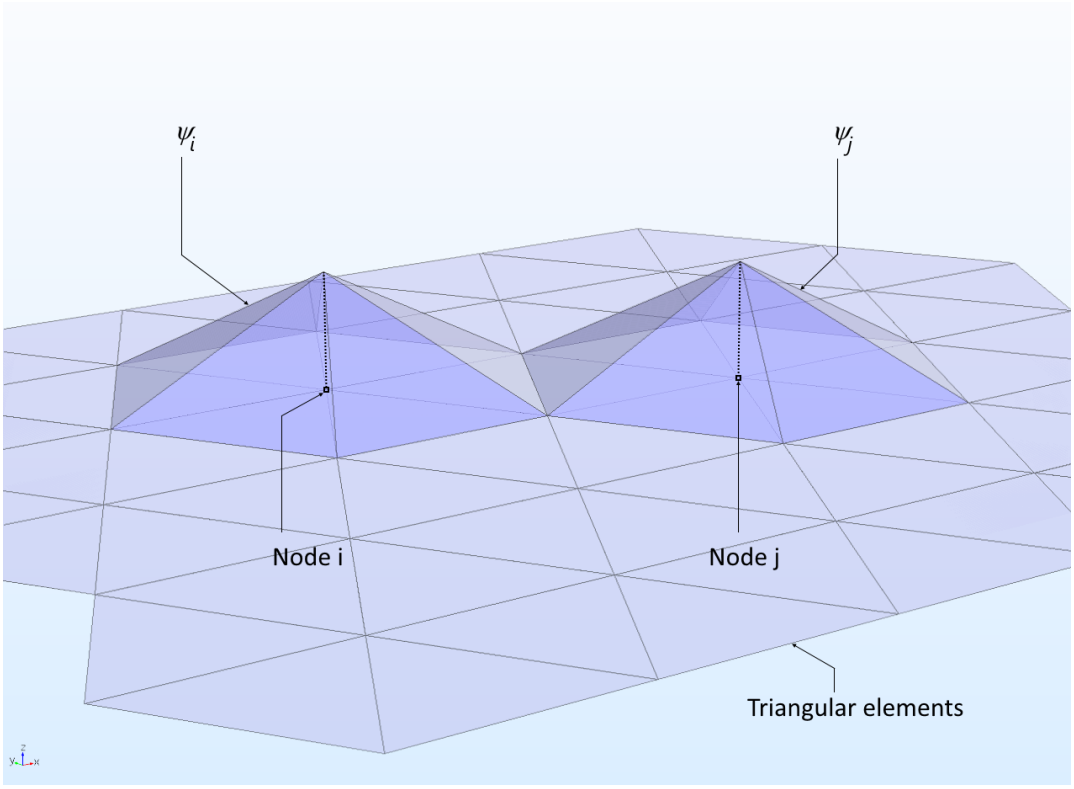
Za linearne funkcije u 2D i 3D, najčešći oblici konačnih elementa prikazani su na slici 1.3. Pomoću konačnih elementa za dvodimenzionalnu analizu prikazanih na slici moguće je opisati ravninsko stanje naprezanja i deformacije pri čemu su nepoznati parametri u čvorovima dvije komponente pomaka. U 2D, četverokutni elementi se često primjenjuju na analizi mehaničkih struktura. Također se mogu koristiti za granični sloj umrežavanja u računalnoj dinamici fluida i modeliranju toplinskog prijenosa. Elementi za trodimenzionalnu analizu s čvorovima po tri komponente pomaka u pravcu Kartezijevih koordinatnih osi koji su kockastog oblika obično se primjenjuju na strukturnu mehaniku i umrežavanje graničnog sloja dok se piramidalni elementi obično postavljaju na vrh elementa rubnog sloja.




Slika 1.3: Položaj čvorova i geometrija za 2D i 3D linearne elemente. [5]

Na slici 1.4 prikazane su po dijelovima linearne funkcije s linearnom osnovom koje

definiraju trokutastu mrežu koja je sastavljena od trokutastih linearnih elementa. Temeljne funkcije izražene su kao funkcije položaja čvorova ($x$ i $y$ u 2D i $x$, $y$ i $z$ u 3D).

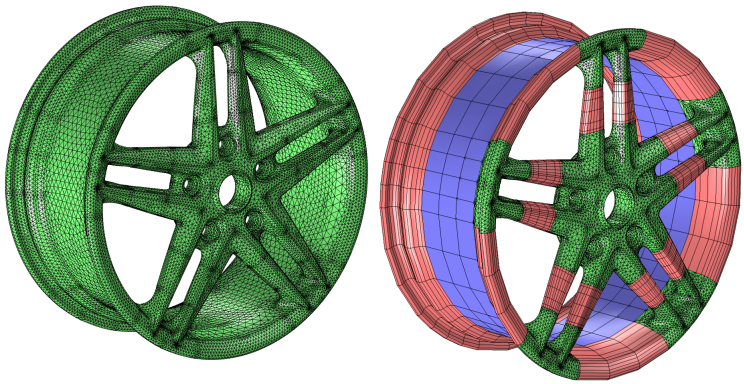



Slika 1.4: Dvije osnovne funkcije koje dijele jedan element vrh, ali se ne preklapaju u 2D domeni. [5]

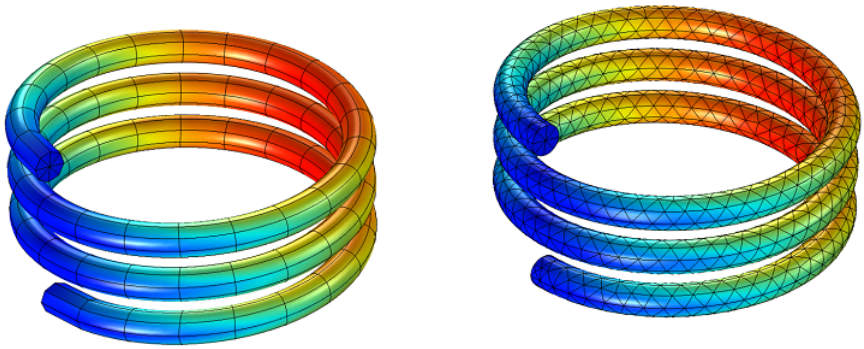
U praksi objekti koje je potrebno modelirati su ne rijetko složeniji te samim time zahtijevaju i složenije konačne elemente. Jedan primjer je element za analizu ljusaka sa šest stupnjeva slobode (položaj čvorova u $x$, $y$ i $z$ osi te rotacije po tim osima). Postoje i složeniji dvostruko zakrivljeni ljuskasti elementi koji u čvorovima mogu imati i znatno više stupnjeva slobode. Također, svaki od prikazanih elementa moguće je proširiti dodavanjem čvorova duž bridova ili po površini elementa. Ukoliko dodani, ti novi čvorovi ne služe za konstrukciju novih elemenata već služe za dobivanje točnijih rezultata tako da omogućuju polinomijalnim funkcijama u tim čvorovima interpolaciju dovoljno glatkih funkcija na domeni.

Nadalje dajmo 2 primjera različitih konstrukcija podijeljenih na konačne elemente na koje su primijenjeni elementi iz gornjeg dijela. Na slici 1.5 dajemo primjer podjele naplatka kotača na konačne elemente. Lijeva podjela sastoji se samo od tetraedara kojih

ima oko 145,000 s oko 730,000 stupnjeva slobode, dok je desna sastavljena od tetraedara (zeleni), kvadara (plavi) i prizmi (rozi). Desna podjela ima oko 78,000 elemenata i nešto manje od 414,000 stupnjeva slobode što ju čini gotovo dvostruko manjom od lijeve. Na idućoj slici, slika 1.6 imamo dvije različite podjele opterećene opruge na konačne elemente koristeći samo prizme odnosno tetraedre. Dok mreža s prizmama ima oko 504 elemenata i 9526 stupnja slobode, mreža s tetraedrima ima 3652 tetraedra i 23,434 stupnja slobode.



Slika 1.5: Primjer mrežastog modela automobilskog naplatka [6]



Slika 1.6: Primjer mrežastog modela opterećene opruge [6]

Kao što je vidljivo iz gornjih slika, čak i za relativno jednostavne konstrukcije mreža brzo postaje prevelika za analognu izradu stoga se oslanjamo na softverske programe za njihovu izradu. Što se samog postupka tiče postoji mnogo različitih pristupa izradi mrežastog

modela, ali klasifikaciju mreže možemo podijeliti na strukturiranu mrežu, ne strukturiranu te hibridnu koja uzima prednosti svake od nje. Postoji mnogo alata koji služe za pretvaranje modela izrađenih u softverima poput FreeCADa [13], AutoCADa [2] i Salomea [30] u mrežasti model. Neki od popularnijih su GMSH [15] i Gridgen [18]. Popularni komercijalni softveri za rješavanje problema pomoću metode konačnih elemenata ujedinjuju dizajniranje modela, pretvorbu modela u mrežasti te računanje metodom konačnih elemenata. Također, radi postizanja točnijih rezultata i manje greške koristi se i h/p-metoda koja mijenja veličinu i stupnjeve slobode elemenata te tako povećava preciznost izračuna na mreži nakon računanja metode konačnih elemenata u kritičnim područjima. Tako dolazimo do iterativnog postupka generiranja mreže koja svakom iteracijom postaje profinjeniji te samim time i točniji.

U ovom radu naglasak je na postupak nakon dobivene mreže, stoga se neće ići u dubinu njezine izrade. Više o izradi mreže može se naći u [34]

## 1.3 Primjer zadatka i rješenja

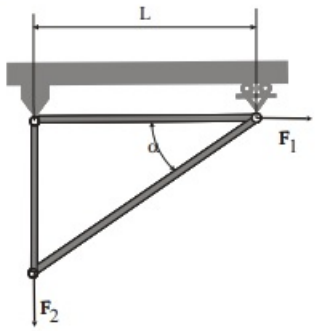
Radi lakšeg razumijevanja metode konačnih elemenata i idućeg potpoglavlja, u ovome ćemo napraviti statički proračun sila na jednostavnom dvodimenzionalnom primjeru sastavljenom od tri štapna elementa. U rješenju zadatka pojavit će se svi koraci koje detaljnije i na apstraktnijoj razini opisujemo u idućem dijelu. Osim toga, pojavit će se i nekoliko stvari na koje nismo stavili naglasak poput vrste materijala i slično. Također, uvodimo i nekoliko jednadžbi za koje ćemo kasnije vidjeti analogone u jednodimenzionalnima zajedno s njihovim opisima. U primjeru rješavamo problem metodom konačnih elemenata sve dok ne dobijemo sile i pomake po štapnim elementima.
Zadatak je napraviti:

1. Statički proračun sila u štapovima i reakcija u osloncima.

2. Pomoću metode konačnih elemenata odrediti pomake čvorova i sile u štapovima

Na slici 1.7 vidimo problem koji moramo riješiti. Sa slike možemo očitati da se problem sastoji od 3 štapna konačna elementa spojena u 3 zajednička čvora. Također, očito je da je gornji lijevi čvor (čvor 1) fiksan, to jest on se ne može micati (ima 0 stupnjeva slobode). Gornji desni čvor (čvor 2) je slobodan samo u jednom smjeru te se može pomicati lijevo-desno. Donji čvor (čvor 3) nema nikakvih ograničenja što mu daje mogućnost pomaka po cijeloj ravnini (2 stupnja slobode).

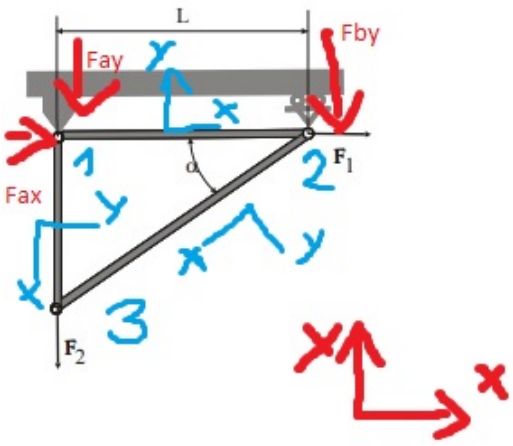
Za prvi dio zadatka koji je preduvjet za računanje pomaka čvorova i sile u štapovima moramo izračunati statičke sile u čvorovima i štapovima.
Radi lakšeg snalaženja konstruiramo vlastitu notaciju tako da je $L_2$ duljina između čvora




$E = 200$ GPa
$\alpha = 30°$
$F_1 = 15$ kN
$F_2 = 5$ kN
$A = 10$ mm$^2$
$L = 2{,}5$ m

Slika 1.7: Zadatak

1 i 3, a $L_3$ duljina između čvorova 2 i 3. Na slici 1.8 su označeni i lokalni koordinatni sustavi za svaki konačni element s plavom bojom te globalni koordinatni sustav s crvenom. Također osim sila $F_1$ i $F_2$ označene su i sile $Fa_x$, $Fa_y$ i $Fb_y$.




Slika 1.8: Slika s oznakama sila, čvorova i koordinatnih sustava

Prvo izračunamo duljine preostalih štapova:

$$L_2 = L \tan \alpha = 2.5 \tan 30 = 1.44337m$$

$$L_3 = \sqrt{L^2 + L_2^2} = 2.88675m$$

Statička sila po koordinatnim osima mora biti jednaka nuli. Statički proračun sila u štapovima i reakcija u osloncima:

$$\sum x = 0$$
$$Fa_x + F_1 = 0$$
$$Fa_x = -F_1 = -15000N$$

$$\sum Mb_x = 0$$
$$-Fa_y L - F_2 L = 0$$
$$Fa_y = \frac{-F_2 L}{L} = -F_2 = -5000N$$

$$\sum y = 0$$
$$-Fb_y - Fa_y - F_2 = 0$$
$$Fb_y = -F_2 - Fa_y = 5000N - 5000N = 0N$$

Da bi mogli koristiti metodu konačnih elemenata moramo uskladiti lokalni i globalni koordinatni sustav tako da se pomaci za svaki čvor računaju u pravom smjeru. Računamo transformacijske matrice i globalne matrice krutosti za sve konačne elemente.

**Konačni element broj 1 (štap između čvorova 1 i 2):**
Os $x$ lokalnog koordinatnog sustava usmjerena je od čvora 1 prema 2 stoga se lokalni i globalni koordinatni sustavi poklapaju te je kut između globalnih i lokalnih os 0°.
Lokalna matrica krutosti:

$$k_1 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \frac{EA}{L} \tag{1.1}$$

Transformacijske matrice:

$$T_1 = I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = T^{-1} \tag{1.2}$$

Globalna matrica krutosti:

$$K_1 = T_1^{-1} k1 T_1 = \begin{bmatrix} 800000 & 0 & -800000 & 0 \\ 0 & 0 & 0 & 0 \\ -800000 & 0 & 800000 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.3}$$

**Konačni element broj 2 (štap između čvorova 1 i 3):**
Os $x$ lokalnog koordinatnog sustava usmjerena je od čvora 1 prema 3 stoga je kut između globalne i lokalne koordinatne osi $x$ jednak 270°.
Lokalna matrica krutosti:

$$k_2 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \frac{EA}{L_2} \tag{1.4}$$

Transformacijske matrice:

$$T_2 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad T_2^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{1.5}$$

Globalna matrica krutosti:

$$K_2 = T_2^{-1} k2 T_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1385646.0921316 & 0 & -1385646.0921316 \\ 0 & 0 & 0 & 0 \\ 0 & -1385646.0921316 & 0 & 1385646.0921316 \end{bmatrix} \tag{1.6}$$

**Konačni element broj 3 (štap između čvorova 2 i 3):**
Os $x$ lokalnog koordinatnog sustava usmjerena je od čvora 2 prema 3 stoga je kut između globalne i lokalne koordinatne osi $x$, pa i $y$ jednak 210°.
Lokalna matrica krutosti:

$$k_3 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \frac{EA}{L_3} \tag{1.7}$$

Transformacijske matrice:

$$T_3 = \begin{bmatrix} -\frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & -\frac{\sqrt{3}}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}, \qquad T_3^{-1} = \begin{bmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & -\frac{\sqrt{3}}{2} & \frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}, \tag{1.8}$$

Globalna matrica krutosti:

$$K_3 = T_3^{-1} k3 T_3 = \begin{bmatrix} 519615 & 300000 & -519615 & -300000 \\ 300000 & 173205 & -300000 & -173205 \\ -519615 & -300000 & 519615 & 300000 \\ -300000 & -173205 & 300000 & 173205 \end{bmatrix} \tag{1.9}$$

Prije zbrajanja matrica krutosti pojedinih elemenata potrebno ih je proširiti na odgovarajuće dimenzije. Time dobivamo iduće matrice za konačne elemente:

$$K_1 = \begin{bmatrix} 800000 & 0 & -800000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -800000 & 0 & 800000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.10}$$

$$K_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1385646.0921316 & 0 & 0 & 0 & -1385646.0921316 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1385646.0921316 & 0 & 0 & 0 & 1385646.0921316 \end{bmatrix} \tag{1.11}$$

$$K_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 519615 & 300000 & -519615 & -300000 \\ 0 & 0 & 300000 & 173205 & -300000 & -173205 \\ 0 & 0 & -519615 & -300000 & 519615 & 300000 \\ 0 & 0 & -300000 & -173205 & 300000 & 173205 \end{bmatrix} \tag{1.12}$$

Sada spajanjem matrica krutosti (1.10), (1.11) i (1.12) dobivamo globalnu matricu krutosti čitave strukture.

$$K = K_1 + K_2 + K_3$$

$$= \begin{bmatrix} 800000 & 0 & -800000 & 0 & 0 & 0 \\ 0 & 1385646.09 & 0 & 0 & 0 & -1385646.09 \\ -800000 & 0 & 1385646.09 & 300000 & -519615 & -300000 \\ 0 & 0 & 300000 & 173205 & -300000 & -173205 \\ 0 & 0 & -519615 & -300000 & 519615 & 300000 \\ 0 & -1385646.09 & -300000 & -173205 & 300000 & 1558851.25 \end{bmatrix} \tag{1.13}$$

U prvom dijelu zadatka izračunali smo sile, te sad možemo pomoću njih konstruirati vektor opterećenja.

$$F = \begin{bmatrix} Fa_x \\ Fa_y \\ F_1 \\ Fb_y \\ 0 \\ F_2 \end{bmatrix} = \begin{bmatrix} -15000 \\ -5000 \\ 15000 \\ 0 \\ 0 \\ -5000 \end{bmatrix} \tag{1.14}$$

Sa slike 1.7 očitali smo da se pomaci mogu dogoditi samo u čvorovima 2 i 3 stoga vektor pomaka koji želimo izračunati izgleda:

$$u = \begin{bmatrix} 0 \\ 0 \\ u_{x2} \\ 0 \\ u_{x3} \\ u_{y3} \end{bmatrix} \tag{1.15}$$

Sad se pomoću vektora opterećenja (1.14) i matrice krutosti (1.3) računa vektor pomaka (1.15) po formuli $F = Ku$. Uklanjanjem dijelova gdje nema pomaka dobivamo idući sustav jednadžbi:

$$\begin{bmatrix} 15000 \\ 0 \\ -5000 \end{bmatrix} = \begin{bmatrix} 1319615.48 \\ -519615 \\ -300000 \\ -519615 \\ 519615 \\ 300000 \\ -300000 \\ 300000 \\ 1558851.25 \end{bmatrix} \begin{bmatrix} u_{x2} \\ u_{x3} \\ u_{y3} \end{bmatrix} \tag{1.16}$$

Rješavanjem sustava dobivamo vektor pomaka

$$u = \begin{bmatrix} 0.01875 \\ 0.0208333 \\ -0.00360843 \end{bmatrix} \tag{1.17}$$

te sada cjeloviti vektor pomaka glasi:

$$u = \begin{bmatrix} 0 \\ 0 \\ 0.01875 \\ 0 \\ 0.0208333 \\ -0.00360843 \end{bmatrix} \tag{1.18}$$

Da bi odredili reakcije oslonaca množimo cjelovitu matricu krutosti (1.3) s cjelovitim vektorom pomaka (1.18) $f = Ku$ i dobivamo

$$f = \begin{bmatrix} -15000 \\ 5000 \\ 15000 \\ 4.5474710^{-13} \\ -1.81898910^{-12} \\ -5000 \end{bmatrix} \tag{1.19}$$

Za određivanje unutrašnjih sila u štapovima (konačnim elementima) potrebni su nam ukupni pomaci po konačnim elemenata što zahtjeva vraćanje pomaka iz globalnog koordinatnog sustava u lokalni koji se dobiva globalnog množenjem pomaka s odgovarajućom matricom transformacije. Za konačni element 1 ukupni pomak glasi

$$\delta u_1 = u_{x2} - u_{x1} \tag{1.20}$$

Pomak u lokalnom koordinatnom sustavu dobivamo s:

$$U_1 = T_1 u_1 = I u_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0.01875 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.01875 \\ 0 \end{bmatrix} \tag{1.21}$$

Apsolutni pomak konačnog elementa dobivamo iz jednadžbe (1.20) i iznosi

$$\Delta u_1 = 0.01875 - 0 = 0.1875$$

Analogno se dobivaju i apsolutni pomaci konačnih elemenata 2 i 3 koji iznose:

$$\Delta u_2 = 0.003608425$$

$$\Delta u_3 = 0$$

Sada imamo sve informacije za izračunati sile u svakom štapu (konačnom elementu) koje iznose:

$$p_1 = \frac{EA}{L}\Delta u_1 = 15000N$$
$$p_2 = \frac{EA}{L_2}\Delta u_2 = 5000N$$
$$p_3 = \frac{EA}{L_3}\Delta u_3 = 0N$$

Ovaj primjer pokazao je na jednostavnoj strukturi što se sve treba izračunati prije dobivanja završnih rezultata. Rješavanje se temeljilo na metodi direktne formulacije konačnih elemenata (Direct stiffness method) o kojoj se više detalja može naći u [11], [10] i u šestom poglavlju [31] koja je razvijena posebno za učinkovito i lako implementiranje u računalni softver za procjenu kompliciranih struktura koje sadrže veliki broj jednostavnih konačnih elemenata. Također, danas se velik broj program za rješavanje metode konačnih elemenata temelji na upravo toj metodi.

