

Универзитет у Београду  
Грађевински факултет



## Семинарски рад

### Виши курс методе коначних елемената

Студенти:

Вукашин Томановић 588/20  
Јован Пантовић 638/20  
Никола Лакић 641/20  
Рајко Бабић 639/20

Ментори:

др. Марија Т. Нефовска-Даниловић  
др. Мирослав Маријановић

# Садржај

<b>1 Увод</b>	<b>1</b>
1.1 Принцип методе коначних елемената . . . . .	1
1.2 Практична примена методе коначних елемената . . . . .	2
1.3 Задаци за израду семинарског рада . . . . .	3
<b>2 Први задатак</b>	<b>5</b>
2.1 Теоријска позадина извођења матрице крутости . . . . .	5
2.2 Изопараметарска формулатија Q9 коначног елемента и ма- трица крутости . . . . .	9
2.3 Примена Python програмског језика за формирање матрице крутости Q9 коначног елемента . . . . .	11
<b>3 Други задатак</b>	<b>15</b>
3.1 Алгоритам прорачуна напона и деформација конзолног но- сача оптерећеног концентрисаном силом на слободном кра- ју методом коначних елемената . . . . .	15
3.2 Примена Python програмског језика за решавање конзолног носача . . . . .	16
3.3 Анализа резултата скрипти и поређење са аналитичким ре- шењем и резултатима SAP2000 комерцијалног програма . . .	21
<b>4 Трећи задатак</b>	<b>26</b>
4.1 Диспозиција и информације о параметрима . . . . .	26
4.2 Дијаграми пресечних сила Nx у зависности од односа димен- зија . . . . .	26
4.3 Закључак . . . . .	37
<b>5 Прилози</b>	<b>38</b>
<b>Литература</b>	<b>39</b>

# 1 Увод

## 1.1 Принцип методе коначних елемената

Метода коначних елемената представља методу нумеричког решавања диференцијалних једначина са граничним условима којима је описан физички дискретизован модел. Дискретизација модела се врши коначним елементима.

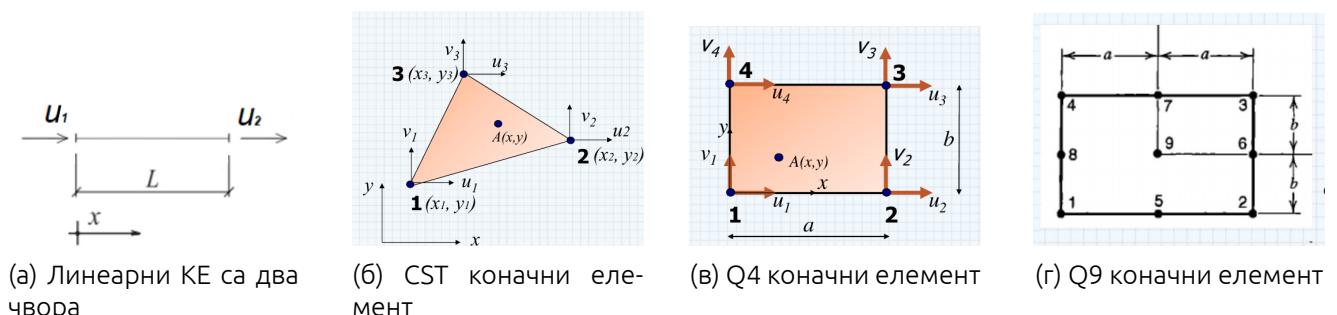
Коначни елемент је мали (геометријски) део континуума модела (плоча, греда, љуска итд) који је део мреже коначних елемената које дискретизују континуум. У једном коначном елементу су дефинисани чворови чији број и распоред зависи од типа коначног елемента, а у чворовима се дефинишу основне непознате које са интерполационим функцијама  $N$  дефинишу поље непознатих. Локални координатни систем коначног елемента може бити глобални  $(x, y)$  или изопараметарски  $(\xi, \eta)$ . Пример за коначни елемент типа  $Q9$  где су за основне непознате усвојена померања којих има два по чвиру (укупно 18), померања  $x_i$  су у  $X$  правцу док су  $y_i$  у  $Y$  правцу:

$$x(x, y) = \sum_{i=1}^9 N_i \cdot x_i \quad (1)$$

$$y(x, y) = \sum_{i=1}^9 N_i \cdot y_i \quad (2)$$

Како познате се узимају гранични услови у чворовима коначног елемента који у општем случају могу бити дати по померањима, силама или мешовито.

Неки типови коначних елемената, за непознате у чворовима су на сликама усвојена померања:



Слика 1: Типови коначних елемената [6]

Интерполационе функције  $N_i$  су дефинисане као функције чија је вредност у тачки посматраног чвора једнака јединици док је у тачкама осталих чвррова та вредност једнака нули. За интерполационе функције се бирају по-

линоми чији ред зависи од типа коначног елемента. Полиноми се бирају из разлога лакше диференцијације и интеграције. Пример интерполовајуће функције  $N_9$  за равански коначни елемент са девет чвора (Q9) у изопараметарском координатном систему:

$$N_9 = (1 - \xi^2)(1 - \eta^2) \quad (3)$$

У чворовима коначних елемената се налазе непознате/познате које од зависности од типа коначног елемента чине уређене н-торке. Обично се за основне непознате коначног елемента узимају генералисана померања јер се њиховим одређивањем лако долази до пресечних сила преко матрице крутости уз помоћ познатог израза:

$$\mathbb{R} = \mathbb{K} \cdot q \quad (4)$$

Даље уз помоћ одређених парцијалних диференцијала поља померања одређујемо и напонско стање унутар коначног елемента.

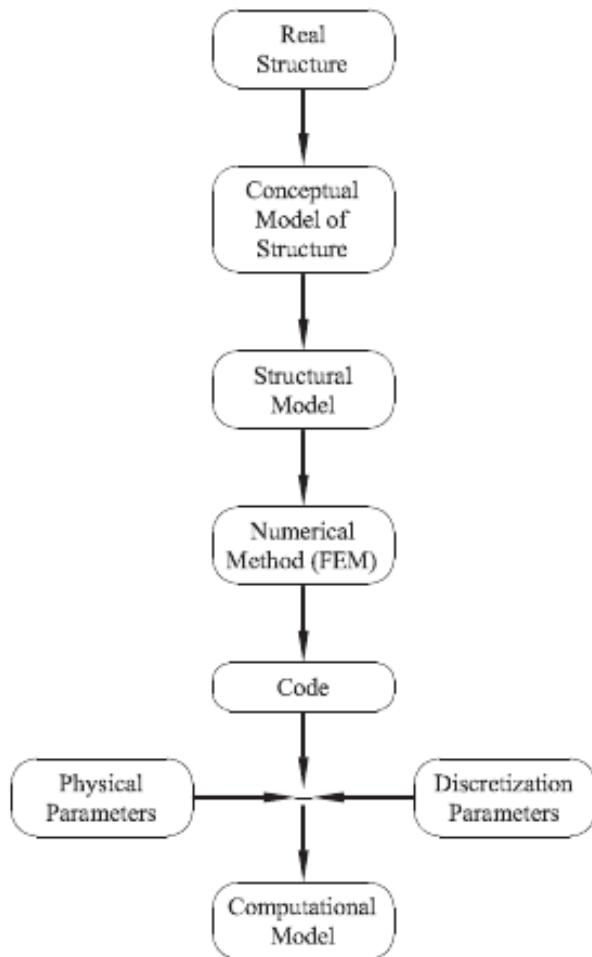
Задато оптерећење коначног елемента се концентрише у његовим чворовима.