## 1.4 Računanje metodom konačnih elemenata

Ovo potpoglavlje temelji se na [31] i [12]. Istaknuli smo da je u osnovi metode konačnih elemenata tražiti rješenje PDJ u diskretnom prostoru po djelovima polinomijalnih funkcija. Takvi prostori funkcija se zadaju koordinatnim zapisom u "lokalnoj" bazi funkcija, tj. svaka funkcija je prikaziva kao linearna kombinacija osnovnih funkcija (vidi sliku 1.4 za osnovne funkcije po dijelovima linearnih elemenata). Pri tome su osnovne funkcije konačnog nosača i opisuju strukturu samo lokalno. Prvi korak je kako povezati koordinate kojima opisujemo funkciju, koje zbog lokalnog nosača osnovnih funkcija nužno imaju samo lokalni doseg, i globalna svojstva funkcije koju želimo konstruirati. Postoje dva osnovna pristupa metodi konačnih elemenata. Prva je metoda sila (metoda fleksibilnosti). U metodi sila osnovne nepoznate veličine u problemu koji se analiziraju su sile. Da bi se dobile jednadžbe strukture, prvo se postavljaju jednadžbe ravnoteže. Rezultat je sustav algebarskih jednadžbi u kojima se nepoznate veličine sile određuju iz jednadžbi. Drugi pristup je metoda pomicanja (metoda krutosti) u kojoj postoje osnovna nepoznata kretanja u čvorovima. Kako bi se postigli uvjeti kompatibilnosti za rješavanje specifičnih problema, potrebno je da su elementi povezani u čvorovima, uz stranice ili odgovarajuće površine prije i nakon opterećenja. Jednadžbe strukture sadrže pomicanje čvora i koriste se jednadžbe ravnoteže i odnos između sile i pomaka. Od dva gore spomenuta pristupa, popularnija je druga metoda, metoda pomaka. Formulacija metode pomaka slična je mnogim strukturnim problemima. Većina softverskih programa se temelji na metodi pomaka.

Nakon što se konstruira mreža međusobno povezanih konačnih elemenata, svakom se elementu pridružuje funkcija pomaka. Svi elementi su izravno ili neizravno povezani,

uključujući čvorove i/ili zajedničke granične linije elemenata i/ili zajedničke površine. Na temelju poznatih vrijednosti napona i deformacije u jednom čvoru i elementu, moguće je odrediti naprezanja i deformacije za bilo koji drugi čvor i element strukture koja se razmatra i čija su svojstva materijala i opterećenja već poznati. Zajedno, strukturne jednadžbe opisuju ponašanje svih čvorova i predstavljaju sustav algebarskih jednadžbi koji je najbolje zapisan u obliku matrice

Prvi korak nakon diskretizacije domene na konačne elemente je izbor funkcije pomaka. Izbor funkcije pomaka obavlja se za svaki element. Funkcija je definirana unutar elementa i koristi vrijednosti izračunate u čvorovima. Za funkcije pomaka najčešće se izabiru linearni, kvadratni ili kubni polinomi. Polinomi se koriste kao funkcije jer su jednostavni za računanje metode konačnih elemenata. Za dvodimenzionalni element, funkcija pomaka je funkcija koordinata u $xy$ ravnini. Funkcije su nepoznate veličine u čvorovima. Za dvodimenzionalne probleme nepoznate veličine su funkcije koordinata $x$ i $y$. Ista funkcija pomaka može se odabrati za svaki element u modelu konačnih elemenata diskretne strukture. Odabrane su funkcije kako bi se postigao kontinuitet pomaka u tijelu korištenjem metode konačnih elemenata između svih elemenata u čvorovima, duž stranica i površine. Nakon odabira funkcije pomaka, uspostavlja se veza između deformacija i pomaka, kao i veza između naprezanja i deformacije.

Nadalje, potrebno je uspostaviti vezu između deformacije i pomaka (kinematičke relacije) te vezu između naprezanja i deformacije (konstitutivne jednadžbe). Ako se ograničimo na jednodimenzionalan problem tada postoji deformacija samo u jednom pravcu (pravac $x$). Tada je deformacija $\epsilon_x$ i ona je povezana s pomakom $u$ u $x$ pravcu. Veza između pomaka i deformacije dana je jednadžbom

$$\epsilon_x = \frac{du}{dx} \tag{1.22}$$

Jednadžba vrijedi za male deformacije i pomake. Jedna od najjednostavnijih konstitutivnih jednadžbi između deformacije i naprezanja je relacija koja se zove Hooke-ov zakon. Za jednodimenzionalni problem veza naprezanja i deformacije je

$$\sigma_x = E\epsilon_x \tag{1.23}$$

gdje je $\epsilon_x$ naprezanje u pravcu $x$, a $E$ modul elastičnosti.

Pomoću prethodnih relacija konstruira se matrica krutosti. U početku, matrice krutosti elemenata i jednadžba elemenata određivane su pomoću koeficijenata krutosti, što je izravno povezano sa strukturnom analizom. Nakon toga razvijeno je nekoliko metoda za određivanje matrice krutosti.

1. Direktan postupak konačnih elemenata (Direct Finite Element Method) - Matrica krutosti povezuje sile u čvorovima elemenata i pomake čvorova. Dobiva se iz ravnotežnih uvjeta za svaki razmatrani element. Izravni pristup računanju matrice kru-

tosti je dobar samo u slučaju jednodimenzionalnih (štapnih) elemenata, međutim, vrlo je prikladan za objašnjenje osnovnog koncepta metode konačnih elemenata.

2. Varijacijski prinicipi (Variational Finite Element Methods) - Prilikom rješavanja složenijih, dvodimenzionalnih i trodimenzionalnih problema teorije elastičnosti, teško je u potpunosti zadovoljiti osnovne relacije konačnih elemenata. To je razlog za primjenu varijacijskih principa u kojima su sadržane relacije teorije elastičnosti pomoću kojih je, uz dopuštenu grešku, moguće provesti približno rješavanje problema. Na taj su način i odgovarajuće jednadžbe teorije elastičnosti približno zadovoljene. Primjer varijacijskih principa su principi virtualnih pomaka i virtualnih sila koji su za rješavanje problema mehanike deformabilnih tijela u elastičnom području ekvivalentni principima minimuma potencijalne energije.

3. Metoda težinskog reziduala (Methods of Weighted Residuals) - Ova se metoda temelji na diferencijalnim jednadžbama razmatranog problema. Metoda se koristi kada nije moguće utvrditi funkcional i za probleme u kojima funkcional uopće ne postoji. Od svih metoda težinskog reziduala najpoznatija je Galerkinova metoda. Na temelju metode reziduala dobivene su jednadžbe koje opisuju ponašanje elemenata. U matričnom obliku to izgleda:

$$\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & \dots & k_{1n} \\ k_{21} & k_{22} & k_{23} & \dots & k_{2n} \\ k_{n1} & k_{n2} & k_{n3} & \dots & k_{nn} \end{bmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

$$f = Kd$$

gdje je:

- $f$ vektor sila u čvorovima elementa,
- $K$ matrica krutosti elemenata i
- $d$ vektor pomaka čvorova elemenata

Osim navedenih postoji još nekoliko metoda, poput Razleigh-Ritzova metode i metode energetske ravnoteže koje nećemo detaljnije opisivati. Korištenjem neke od gore navedenih metoda dobiju se matrice krutosti i jednadžbe pojedinih konačnih elemenata te je kao idući korak potrebno izračunati globalnu matricu krutosti.

Primjenom metode direktne superpozicije (Direct Stiffness Method), jedne od metoda koja se danas najčešće primjenjuje za izvođenje globalne jednadžbe konačnih elemenata i globalne matrice krutosti, matrice pojedinih elemenata se kombiniraju. Prilikom primjene

metode mora se poštivati koncept kontinuiteta ili kompatibilnosti, što zahtijeva da struktura zadržava cjelovitost to jest, ne smije doći do prekida strukture. Globalna jednadžba strukture u obliku matrice je:

$$f = Kd \tag{1.24}$$

gdje je:

- $f$ vektor opterećenja u globalnom koordinatnom sustavu,
- $K$ globalna matrica krutosti i
- $d$ vektor poznatih i nepoznatih pomaka svih čvorova strukture.

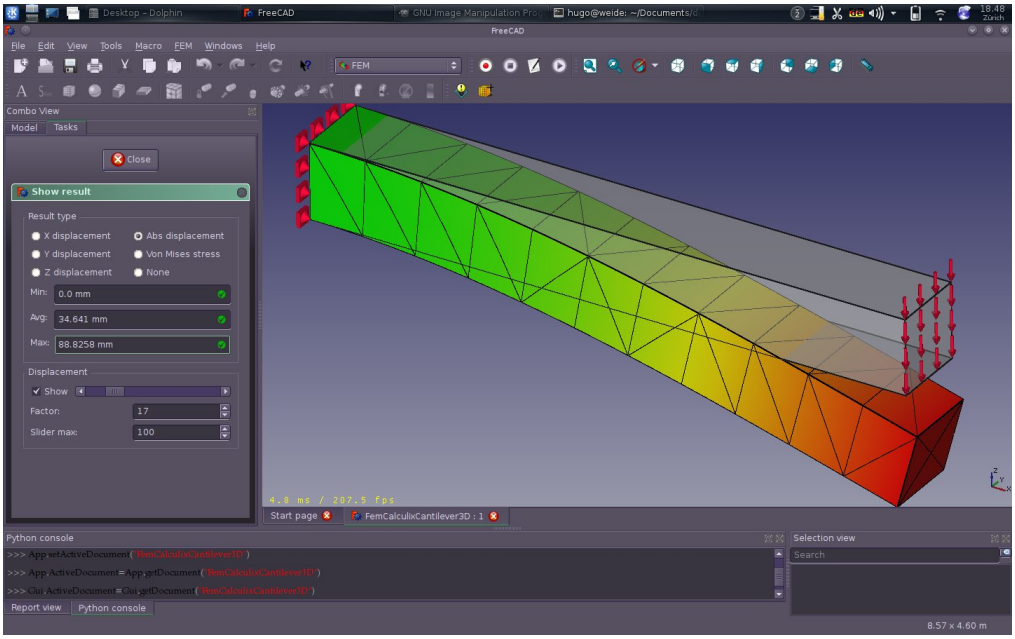
Globalna matrica krutosti $K$ je simetrična i singularna matrica. Problem singulariteta matrice rješava se uvođenjem rubnih uvjeta tako da struktura zadrži postojeće mjesto i da se ne kreće kao kruto tijelo. Tako matrica krutosti postaje pozitivno definitna što je uvjet za mogućnost rješavanja nehomogenog sustava jednadžbi. Formiranje globalne matrice krutosti danas se u računalu najčešće provodi takozvanim postupkom popunjavanja, pri čemu se kinematička matrica transformacije zamjenjuje tablicom iz koje je vidljivo koji lokalni stupnjevi slobode odgovaraju globalnim stupnjevima slobode. Počinje se s nulmatricom te se nadalje popunjava.

Rješavanjem gornje jednadžbe matrične strukture (1.24) u koju su uneseni rubni uvjeti dobivamo vrijednosti globalnog vektora pomaka $d$. Rješavanje sustava provodi se Gaussovom metodom eliminacije ili primjenom neke od iterativnih metoda. Nepoznanice koje se dobiju rješavanjem su pomaci u čvorovima. To su prvi rezultati koji se dobiju primjenom metode konačnih elemenata.

Nakon rješavanja globalnog sustava jednadžbi i izračunavanja globalnog vektora pomaka $d$, potrebno je izračunati naprezanje u konačnim elementima. Za dobivanje tih vrijednosti koristimo veze između deformacija, naprezanja i pomaka iz (1.22) i (1.23). Pomoću navedenih izraza moguće je odrediti naprezanje u proizvoljnoj točki konačnog elementa. Naprezanja se izračunavanju u težištu elementa, u čvorovima ili u točkama numeričke integracije. Važno je naglasiti da se pri gruboj diskretizaciji naprezanja po rubovima susjednih elemenata mogu znatno razlikovati, odnosno naprezanja koja pripadaju različitim susjednim elementima duž zajedničkog ruba mogu biti različitog iznosa. Razlike se mogu smanjiti usitnjavanjem mreže konačnih elemenata. U slučaju različitih vrijednosti duž rubova, za procjenu stanja naprezanja izračunava se srednja vrijednost.

Rezultati dobiveni primjenom metode konačnih elemenata se interpretiraju tako da se odrede mjesta djelovanja najvećih naprezanja i deformacija. Na temelju znanja o materijalima i strukturi tada inženjer donosi daljnje odluke mora li se, i želi li se dizajn strukture mijenjati te se ovisno o odgovoru postupak ponavlja (iterira). Računalni programi za naknadnu obradu (vizualizaciju) pružaju inženjerima pomoć pri analizi izradom toplinskih

karata ”heat map” tako da se dijelovi strukture s većim naprezanjima/deformacijama bojaju u toplije a područja s manjim u hladnije boje. Velik broj softverskih alata za rješavanje metode konačnih elemenata dolazi s ugrađenim softverom za vizualizaciju stoga iteriranje prilikom izrade modela postaje prirodan tok dizajniranja novih struktura.




Slika 1.9: Primjer vizualizacije opterećenja na gredu u softverskom alatu FreeCAD [14]

# Poglavlje 2

# OOFEM

OOFEM je objektno orijentirani program otvorenoga koda (open source) dizajniran za rješavanje mehaničkih, transportnih i fluidnih problema pomoću konačnih elemenata koji radi na različitim platformama [29]. U ovom poglavlju detaljno opisujemo softverski program OOFEM. U početku dajemo kratak opis kako se je program razvijao, te njegovo trenutno stanje. U drugom dijelu dajemo opis strukture koda te arhitektonske odluke. Zatim slijede dijelovi gdje se opisuju postojeći konačni elementi i materijali za te konačne elemente te kako dodati nove. Nakon što završimo opis samog programa slijedi što OOFEM očekuje kao ulazne podatke i mogućnosti što sve OOFEM može analizirati. Kako je za OOFEM predviđeno da čita ulazne podatke iz datoteke autori su radi trenutnog stanja u praksi gdje programski jezik Python dobiva sve više na važnosti i popularnosti odlučili omogućiti zadavanje ulaznih podataka programski putem Pythona. U dijelu 2.6 objašnjavamo što sve i kako možemo koristiti Python pri zadavanju ulaznih podataka. U posljednjem dijelu poglavlja prikazujemo kako analizirati dobivene rezultate što uključuje validaciju vrijednosti i naknadnu analizu putem vizualizacijskih alata radi lakšeg snalaženja u moru vrijednosti sila i naprezanja. Ovo poglavlje temelji se na web stranici i dokumentaciji OOFEM-a. [28] [29]

## 2.1 O OOFEM-u

Cilj OOFEM-a je razviti djelotvoran i robustan alat za izračune metode konačnih elemenata te osigurati modularno i proširivo okruženje za budući razvoj. OOFEM je izdan pod licencom za slobodan softver, GNU Lesser General Public License (LGPL). Više o tipu licence može se pronaći na [17], ali ukratko ova licenca dopušta slobodno korištenje, modifikaciju i redistribuciju softvera. OOFEM je u kontinuiranom razvoju od 1997. godine. Projekt je započeo 1993. godine kao dio doktorske disertacije Boreka Patzaka o računalnom modeliranju betonskih konstrukcija i financiran je djelomično od strane ministarstva zna-

nosti i znanstven zaklade Češke Republike, fondova Europske unije i industrije (LMAT, www.lmat-uk.com). Prema Github-u [26] i Openhub-u [23] projekt sadrži nešto manje od 300,000 linija koda i na njemu su radila 42 suradnika. Openhub procjenjuje da je u taj projekt uloženo 76 godina rada (COCOMO model) što ga s prosječnom cijenom rada stručne osobe od 200,000 kn po godini čini projekt vrijedan 15,200,000 kuna. Važno je još napomenuti da je većina koda napisana u C++-u koji čini 94% projekta i da je projekt dobro dokumentiran (28% izvornog koda su komentari, što ga stavlja iznad prosjeka). OOFEM ima mnogo značajki, od kojih nama najvažnije su iduće:

- Objektno orijentirana arhitektura.
- Modularna i proširiva jezgra za računanje metode konačnih elemenata (OOFEMlib).
  - Potpuno proširiva. Jezgra programa se može proširiti u bilo kojem "smjeru". Moguće je dodavanje nove vrste konačnih elementa, novog modela materijala s bilo kojim tipom i brojem internih parametara povijesti, novih graničnih uvjeta, numeričkih algoritama ili modula analize, kao i mogućnost dodavanja i upravljanja proizvoljnim stupnjevima slobode.
  - Neovisna formulacija problema, numeričkog rješavanja i pohrane podataka. Jezgra pruža neovisne apstrakcije za analizu, opću numeričku metodu i pohranu podataka (rijetke "sparse" matrice). Koncept mapiranja komponenata omogućuje samostalno formuliranje problema i numeričke metode i omogućuje korištenje bilo koje prikladne numeričke metode za rješavanje problema bez promjene. Ovaj koncept dodatno je poboljšan apstraktnim sučeljem za rijetke matrice, koje omogućuje samostalno formuliranje numeričkih metoda nad rijetkim matricama.
  - Potpuna podrška za ponovno pokretanje. Jezgra podržava potpuno ponovno pokretanje iz bilo kojeg prethodno spremljenog stanja.
  - Stupnjevita analiza. Omogućuje grupiranje osnovnih problema, prijenos i dijeljenje rješenja polja između osnovnih potproblema. Opći dizajn omogućuje korištenje različitih diskretizacija za osnovne potprobleme.
  - Paralelna podrška za procesiranje. Temeljena na rastavljanju domene, paradigmama slanja poruka i dinamičnom balansiranju opterećenja računanja. Mnoge strukturne analize mogu se izvoditi paralelno čime se dobivaju velika ubrzanja.
  - Efikasni algoritmi za rješavanje rijetkih matrica. Dostupni su izravni i iterativni algoritmi. Metode izravnog rješavanja uključuju simetrično i nesimetrično rješavanje. Iterativne metode podržavaju mnoge rijetke oblike pohrane i dolaze s nekoliko pretpostavki. Dostupna su i sučelja za biblioteke linearnih metoda za programe poput IML, PETSc (serijski i paralelni), PARDISO, SuperLu i SPOOLES.

  - Podrška za proširenu metodu konačnih elemenata (eXtended Finite Element Method XFEM). Ugrađena reprezentacija za globalne funkcije obogaćivanja i njihov geometrijski opis. Generalna integracijska pravila pružaju podršku za implementiranje XFEM baziranih algoritama.

- Modul za strukturnu mehaniku.
  - Različite procedure za analizu. Linearna statička i linearnu dinamička (analiza vrijednosti svojstvenih vektora, direktne metode integracije - implicitno i eksplicitno); nelinearna statička (CALM rješavanje), nelinearna dinamička (eksplicitna, paralelna verzija).
  - Velika biblioteka materijala, uključujući najsuvremenije modele za nelinearne mehanike prijeloma kvazi lomljivih materijala. Više o tome u 2.3.
  - Prilagodiva analiza. Linearna i nelinearna.
  - Napredne značajke modeliranja. Čvorovi robovi, lokalni koordinatni sustavi, aktivacija / deaktivacija elemenata i još mnogo toga.
  - Paralelna analiza. Eksplicitna nelinearna dinamika koja koristi metodu razlaganja domene, linearna i nelinearna statička (zahtijeva PETSC).

- Naknadna analiza (post processing).
  - Ugrađeno X-window procesiranje.
  - Pretvorba izlaza u VTK format za korištenje vizualizacijskih alata poput MayaVi ili ParaView za vizualizaciju na različitim platformama (Windows, MacOS i Unix).

- Sučelje prema generatorima mreža T3d i Targe2, te alat za unos podataka iz UNV formata.

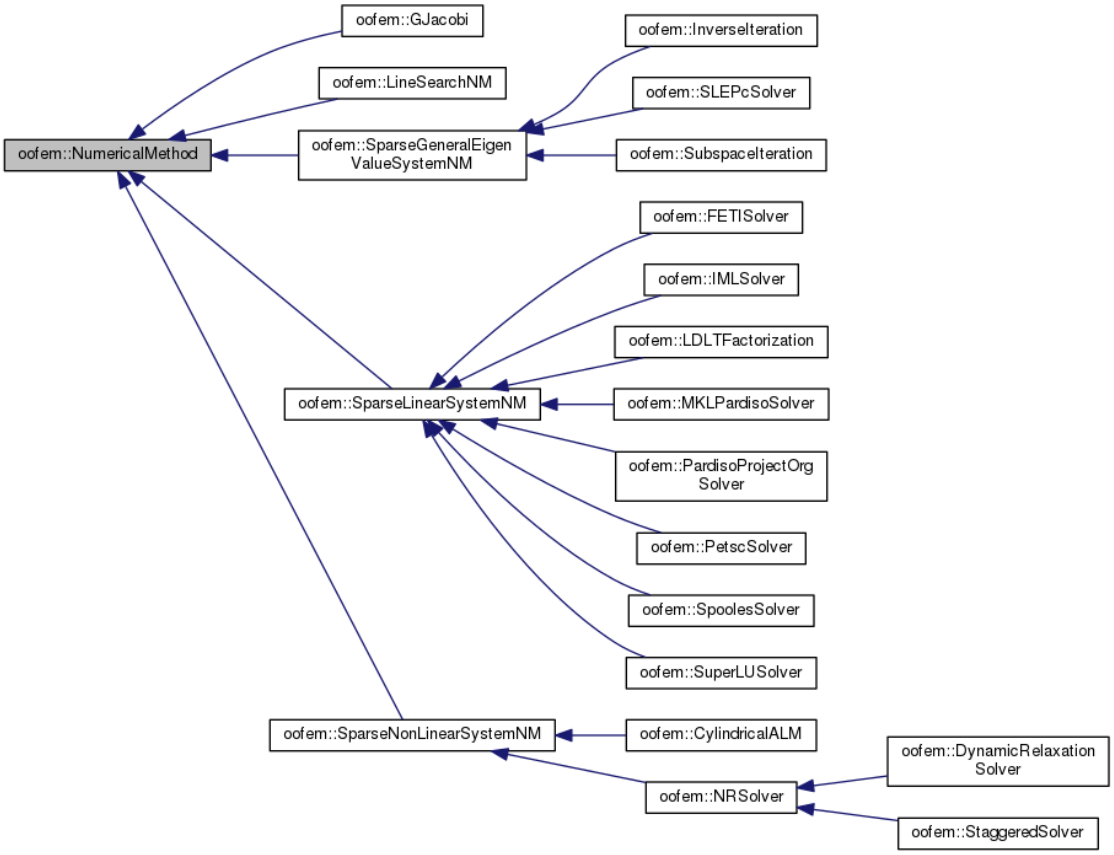
- Sučelje prema bibliotekama za dobivanje svojstvenih vrijednosti (trenutno su podržani PETSc, SLEPc, IML, PARDISo, SuperLu i SPOOLES).

Osim već nabrojanih OOFEM ima i module za rješavanje problema dinamike fluida i prijenosa topline.

## 2.2 Struktura koda OOFEM-a

U poglavlju o metodi konačnih elemenata vidjeli smo da postoji više načina rješavanja i nekoliko različitih problema (strukturna analiza, analiza fluida, analiza prijenosa topline, elektrostatički problemi) koji se rješavaju na gotovo isti način, kao i velik broj različitih

konačnih elemenata koji moraju poštivati zajednička svojstva. Sve to nam nameće ideju o objektno orijentiranom pristupu. Tim su se principom vodili i arhitekti OOFEM programa. Tako se cijela arhitektura OOFEM-a bazira na nasljeđivanju općenitih apstraktnih klasa koje implementiraju specifična svojstva za određeni problem. Na slici 2.1 vidimo jedan takav primjer. Na slici 2.1 je prikazan dijagram nasljeđivanja za osnovnu klasu koja opi-




Slika 2.1: Dijagram nasljeđivanja apstraktne klase *NumericalMethod* [25]

suje što sve numeričke metode za specifične probleme poput rješavanja linearnog sustava moraju implementirati. Također na slici vidimo da nasljeđivanje ide u dubinu do čak 3 razine čemu je razlog to što se na prvoj razini grana na numeričke metode koje rješavaju različite probleme, a tek kasnije uvode implementacije za rješavanje tih problema.

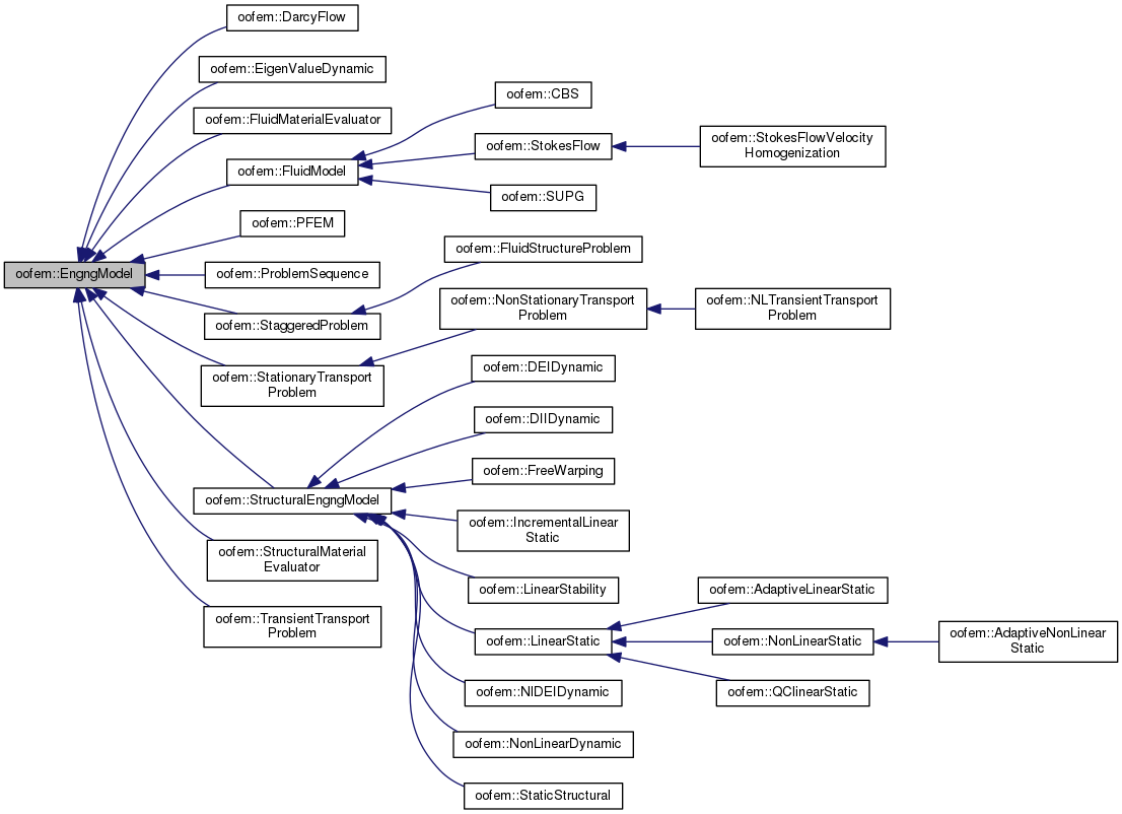
Nadalje, dat ćemo opis svih najvažnijih klasa u OOFEM-u, njihove uloge, međusobnu komunikaciju i stabla nasljeđivanja. Informacije su dobivene iz izvornog koda na [26] i automatsko kreiranoj dokumentaciji na [25] i [24]. Okosnica program su iduće klase:

*EngngModel*, *Domain*, *NumericalMethod* i *FEMComponent*. Iz tih klasa izniču gotovo sve ostale funkcionalnosti te su upravo one zaslužne za apstrakciju i objektno orijentirani pristup.

Klasa koja obuhvaća, te kasnije i inicijalizira sve ostale je ***EngngModel***. *EngngModel* je apstrakcija za problem koji se razmatra. Ona je prva klasa koji se inicijalizira i predstavlja vrstu analize koju treba izvršiti, te implementira komunikaciju sa svim svojim atributima. Izvedene klase poput FluidModel i LinearStatic "znaju" sastaviti jednadžbe i fizičko značenje pojedinih komponenti. One su odgovorne za formiranje upravljačke jednadžbe za svaki korak rješavanja, obično zbrajanjem doprinosa pojedinih elemenata i čvorova. Glavne funkcije klase *EngngModel* su:

- Parsiranje ulaza (ulazne datoteke) na zasebne domene i spremanje rezultata.
- Spremanje i vraćanje stanja u/iz kontekstnih datoteka.
- Sastavljanje jednadžbi zbrajanjem doprinosa iz problemskih domena (obično od čvorova i elemenata).
- Rješavanje problema opisanih jednadžbama pomoću prikladnih numeričkih metoda. To zahtijeva povezivanje numeričkih metoda karakterističnih elemenata s komponentama jednadžbi. *EngngModel* mora mapirati svaku komponentu jednadžbe (koja ima fizičko značenje). odgovarajućoj numeričkoj komponenti numeričke metode.
- Prekinuti vremenski korak ažuriranjem vrijednosti čvorova i elementa (uključujući ažuriranje integracijskih točaka).

Jedna od najvažnijih (virtualnih) funkcija koje *EngngModel* definira je *solveYourself* koja je zaslužna za pozivanje analize modela. Ona poziva funkciju *solveYourselfAt* koja prihvaća vremenski korak kao argument. Tako ju funkcija *solveYourself* poziva za svaki vremenski korak specificiran iz ulaznih podataka. Funkcija *solveYourselfAt* implementirana je za svaki problem zasebno te je upravo ta funkcija ona koja detaljno opisuje kako se pojedini fizikalni problem rješava. Tako primjerice klasa *StaticStructural* implementira tu metodu između 259. i 395. linije u datoteci staticstructural.c. Metoda dohvaća sve potrebne elemente poput matrice krutosti koja je za ovaj problem rijetka matrica, unutarnje i vanjske sile, opterećenja i ostale bitne sastavnice potrebne za rješavanje problema. Po dohvaćanju svih potrebnih elemenata, metoda poziva metodu *solve* nad atributom koji nasljeđuje apstraktnu klasu *NumericalMethod*. U slučaju klase *StaticStructural* ako u ulaznim podacima nije specificirano drugačije klasa koja rješava numerički zadatak (sustav jednadžbi) je *NRSolver* koja implementira Newton-Raphson metodu rješavanja. Koje sve probleme OOFEM "zna" riješiti možemo vidjeti tako da pogledamo tko sve implementira klasu EngngModel. Grafički prikaza toga dan je na slici 2.2.




Slika 2.2: Dijagram nasljeđivanja apstraktne klase EngngModel [25]

Iduća važna klasa koja je sadržana kao atribut unutar *EngngModel* klase je ***Domain***. Mreža čvorova i elemenata je reprezentirana pomoću nje, te su unutar *Domain* klase spremljene sve komponente modela metode konačnih elemenata poput čvorova, stupnjeva slobode (Degree Of Freedom, DOF), materijala, rubnih uvjeta itd. Svaki *EngngModel* ima barem jednu domenu koja omogućuje apstrakciju te se brine za daljnju komunikaciju s njezinim elementima. Najvažnije funkcije domene su:

- Konstruiranje objekata pročitanih s ulaza. To uključuje čitanje i parsiranje mreže s ulaza te konstrukciju odgovarajućih komponenata pravoga tipa.
- Pružanje generaliziranog pristupa pojedinim komponentama domene.
- Filtriranje izlaza za određene korake, elemente i čvorove tijekom rješavanja

Kao što smo već vidjeli na slici 2.1 apstraktna klasa koja omogućava lagano dodavanje novih matematičkih metoda rješavanja pojedinih problema je ***NumericalMethod***. Kao što

je rečeno u opisu *EngngModela*, *EngngModel* može koristiti različite numeričke metode, ovisno, na primjer, o veličini problema ili prethodnoj konvergenciji. Svaka pojedina instanca klase *EngngModel* odgovorna je za mapiranje njezinih jednadžbi na odgovarajuće numeričke komponente. Takvo mapiranje omogućuje implementiranje numeričkih metoda neovisno od određenog fizičkog problema. Samim time, numerička metoda može predstavljati i sučelje za algoritam napisan u C-u ili Fortranu. Klase izvedene iz numeričke metode trebale bi objaviti sučelje za specifičnu vrstu problema (poput rješenja linearnog sustava). Treba naglasiti da sve numeričke metode koje rješavaju isti problem koriste isto sučelje (isto mapiranje) - to se provodi korištenjem iste osnovne klase osnovnih problema. Primjer toga je problem rješavanja rijetkog linearnog sustava deklariranog klasom *SparseLinearSystemNM*. Stoga je moguće riješiti problem s bilo kojom prikladnom numeričkom metodom za klasifikaciju i ostaviti cijeli model, uključujući i mapiranje, nepromijenjen jer sve instance klase pružaju zajedničko sučelje. Tako primjerice za problem rješavanja rijetkog linearnog sustava (*SparseLinearSystemNM*) postoji 8 različitih implementacija numeričkih metoda što je vidljivo sa slike 2.1 (u novoj verziji za koju još ne postoji dokumentacija dodana je još jedna, deveta metoda koja koristi metodu direktne faktorizacije rijetke matrice). Glavni zadatak tih metoda je rješavanje sustava jednadžbi $Ax = b$.

Kako izgleda implementacija tih numeričkih metoda prikazat ćemo na primjeru klase *IMLSolver* koja za rješavanja sustava jednadžbi $Ax = b$ koristi iterativne metode IML++ biblioteke te može raditi sa svim implementacijama rijetkih matrica (npr. *SpoolesSolver* zahtjeva točno određeni format rijetke matrice). Tako klasa *IMLSolver* implementira dvije glavne metode. Prva je čitanje specifičnih ulaznih podataka poput prihvatljive greške, maksimalnog broj iteracija i tipa iterativne metode za rješavanje koje mogu biti ili GMRES (generalizirana metoda minimalnih reziduala) ili CG (metoda konjugiranih gradijenta), a druga je metoda *solve* koja provjeri preduvjete, pokrene štopericu i ovisno o odabranoj metodi pozove GMRES ili CG algoritam. Algoritam GMRES služi za rješavanje nesimetričnog linearnog sustava, dok CG rješava simetričan pozitivno definitan slučaj. U nastavku dajemo kratak opis metode, pseudokod i opis konvergencije za CG algoritam. Opis prati algoritam opisan u [3].

Metoda konjugiranih gradijenta je efektivna metoda za rješavanje simetričnih pozitivno definitnih sustava. Metoda generira nizove vektora aproksimacija rješenja, ostataka te smjer traženja koji se koristi kod računanje iduće aproksimacije rješenja. U svakoj iteraciji izvedu se dva unutarnja množenja za računanje skalara za iduću aproksimaciju koji zadovoljavaju određene uvjete ortogonalnosti. Na simetrično pozitivno definiranom linearnom sustavu ti uvjeti podrazumijevaju da je udaljenost do pravog rješenja minimalizirana u nekoj normi. i-ta aproksimacija $x^{(i)}$ se mijenja za umnožak skalara ($\alpha_i$) i vektora smjera $p^(i)$:

$$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} \tag{2.1}$$

Odnosno ostaci se računaju s $r^{(i)} = b - Ax^{(i)}$ to jest:

$$r^{(i)} = r^{(i-1)} - \alpha q^{(i)} \qquad \text{gdje je} \qquad q^{(i)} = Ap^{(i)} \tag{2.2}$$

$\alpha$ se izabire tako da minimizira ${r^{(i)}}^T A^{-1} r^{(i)}$ za svaki odabrani $\alpha$ u jednadžbi 2.2. Smjer traženja izabire se koristeći ostatke

$$p^{(i)} = r^{(i)} + \beta_{i-1} p^{(i-1)} \tag{2.3}$$

gdje se odabir $\beta_i = ({r^{(i)}}^T r^{(i)})/({r^{(i-1)}}^T r^{(i-1)})$ vrši osiguravajući da $p^{(i)}$ i $Ap^{(i-1)}$, odnosno ekvivalentno tome $r^{(i)}$ i $r^{(i-1)}$ budu ortogonalni. Štoviše, može se pokazati da takav odabir $\beta_i$ čini $p^{(i)}$ ortogonalnima na sve prijašnje $Ap^{(j)}$ i analogno $r^{(i)}$ ortogonalnim na sve $r^{(j)}$. U nastavku slijedi pseudokod za algoritam.