Битно је нагласити да је метод коначних елемената апроксимативан метод јер су уведене априксимације дискретизацијом модела коначним елементима, интерполовационим функцијама, концептисањем оптерећења у чворовима и претпоставком линеарног понашања материјала које за резултат дају **приближне** вредности утицаја у чворовима коначних елемената. Тачност се може повећати прогушћавањем мреже коначних елемената али треба имати у виду да то доводи до веће потражње прорачунске моћи рачунара као и проблема које се могу јавити у неким карактеристичним тачкама континуума [4].

## 1.2 Практична примена методе коначних елемената

Познавањем матрице крутости једног елемента може се доћи до матрице крутости целог система (континуума) сабирањем адекватних елемената појединачних матрица крутости. За системе који су дискретизовани великим бројем коначних елемената тај посао постаје практично немогућ за "ручно" сабирање, као и решавање система једначина које формирају инвертована матрица крутости и вектор оптерећења да би одредили векторе померања у чворовима.

За решавање великог броја система једначина користимо рачунаре и да би рачунари служили тој сврси морамо прво применити алгоритам да реалну конструкцију сведемо на математички модел и коначно на код који ће нам решити системе једначина и приказати жељене резултате:



Слика 2: Алгоритам који приказује кораке потребне да би се дошло до применљивог кода од реалне конструкције [4]

Данас је немогуће замислiti брз и ефикасан прорачун конструкција без примене методе коначних елемената који се користе у софтверским пакетима. Технолошки напредак који су нам донели рачунари у грађевинарству су олакшали живот модерног инжењера али тај технолошки напредак који доноси доста аутоматизације је мач са две оштрице јер чине корисника лењим за критичким сагледавањем резултата. Овим семинарским радом ћемо показати колико места за грешку је могуће направити решавањем једноставних модела и зашто се на крају дана морамо ослањати на своје инжењерско знање стечено на факултету.

### 1.3 Задаци за израду семинарског рада

Први задатак се састоји из прорачуна и приказивања матрице крутости коначног елемента типа Q9 применом неког од програмских језика (Matlab, Mathcad, Python итд).

Други задатак је прорачун деформације и напонског стања статичког система конзоле оптерећене концетрисаном силом на слободном крају коришћењем једног од претходно наведених програмских језика за срачунату

матрицу крутости из првог задатка.

За трећи задатак је потребно коришћењем једног комерцијалног програма извући упоредне резултате варирањем димензија греде, стубова и растојања стубова континуалне греде оптерећене једнакоподељеним оптерећењем и дати закључак.

## 2 Први задатак

### 2.1 Теоријска позадина извођења матрице крутости

Приказаћемо поступак добијања матрице крутости уз помоћ основних принципа механике.

Основне једначине теорије еластичности могу да се опишу на два начина:

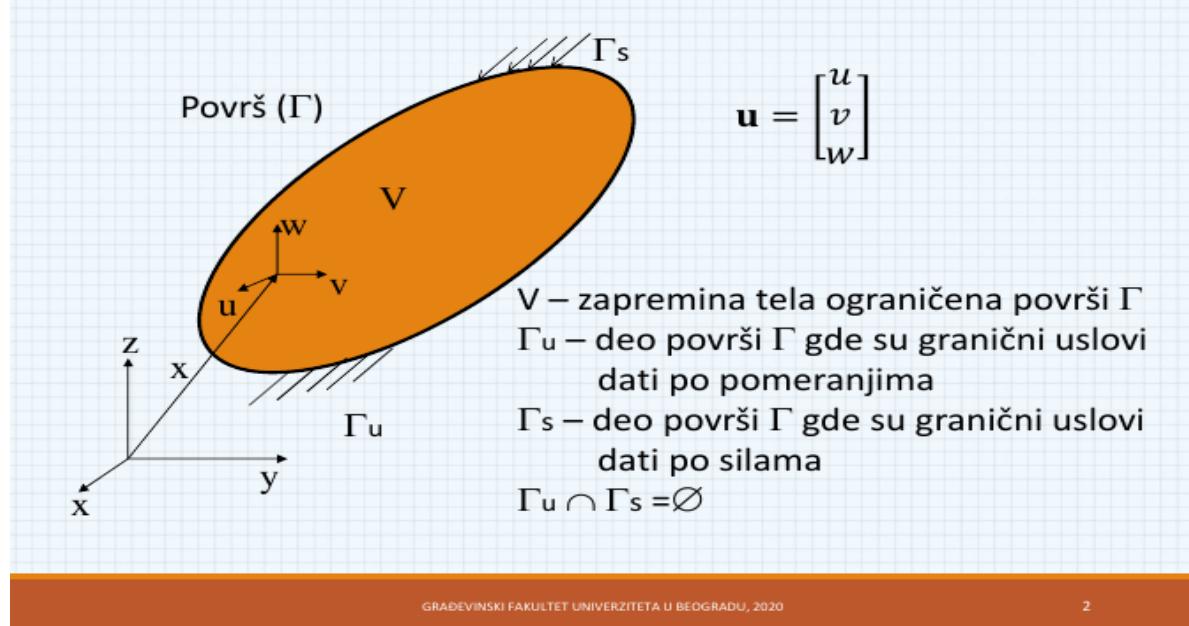
1. Математичком формулацијом које са граничним условима чине потпуно дефинисан проблем
2. Варијационим формулама

Математичким формулацијама са граничним условима можемо да решимо само једноставне проблеме, за комплексније проблеме нам стоји на располагању варијациона формулација у којој спада принцип виртуалног рада односно принцип виртуалних померања.

Виртуелна померања  $\delta u$  [6] су произвољна, бесконачно мала померања (непрекидне, диференцијабилне функције координата тачака) која задовољавају граничне услове по померањима.

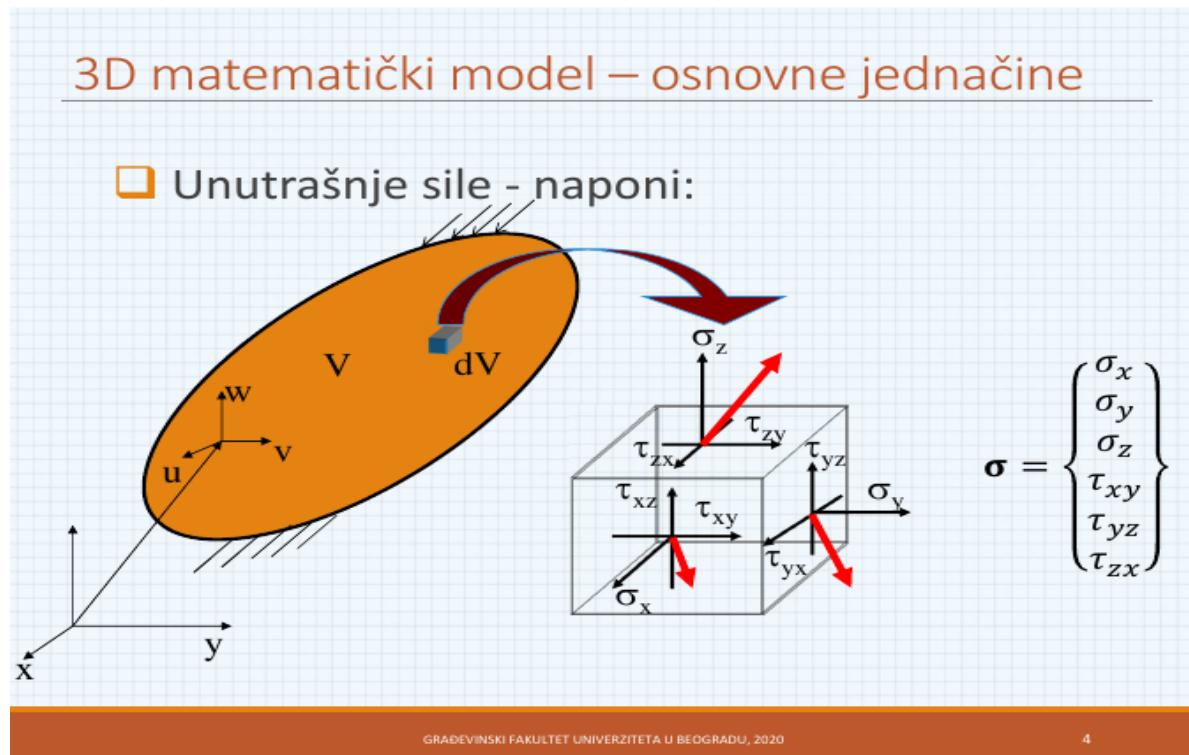
Нека је тело запремине  $V$  ограничено површи  $\Gamma$  оптерећено запреминским  $f$  и површинским  $r$  оптерећењем са граничним условима које су на граничној површи (контури) дати по померањима  $\Gamma_u$  и по силама  $\Gamma_s$ .

### 3D математички модел – основне једначине



Слика 3: Тело ограничено површи  $\Gamma$  [6]

Из тродимензионалог тела издвајамо елементарни део чија је запремина

$dV$ :

Слика 4: Тензор напона за елементарни део тела [6]

За систем кажемо да је у равнотежи ако је збир парцијалних извода напона  $\sigma$  са вектором оптерећења  $f$  за одређену раван једнак нули, тј:

$$\mathbb{L}^T \cdot \sigma + f = 0 \quad (5)$$

Где су:

$$\mathbb{L}^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix}$$

- матрица оператор, матрица парцијелних извода

$$\sigma = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix}$$

- тензор напона елементарног тела  $dV$

$$f = \begin{bmatrix} fx \\ fy \\ fz \end{bmatrix}$$

- вектор запреминског оптерећења

Када се једначина (5) множи се са вектором виртуалних померања  $\delta u$  и интегри по читавој запремини тела  $V$  добија се следеће:

$$\int_V L^T \sigma \delta u dV + \int_V f \delta u^T dV = 0 \quad (6)$$

За део контуре код којег су гранични услови дати по силама, услов равнотеже између спољашњих и унутрашњих сила дат је једначинама познатијим као Кошијеви површински услови [7]:

$$\sigma_x \cdot n_x + \tau_{xy} \cdot n_y + \tau_{xz} \cdot n_z = p_x \quad (7)$$

$$\tau_{yx} \cdot n_x + \sigma_y \cdot n_y + \tau_{yz} \cdot n_z = p_y \quad (8)$$

$$\tau_{zx} \cdot n_x + \tau_{zy} \cdot n_y + \sigma_z \cdot n_z = p_z \quad (9)$$

Или матрично:

$$\sigma n = p \quad (10)$$

Где су:

$$n = \begin{bmatrix} n_x & 0 & 0 & n_y & 0 & n_z \\ 0 & n_y & 0 & n_x & n_z & 0 \\ 0 & 0 & n_z & 0 & n_y & n_x \end{bmatrix} \text{ - матрица триг. функција углова нормале } n$$

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \text{ - вектор површинских сила у правцима } x, y, z$$

Слично као и са једначином (5) и Кошијеви површински услови се множе са вектором виртуалних померања  $\delta u^T$  и интеграле по контури где су гранични услови дати по силама  $\Gamma_S$ :

$$\int_{\Gamma_S} (\sigma n - p) \delta u^T dA = 0 \quad (11)$$

Сабирањем (5) са (11) при чему су:

$$L \delta u = \delta \epsilon \quad (12)$$

$$L^T(\sigma \delta u^T) = L^T \sigma \delta u^T + \sigma L \delta u \Rightarrow L^T \sigma \delta u^T = L^T(\sigma \delta u^T) - \sigma \delta \epsilon^T \quad (13)$$

$$\int_V L^T(\sigma \delta u^T) dV = \int_{\Gamma_S} \sigma \delta u^T n dA \quad (14)$$

Добијамо израз:

$$\int_V \sigma \delta \epsilon^T dV - \int_V f \delta u^T dV - \int_{\Gamma_S} p \delta u^T dA = 0 \quad (15)$$

Израз (15) представља принцип виртуалног рада који гласи:

За тело у равнотежи важи да је збир радова спољашњих и унутрашњих сила при виртуелним померањима једнак нули [6], тј:

$$\delta W_{ext} + \delta W_{int} = 0 \quad (16)$$

Где су:

$$\begin{aligned}\delta W_{ext} &= - \int_V f \delta u^T dV - \int_{\Gamma_s} p \delta u^T dA - \text{рад спољашњих сила} \\ \delta W_{int} &= \int_V \sigma \delta \epsilon^T dV - \text{рад унутрашњих сила}\end{aligned}$$