**Algorithm 1** Pretkondicionirani algoritam za metodu konjugiranih gradijenata

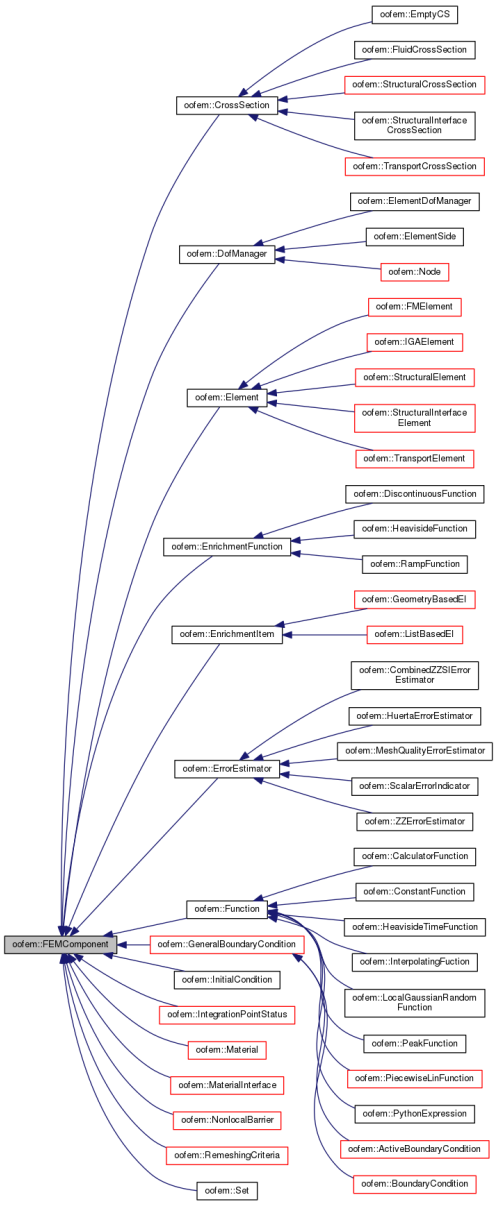
Izračunaj $r^{(0)} = b - Ax^{(0)}$ za neki inicijalni $x^{(0)}$
**for** $i = 1, 2, \ldots$ **do**
    $z^{(i-1)} = M^{-1} r^{(i-1)}$
    $\rho_{(i-1)} = {r^{(i-1)}}^T z^{(i-1)}$
    **if** $i = 1$ **then**
        $p^{(1)} = z^{(0)}$
    **else**
        $\beta_{(i-1)} = \rho_{(i-1)}/\rho_{(i-2)}$
    **end if**
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{(i-1)}/{p^{(i)}}^T q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    provjera konvergencije i nastavak u slučaju potrebe
**end for**

Pogreška metode može se ograničiti u smislu broja spektralnog stanja $\kappa_2$ matrice $M^{-1}A$. Ako su $\lambda_{max}$ i $\lambda_{min}$ najveća i najmanja svojstvena vrijednost simetrične pozitivno definitne matrice $B$, onda je broj spektralnog stanja matrice $B$ jednak $\kappa_2(B) = \lambda_{max}(B)/\lambda_{min}(B)$. Ako je $\tilde{x}$ egzaktno rješenje sustava $Ax = b$ gdje je A simetrična i pozitivno definitna matrica tada za metodu CG sa simetričnom i pozitivno definitno pretkondicioniranom matricom $M$ vrijedi:

$$\|x^{(i)} - \tilde{x}\|_A \leqslant 2\alpha^i \|x^{(0)} - \tilde{x}\|_A \tag{2.4}$$

gdje je $\alpha = (\sqrt{\kappa_2} - 1)/(\sqrt{\kappa_2} + 1)$ i $\|y\|_A^2 \equiv (y, Ay)$. Iz toga slijedi da je broj iteracija za dostizanje relativnog smanjenja greške proporcionalan s $\sqrt{\kappa_2}$. Ovdje smo opisali jednu implementaciju *NumericalMethod* klase kojoj je glavna uloga rješavanje (ne)linearnog sustava (ne)jednadžbi $Ax = b$.

Klasa s najvećim brojem nasljeđivanja i najdubljim stablom je ***FEMComponent*** klasa. Na slici 2.3 se može vidjeti to stablo koja radi svoje veličine nije moglo stati stoga nekim čvorovima nisu sva djeca prikazana.



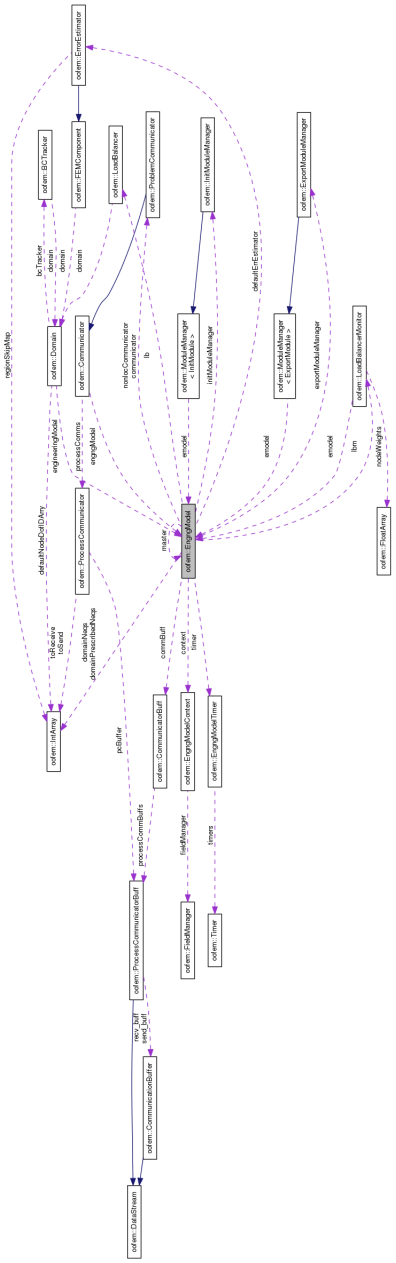

Slika 2.3: Dijagram nasljeđivanja apstraktne klase *FEMComponent*. Čvorovi s crvenim okvirom su klase kojima nisu prikazana djeca. [25]

*FEMComponent* klasu implementiraju svi objekti koji su u vezi s konačnom metodom elemenata. Tako se među klasama koje implementiraju mogu pronaći rubni uvjeti, inicijalni uvjeti, funkcije sa silama, materijali konačnih elemenata te i sami konačni elementi. *FEMComponent* definira atribute i metode zajedničke svim komponentama mreže: elementima, čvorovima, vremenskim koracima, materijalima, opterećenjima i funkcijama učitavanja. Ova klasa definira dva atributa zajednička svim komponentama konačnih elemenata. "Broj" koji se prvenstveno koristi za čitanje podataka u podatkovnoj datoteci i "domena" koja se koristi za komunikaciju s drugim komponentama (npr. za element koji će dobiti svoj materijal), za pristup koordinatnom sustavu i podatkovnoj datoteci. U idućim potpoglavljima detaljnije ćemo opisati klase koje opisuju materijale od kojih se konačni elementi mogu sastojati i klase konačnih elemenata.

Zanimljivo je još i vidjeti kako sve te klase međusobno komuniciraju, to jest kako su one povezane. Dobar uvid u to nam daje slika 2.4 gdje su prikazane poveznice između klasa. Na slici se vidi kako na apstraktnoj razini klase međusobno komuniciraju, to jest kako čitav softver zapravo i radi. Što se raspodijele koda u direktorije tiče, u repozitoriju [26] se nalaze mnogi direktoriji. Koncentrirat ćemo se u većoj ili manjoj mjeri na iduće: **bindings**, **doc**, **src** i **tools**. U direktoriju doc se nalazi se dokumentacija u TeX obliku koja se automatski nadopunjuje iz komentara unutar koda. Generirana dokumentacija se nalazi i na [29] te je bitan izvor svih informacija. Više o direktoriju bindings bit će rečeno u potpoglavlju 2.6 jer se tu nalazi sve potrebno za korištenje OOFEM softvera pomoću Pythona. Direktorij tools sadrži različite alate za modifikaciju ulaznih i izlaznih podataka od kojih ćemo koristiti *unv2oofem.py*. Posljednji i najvažniji direktoriji je src koji sadrži sam kod softvera. Unutar tog direktorija nama najvažniji bit će sm koji se odnosi na strukturnu analizu. Unutar njega nalaze se svi potrebni i nama zanimljivi elementi poput materijala, konačnih elemenata i modela rješavanja.

## 2.3 Materijali u OOFEM-u

Osnovna klasa za sve modele materijala je klasa *Material*, izvedena iz *FEMComponent* klase. Apstraktna klasa *Material* implementira sučelje neovisno o analizi koja se provodi. Dio sučelja potreban za analizu trebaju dodati izvedene klase, koje predstavljaju osnovne klase za određeni problem. Tipičan primjer je klasa **StructuralMaterial** za strukturni materijal koji izlaže sve usluge potrebne za strukturnu analizu i na koju ćemo se kao što je rečeno u prethodnom dijelu najviše koncentrirati. Izravno izvedene klase poput *SturucturalMaterial* moraju implementirati specifične dijelove konačnih elemenata i poprečnog presjeka za problematiku domene što su na primjeru klase *StructuralMaterial* klase *StructuralElement* i *StructuralCrossSection* koje su apstraktne osnovne klase za sve "strukturne" konačne elemente. Kako bi se pohranile sve potrebne varijable povijesti modela materijala, usvaja se takozvani koncept stanja. Stanje materijala može se smatrati spremnikom




Slika 2.4: Dijagram s poveznicama između apstraktnih klasa OOFEM-a. [25]

svih potrebnih povijesnih varijabli. Obično se pohranjuju dvije vrste varijabli, privremene i trajne. Privremene se odnose na stvarno stanje integracijske točke, ali ne moraju odgovarati

globalnoj ravnoteži. Te se privremene varijable mijenjaju tijekom iteracija pretraživanja ravnoteže. Trajne varijable odnose se na prethodno konvergirano stanje. Za svaki model materijala mora biti definiran odgovarajuće stanje i mora biti stvorena i pridružena jedinstvena instanca za svaku integracijsku točku. Za prije navedeni materijal *SturcturalMaterial* to je klasa *StructuralMaterialStatus* koja je također apstraktna te predstavlja osnovnu za implementaciju stanja pojedinih materijala. U idućem dijelu dajemo detaljan opis jednog materijala za strukturnu analizu te što je sve potrebno za njegovu implementaciju.

## Primjer modela materijala

U ovom odjeljku opisat će se implementacija materijala izotropnog oštećenja. Za pokrivanje raznih modela temeljenih na izotropnom konceptu oštećenja, najprije se definira osnovna klasa *IsotropicDamageMaterial* koja implementira sve zajedničke značajke. Izvedene klase onda samo implementiraju određene zakone o evoluciji oštećenja. Modeli izotropnih oštećenja temelje se na pojednostavljenoj pretpostavci da je degradacija krutosti izotropna, to jest moduli krutosti koji odgovaraju različitim smjerovima smanjuju se proporcionalno i neovisno o smjeru djelovanja sila. Posljedično, matrica krutosti oštećenog materijala izražava se kao:

$$D = (1 - \omega)D_\epsilon$$

gdje je $D_\epsilon$ elastična matrica krutosti neoštećenog materijala i $\omega$ je parametar oštećenja. U početku, $\omega$ je postavljen na nulu, što predstavlja "novi" neoštećeni materijal i reagira linearno elastično. Budući da materijal prolazi kroz deformaciju, širenje mikro nedostataka smanjuje krutost, što je predstavljeno rastom parametra $\omega$ oštećenja. Za $\omega = 1$, krutost u potpunosti nestaje. Slično teoriji plastičnosti, uvodi se funkcija sile $f$. U teoriji oštećenja prirodno je raditi u prostoru naprezanja i stoga funkcija sile ovisi o naprezanju i o dodatnom parametru $\kappa$, opisujući evoluciju štete. Fizikalno, $\kappa$ je skalar najveće razine napetosti koju je materijal dosegao. Funkcija sile obično ima oblik

$$f(\epsilon, \kappa) = \widetilde{\epsilon}(\epsilon) - \kappa$$

gdje je $\widetilde{\epsilon}$ jednak naprezanju. Ostaje povezati varijablu $\kappa$ s parametrom $\omega$ oštećenjem. Budući da oba $\kappa$ i $\omega$ rastu monotono, prikladno je pretpostaviti eksplicitnu jednadžbu

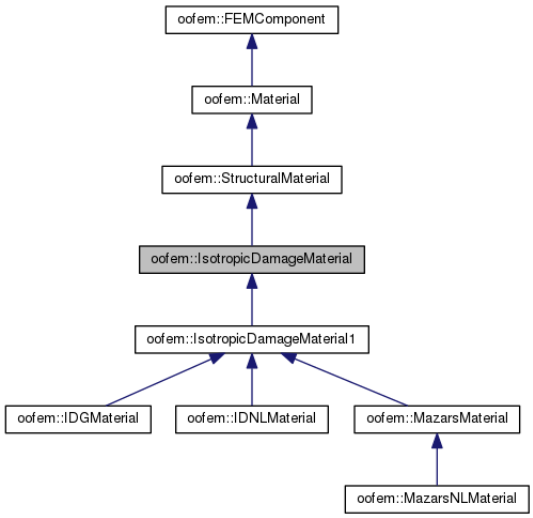
$$\omega = g(\kappa).$$

Važna prednost ove eksplicitne formulacije je da se sila koja odgovara danom naprezanju može izravno procijeniti, bez potrebe za rješavanjem nelinearnog sustava jednadžbi. Ovaj opći okvir za računanje naprezanja i matrice krutosti je uobičajen za sve materijalne modele ove vrste. Dakle, prirodno je uvesti osnovnu klasu za sve modele izotropnih oštećenja koji osiguravaju opću implementacija algoritama za evaluaciju matrice sila i krutosti. Pojedini

modeli zatim pružaju samo njihov ekvivalent naprezanju i oštećenju. Tako je na primjer za definiciju naprezanja po Mazarsu (1984) dovoljno uvrstiti njegovu formulu naprezanja zasnovanu na normi pozitivnih vrijednosti naprezanja:

$$\widetilde{\epsilon} = \sqrt{\sum_{I=1}^{3} \langle \epsilon_I \rangle^2}$$

gdje su $\langle \epsilon_I \rangle$ pozitivne vrijednosti tenzora naprezanja $\epsilon$. Također, kao što je prije i navedeno za svaki materijal potrebno je implementirati i klasu stanja pa se tako implementira i *IsotropicDamageMaterialStatus*. Za izotropsko bazirane modele oštećenja, jedina varijabla koju je potrebno pamtiti jest najveće dostignuto naprezanje ($\kappa$). Osim toga implementira se i metoda za pristupanje i promjenu naprezanja.




Slika 2.5: Dijagram apstraktne klase nasljeđivanja materijala izotropskog oštećenja [25]

## 2.4 Konačni elementi u OOFEM-u

Glavna osnovna apstraktna klasa za sve konačne elemente je **Element**. Svrha ove klase jest implementacija osnovnih značajki koje su zajedničke svim konačnim elementima (kao što su pohranjivanje referenci na materijal elementa, čvorovi, opterećenja, sastavljanje polja lokacije, pohranjivanje iz/u kontekstnu datoteku i slično). Klasa *Element* ne implementira niti jednu metodu koja se odnosi na određenu analizu već ih samo deklarira te dopušta

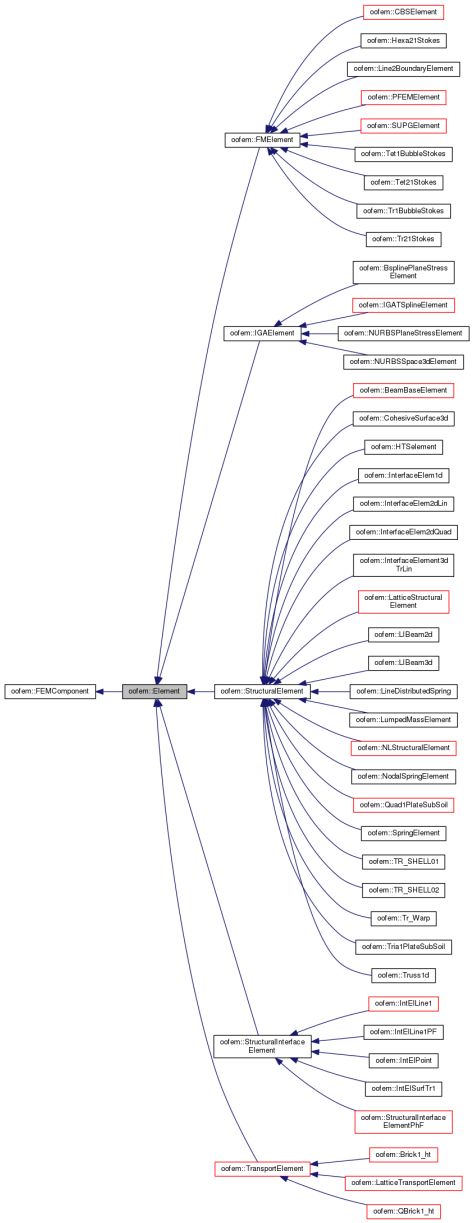
izvedenim klasama da ih implementiraju. Direktna djeca obično definiraju opće usluge potrebne za određenu svrhu analize poput matrica krutosti i mase za strukturnu analizu ili matrice procjene kapaciteta i provodljivosti za analizu prijenosa topline. Na slici 2.6 vidimo kako izgleda stablo nasljeđivanja klase *Element*. Nadalje, kao i ranije koncentrirat ćemo se na strukturnu analizu, to jest konačne elemente koji nasljeđuju klasu ***StructuralElement*** koja je osnovna klasa za sve konačne elemente strukturne analize. Konkretnije, nakon opisa klase *StructuralElement* opisat ćemo i klasu *Truss1d* koja opisuje štapni element u dvodimenzionalnom prostoru te je jedan primjer kako implementacija konačnog elementa izgleda.

Klasa *StrukturalElement* deklarira sve potrebne metode i atribute koje zahtijeva strukturna analiza poput metode računanja matrica krutosti, opterećenja, naprezanja i vektora sila. Osnovni zadaci strukturnog elementa su izračunavanje doprinosa globalnim jednadžbama ravnoteže (matrice mase i krutosti, različitih vektora opterećenja zbog graničnih uvjeta, sile opterećenja, toplinskog opterećenja itd.) i izračunavanje odgovarajućih naprezanja i sila iz pomaka čvorova. Radi toga se deklariraju odgovarajuće virtualne metode za računanje tih doprinosa. Ti standardni doprinosi mogu se izračunati numeričkom integracijom odgovarajućih struktura, koji tipično ovise o interpolaciji elemenata ili tipu materijala, u odnosu na volumena elemenata. Stoga je moguće osigurati opće definicije tih metoda, pod uvjetom da se primjenjuju odgovarajuće metode za računanje interpolacije materijala i da su inicijalizirana odgovarajuća integracijska pravila i uvjeti. Kao primjer toga dajemo računanje matrice krutosti. Budući da matrica krutosti elemenata doprinosi globalnoj ravnoteži, krutost će biti zatražena korištenjem metode *giveCharacteristicMatrix*. Implementacija metode strukturnog elementa uzima u obzir samo materijalnu nelinearnost (geometrijska nelinearnost se ne uzima u obzir te se za elemente u kojima geometrijska nelinearnost igra ulogu očekuje da se ova metoda nadjača). Matrica krutosti elementa može se izračunati koristeći se formulom

$$K = \int_V B^T DB dV$$

gdje je $B$ takozvana geometrijska matrica koja sadrži derivacije osnovnih funkcija, a $D$ je matrica krutosti materijala. Numerička integracija koristi se za procjenu tog integrala. Za numeričku integraciju koristi se klasa *IntegrationRule*. Formule za numeričku integraciju za određeni element kreiraju se tijekom inicijalizacije elementa i pohranjuju se u polju *integrationRulesArray* kao atribut klase, naslijeđen iz klase *Element*.

Nadalje slijede neki implementacijski detalji klase *Truss1d*. Nju uzimamo za primjer što je sve potrebno implementirati za jedan konačan element ako želimo dodati novi koji još nije implementiran. Također, vjerojatnost potrebe za dodavanjem novih konačnih elemenata nije prevelika jer OOFEM već sadrži oko 70-tak različitih konačnih elemenata. *Truss1d* (štapni element u 1D) predstavlja linearni izoparametarski rešetkasti element u




Slika 2.6: Dijagram nasljeđivanja osnovne klase Element [25]

1D određen s dva čvora. Pretpostavlja se da se elementi nalaze duž x-osi. Element zahtjeva da mu je područje poprečnog presjeka određeno. Sama implementacija izvedena je u 400-tinjak linija koda, a mi dajemo kratki opis datoteke zaglavlja i najvažnije metode koje

zaglavlje deklarira kao i neke zanimljivije implementacije jednostavnijih metoda. *Truss1d* (štapni element) služi pri strukturnoj analizi stoga nasljeđuje klasu *StructuralElement*. Od važnijih metoda implementira virtualnu metodu *computenumberOfDofs* koja vraća broj 2, a označava broj stupnjeva slobode. Zatim slijede implementacija za računanje interpolacijske i geometrijske matrice. Obje metode računaju rješenja na danim integracijskim točkama koje dobivaju kao parametre. Dajemo kod za računanje interpolacijske matrice pomaka:

Kod 2.1: Računanje interpolacijske matrice za štapni element u 1D

```
1 void
2 Truss1d :: computeNmatrixAt(const FloatArray &iLocCoord, FloatMatrix &answer)
3 // Returns the displacement interpolation matrix {N} of the receiver,
4 // evaluated at gp.
5 {
6     FloatArray n;
7     this->interp.evalN( n, iLocCoord, FEIElementGeometryWrapper(this) );
8     // Reshape
9     answer.resize(1, 2);
10    answer.at(1, 1) = n.at(1);
11    answer.at(1, 2) = n.at(2);
12 }
```

Samo zaglavlje elementa izgleda ovako:

Kod 2.2: Datoteka zaglavlja za štapni element u 1D

```
1 #ifndef truss1d_h
2 #define truss1d_h
3
4 ...
5
6 namespace oofem {
7 class FEI1dLin;
8
9 /**
10  * This class implements a two-node truss bar element for one-dimensional
11  * analysis.
12  */
13 class Truss1d : public StructuralElement{
14
15 ...
16
17 public:
18     Truss1d(int n, Domain * d);
19     virtual ~Truss1d() { }
20
21     virtual FEInterpolation *giveInterpolation() const;
22
23     virtual void computeLumpedMassMatrix(FloatMatrix &answer, TimeStep *tStep);
24     virtual void computeMassMatrix(FloatMatrix &answer, TimeStep *tStep)
25     { computeLumpedMassMatrix(answer, tStep); }
26
27     virtual int computeNumberOfDofs() { return 2; }
28     virtual void giveDofManDofIDMask(int inode, IntArray &) const;
29
30     // characteristic length (for crack band approach)
```

```
31      virtual double giveCharacteristicLength(const FloatArray &normalToCrackPlane)
32      { return this->computeLength(); }
33
34      virtual double computeVolumeAround(GaussPoint *gp);
35
36      virtual void computeStressVector(FloatArray &answer, const FloatArray &strain,
            GaussPoint *gp, TimeStep *tStep);
37      virtual void computeConstitutiveMatrixAt(FloatMatrix &answer, MatResponseMode rMode,
            GaussPoint *gp, TimeStep *tStep);
38
39      virtual int testElementExtension(ElementExtension ext) { return 0; }
40
41      virtual Interface *giveInterface(InterfaceType it);
42
43      virtual MaterialMode giveMaterialMode() { return _1dMat; }
44
45      ...
46
47  protected:
48      virtual void computeBmatrixAt(GaussPoint *gp, FloatMatrix &answer, int = 1, int =
            ALL_STRAINS);
49      virtual void computeBHmatrixAt(GaussPoint *gp, FloatMatrix &answer);
50      virtual void computeNmatrixAt(const FloatArray &iLocCoord, FloatMatrix &answer);
51      virtual void computeGaussPoints();
52
53  };
54  } // end namespace oofem
55  #endif // truss1d_h
```

Iz same datoteke zaglavlja vidimo da je poveći broj metoda već implementiran i to samo jednom linijom, te preostale poput već navedene *computeBmatrixAt* svode se na pozivanje već gotovih metoda nad-klasa. Važno je još napomenuti metodu *computeGaussPoints* koja inicijalizira integracijsko pravilo za štapni element i čija implementacije izgleda ovako:

Kod 2.3: Inicijalizacija integracijskog pravila za štapni element

```
1 void Truss1d :: computeGaussPoints()
2 // Sets up the array of Gauss Points of the receiver.
3 {
4     if ( integrationRulesArray.size() == 0 ) {
5         integrationRulesArray.resize(1);
6         integrationRulesArray[0] = std::make_unique<GaussIntegrationRule>(1, this, 1, 2);
7         this->giveCrossSection()->setupIntegrationPoints(* integrationRulesArray[0], 1,
              this);
8     }
9 }
```

Kada se element stvori (pomoću klase *Domain*), zove se zadani konstruktor. Implementacija elementa najprije treba nazvati implementaciju roditelja kako bi se osiguralo da se atributi deklarirani na razini roditelja pravilno iniciraju. Zatim element mora inicirati atribute koji su sami deklarirani, kao i postaviti pravila integracije. U našem primjeru, posebna metoda *computeGaussPoints* poziva se za inicijalizaciju pravila integracije. U tom se slučaju stvara samo jedna pravilo integracije. To je tipa *GaussIntegrationRule*, što ukazuje da se koristi Gaussova integracija. Jednom kada se stvori pravilo integracije, njegove

su integracijske točke stvorene da predstavljaju linijski integral s jednom integracijskom točkom. Integracijske točke bit će povezane s elementom koji se razmatra i imat će 1D način rada (koji određuje vrstu odgovora na model materijala).

Na ovom jednostavnom primjeru prikazali smo kako implementirati konačni element. Implementirani konačni element ima svojstva opisana u tablici 2.1:

| **Ključna riječ** | **truss1d** |
|---|---|
| Opis | 1D štapni element |
| Specifični parametri | - |
| Nepoznanice | Jedan stupanj slobode u svakom čvoru |
| Aproksimacije | Linearna aproksimacija pomaka i geometrije |
| Integracija | Egzaktna |
| Značajke | Potpuna podrška za dinamičku analizu, podrška za prilagodbe |
| Svojstva poprečnog presjeka | Površina je obavezna |
| Opterećenja | Opterećenja na element su podržana. Granična opterećenja nisu podržana u trenutačnoj implementaciji |
| Stanje | Pouzdan |

Tablica 2.1: Sažetak svojstva štapnog 1D elementa

## 2.5 Ulazni podaci

Predviđen način rada OOFEM-a je takav da se prvo pomoću njegovih alata za predobradu poput *unv2oofem.py* ili ručnim unosom konstruira datoteka koju će program uzeti kao ulaz, u kojoj se nalaze svi potrebni podaci za analizu u prigodnom obliku. U ovom potpoglavlju opisat ćemo kako ta datoteka treba izgledati. Prilikom opisa koncentrirat ćemo se na izgleda datoteke za strukturnu analizu. Potpuna dokumentacija za sve tipove analiza kao i za paralelnu obradu može se pronaći na [29].

Kao što je već navedeno, idući opis opisuje ulazne podatke za strukturnu analizu. Prilikom opisa tekst unutar navodnika je primjer teksta koji se zapravo unosi u datoteku s ulaznim podacima. Prilikom navođenja niza brojeva (bez obzira je li niz realnih ili cijelih brojeva) prvo se navodi duljina niza i zatim vrijednosti. Primjer s koordinatama čvora izgleda ovako: *coords 3 0. 2.7 10.* gdje je 3 duljina niza a 0, 2.7 i 10 su vrijednosti. Poredak unutar datoteke nije važan ali radi jasnijeg opisa kao i čitanja preporučuje se idući redoslijed:

1. Ime datoteke izlaza. Npr. ”izlazni_podaci.out”

2. Opis analize. Proizvoljan tekst koji je će ispisati u izlaznoj datoteci npr. ”Analiza tri štapna elementa”.

3. Tip analize s atributima. Nužni parametri su nmsteps koji predstavlja niz koraka rješenja s istim zajedničkim atributima te atributi. Ako želimo izvesti rješenje potrebno je nadodati broj modula za izvoz s ključnom riječi nmodules i broj, npr. ”StaticStructural nsteps 1 nmodules 2”.

4. Modul izvoza podataka (dodatni parametar). Ako želimo izvesti rješenje za naknadnu analizu u nekom drugom programu izabire se željeni tip izvoza podataka te se kao što je navedeno u prethodnoj stavci (tip analize) poveća broj nmodules za jedan, npr. ”vtkxml tstep_all, domain_all”.

5. Opis domene zadatka. Najopsežniji dio datoteke, ovisno o analizi može biti i više domena, ali u našem slučaju bit će jedna. U slučaju više domena ulaz se zadaje analogno. U opisu domene nalazi se sve što je u dijelu poglavlja 2.2 kod opisa klase *Domain* opisano.

   a) Tip domene. Neke od postojećih domena su: *2dPlaneStress* ili *2d-Truss* za analize s dva stupnja slobode po čvoru (pomaci *u* i *v*), *3d* za analize s tri stupnja slobode (pomaci po *u*, *v* i *w*, *3dShell* za analize sa šest stupnjeva slobode (pomaci i rotacije po svakoj osi). Za primjer možemo uzeti ”domain 2dPlaneStress”.

   b) Način ispisa. Filter koji određuje koje će se sve informacije ispisivati, to jest za koje čvorove, elemente i korake želimo informacije. Ako želimo dobiti informacije o svim stavkama zadajemo: ”OutputManager tstep_all dofman_all element_all”.

   c) Veličina domene. Broje pojedinih dijelova unutar domene, to jest broj čvorova, elemenata, poprečnih presjeka, materijala, rubnih uvjeta zajedno s opterećenjima, početnih uvjeta, funkcija vremena i skupova podataka. U primjeru dani su brojevi za dijelove domene u istom redoslijedu: ”ndofman 6 nelem 5 ncrosssect 1 nmat 1 nbc 3 nic 0 nltf 1 nset 3”.

   d) Skup stupnjeva slobode. Ovaj tip parametra sakuplja više stupnjeva slobode u jedan element. Najčešći tip je čvor. Osim čvora postoje i viseći čvorovi, stijenke elemenata, PFEM čestice i drugi. Nadalje koristimo čvorove. Čvor prihvaća koordinate u globalnom koordinatnom sustavu (ako drugačije nije navedeno) te neobavezne varijable za početne i rubne uvijete kao i opterećenje. ”node 1 coords 3 0. 0. 10.”.

   e) Konačni element. Jedan od elemenata opisanih u potpoglavlju 2.4. Konačni element zahtjeva čvorove koji ga opisuju npr. ”Truss2d 1 nodes 2 1 4”.

f) Skupovi. Mogu biti skupovi čvorova, konačnih elemenata itd. Koristi se za spajanje regija elemenata s rubnim uvjetima, poprečnim presjecima i inicijalnim uvjetima kao na primjer kad želimo uključiti utjecaj gravitacije na sve elemente. Dajemo dva primjera jer se skupovi mogu koristiti na različite načine: "Set 1 elementranges {(1 3)}", "Set 2 nodes 3 1 2 3".

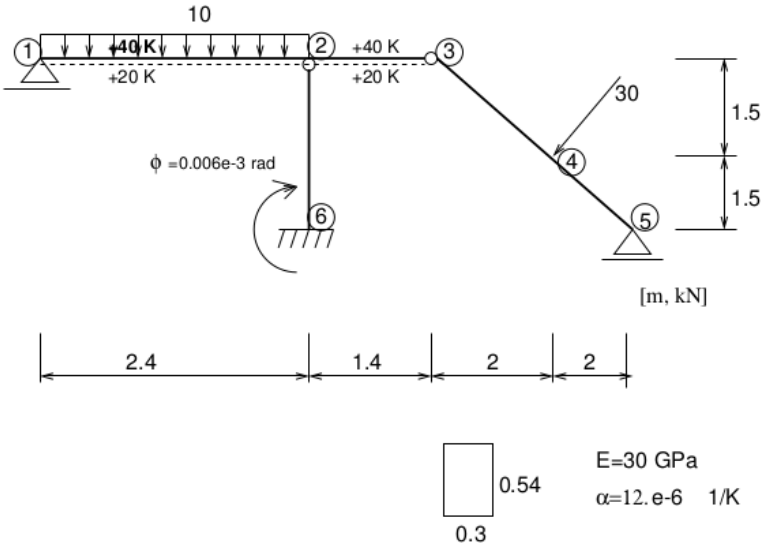
g) Poprečni presjek. Za varijable poprečnog presjeka mogu se koristiti *SimpleCS*, *VariableCS*, *LayeredCS*, *FiberedCS* i *WarpingCS*. Najjednostavniji je *SimpleCS* koji opisuje poprečni presjek s konstantnim značajkama i zadaje se sa širinom i duljinom, a u 3D analizi odgovarajući volumen i ostale dimenzije automatski se dobivaju. Osim toga mogu mu se pridodati materijal i konačne elemente na koje se taj presjek odnosi. Primjer, "SimpleCS 1 thick 0.1 width 1.0 material 1 set 1".

h) Materijal. Neki od materijala opisanih u 2.3 dijelu. Nakon tipa materijala dolazi njegova gustoća i specifične značajke ovisno o materijalu. Tako za prijašnji primjer za izotropski linearni elastični materijal definicija izgleda "IsoLE 1 d 1. E 1.0 n 0.2 tAlpha 0.000012" gdje je E Yungova konstanta, n Poissonov omjer i tAlpha koeficijent toplinskog širenja.

i) Rubni uvjeti i opterećenje. Dio sa svim tipovima opterećenja i uvjeta. Što se rubnih uvjeta tiče, postoji ih nekolicina, a mi ćemo izdvojiti: *BoundaryCondition* (Dirichletov rubni uvjet), *NodalLoad* (koncentrirano opterećenje u pojedinom čvoru) i LinearConstraintBC (rubni uvjet koji implementira ograničenje u obliku $\sum_i w_i r_i = c$ gdje su $r_i$ nepoznanice vezane uz stupnjeve slobode određene čvorovima dok se težine $w_i$ unosi). Primjer jednog rubnog uvjeta koji fiksira čvorove iz skupa 2 je "BoundaryCondition 1 loadTimeFunction 1 dofs 2 1 3 values 2 0.0 0.0 set 2". Osim rubnih uvjeta u ovu skupinu parametara pripadaju i opterećenja poput *DeadWeight*, *StructEigenstrainLoad*, *ConstantEdgeLoad* i *LinearEdgeLoad*. Za primjer *ConstantEdgeLoad* označava opterećenje po rubu konačnog elementa i prihvaća iduće tipove uvjeta s njihovim kodovima, Neumannove (2), Newtonove (3) i Stefan-Boltzmannove (7) rubne uvjete. Kao i kod rubnih uvjeta, opterećenja također imaju parametar *loadTimeFunction* koja prihvaća identitet vremenske funkcije s kojom se uvjeti skaliraju. Kao primjer opterećenja dajemo iduće opterećenje: "LinearEdgeLoad 3 loadTimeFunction 1 dofs 2 1 3 Components 4 1.0 0.0 1.0 0.0 loadType 3".

j) Vremenske funkcije. Kao što je već navedeno, vremenske funkcije nam omogućavaju prijelom uvjeta kroz vrijeme kod analize u više koraka. Neke od funkcija vremena su *ConstantFunction* u kojoj se zadaje konstanta i *PeakFunction* u kojoj je zadana vrijednost za točno jednu vremensku točku te je za sve ostale jednaka 0. Primjer konstantne funkcije je "ConstantFunction 1 f(t) 1.0".

Važno je još napomenuti da unutar datoteke ulaza svaka linija predstavlja jedan ulazni podatak, te linije koje počinju sa znakom # smatraju se komentarima te se zanemaruju. Radi boljeg razumijevanja dajemo sadržaj jedne takve datoteke ulaza. Da bi primjer bio jasniji prilažemo i strukturu po kojoj je ulaz modeliran.




Slika 2.7: Model strukture koju je potrebno analizirati. [29]

Kod 2.4: Primjer datoteke ulaza za OOFEM za problem sa slike 2.7

```
analiza_greda_u_2d.out
# uzima se u obzir samo utjecaj momenta na pomake
# beamShearCoeff je namjerno neproporcijonalno povecan
StaticStructural nsteps 3 nmodules 1
errorcheck
domain 2dBeam
OutputManager tstep_all dofman_all element_all
ndofman 6 nelem 5 ncrosssect 1 nmat 1 nbc 6 nic 0 nltf 3 nset 7
node 1 coords 3 0.  0.  0.
node 2 coords 3 2.4 0.  0.
node 3 coords 3 3.8 0.  0.
node 4 coords 3 5.8 0.  1.5
node 5 coords 3 7.8 0.  3.0
node 6 coords 3 2.4 0.  3.0
Beam2d 1 nodes 2 1 2
Beam2d 2 nodes 2 2 3 DofsToCondense 1 6
Beam2d 3 nodes 2 3 4 DofsToCondense 1 3
Beam2d 4 nodes 2 4 5
Beam2d 5 nodes 2 6 2 DofsToCondense 1 6
SimpleCS 1 area 1.e8 Iy 0.0039366 beamShearCoeff 1.e18 thick 0.54 material 1 set 1
IsoLE 1 d 1. E 30.e6 n 0.2 tAlpha 1.2e-5
```

```
22 BoundaryCondition 1 loadTimeFunction 1 dofs 1 3 values 1 0.0 set 4
23 BoundaryCondition 2 loadTimeFunction 1 dofs 1 5 values 1 0.0 set 5
24 BoundaryCondition 3 loadTimeFunction 2 dofs 3 1 3 5 values 3 0.0 0.0 -0.006e-3 set 6
25 ConstantEdgeLoad 4 loadTimeFunction 1 Components 3 0.0 10.0 0.0 loadType 3 set 3
26 NodalLoad 5 loadTimeFunction 1 dofs 3 1 3 5 Components 3 -18.0 24.0 0.0 set 2
27 StructTemperatureLoad 6 loadTimeFunction 3 Components 2 30.0 -20.0 set 7
28 PeakFunction 1 t 1.0 f(t) 1.
29 PeakFunction 2 t 2.0 f(t) 1.
30 PeakFunction 3 t 3.0 f(t) 1.
31 Set 1 elementranges {(1 5)}
32 Set 2 nodes 1 4
33 Set 3 elementedges 2 1 1
34 Set 4 nodes 2 1 5
35 Set 5 nodes 1 3
36 Set 6 nodes 1 6
37 Set 7 elements 2 1 2
```

## 2.6 Korištenje OOFEM alata iz Pythona

Nakon što smo opisali kako je OOFEM predviđen za korištenje, dajemo i nama najvažnije opis, kako koristiti OOFEM iz programskog jezika Python. U prvom poglavlju ukratko smo opisali što je metoda konačnih elemenata i kako ju koristiti, a u drugom pokazali smo kako koristiti OOFEM. Sada ćemo pokazati 2 načina na koje se može koristiti OOFEM iz Pythona. Prvi način koji nam nije pretjerano zanimljiv bazira se u potpunosti na načinu korištenja kao što se koristi i putem komandne linije učitavanja svih parametra iz datoteke ulaza. Rješavanje problema tada izgleda kao u kodu 2.5.