Из конститутивних једначина је позната веза:

$$\sigma = E\epsilon \quad (17)$$

Па је након сређивања заменом  $\delta u^T L^T = \delta \epsilon^T$  рад унутрашњих сила коначно једнак:

$$\delta W_{int} = \int_V \delta u^T L^T E L u dV \quad (18)$$

Поље померања  $u$  се апроксимира интерполяционим функцијама  $N$ :

$$u = Nq \quad (19)$$

За виртуелно померање  $\delta u$  следи:

$$\delta u = N \delta q \Rightarrow \delta u^T = \delta q^T N^T \quad (20)$$

Како је:

$$\epsilon = Lu \Rightarrow {}^{u=Nq} \epsilon = LNq = Bq \quad (21)$$

Коначно је израз за напон  $\sigma$  једнак:

$$\sigma = E\epsilon = EBq \quad (22)$$

Заменом претходних израза у једначини (3) и дељењем једначине са  $\delta q^T$  добијамо:

$$\left( \int_V B^T EB dV \right) q = \int_V N^T f dV + \int_{\Gamma_s} N^T p dA \quad (23)$$

Како је  $Kq = Q$  примећујемо да израз  $(\int_V B^T EB dV)$  представља матрицу крутости  $K$ , тј:

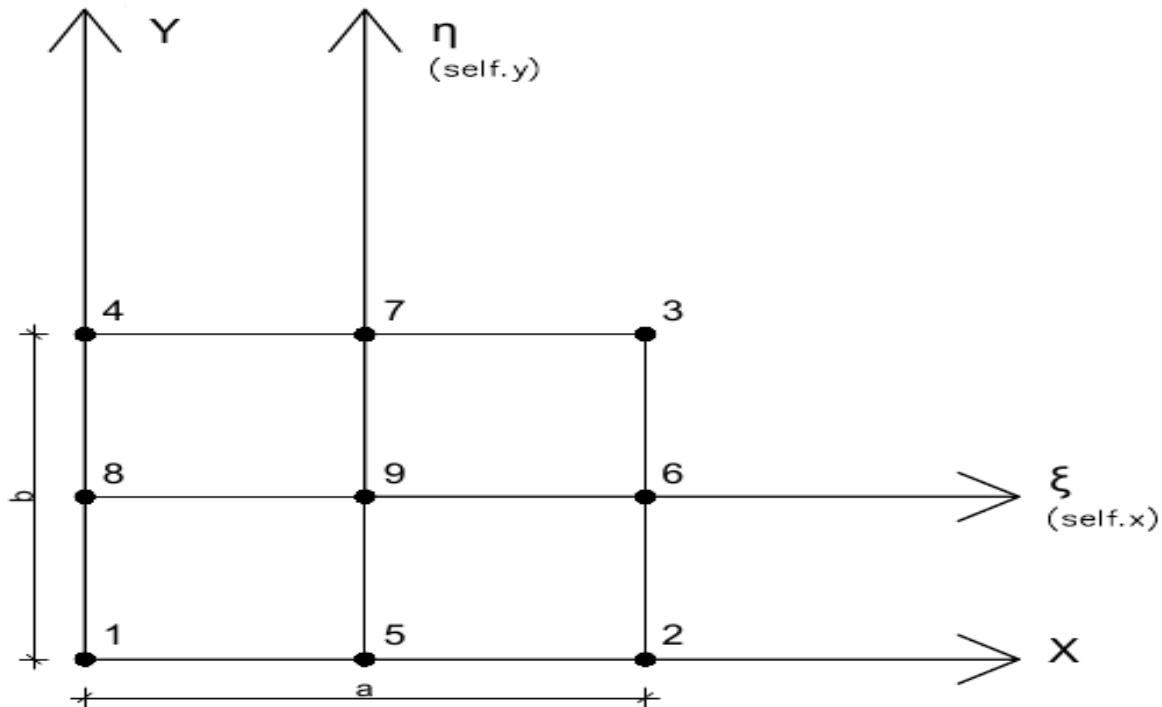
$$K = \int_V B^T EB dV \quad (24)$$

За равански коначни елемент Q9 са константном дебљином  $t$  матрица крутости има следећи облик:

$$K = t \int_A B^T EB dA \quad (25)$$

## 2.2 Изопараметарска формулатија Q9 коначног елемента и матрица крутости

Природни координатни систем  $(\xi, \eta)$  за  $Q9$  коначни елемент је приказан на следећој слици:



Слика 5: Природни координатни систем  $Q9$  коначног елемента

Идеја увођења оваквог система је лакша интеграција произвољног правоугаоног коначног елемента са 9 тачака пошто су границе интеграције дефинисане од  $[-1, 1]$  у оба ортогонална правца за природни координатни систем. Приликом интеграције се природни координатни систем трансформише у глобални уз помоћ Јакобијеве матрице.

Координате тачака  $i$  су дате у глобалном координатном систему  $X, Y$  дефинисаним на слици 5 (тачка 1 нпр има координате  $(0, 0)$ , тачка 2  $(a, 0)$ , итд) док су интерполяционе функције  $N_i(\xi, \eta)$  дефинисане у природном координатном систему.

Поља померања  $u(\xi, \eta)$  и  $v(\xi, \eta)$  су дефинисана на следећи начин:

$$u(\xi, \eta) = \sum_{i=1}^9 N_i(\xi, \eta) x_i \quad (26)$$

$$v(\xi, \eta) = \sum_{i=1}^9 N_i(\xi, \eta) y_i \quad (27)$$

Где су  $x_i$  и  $y_i$  померања тачке  $i$  у  $X$  односно  $Y$  правцу и оне чине основне непознате коначног елемента, интерполяционе функције  $N_i$  су дефиниса-

не на следећи начин:

$$N_1 = 0.25(1 - \xi)(1 - \eta) - 0.5N_5 - 0.5N_8 - 0.25N_9 \quad (28)$$

$$N_2 = 0.25(1 + \xi)(1 - \eta) - 0.5N_5 - 0.5N_6 - 0.25N_9 \quad (29)$$

$$N_3 = 0.25(1 + \xi)(1 + \eta) - 0.5N_6 - 0.5N_7 - 0.25N_9 \quad (30)$$

$$N_4 = 0.25(1 - \xi)(1 + \eta) - 0.5N_7 - 0.5N_8 - 0.25N_9 \quad (31)$$

$$N_5 = 0.5(1 - \xi^2)(1 - \eta) - 0.5N_9 \quad (32)$$

$$N_6 = 0.5(1 + \xi)(1 - \eta^2) - 0.5N_9 \quad (33)$$

$$N_7 = 0.5(1 - \xi^2)(1 + \eta) - 0.5N_9 \quad (34)$$

$$N_8 = 0.5(1 - \xi)(1 - \eta^2) - 0.5N_9 \quad (35)$$

$$N_9 = (1 - \xi^2)(1 - \eta^2) \quad (36)$$

Поље деформација је дефинисано у глобалном координатном систему оса као:

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} \quad (37)$$

Како су поља померања  $u$  и  $v$  у дефинисана у изопараметарском координатном систему  $(\xi, \eta)$  а поља деформација у глобалном, тј поља померања из којих следе поља деформација у координатном систему  $(X, Y)$  потребно је пронаћи одговарајућу везу између њих, то радимо на следећи начин:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial \xi} \cdot \frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta} \cdot \frac{\partial \eta}{\partial x} \quad (38)$$