Kod 2.5: Pokretanje OOFEM alata pomoću Pythona i učitavanje ulaznih podataka iz datoteke

```
1 import liboofem
2
3 podaci = liboofem.OOFEMTXTDataReader("analiza_stapa_u_2d.in")
4 problem = liboofem.InstanciateProblem(podaci, liboofem.problemMode._processor, 0)
5 problem.checkProblemConsistency()
6 problem.setRenumberFlag()
7 problem.solveYourself()
8 problem.terminateAnalysis()
```

Prije nego što opišemo drugi način korištenja gdje se svi ulazni podaci konstruiraju pomoću Python objekata, ukratko ćemo opisati što je sve potrebno učiniti da bi mogli koristiti biblioteku *liboofem*, kako je to omogućeno i što nam ona nudi.

Da bi se C++ kod mogao koristiti putem Pythona potrebno je zadovoljiti neke preduvjete. Za to ostvariti postoji nekoliko načina, a autori OOFEM-a odlučili su se za korištenje Boost.Python biblioteke koja je dio boost C++ kolekcije. Boost je velik projekt koji sadrži više od 150 biblioteka i više od 23 milijuna linija koda za C++. Boost.Python je C++ biblioteka koja omogućuje besprijekornu interoperabilnost između C++-a i programskog jezika Python. Boost.Python omogućuje brz i lagan izvoz C++ koda u Python tako da

se novonastalo sučelje gotovo ni ne mijenja u odnosu na C++ sučelje. Korištenjem Boost.Python biblioteke jedino potrebno za korištenje OOFEM-a jest definirati sve klase, metode i funkcije koje će se koristiti direktnim pozivanjem iz Pythona. To je učinjeno u datoteci *oofemlib.cpp* koju se može pronaći na adresi [27]. Datoteka s definicijama sadrži gotovo 2000 linija te definira Python biblioteku liboofem koju smo vidjeli u isječku koda 2.5. Važno je još reći da se prilikom kompiliranja OOFEM-a mora postaviti varijabu "USE_PYTHON_BINDINGS" na "ON" tako da se kompilira i datoteka *oofemlib.cpp*. Također, potrebno je i onda dobivenu binarnu datoteku *liboofem.so* postaviti negdje gdje Python "očekuje" module tako se liboofem može uključiti "importati" u Python. Više informacija o Boost.Pythonu i generalno korištenju C++ koda u Pythonu može se pronaći na [4], [9] i [32].

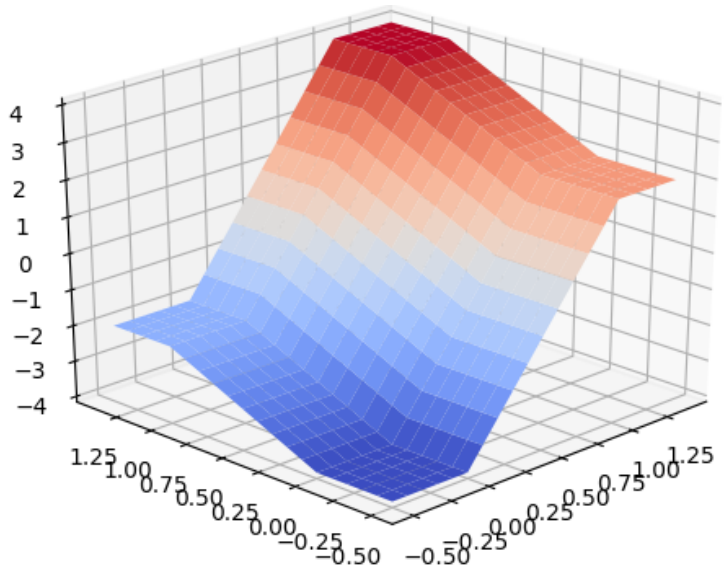
Nakon što smo opisali što je sve potrebno da se kod napisan u C++-u uopće može koristiti u Pythonu, opisat ćemo neke od mogućnosti koje nam *liboofem* Python biblioteka nude, te dajemo primjer kako je i koristiti. Kao što smo i opisali Boost.Python je omotač oko C++ koda stoga je glavna motivacija prijašnjeg (2.5) potpoglavlja bila upoznavanje mogućnosti OOFEM programa. Sada ćemo pokazati kako te iste funkcionalnosti koristiti iz Pythona, ali i neke dodatne mogućnosti koje nam samo Python okruženje nudi, poput interaktivnog ispitivanja pojedinih elemenata te grafički prikaz željenih vrijednosti.

Primjer kombinacije Pythonovih mogućnosti (točnije korištenje matplotlib biblioteke) s mogućnostima OOFEM-a moguće je konstruirati mrežasti model i odmah ga i vizualizirati. Pokretanjem koda 2.6, dobiva se slika 2.8.

Kod 2.6: Konstrukcija mrežastog modela i vizualizacija

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

import liboofem

# inicijalizacija uniformne mreze
mreza = liboofem.UniformGridField()

# postavljanje mreze na 2D geometriju
mreza.setGeometry(lo=(0,0),hi=(1,1),div=(2,2))

# mreza se proteze na 2x2 znaci da imamo 3x3 cvora, potrebno nam je 9 vrijednosti
mreza.setValues([-4, -3, -2, -1, 0, 1, 2, 3, 4])

# vizualizaciju zelimo na malo sirem podrucju
X,Y=np.meshgrid(np.arange(-.5,1.5,.1),np.arange(-.5,1.5,.1))

# funkcija koja vraca vektor vrijednosti u koordinatama
@np.vectorize
def vrijednost_na_koordinatama(x,y):
    return mreza.evaluateAtPos((x,y))[0]

Z = vrijednost_na_koordinatama(X,Y)
```

```
26
27 # vizualizacija pomocu matplotliba
28 fig = plt.figure()
29 ax = fig.gca(projection='3d')
30 surf = ax.plot_surface(X, Y, Z, linewidth=1, rstride=1, cstride=1,cmap=cm.coolwarm, shade=
       True)
31 plt.show()
```




Slika 2.8: Vizualizacija uniformne pravokutne mreže konstruirane pomoću *liboofem* biblioteke putem Pythona

Sama mreža čvorova, kao i svi ostali elementi analize mogu se i individualno konstruirati što ćemo pokazati na primjeru koda 2.7. Kod usko prati sve navedene korake iz prijašnjega potpoglavlja 2.5 što će biti jasno vidljivo, i modelira primjer sa slike 2.7 te je analogon ulaznoj datoteci 2.4 za analizu pomoću OOFEM-a putem komandne linije. Nakon same analize moguće je, kao i u prošlom primjeru konstruirati razne vizualizacije što prepuštamo čitateljima radi toga da sam kod bude što čišći i relevantniji samoj zadaći koja je opisivanje korištenja *liboofem-a* radi analize konačnih elemenata.

Kod 2.7: Korištenje OOFEM putem Pythonovog sučelja

```
1 import liboofem
2
3 # Definicija analize koja ce se provesti
4 problem = liboofem.linearStatic(nSteps=3, outFile="izlazni_podaci2.out")
5
6 # Domena i izlazni podaci koji ce se sacuvati
```

```
domena = liboofem.domain(1, 1, problem, liboofem.domainType._2dBeamMode, tstep_all=True,
                         dofman_all=True, element_all=True)
problem.setDomain(1, domena, True)

# Za svaki objekt koji nasljeduje klasu FEMComponent ukoliko priliko inicijalizacije
# nema definiranu domenu pridruzuje se zadnjoj inicijaliziranoj

# Slicno kao i gore, ako nije odreden broj komponente dodjeljuje mu se iduci po redu
# npr. za peakFunction dodjeljuje joj se domain.numberOfLoadTimeFunctions + 1
# Analogno i za sve ostale klase nasljednice FEMComponent klase

vremenska_funk_1 = liboofem.peakFunction(1, domena, t=1, f_t=1)
vremenska_funk_2 = liboofem.peakFunction(2, domena, t=2, f_t=1)
vremenska_funk_3 = liboofem.peakFunction(3, domena, t=3, f_t=1)
vremenske_funkcije = (vremenska_funk_1, vremenska_funk_2, vremenska_funk_3)

# loadTimeFunction parametar moze se zadati putem broja ili direktno (vidi primjere)
# Analogno vrijedi i za sve ostale objekte koji implementiraju metodu giveNumber()

# Ukoliko se od nekog parametra ocekuje niz, zadaje se pomocu liste i n-torke

# Rubni uvjeti
rubni_uvjet_1 = liboofem.boundaryCondition(1, domena, loadTimeFunction=1,
                                           prescribedValue=0.0)
rubni_uvjet_2 = liboofem.boundaryCondition(2, domena, loadTimeFunction=2,
                                           prescribedValue=-.006e-3)

opterecenje_po_rubu = liboofem.constantEdgeLoad(3, domena, loadTimeFunction=1,
                                                components=(0., 10., 0.), loadType=3,
                                                ndofs=3)

opterecenje_u_cvoru = liboofem.nodalLoad(4, domena, loadTimeFunction=1,
                                         components=(-18., 24., 0.))

opterecenje_temper = liboofem.structTemperatureLoad(5, domena,
                                                    loadTimeFunction=vremenska_funk_3,
                                                    components=(30., -20.))
uvjeti_i_opterecenja = (
    rubni_uvjet_1, rubni_uvjet_2, opterecenje_po_rubu, opterecenje_u_cvoru,
    opterecenje_temper)

# Cvorovi
cvor_1 = liboofem.node(1, domena, coords=(0., 0., 0.), bc=(0, 1, 0))
cvor_2 = liboofem.node(2, domena, coords=(2.4, 0., 0.), bc=(0, 0, 0))
cvor_3 = liboofem.node(3, domena, coords=(3.8, 0., 0.), bc=(0, 0, rubni_uvjet_1))
cvor_4 = liboofem.node(4, domena, coords=(5.8, 0., 1.5), bc=(0, 0, 0), load=(4,))
cvor_5 = liboofem.node(5, domena, coords=(7.8, 0., 3.0), bc=(0, 1, 0))
cvor_6 = liboofem.node(6, domena, coords=(2.4, 0., 3.0),
                       bc=(rubni_uvjet_1, 1, rubni_uvjet_2))
cvorovi = (cvor_1, cvor_2, cvor_3, cvor_4, cvor_5, cvor_6)

# Materijali i poprecni presjeci
materijal = liboofem.isoLE(1, domena, d=1., E=30.e6, n=0.2, tAlpha=1.2e-5)
poprecni_presjek = liboofem.simpleCS(1, domena, area=0.162, Iy=0.0039366,
                                     beamShearCoeff=1.e18, thick=0.54)

# Konacni elementi
greda_1 = liboofem.beam2d(1, domena, nodes=(1, cvor_2), mat=1, crossSect=1,
                          boundaryLoads=(3, 1), bodyLoads=(5,))
greda_2 = liboofem.beam2d(2, domena, nodes=(2, 3), mat=materijal, crossSect=1,
```

```
67                              DofsToCondense=(6,), bodyLoads=[opterecenje_temper])
68 greda_3 = liboofem.beam2d(3, domena, nodes=(cvor_3, 4), mat=1,
69                              crossSect=poprecni_presjek, dofstocondense=[3])
70 greda_4 = liboofem.beam2d(4, domena, nodes=(cvor_4, cvor_5), mat=materijal,
71                              crossSect=poprecni_presjek)
72 greda_5 = liboofem.beam2d(5, domena, nodes=(cvor_6, 2), mat=1, crossSect=1,
73                              DofsToCondense=(6,))
74 konacni_elementi = (greda_1, greda_2, greda_3, greda_4, greda_5)
75
76 # Potrebno je sad popuniti domenu s novoinicijaliziranim komponentama
77
78 # Radi dobivanja na brzini koristi se resize tako da ne dolazi do dinamicne
79 # realokacije prilikom dodavanja svake nove komponenete
80
81 domena.resizeDofManagers(len(cvorovi))
82 for cvor in cvorovi:
83     domena.setDofManager(cvor.number, cvor)
84
85 domena.resizeElements(len(konacni_elementi))
86 for konacni_element in konacni_elementi:
87     domena.setElement(konacni_element.number, konacni_element)
88
89 domena.resizeMaterials(1)
90 domena.setMaterial(1, materijal)
91
92 domena.resizeCrossSectionModels(1)
93 domena.setCrossSection(1, poprecni_presjek)
94
95 domena.resizeBoundaryConditions(len(uvjeti_i_opterecenja))
96 for uvjet_opterecenje in uvjeti_i_opterecenja:
97     domena.setBoundaryCondition(uvjet_opterecenje.number, uvjet_opterecenje)
98
99 domena.resizeFunctions(len(vremenske_funkcije))
100 for vremenska_funkcija in vremenske_funkcije:
101     domena.setFunction(vremenska_funkcija.number, vremenska_funkcija)
102
103 problem.checkProblemConsistency()
104 problem.init()
105 problem.postInitialize()
106 problem.setRenumberFlag()
107 problem.solveYourself()
108 problem.terminateAnalysis()
```

Važno je još napomenuti da omotač oko C++ koda programa OOFEM ne obuhvaća sve moguće značajke u jednakom obimu. Tako smo primjerice iz gornjeg koda vidjeli da se može koristiti konačni element greda u dvodimenzionalnom prostoru (*liboofem.beam2d*), ali radi lakšeg prototipiranja i bolje apstrakcije koda ostali konačni elementi nisu tako definirani. Definiciju *beam2d* konačnog elementa može se pronaći u datoteci *oofemlib.cpp* na adresi [27] između 1544-te i 1585-e linije koji izgleda ovako:

Kod 2.8: Implementacija omotača oko generalnog konačnog elementa i specifičnog konačnog elementa *beam2d*

```
1 /*****************************************************
2 * Element
3 *****************************************************/
```

```
4 // element(aClass,domain=defaultDomain,**kw)
5 object element(bp::tuple args, bp::dict kw)
6 {
7     // extracts first python argument (string element type)
8     string aClass = extract<string>(args[0])();
9     // extracts values from args if they are specified
10    int number =     len(args)>1? extract<int>(args[1])() : 0;
11    Domain *domain = len(args)>2? extract<Domain*>(args[2])() : nullptr;
12    /* ???????????????
13    // if no material is specified, set it to 1
14    if (!kw.has_key("mat")) { kw["mat"] = 1; }
15    // if no cross section is specified, set it to 1
16    if (!kw.has_key("crosssect")) { kw["crosssect"] = 1; }
17    // if no domain is specified and one already exists in the script, use that one
18    if (domain == nullptr && temp_global.has_key("defaultDomain")) { domain = extract<
          Domain*>(temp_global["defaultDomain"])(); }
19    if (domain==nullptr) { LOG_ERROR(oofem_errLogger,"wrong Domain"); }
20    ????????????????????? */
21    // create Element (convert aClass to char* - expected by classFactory.createElement)
22    auto elem = classFactory.createElement(aClass.c_str(),number,domain);
23    // if elem==nullptr, something was wrong
24    if (!elem) { OOFEM_LOG_ERROR("element: wrong input data"); }
25    // sets globalNumber == number befor initializeFrom
26    elem->setGlobalNumber(number);
27    // construct OOFEMTXTInputRecord from bp::dict **kw
28    OOFEMTXTInputRecord ir = makeOOFEMTXTInputRecordFrom(kw);
29    // pass input record to elem
30    elem->initializeFrom(&ir);
31    // convert element to PyObject (expected by raw_function, which enables fun(*args,**kw
          ) syntax in python)
32    return object(ptr(elem.release()));
33 }
34
35 // auxiliary constructor for specific element type (name is passed as first argument)
36 object createElementOfType(const char* type, bp::tuple args, bp::dict kw)
37 {
38    args = len(args)>1? bp::make_tuple(type,args[0],args[1]) : len(args)>0? bp::make_tuple
          (type,args[0]) : bp::make_tuple(type);
39    return element(args, kw);
40 }
41 // specific elements
42 object beam2d(bp::tuple args, bp::dict kw) { return createElementOfType("beam2d",args,kw);
       }
43
44 ...
45
46 BOOST_PYTHON_MODULE(liboofem)
47 {
48
49 ...
50
51    def("element", raw_function(element,1));
52    def("beam2d", raw_function(beam2d,0));
53
54 ...
55
56 }
```

Iz koda je vidljivo da se korisnicima ostavlja mogućnost korištenja svih konačnih ele-

menata OOFEM-a, ali kroz malo drugačije sučelje. Tako se primjerice inicijalizacija ostalih konačnih elemenata odvija na idući način:

Kod 2.9: Primjer konstrukcije konačnih elemenata koji nemaju implementiran direktan poziv

```
greda_direktno = liboofem.beam2d(1, domain, nodes=(1,2), ...)
greda_indirektno = liboofem.element('beam2d', 1, domain, nodes=(1,2), ...)
greda_indirektno2 = liboofem.InputRecord(1, "beam2d", {"nodes": (1,2), ...})
stap_indirektno = liboofem.element('truss2d', 1, domain, nodes=(1,2), ...)
```

Osim za konačne elemente, identičan postupak se odvija i za ostale komponente koje nasljeđuju klasu *FEMComponent* opisanu u potpoglavlju 2.2 i prikazanu na slici 2.3 poput materijala (*Material*), poprečnih presjeka (*CrossSection*), vremenskih funkcija itd.

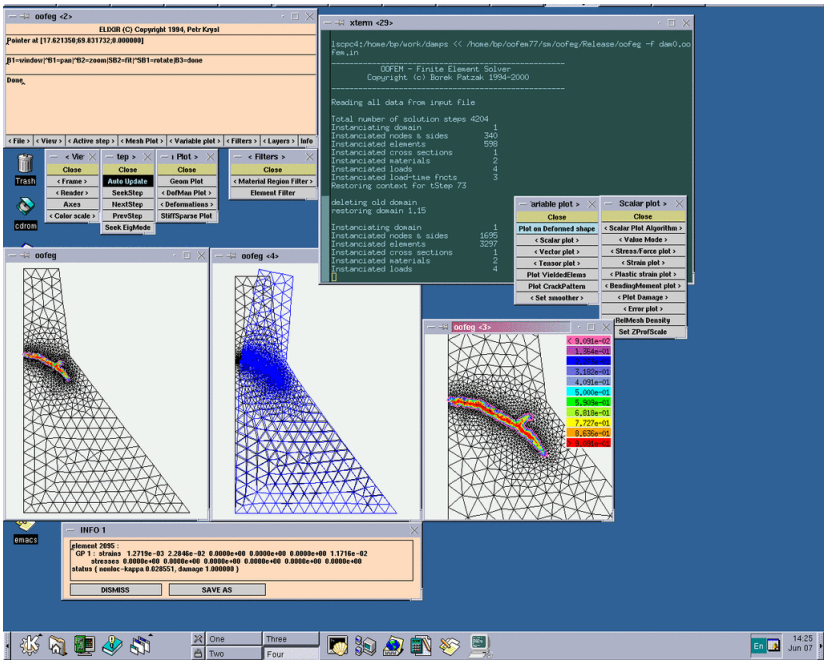
U ovom dijelu prikazali smo više načina korištenja OOFEM-a putem Pythona te nekoliko ideja za nadogradnju što se još može postići. Pokazali smo kako je i gdje implementiran omotač oko C++ koda te kako i gdje u slučaju potrebe i želje treba definirati dodatne metode koje trenutno nisu podržane za interakciju kroz Python. U posljednjem poglavlju pokazat ćemo još jednom kako koristiti OOFEM i *liboofem* biblioteku za Python za analizu metode konačnih elemenata.

## 2.7 Vizualizacija rezultata

OOFEM pruža nekoliko načina kako dobivene rezultate prikazati. Opisat ćemo 3 različita načina i dati nekoliko primjera kako te vizualizacije izgledaju.

1. OOFEG. OOFEM izvorni kod dolazi s OOFEG, X-windows baziranim vizualizacijskim alatom. Kako je OOFEG baziran na X-windows bibliotekama upotreba mu je ograničena na Unix platforme. OOFEG je razvijen pomoću Elixir i Ckit biblioteka fiksnih verzija koje više nisu aktualne stoga grafičko sučelje izgleda kao ”zamrznuto” u vremenu kasnih 90-tih i daljnji razvoj više nije lako ostvariv. OOFEM je isto limitiran u mogućnostima 3D vizualizacije. Radi svega toga OOFEG je pogodan za upotrebu kod relativno jednostavnih primjera ali ubrzo postaje nedovoljno moćan. OOFEG ima jedan dobar slučaj upotrebe, a to je za vizualizaciju mreže te se nakon što su instalirane sve njegove potrebne komponente koristi s *oofeg -f ulazna_datoteka.in*.

2. ParaView. OOFEM omogućava na jednostavan način pretvorbu izlazni podataka u VTK format što omogućava vizualizaciju putem softvera treće strane poput ParaViewa i MayaVia. Dobra strana toga je to što su takvi alati specijalizirani za vizualizaciju, omogućuju znatno više značajki te se u odnosu na OOFEG i dalje razvijaju i napreduju. Kao što smo naveli u potpoglavlju 2.5 pod rednim brojem četiri, za izvoz podataka u VTK format dovoljno je dodati jednu liniju. Jednako tako dodavanje

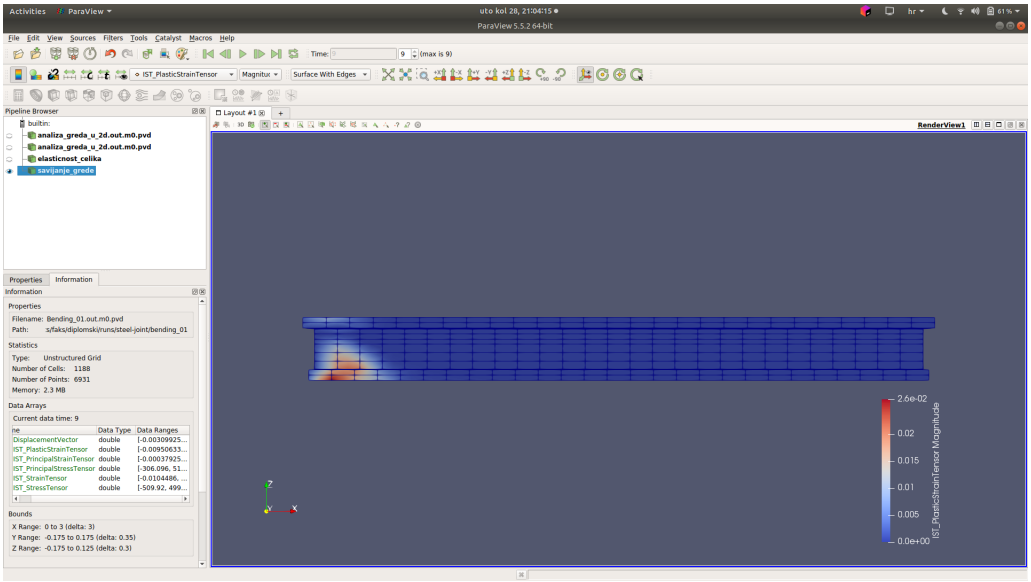


Slika 2.9: Vizualizacija pomoću OOFEG grafičkog procesora [29]

samo jedne linije u Python kod rezultati će se ispisati u datoteku i u VTK formatu. Nakon završetka analize, konstruirat će se datoteke s informacijama za svaki korak koje se uvezu u našem slučaju u ParaView. Nadalje, za uvesti podatke u ParaView potrebno je samo označiti datoteku koja ima ekstenziju *.pvd* i koja ima zapisane putanje do datoteka s podacima o rezultatima analize. Na slici 2.10 dajemo primjer strukture grede I profila vizualizirane pomoću ParaView softvera. Mreža grede je konstruirana u Salome programu i preuzeta sa stranice [29]. Datoteka je potom pomoću OOFEM konvertera ulazne datoteke pretvorena u OOFEM-u čitljiv format. Mreža se sastoji od 6931 čvora i 1188 kvadratnih elemenata. Analiza te ulazne strukture bez nikakvih optimizacija trajala je oko 90 minuta na računalu s i5 procesorom da bi se po završetku konstruirala datoteke potrebne ParaViewu za vizualizaciju.

3. Python. Kao što smo pokazali na gornjem primjeru, ekosustav Pythona dolazi s velikom količinom alata koji nam mogu pomoći u vizualizaciji bilo kakvih podataka. Tako smo na slici 2.8 prikazali mrežu konačnih elemenata konstruiranih pomoću OOFEM-a. Jedna od bitnijih biblioteka za vizualizaciju je *matplotlib* o kojoj se više može pronaći na web stranici [22].

Iz svega navedenoga preporuka za vizualizaciju jest koristiti već postojeće alate pomoću




Slika 2.10: Vizualizacija grede na koju se vrši pritisak pomoću ParaView softvera

jednostavne funkcije izvoza rješenja, te u slučaju potrebe za nekom dodatnom vizualizacijom koja obuhvaća nešto što se iz uobičajenih rezultata ne dobije poput matrice krutosti pojedinih elemenata koristiti Pythonov ekosustav koji pruža dobru podršku za proizvoljne posebne slučajeve.

# Poglavlje 3

# Testni primjer

U ovom poglavlju pokazat ćemo kako riješiti zadatak sa slike 1.7 koji smo ručno riješili u poglavlju 1.3. Zadatak ćemo riješiti na 3 načina pomoću OOFEM-a te usporediti rezultate u odnosu na ručno rješavanje. Prvo ćemo sastaviti datoteku ulaza s potpunim problemom te pokrenuti OOFEM s njome. Za drugi način koristit ćemo Python i pomoću njega učitati ulaznu datoteku koju smo koristili i u prvom načinu. Za treći način rješavanja u potpunosti ćemo definirati ulazne podatke u Pythonu i dobiti rješenje. Za vizualizaciju rješenja koristit ćemo ParaView.

## 3.1 Rješavanje učitavanje datoteke u OOFEM putem terminala

Kao što smo već pokazali na prijašnjim primjerima rješavanjem putem učitavanje iz datoteke temelji se na izradi valjane datoteke. Osim ručne izrade u praksi se češće izrađuje model pomoću programa poput Salomea koji se izveze u izlaznu datoteku univerzalnog formata (.UNV) za koji OOFEM ima prikladan program za pretvorbu u njemu razumljiv oblik. Kako je naš problem bio dovoljno jednostavan upisujemo ga ručno i dobivamo iduću datoteku ulaza.

Kod 3.1: Datoteka ulaza za zadatak sa slike 1.7

```
1 rezultat.out
2 Analiza sila i pomaka u 3 stapna elementa
3
4 StaticStructural nsteps 1 nmodules 1
5
6 vtkxml tstep_all domain_all primvars 1 1
7
8 domain 2dtruss
9 OutputManager tstep_all dofman_all element_all
10
```

```
11 ndofman 3 nelem 3 ncrosssect 1 nmat 1 nbc 4 nic 0 nltf 1 nset 4
12
13 node 1 coords 3 0.0  0.0  0.0
14 node 2 coords 3 2.5  0.0  0.0
15 node 3 coords 3 0.0  0.0  1.44
16
17 truss2d 1 nodes 2 1 2
18 truss2d 2 nodes 2 1 3
19 truss2d 3 nodes 2 2 3
20
21 SimpleCS 1 area 1 material 1 set 1
22 IsoLE 1 d 1. E 200.0e4 n 0.2 tAlpha 1.2e-5
23
24 BoundaryCondition 1 loadTimeFunction 1 dofs 2 1 3 values 2 0.0 0.0 set 2
25 BoundaryCondition 2 loadTimeFunction 1 dofs 1 3 values 1 0.0 set 3
26
27 NodalLoad 3 loadTimeFunction 1 dofs 1 1 Components 1 15000.0 set 3
28 NodalLoad 4 loadTimeFunction 1 dofs 1 3 Components 1 5000.0 set 4
29
30 ConstantFunction 1 f(t) 1.
31
32 Set 1 elements 3 1 2 3
33 Set 2 nodes 1 1
34 Set 3 nodes 1 2
35 Set 4 nodes 1 3
```

## 3.2 Korištenje Pythona za učitavanje datoteke ulaza

Slično kao kod rješavanja pomoću pokretanja programa iz terminala izgleda korištenje Pythona za učitavanje datoteke ulaza kao što je i bilo prikazano u 2.5. Python skripta za rješavanje na ovaj način izgleda ovako:

Kod 3.2: Python program za rješavanje zadatak sa slike 1.7 čitanjem datoteke ulaza

```
1 import liboofem
2
3 podaci = liboofem.OOFEMTXTDataReader("zadatak_terminal.in")
4
5 problem = liboofem.InstanciateProblem(podaci, liboofem.problemMode._processor, 0)
6
7 problem.checkProblemConsistency()
8 problem.setRenumberFlag()
9 problem.solveYourself()
10 problem.terminateAnalysis()
```

## 3.3 Korištenje Pythona za definiranje i rješavanje zadatka

Analogno s primjerom 2.7 postavljanje i rješavanje zadatka sa slike 1.7 putem Pythona konstruira se iduća skripta:

Kod 3.3: Python program za rješavanje zadatak sa slike 1.7 inicijalizacijiom svih elemenata putem liboofem biblioteke

```
import liboofem

problem = liboofem.linearStatic(nSteps=1, outFile="rezultat_python_full.out")

domena = liboofem.domain(1, 1, problem, liboofem.domainType._2dTrussMode, tstep_all=True,
                         dofman_all=True, element_all=True)
problem.setDomain(1, domena, True)

vremenska_funkcija = liboofem.loadTimeFunction('ConstantFunction', 1, domena, f_t=1)
domena.resizeFunctions(1)
domena.setFunction(vremenska_funkcija.number, vremenska_funkcija)

rubni_uvjet_1 = liboofem.boundaryCondition(1, domena, loadTimeFunction=1,
                                           prescribedValue=0.0)
opterecenje_u_cvoru_1 = liboofem.nodalLoad(2, domena, loadTimeFunction=1,
                                           components=(15000., 0.))
opterecenje_u_cvoru_2 = liboofem.nodalLoad(3, domena, loadTimeFunction=1,
                                           components=(0., 5000.))
uvjeti_i_opterecenja = (rubni_uvjet_1, opterecenje_u_cvoru_1, opterecenje_u_cvoru_2)
domena.resizeBoundaryConditions(len(uvjeti_i_opterecenja))
for uvjet_opterecenje in uvjeti_i_opterecenja:
    domena.setBoundaryCondition(uvjet_opterecenje.number, uvjet_opterecenje)

cvor_1 = liboofem.node(1, domena, coords=(0., 0., 0.), bc=(1, 1))
cvor_2 = liboofem.node(2, domena, coords=(2.5, 0., 0.), bc=(0, 1), load=(2,))
cvor_3 = liboofem.node(3, domena, coords=(0., 0., 1.44), bc=(0, 0), load=(3,))
cvorovi = (cvor_1, cvor_2, cvor_3)
domena.resizeDofManagers(len(cvorovi))
for cvor in cvorovi:
    domena.setDofManager(cvor.number, cvor)

materijal = liboofem.isoLE(1, domena, d=1., E=200.e4, n=0.2, tAlpha=1.2e-5)
domena.resizeMaterials(1)
domena.setMaterial(1, materijal)

poprecni_presjek = liboofem.simpleCS(1, domena, area=1)
domena.resizeCrossSectionModels(1)
domena.setCrossSection(1, poprecni_presjek)

greda_1 = liboofem.element('truss2d', 1, domena, nodes=(1, 2), mat=1, crossSect=1)
greda_2 = liboofem.element('truss2d', 2, domena, nodes=(1, 3), mat=1, crossSect=1)
greda_3 = liboofem.element('truss2d', 3, domena, nodes=(2, 3), mat=1, crossSect=1)
konacni_elementi = (greda_1, greda_2, greda_3)
domena.resizeElements(len(konacni_elementi))
for konacni_element in konacni_elementi:
    domena.setElement(konacni_element.number, konacni_element)

problem.checkProblemConsistency()
problem.init()
problem.postInitialize()
problem.setRenumberFlag()
problem.solveYourself()
problem.terminateAnalysis()
```

## 3.4 Korištenje PyGmsh-a za izradu mreže i liboofem-a za rješavanje zadatka

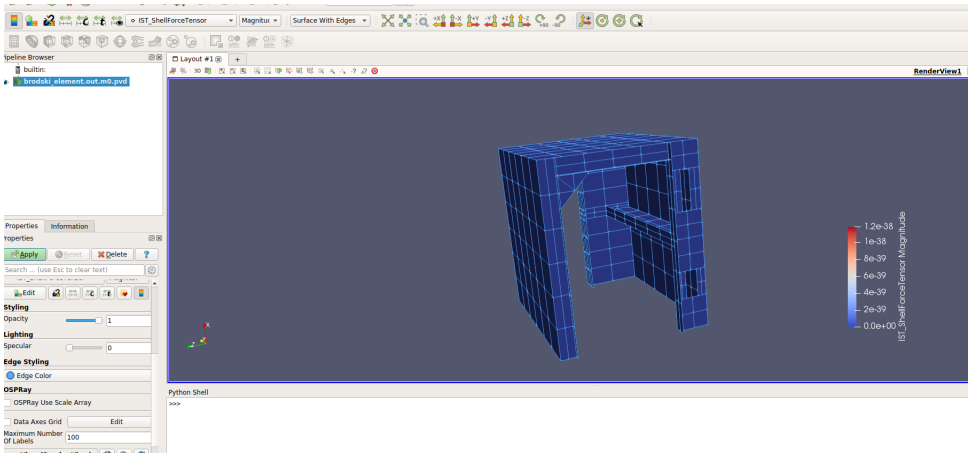
Ako imamo datoteku s modelom na kojem želimo raditi izračune u formatu koji Gmsh prepoznaje i s kojim zna raditi možemo iskoristiti PyGmsh [16] koji je Python omotač oko Gmsh-a za izradu mrežastog modela iz danog početnog modela. Tako možemo ne mičući se od Pythona, od početnog modela izraditi mrežu po željenim specifikacijama, na nju dodati rubne uvjete, opterećenja, materijale i ostale elemente potrebne za metodu konačnih elemenata te pomoću liboofem-a napraviti izračun koji ćemo kasnije vizualizirati u nekom od za to specifičnih alata poput ParaViewa. U nastavku dajemo kod koji radi upravo navedeno. Kod koristi materijale iz [34], to jest ulazni podaci te proces izrade mreže su preuzeti u potpunosti, ali je provjere kvalitete mreže radi čitkosti koda zanemarena. Konstruirana mreža sastoji se od elemenata 4 elementa s 3 čvora te 332 elementa s četiri čvora gdje svaki čvor ima šest stupnjeva slobode, te koristimo ljuskaste elemente. U danom kodu ne zadajemo nikakve rubne uvjete za analizu ni opterećenja već puštamo liboofem-u da samo konstruira izlazne podatke koje ćemo vizualizirati pomoću ParaViewa. Vizualizacijom koda 3.4 dobiva se slika 3.1.

Kod 3.4: Konstrukcija mrežastog modela pomoću PyGmsh-a te učitavanje mreže u OOFEM putem liboofem-a

```
import liboofem
import pygmsh


def kreiraj_mrezu(datoteka_s_cad_modelom):
    geom = pygmsh.built_in.Geometry()

    with open(datoteka_s_cad_modelom, 'r') as myfile:
        for line in myfile:
            geom.add_raw_code(line)

    geom.add_raw_code('Mesh.Algorithm=8;')

    points, cells, point_data, cell_data, field_data = pygmsh.generate_mesh(geom)

    return points, cells


def kreiraj_vremenske_funkcije(domena):
    vremenska_funkcija = liboofem.loadTimeFunction('ConstantFunction', 1, domena, f_t=1)
    domena.resizeFunctions(1)
    domena.setFunction(vremenska_funkcija.number, vremenska_funkcija)


def kreiraj_cvorove(gmsh_cvorovi, domena):
    domena.resizeDofManagers(len(gmsh_cvorovi))
    for indeks, koordinate in enumerate(gmsh_cvorovi, 1):
        domena.setDofManager(indeks,
                             liboofem.node(indeks, domena, coords=tuple(koordinate)))

```

```
31
32 def kreiraj_materijale_i_presjeke(domena):
33     poprecni_presjek = liboofem.simpleCS(1, domena, area=1, thick=1)
34     domena.resizeCrossSectionModels(1)
35     domena.setCrossSection(1, poprecni_presjek)
36
37     materijal = liboofem.isoLE(1, domena, d=1., E=200.e4, n=0.2, tAlpha=1.2e-5)
38     domena.resizeMaterials(1)
39     domena.setMaterial(1, materijal)
40
41
42 def kreiraj_konacne_elemente(gmsh_elementi, domena):
43     # Za trokutaste elemente koristimo tr_shell01. Ljuske koje kombiniraju CCT3D element
44     # (Mindlinova hipoteza) s trokutastim elementom opterecenja i 6 stupnjeva slobode.
45     domena.resizeElements(len(gmsh_elementi['triangle']) + len(gmsh_elementi['quad']))
46     for indeks, trokut in enumerate(gmsh_elementi['triangle'] + 1, 1):
47         domena.setElement(indeks, liboofem.element('tr_shell01', indeks, domena,
48                                                    nodes=tuple(trokut), mat=1,
49                                                    crossSect=1, nip=4))
50
51     last_indeks = indeks
52     # Za cetverokutne elemente koristimo mitc4shell koji su bilinearni elementi
53     # sastavljeni od cetiri cvora po uzoru na MITC tehniku.
54     for indeks, cetverokut in enumerate(gmsh_elementi['quad'] + 1, last_indeks + 1):
55         domena.setElement(indeks, liboofem.element('quad1mindlin', indeks, domena,
56                                                    nodes=tuple(cetverokut), mat=1,
57                                                    crossSect=1))
58
59
60 def kreiraj_opterecenja(domena):
61     pass
62
63
64 def inicijaliziraj_ulazne_podatke(gmsh_cvorovi, gmsh_elementi, datoteka_izlaza):
65     problem = liboofem.linearStatic(nSteps=1, outFile=datoteka_izlaza)
66
67     domena = liboofem.domain(1, 1, problem, liboofem.domainType._3dShellMode,
68                              tstep_all=True, dofman_all=True, element_all=True)
69     problem.setDomain(1, domena, True)
70
71     kreiraj_cvorove(gmsh_cvorovi, domena)
72     kreiraj_konacne_elemente(gmsh_elementi, domena)
73     kreiraj_materijale_i_presjeke(domena)
74     kreiraj_opterecenja(domena)
75     kreiraj_vremenske_funkcije(domena)
76
77     return problem
78
79
80 def napravi_analizu(problem):
81     problem.checkProblemConsistency()
82     problem.init()
83     problem.postInitialize()
84     problem.setRenumberFlag()
85     problem.solveYourself()
86     problem.terminateAnalysis()
87
88
89 gmsh_cvorovi, gmsh_element = kreiraj_mrezu('gmsh_model.txt')
90 problem = inicijaliziraj_ulazne_podatke(gmsh_cvorovi, gmsh_element, 'rjesenje.out')
```