Издвајањем  $\frac{\partial u}{\partial \xi}$  и  $\frac{\partial u}{\partial \eta}$  се добија:

$$\frac{\partial u}{\partial \xi} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial \xi} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial \xi} \quad (39)$$

$$\frac{\partial u}{\partial \eta} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial \eta} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial \eta} \quad (40)$$

Или матрично:

$$\begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} \quad (41)$$

Где је:

$$\mathbb{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} - \text{Јакобијева матрица} \quad (42)$$

Елементи Јакобијеве матрице су у ствари одговарајући парцијални изводи поља померања  $x$  и  $y$  где су:

$$\frac{\partial x}{\partial \xi} = \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \xi} \cdot x_i \quad \frac{\partial y}{\partial \xi} = \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \xi} \cdot y_i \quad (43)$$

$$\frac{\partial x}{\partial \eta} = \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \eta} \cdot x_i \quad \frac{\partial y}{\partial \eta} = \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \eta} \cdot y_i \quad (44)$$

Па се Јакобијева матрица може приказати и као:

$$\mathbb{J} = \begin{bmatrix} \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \xi} \cdot x_i & \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \xi} \cdot y_i \\ \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \eta} \cdot x_i & \sum_{i=1}^9 \frac{\partial N_i(\xi, \eta)}{\partial \eta} \cdot y_i \end{bmatrix} \quad (45)$$

Како су поља деформација  $\varepsilon$  (37) дефинисана као одговарајући парцијелни изводи поља померања по  $x$  и  $y$  а поља померања су функције интерполяционих функција у глобалном координатном систему потребно је успоставити везу између интерполовајућих функција у глобалном координатном систему ( $X, Y$ ) и интерполовајућих функција у природном координатном систему ( $\xi, \eta$ ). Везу између два координатна система, тј интерполовајућих функција у два различита координатна система успоставља претходно изведена Јакобијева матрица  $\mathbb{J}$ :

$$\left\{ \begin{array}{c} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{array} \right\} = \mathbb{J}^{-1} \left\{ \begin{array}{c} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{array} \right\} \quad (46)$$

Са претходном везом можемо да формирамо матрицу  $\mathbb{B}$ :

$$\mathbb{B}(x, y) = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \dots & \frac{\partial N_9}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & \dots & 0 & \frac{\partial N_9}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \dots & \frac{\partial N_9}{\partial y} & \frac{\partial N_9}{\partial x} \end{bmatrix} \quad (47)$$

Коначно матрица крутости (25) се може представити на следећи начин:

$$K = t \int_{-1}^1 \int_{-1}^1 B(x, y)^T E B(x, y) |J| d\xi d\eta \quad (48)$$

### 2.3 Примена Python програмског језика за формирање матрице крутости Q9 коначног елемента

Приликом писања скрипти је коришћен објектно оријентисани приступ искључиво из разлога изолације променљивих унутар метода пошто Python уме да "заборави" променљиве приликом њихове алокације у меморији. За симболичку интеграцију се користи библиотека *sympy* [5] уз стандардне библиотеке које нуди Python3 програмски језик. Приликом прорачуна

матрице крутости матрица материјала  $\mathbb{E}$  је дата за равно стање напона:

$$\mathbb{E} = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \quad (49)$$

Која је у коду представљена у *init* методи:

---

```
self.Ematrica = sp.Matrix([[self.E/(1-self.ni**2), self.ni*self.E/(1-self.ni**2), 0],
                           [self.ni*self.E/(1-self.ni**2), self.E/(1-self.ni**2), 0],
                           [0, 0, (1-self.ni)/2*self.E/(1-self.ni**2)]])
```

---

Низови координата тачака  $x_niz$  и  $y_niz$  у глобалном координатом систему ( $X, Y$ ) коначног елемента какав је приказан на слици 5 су дати у помоћној скрипти *mapiranjeQ9.py* као методе:

---

```
def x_niz(self): # Formiranje koordinata tacaka Xi
    niz = sp.Matrix([0, self.a, self.a, 0, 0.5 * self.a, self.a, 0.5 * self.a, 0, 0.5 * self.a])
    return niz

def y_niz(self): # Formiranje koordinata tacaka Yi
    niz = sp.Matrix([0, 0, self.b, self.b, 0, 0.5 * self.b, self.b, 0.5 * self.b, 0.5 * self.b])
    return niz
```

---

Јакобијева матрица, њена инверзна и детерминанта су приказане у следећим методама у скрипти *mapiranjeQ9.py*:

---

```
def Jakobijan(self):
    niz_interpolacionih = sp.Matrix([self.N1(), self.N2(), self.N3(), self.N4(),
                                      self.N5(), self.N6(), self.N7(), self.N8(), self.N9()])
    x_niz = self.x_niz()
    X = sp.transpose(x_niz)*niz_interpolacionih # Polje pomeranja u X pravcu
    X = X[0]
    y_niz = self.y_niz()
    Y = sp.transpose(y_niz)*niz_interpolacionih # Polje pomeranja u Y pravcu
    Y = Y[0]
    J = sp.Matrix([[sp.diff(X, self.x), sp.diff(Y, self.x)], [sp.diff(X, self.y),
                                                               sp.diff(Y, self.y)]])
    return J

def JakobijanInv(self):
    J = self.Jakobijan()
    J = J.inv()
    return J

def JakobijanDet(self):
    J = self.Jakobijan()
    J = J.det()
    return J
```

---

Формирање матрице  $\mathbb{B}$  спајањем матрица  $\mathbb{B}_i$  по колонама је представљено у методи:

---

```
# Formiranje matrice B spajanjem matrica Bi:
def Bmatrica(self):
    b = self.mp.B1().col_insert(2, self.mp.B2())
    b = b.col_insert(4, self.mp.B3())
    b = b.col_insert(6, self.mp.B4())
    b = b.col_insert(8, self.mp.B5())
    b = b.col_insert(10, self.mp.B6())
    b = b.col_insert(12, self.mp.B7())
    b = b.col_insert(14, self.mp.B8())
    b = b.col_insert(16, self.mp.B9())
    return b
```

---

Где су матрице  $\mathbb{B}_i$  матрице из скрипте *mapiranjeQ9.py*. Пример дела кода из скрипте *mapiranjeQ9.py* којом је представљена матрица  $\mathbb{B}_1$ :

---

```
#Kolone matrice B pri cemu su kolone matrice reda 3x2
def B1(self):
    vektor = self.JakobijanInv()*sp.Matrix([[sp.diff(self.N1(), self.x)],
                                             [sp.diff(self.N1(), self.y)]])
    x = vektor[0]
    y = vektor[1]
    matrica = sp.Matrix([[x, 0], [0, y], [y, x]])
    return matrica
```

---

Подинтегрална функција и њена интеграција са могућношћу исписивања елемента матрице крутости у симболичкој или интегралној форми као и исписивање целе матрице крутости је приказано у следећој методи:

---

```
# Formiranje podintegralne f-je i integracije:
def integracija(self):
    BT = self.BT()
    B = self.Bmatrica()
    upit = str(input('Numericka ili simbolicka vrednost koeficijenta krutosti matrice?
                      (recima unesi "numericka" ili "simbolicka", bez navodnika): '))
    upit = upit.lower()
    if upit == 'simbolicka':
        i = input('Unesi red matrice i: ')
        j = input('Unesi kolonu matrice j: ')
        if i == 'sve' and j == 'sve':
            podintegralna = BT*self.Ematrica*B*self.t*self.mp.JakobijanDet()
            integral = sp.integrate(sp.integrate(podintegralna, (self.x, -1, 1)), (self.y,
                -1, 1))
            print('\nK = ', integral)
        else:
            i = int(i)
            j = int(j)
            if i > 18 and j > 18:
                print('\ni i j moraju biti manji od 18! ')
                exit()
            else:
                podintegralna = BT.row(i - 1)*self.Ematrica*B.col(j -
                    1)*self.t*self.mp.JakobijanDet()
                integral = sp.integrate(sp.integrate(podintegralna, (self.x, -1, 1)),
                    (self.y, -1, 1))
                koeficijent_matrice = integral[0]
                print(f'\nk{i}_{j} = ', koeficijent_matrice)
    elif upit == 'numericka':
        i = input('Unesi red matrice i: ')
        j = input('Unesi kolonu matrice j: ')
        if i == 'sve' and j == 'sve':
```

---

```

a = float(input('Unesi dimenziju konacnog elementa a [m]: '))
b = float(input('Unesi dimenziju konacnog elementa b [m]: '))
E = float(input('Unesi moduo elasticnosti E [GPa]: '))
E = E * 10 ** 6
ni = float(input('Unesi Poasonov koeficijent \u03bd: '))
t = float(input('Unesi debljinu konacnog elementa t [m]: '))
podintegralna = BT*self.Ematrica*B*self.t*self.mp.JakobijanDet()
integral = sp.integrate(sp.integrate(podintegralna, (self.x, -1, 1)), (self.y,
    -1, 1))
integral = integral.subs({self.E: E, self.ni: ni, self.t: t, self.a: a, self.b:
    b})
print('\nK = ', integral)

elif i != 'sve' and j != 'sve':
    i = int(i)
    j = int(j)
    if i > 18 or j > 18:
        print('\ni i j moraju biti manji od 18! ')
        exit()
    else:
        a = float(input('Unesi dimenziju konacnog elementa a [m]: '))
        b = float(input('Unesi dimenziju konacnog elementa b [m]: '))
        E = float(input('Unesi moduo elasticnosti E [GPa]: '))
        E = E*10**6
        ni = float(input('Unesi Poasonov koeficijent \u03bd: '))
        t = float(input('Unesi debljinu konacnog elementa t [m]: '))
        podintegralna = BT.row(i - 1) * self.Ematrica * B.col(j - 1) *
            self.t*self.mp.JakobijanDet()
        integral = sp.integrate(sp.integrate(podintegralna, (self.x, -1, 1)),
            (self.y, -1, 1))
        integral = integral.subs({self.E: E, self.ni: ni, self.t: t, self.a: a,
            self.b: b})
        koeficijent_matrice = integral[0]
        print(f'\nk{i}_{j} = ', koeficijent_matrice)
    else:
        print('\n i i j moraju biti rec "sve" ili broj manje ili jednak 18!')
        exit()
else:
    print('\nUnesi lepo "simbolicka" ili "numericka" pri upitu.')
    exit()

```

Приликом читања кода имати у виду да се променљиве `self.x` и `self.y` односе на променљиве изопараметарског координатог система тј  $\text{self.x} = \xi$  и  $\text{self.y} = \eta$ .

Приликом уношења "sve" за упите променљивих  $i$  и  $j$  имати у виду да ће приказ резултата потрајати у зависности од радног такта процесора рачунара.

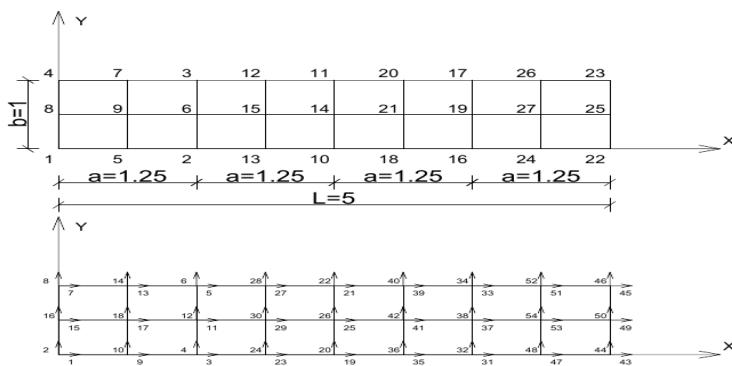
### 3 Други задатак

#### 3.1 Алгоритам прорачуна напона и деформација конзолног носача оптерећеног концентрисаном силом на слободном крају методом коначних елемената

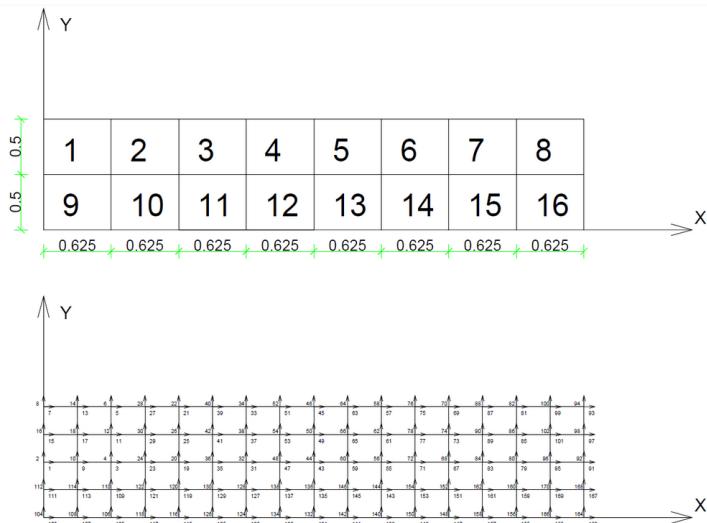
Алгоритам коришћен за прорачун конзолног носача је следећи:

1. Поделити носач на коначне елементе водећи рачуна о њиховој регуларности.
2. Нумерисати померања  $q$  у чворовима у оба правца и формирати кодиран вектор померања.
3. Заменити позната померања  $q_p$  њиховим вредностима.
4. Формирати вектор концентрисаних сила кодиран као и вектор померања и задати вредност концентрисане силе у правцу одређеног по мерања.
5. Формирати кодирану матрицу крутости за цео систем.
6. Одредити непозната померања.
7. Одредити деформације у чворовима коначних елемената.
8. Одредити напоне у чворовима коначних елемената и упросечити их ако је чвр заједнички за више коначних елемената.
9. Приказати резултате.

Прорачун је спроведен за две различите густине мреже коначних елемената: ређом са укупно 4 коначних елемената и са односом страна  $\frac{b}{a} = \frac{1}{1.25} = 0.8$  и гушћом са укупно 16 коначних елемената и са односом страна  $\frac{b}{a} = \frac{0.5}{0.625} = 0.8$ . Како је однос страна исти вредности елемената матрице крутости за један коначни елемент се неће променити за ове две различите густине. Усвојена конвенција померања и коначних елемената за различите густине су приказане на следећим slikama:



Слика 6: Нумерација чврова конзолног носача са мрежом од 4 коначна елемента (слика горе), нумерација (кодни бројеви) померања у чворовима конзолног носача са мрежом од 4 коначна елемента (слика доле)



Слика 7: Нумерација коначних елемената конзолног носача са мрежом од 16 коначних елемената (слика горе), нумерација (кодни бројеви) померања конзолног носача са мрежом од 16 коначних елемената (слика доле)

Скрипта као резултат враћа дијаграме померања и напона за различите густине коначног елемента и то за горње влакно носача, напони су упресечени у тачакма поклапања коначних елемената.

Приликом покретања скрипте за жељену густину откоментарисати `obj.plot()` или `obj.plot2x()` брисањем карактера #.

### 3.2 Примена Python програмског језика за решавање конзолног носача

Додатне библиотеке за операције са низовима, матрицама и њихово кодирање омогућавају `numpy` [3] и `pandas` [2], библиотека за штампање дијаграма омогућава `matplotlib` [1].

Како постоје посебне методе за различите густине мреже коначних елемената биће приказане само методе и њихово функционисање за мрежу са 4 коначна елемента, аналогно важи код и за гушћу мрежу само са разликом у броју померања.

Првенствено у `init(self)` методи се складишти као атрибут `self.niz_elemenata` низ вредности елемената матрице крутости читајући колону "Elementi" из `ElementiMK.csv` фајла.

Методом `index(self)` се врши кодирање померања коначних елемената где су врсте посебно померања чвррова једног коначног елемента, метода враћа само матрицу кодова (индекса) померања коначних елемената:

```
def index(self):
    matrica = np.array([
        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18], # KE 1
        [3, 4, 19, 20, 21, 22, 5, 6, 23, 24, 25, 26, 27, 28, 11, 12, 29, 30], # KE 2
        [19, 20, 31, 32, 33, 34, 21, 22, 35, 36, 37, 38, 39, 40, 25, 26, 41, 42], # KE 3
        [31, 32, 43, 44, 45, 46, 33, 34, 47, 48, 49, 50, 51, 52, 37, 38, 53, 54] # KE 4
    ])
```

---

```
    return matrica
```

---

Метода `indeksiranematrice(self)` формира матрицу крутости  $18 \times 18$  трансформишући претходно учитани низ елемената матрица крутости и коначно враћа низ 4 кодираних матрица крутости ( $4 \times 18 \times 18$ ) кодиране како и у `index(self)` методи:

---

```
def indeksiranematrice(self):
    matrica_krutosti = self.niz_elemenata.reshape(18, 18)
    indeks_matrica = pd.DataFrame(self.index())
    konacni_elementi = np.arange(0, 4)
    niz_matrica = []
    for i in konacni_elementi:
        indeks_matrice_i = np.array(indeks_matrica.loc[i])
        matrica = pd.DataFrame(matrica_krutosti, index=indeks_matrice_i,
                               columns=indeks_matrice_i)
        niz_matrica.append(matrica)
    return niz_matrica
```

---

Када имамо низ кодираних матрица крутости можемо да формирамо матрицу крутости  $\mathbb{K}$  целог система:

---

```
def ksystem(self):
    pomeranja = np.array(np.arange(1, 55))
    ksistema = pd.DataFrame(np.zeros((54, 54)), columns=pomeranja, index=pomeranja)
    matrice_krutosti = self.indeksiranematrice()
    for m in matrice_krutosti:
        for i in ksistema.index:
            for j in ksistema.columns:
                try:
                    ksistema.loc[i][j] = ksistema.loc[i][j] + m.loc[i][j]
                except Exception:
                    pass
    return ksistema
```

---

Са формираном матрицом крутости целог система потребно је формирати матрицу крутости уз непозната померања  $\mathbb{K}_{nn}$  избацивањем редова и колона са индексима (познатим померањима) (1, 2, 15, 16, 7, 8) и на исти начин формирати вектор чврних сила  $P_n$ . Вектор чврних сила  $P_n$  за све своје елементе има вредност 0 осим за елемент са индексом (кодом померања) 50 чија је вредност једнака  $-40kN$  по поставци задатка. Резултат методе `unknown_displacements(self)` је вектор непознатих померања  $q_n$  који се добија познатом везом:

$$q_n = \mathbb{K}_{nn}^{-1} P_n \quad (50)$$

Метода `unknown_displacements(self)`:

---

```
def unknown_displacements(self):
    vektor_cvornih_sila = np.array(np.zeros(54), dtype=np.int64)
    pomeranja = np.array(np.arange(1, 55))
    cvorne_sile = pd.Series(vektor_cvornih_sila, index=pomeranja)
    for i in cvorne_sile.index:
```

---

```

if i == 50: # i za 4 KE je 50
    cvorne_sile.loc[50] = cvorne_sile.loc[50] - 40
poznata_pomeranja = np.array([1, 2, 15, 16, 7, 8])
nepoznata_pomeranja = pd.Series(pomeranja, index=pomeranja).drop(poznata_pomeranja)
pnn = cvorne_sile.drop(poznata_pomeranja)
knn = self.ksystem().drop(index=poznata_pomeranja, columns=poznata_pomeranja)
qnn = np.dot(np.linalg.inv(knn), pnn)
qnn = pd.Series(qnn, index=nepoznata_pomeranja)
return qnn

```

---

Када смо одредили непозната померања можемо да формирајмо вектор свих померања чворова  $q$ , метода `displacements(self)` враћа кодирани вектор померања свих чворова система:

---

```

def displacements(self):
    qnn = self.unknown_displacements2x()
    q = pd.Series(np.zeros(54), index=np.arange(1, 55))
    for i in q.index:
        try:
            q.loc[i] = q.loc[i] + qnn.loc[i]
        except Exception:
            pass
    return q