```
91 napravi_analizu(problem)
```




Slika 3.1: Vizualizacija modela sastavljenog od ljuskastih elemenata s tri i četiri čvora pomoću ParaViewa nakon izrade mreže u PyGmsh-u i učitavanja u OOFEM

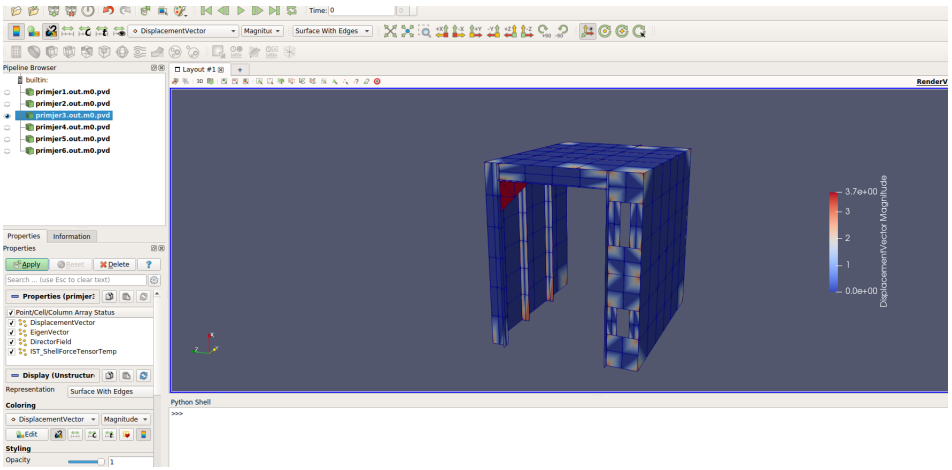
Ako se kao u gornjem primjeru izradom mrežastog model od učitanog CAD modela dobije neregularna mreža onda prilikom rješavanja fizikalnog problema na njoj nije moguće doći do rješenja što se vidi sa slike 3.2 koja je učitana prošla geometrija sa silama na ponekim čvorovima. Slika prikazuje jedan od međukoraka ka rješenju, ali kako izračun ne konvergira rješenje nije konačno ni ispravno.

## 3.5 Dodatne mogućnosti biblioteke liboofem

Kao što smo bili napomenuli *liboofem* u teoriji pruža mogućnost dohvaćanja svih objekata OOFEM-a iz C++-a putem Pythona. U nastavku dat ćemo jedan primjer s opisom kako dohvatiti još neke od zanimljivijih podataka.

Kod 3.5: Python program za rješavanje zadatak sa slike 1.7 zajedno s kodom koji ispituje elemente analize i vraća dodatne informacije o njima

```
1 import liboofem
2
3 podaci = liboofem.OOFEMTXTDataReader("zadatak_terminal.in")
4
5 problem = liboofem.InstanciateProblem(podaci, liboofem.problemMode._processor, 0)
6
7 problem.checkProblemConsistency()
8 problem.setRenumberFlag()
9 problem.solveYourself()
```




Slika 3.2: Vizualizacija prijašnjeg modela sa silama opterećenja u nekim čvorovima

```
10 problem.terminateAnalysis()
11
12 problem.printYourself()
13 '''
14 EngineeringModel: instance StaticStructural
15 number of steps: 1
16 number of eq's : 3
17 '''
18 print(problem.outputBaseFileName)  # rezultat.out
19
20 vremenski_korak = problem.giveCurrentStep()
21 print(vremenski_korak.giveTimeIncrement())  # 1.0
22 print(vremenski_korak.giveIntrinsicTime())  # 1.0
23 print(vremenski_korak.giveTargetTime())  # 1.0
24
25 # Prvo dohvatimo domenu koja ce nam sluziti kao izvor podataka
26 domena = problem.giveDomain(1)
27
28 # Dohvatimo prvi cvor
29 node = domena.giveDofManager(1)
30 node.printYourself()
31 '''
32 Node 1     coord : x 0.000000   y 0.000000   z 0.000000
33 dof 1  of Node 1 :
34 equation -1     bc 1
35 dof 3  of Node 1 :
36 equation -2     bc 1
37 load array : IntArray of size : 0
38 '''
39
40 # Dohvatimo prvi element domene
41 element = domena.giveElement(1)
42 print(element.giveClassName())  # Truss2d
43 print(element.computeLength())  # 2.5
44 print(element.giveNumberOfDofManagers())  # 2
```

```
45 print(element.giveDofManager(1).coordinates.printYourself())  # FloatArray (0, 0, 0)
46 print(element.giveDofManager(1).coordinates.printYourself())  # FloatArray (2.5, 0, 0)
47 print(element.geometryType)  # liboofem.Element_Geometry_Type.EGT_line_1
48
49 integration_rule = element.defaultIntegrationRule
50 print(integration_rule.giveNumberOfIntegrationPoints())  # 1
51
52 integration_point = integration_rule.getIntegrationPoint(0)
53 print(integration_point)  # <liboofem.GaussPoint object at 0x7f6be5ace910>
54
55 '''
56 Izvorna biblioteka liboofem ne implementira CharType klasu stoga bez modifikacije
57 nemoguce je koristiti metode na konacnim elementima poput
58 giveCharacteristicMatrix, giveCharacteristicValue i element.giveCharacteristicVector
59 jer je potpis tih funkcija na primjeru za dohvacanje matrice:
60 giveCharacteristicMatrix(oofem::PyElement {lvalue}, oofem::FloatMatrix {lvalue},
61                          oofem::CharType, oofem::TimeStep*)
62 '''
63 # trazena_matrica = liboofem.CharType(1)  # kod matrice krutosti je 1, vise u chartype.h
64 # matrica_krutosti = liboofem.FloatMatrix()
65 # matrica_krutosti.resize(4, 4)
66 # element.giveCharacteristicMatrix(matrica_krutosti, trazena_matrica, vremenski_korak)
67
68 poprecni_presjek = element.giveCrossSection()
69 poprecni_presjek.printYourself()
70 '''
71 Cross Section with properties :
72 Dictionary :
73    Pair (403,1.000000)
74    Pair (404,0.000000)
75    Pair (405,0.000000)
76    Pair (406,0.000000)
77    Pair (402,0.000000)
78    Pair (407,0.000000)
79    Pair (408,0.000000)
80    Pair (409,0.000000)
81    Pair (410,0.000000)
82    Pair (411,0.000000)
83 '''
84
85 print(domena.giveNumberOfBoundaryConditions())  # 4
86 rubni_uvjet = domena.giveBc(1)
87 print(rubni_uvjet.isImposed(vremenski_korak))  # True
```

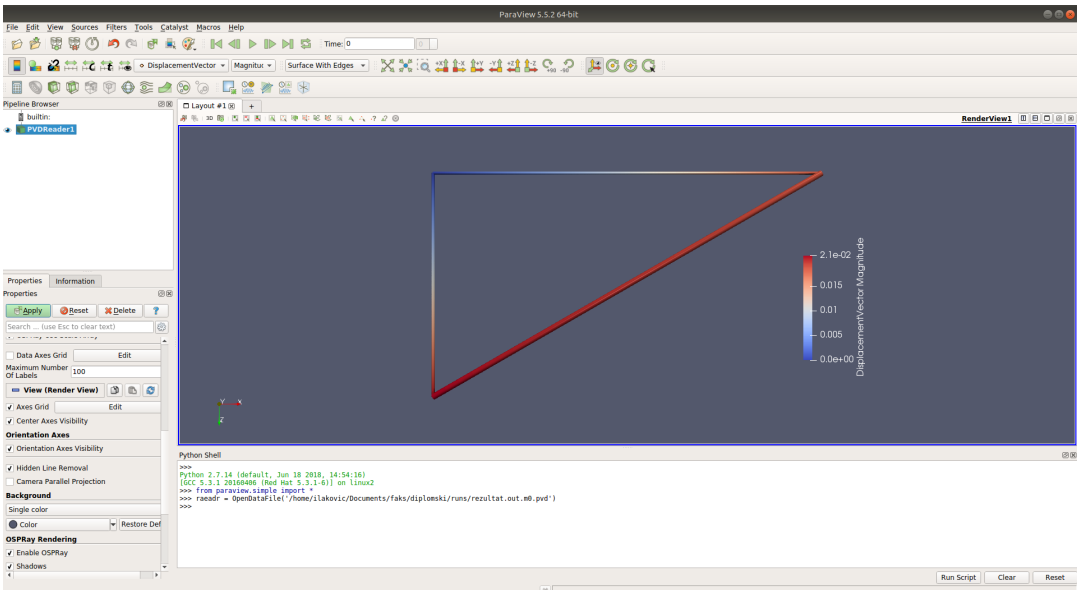
## 3.6 Vizualizacija dobivenih rezultata

U ovom dijelu prikazat ćemo jedan primjer kako se modeliranje može provoditi, to jest napisat ćemo skriptu koja nakon konstrukcije modela u alatu poput Salomea automatski pretvori izlazne podatke u pogodan ulazni format za OOFEM, napravi analizu te vizualizira podatke u softverskom programu ParaView. Za pomoć pri tome koristit ćemo Python skriptu za pretvorbu podataka, pokretanje OOFEM-a putem Pythona i Python sučelje koje nam ParaView daje na raspolaganje.

Kod 3.6: Python skripta za automatsku obradu i vizualizaciju modela konstruiranog sa softverom treće strane poput programa Salome

```
from subprocess import run

from paraview.simple import OpenDataFile
import liboofem

PUT_DO_OOFEM_KONVERTERA = '../oofem/tools/unv2oofem/unv2oofem.py'

UNV_DATOTEKA = 'salome_izlazna_datoteka.unv'
KONTROLNA_DATOTEKA = 'kontrolna_datoteka.ctrl'
OOFEM_ULAZ = 'oofem_ulazna_datoteka.in'

# Pretvaranje ulaznih podataka u pogodan oblik
run(['python', PUT_DO_OOFEM_KONVERTERA, UNV_DATOTEKA, KONTROLNA_DATOTEKA, OOFEM_ULAZ])

podaci = liboofem.OOFEMTXTDataReader(OOFEM_ULAZ)

problem = liboofem.InstanciateProblem(podaci, liboofem.problemMode._processor, 0)

'''
Dio koda u kojem se mogu napraviti dodatne izmjene poput dodavanje novih materijala,
rubnih uvjeta, konacnih elemenata i slicno.
Primjer dodavanja novog elementa
domena = problem.giveDomain(1)
novi_broj_elemenata = domena.giveNumberOfElements() + 1
novi_element = liboofem.element('truss2d', novi_broj_elemenata, domena, nodes=(1, 2))
domena.resizeElements(len(konacni_elementi))
domena.setElement(novi_element.number, novi_broj_elemenata)
'''

# Rjesavanje problema
problem.checkProblemConsistency()
problem.setRenumberFlag()
problem.solveYourself()
problem.terminateAnalysis()

IZLAZNA_DATOTEKA = problem.outputBaseFileName

# Ucitavanje podataka u ParaView koja odraduje vizualizaiju
OpenDataFile(IZLAZNA_DATOTEKA)
```

Ako rezultate prethodnih kodova (3.1, 3.2 i 3.3) koji rješavaju problem 1.7 iz dijela 1.3 želimo ručno učitati i prikazati u ParaViewu rezultat je slika 3.3




Slika 3.3: Vizualizacija OOFEM rješenja zadatka 1.7 pomoću ParaViewa

# Bibliografija


[1] Anisim Open Source Engineering Software, *FreeCAD FEM Tutorial Self-Weight Analyse eines Carports*, https://www.youtube.com/watch?v=2N-oPw1YuXM, posjećena 2018-08-15.

[2] *AutoCAD For Mac & Windows | CAD Software | Autodesk*, https://www.autodesk.com/products/autocad/overview, posjećena 2018-09-06.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine i H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, 1994 (en).

[4] *Boost.Python - 1.68.0*, https://www.boost.org/doc/libs/1_68_0/libs/python/doc/html/index.html, posjećena 2018-08-26.

[5] *Detailed Explanation of the Finite Element Method (FEM)*, https://www.comsol.com/multiphysics/finite-element-method, posjećena 2018-08-16.

[6] *Meshing your Geometry: When to Use the Various Element Types*, https://www.comsol.com/blogs/meshing-your-geometry-various-element-types/, posjećena 2018-08-16.

[7] R. Courant, *Variational methods for the solution of problems of equilibrium and vibrations*, Bulletin of the American Mathematical Society **49** (1943), br. 1, 1–23, ISSN 0002-9904, 1936-881X, http://www.ams.org/home/page/, posjećena 2018-08-15 (en-US).

[8] *The Finite Element Method for Problems in Physics | Coursera*, https://www.coursera.org/learn/finite-element-method, posjećena 2018-08-14.

[9] *16.16. ctypes — A foreign function library for Python — Python 3.6.6 documentation*, https://docs.python.org/3.6/library/ctypes.html, posjećena 2018-08-26.

[10] J. Dean, *Introduction to the Finite Element Method (FEM)*, https://www.ccg.msm.cam.ac.uk/images/FEMOR_Lecture_1.pdf, posjećena 2018-08-18.

[11] C. A. Felippa, *A Historical Outline of Matrix Structural Analysis: A Play in Three Acts*, Computers  Structures (2001) (en).

[12] *Modeliranje konstrukcija primjenom računara*, http://www.am.unze.ba/mkpr/, posjećena 2018-08-20.

[13] *FreeCAD: Your Own 3D Parametric Modeler*, https://www.freecadweb.org/, posjećena 2018-09-06.

[14] *FEM CalculiX Cantilever 3D - FreeCAD Documentation*, https://www.freecadweb.org/wiki/FEM_CalculiX_Cantilever_3D, posjećena 2018-08-18.

[15] *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*, http://gmsh.info/, posjećena 2018-09-06.

[16] *nschloe/pygmsh: Python interface for Gmsh*, https://github.com/nschloe/pygmsh, posjećena 2018-09-03.

[17] *GNU General Public License*, https://www.gnu.org/licenses/gpl-3.0.en.html, posjećena 2018-08-20.

[18] *USGS GRIDGEN*, https://water.usgs.gov/ogw/gridgen/, posjećena 2018-09-06.

[19] A. Hrennikoff, *Solution of problems of elasticity by the framework method*, Journal of applied mechanics **8.4** (1941), 169–175 (en-US).

[20] Ch.; Koronios A.; Mathew J. Kiritsis, D.; Eemmanouilidis, *Engineering Asset Management" Proceedings of the 4th World Congress on Engineering Asset Management (WCEAM)*, 2009 (en).

[21] Jaroslav Mackerle, *FEM and BEM in the context of information retrieval*, **80** (2002), 1595–1604 (en).

[22] *Tutorials — Matplotlib 2.2.3 documentation*, https://matplotlib.org/tutorials/index.html, posjećena 2018-08-28.

[23] *The oofem Open Source Project on Open Hub*, https://www.openhub.net/p/oofem, posjećena 2018-08-20.

[24] *OOFEM: Programmer's Manual*, http://www.oofem.org/resources/doc/programmer/programmer.pdf, posjećena 2018-08-22.

[25] *OOFEM: Reference Manual*, http://www.oofem.org/resources/doc/oofemrefman/index.html, posjećena 2018-08-22.

[26] *OOFEM repozitorij*, https://github.com/oofem/oofem, posjećena 2018-08-20.

[27] *Source code for oofemlib.cpp, github - OOFEM*, https://github.com/oofem/oofem/blob/master/bindings/python/oofemlib.cpp, posjećena 2018-08-27.

[28] B. Patzák, *OOFEM - an object-oriented simulation tool for advanced modeling of materials and structures*, Acta Polytechnica **52** (2012), br. 6, 59–66 (en).

[29] B. Patzák, *OOFEM home page*, http://www.oofem.org/resources/doc/oofemrefman/index.html, posjećena 2018-08-18.

[30] *Welcome to the www.salome-platform.org — SALOME Platform*, http://www.salome-platform.org/, posjećena 2018-09-06.

[31] J. Sorić, *Metoda konačnih elemenata*, Golden marketing, 2004 (hr).

[32] *Simplified Wrapper and Interface Generator*, http://www.swig.org/, posjećena 2018-08-26.

[33] *List of finite element software packages*, srpanj 2018, https://en.wikipedia.org/w/index.php?title=List_of_finite_element_software_packages&oldid=848886390, posjećena 2018-08-13.

[34] T. I. Zečević, *Objektno orijentirane biblioteke za realizaciju metode konačnih elemenata s primjenom u brodogradnji*, Diplomski rad, 2018 (hr).

[35] Gallagher R. H. Zienkiewicy, O. C. i R. W. Lewis, *The first 25 years and the future*, International Journal for Numerical Methods in Engineering (1994) (en).

# Sažetak

Metoda konačnih elemenata je jedna od važnijih numeričkih metoda za rješavanje problema u inženjerstvu poput strukturne analize objekata. Korisnost metode može sve vidjeti po količini alata koji su razvijeni za nju te po tome što je standard u industrijama poput zrakoplovne i automobilske. Cilj ovog diplomskog rada bio je predstaviti OOFEM (Object Oriented Finite Element Method), jedan takav alat i strukturu njegovog otvorenog koda tako da se budući korisnici mogu lakše snaći u njegovom korištenju, a posebice ako im se ukaže potreba za nadogradnjom i implementacijom novih funkcionalnosti. U opisu OOFEM-a poseban naglasak stavljen je na strukturnu analizu te konačne elemente i materijale potrebne koje strukturna analiza zahtjeva poput greda i šipki u dvodimenzionalnom sustavu analize. Središnji dio rada je opis korištenja OOFEM-a putem njegovog sučelja za korištenje iz programskog jezika Python. Osim generalnih informacija što se sve i kako može koristiti putem Pythona, priloženo je i nekoliko primjera te je čak i ukratko opisan postupak kako je izrađeno sučelje iz C++-a tako da se može koristiti u Pythonu. Motivacija iza toga je ta što nisu sve značajke izložene putem Python sučelja, te se ukratko pokazuje kako bi se mogle izložiti nove funkcionalnosti u slučaju potrebe.

# Summary

The finite element method is one of the most important numerical methods for solving engineering problems such as structural analysis of objects. The usefulness of the method can be seen by the amount of tools developed for it and by the fact that it became standard in industries such as aviation and automotive. The aim of this thesis was to present OOFEM (Object Oriented Finite Element Method), enabling future users to use it more easily, especially if they need to upgrade existing or implement new functionalities. In the description of OOFEM, a special emphasis was placed on the structural analysis method and finite elements and materials needed for it as beams and trusses in two-dimensional analysis. The central part of the thesis describes the use of OOFEM through its interface for use in the Python programming language. In addition to general information about what and how to use it through Python, a few examples are included, and a brief description of how a C++ interface is made so that it can be used in Python. The motivation behind it is that not all features are exposed through Python interface, so there is brief introduction how new functionalities could be exposed if necessary.

# Životopis

Ivan Laković rođen 14. studenog 1993. godine u Puli. Završava osnovnu školu u Svetom Lovreču, a zatim i opću gimnaziju u Pazinu. Na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu 2012. godine upisuje preddiplomski studij matematike. Izobrazbu nastavlja na istom fakultetu, na diplomskom studiju Računarstvo i matematika. Jedan semestar studija provodi na razmjeni na Sveučilištu u Wroclawu (Uniwersytet Wrocławski, Instytut Informatyki). Za trajanja studija, osim studiranja natječe se i postiže dobre rezultate na studentskim natjecanjima *Mozgalo, Airtravel Hackathon, AI Battleground, Try{code}catch i tehnološkoj areni STEM games-a* te pohađa i kolegij Raspodijeljeni razvoj programske potpore (Distributed software development) koje se održava na FER-u u suradnji sa sveučilištima Mälardalen University u Švedskoj i Politecnico di Milano u Italiji. Diplomski studiji završava 2018. godine pod mentorstvom prof. dr. sc. Luke Grubišića.