```

---

Са формираним вектором померања чворова постоји све што је потребно да се одреде деформације а потом и напони у чворовима коначних елемената. Како су поља деформација  $\varepsilon$  (37) одређени парцијелни изводи поља померања  $u(x, y)$  и  $v(x, y)$  а поља померања су дефинисана у природном координатном систему  $(\xi, \eta)$  тј:

$$\begin{aligned}
 u(\xi, \eta) &= N_1 q_{x1} + N_2 q_{x2} + N_3 q_{x3} + N_4 q_{x4} + N_5 q_{x5} + N_6 q_{x6} + N_7 q_{x7} + N_8 q_{x8} + N_9 q_{x9} \\
 v(\xi, \eta) &= N_1 q_{y1} + N_2 q_{y2} + N_3 q_{y3} + N_4 q_{y4} + N_5 q_{y5} + N_6 q_{y6} + N_7 q_{y7} + N_8 q_{y8} + N_9 q_{y9} \\
 N_i &\text{ -- интерполационна функција } i \\
 q_{xi} &\text{ -- померање } i \text{ у X правцу} \\
 q_{yi} &\text{ -- померање } i \text{ у Y правцу}
 \end{aligned}$$

Потребно је дефинисати изразе за поља померања у природном координатном систему. Без посебног извођења она су дефинисана као:

$$\varepsilon(\xi, \eta) = tM \begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{bmatrix} \quad (51)$$

Где су:

$$t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad (52)$$

$$M = \begin{bmatrix} [\mathbb{J}^{-1}] & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & [\mathbb{J}^{-1}] \end{bmatrix} \quad (53)$$

$$\mathbb{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} - \text{Јакобијева матрица} \quad (54)$$

Симболички прорачун деформацијских величина у тачкама од 1-9 како су приказане на слици 5 посебно за сваку тачку раде методе `epsilon1` из скрипте *mapiranjeQ9.py*, методе враћају низ (вектор) од 3 елемента деформацијских величина у симболичкој форми. Пример једне такве методе за тачку 1 коначног елемента:

---

```
def epsilon1(self):
    Jakobian = sp.Matrix([[sp.diff(self.polje_pomeranja_x(), self.x)],
                          [sp.diff(self.polje_pomeranja_x(), self.y)], [sp.diff(self.polje_pomeranja_y(),
                           self.x)], [sp.diff(self.polje_pomeranja_y(), self.y)]])
    inv_J = self.JakobianInv()
    M = sp.Matrix([[inv_J[0], inv_J[1], 0, 0], [inv_J[2], inv_J[3], 0, 0], [0, 0,
                           inv_J[0], inv_J[1]], [0, 0, inv_J[2], inv_J[3]]])
    t_matrica = sp.Matrix([[1, 0, 0, 0], [0, 0, 0, 1], [0, 1, 1, 0]])
    epsilon = t_matrica*M*Jakobian
    epsilon = epsilon.subs({self.x: -1, self.y: -1})
    epsilon = np.array([epsilon[0], epsilon[1], epsilon[2]], dtype=str)
    return epsilon
```

---

Са изразима за деформацијске величине можемо одредити напоне у тачкама коначних елемената уз помоћ Хуковог закона:

$$\sigma = E\epsilon \quad (55)$$

У коду тај прорачун врши метода `stress(self, ke, tacka, q, napon)` која за дате параметре враћа низ (вектор) напона:

$$\sigma = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} \quad (56)$$

Параметар `ke` представља број коначног елемента (за 4 КЕ тај број је 1-4), параметар `tacka` је у ствари деформацијско стање у тачки коначног елемента какве су приказане на слици 5 коју враћају методе `epsilon1`, параметар `q` је вектор померања система, и параметар `napon` има вредности: 0 - за приказ напона  $\sigma_x$ , 1 - за приказ напна  $\sigma_y$  и 2 - за приказ напона  $\tau_{xy}$ :

---

```

def stress(self, ke, tacka, q, napon):
    indeksi = self.index()
    Q = [q[j] for j in indeksi[ke - 1]]
    X = [Q[2 * j] for j in range(9)]
    Y = [Q[2 * j + 1] for j in range(9)]
    a = self.a
    b = self.b
    qx1 = X[0]
    qx2 = X[1]
    qx3 = X[2]
    qx4 = X[3]
    qx5 = X[4]
    qx6 = X[5]
    qx7 = X[6]
    qx8 = X[7]
    qx9 = X[8]
    qy1 = Y[0]
    qy2 = Y[1]
    qy3 = Y[2]
    qy4 = Y[3]
    qy5 = Y[4]
    qy6 = Y[5]
    qy7 = Y[6]
    qy8 = Y[7]
    qy9 = Y[8]
    epsilon1 = eval(tacka[0])
    epsilon2 = eval(tacka[1])
    epsilon3 = eval(tacka[2])
    konacno = np.dot(self.E_matrica(), np.array([[epsilon1], [epsilon2], [epsilon3]]))
    konacno = konacno[napon][0]
    return konacno

```

---

Штампање дијаграма врши метода `plot(self)` формирајући низове вредности у тачкама горњег влакна:

---

```

def plot(self):
    q = self.displacements()
    kt = mp.KoordinateTacaka()
    q = pd.Series(q, index=np.arange(1, 55))
    ugib = [q.loc[16], q.loc[18], q.loc[12], q.loc[30], q.loc[26], q.loc[42], q.loc[38],
            q.loc[54], q.loc[50]]
    sigma_x = np.array([self.stress(ke=1, tacka=kt.epsilon4(), q=q, napon=0),
                       self.stress(ke=1, tacka=kt.epsilon7(), q=q, napon=0),
                       self.average(a=self.stress(ke=1, tacka=kt.epsilon3(), q=q, napon=0),
                                    b=self.stress(ke=2, tacka=kt.epsilon4(), q=q, napon=0)),
                       self.stress(ke=2, tacka=kt.epsilon7(), q=q, napon=0),
                       self.average(a=self.stress(ke=2, tacka=kt.epsilon3(), q=q, napon=0),
                                    b=self.stress(ke=3, tacka=kt.epsilon4(), q=q, napon=0)),
                       self.stress(ke=3, tacka=kt.epsilon7(), q=q, napon=0),
                       self.average(a=self.stress(ke=3, tacka=kt.epsilon3(), q=q, napon=0),
                                    b=self.stress(ke=4, tacka=kt.epsilon4(), q=q, napon=0)),
                       self.stress(ke=4, tacka=kt.epsilon7(), q=q, napon=0),
                       self.stress(ke=4, tacka=kt.epsilon3(), q=q, napon=0)
                      ])
    sigma_y = np.array([self.stress(ke=1, tacka=kt.epsilon4(), q=q, napon=1),
                       self.stress(ke=1, tacka=kt.epsilon7(), q=q, napon=1),
                       self.average(a=self.stress(ke=1, tacka=kt.epsilon3(), q=q, napon=1),
                                    b=self.stress(ke=2, tacka=kt.epsilon4(), q=q, napon=1)),
                       self.stress(ke=2, tacka=kt.epsilon7(), q=q, napon=1),
                       self.average(a=self.stress(ke=2, tacka=kt.epsilon3(), q=q, napon=1),
                                    b=self.stress(ke=3, tacka=kt.epsilon4(), q=q, napon=1)),
                       self.stress(ke=3, tacka=kt.epsilon7(), q=q, napon=1),
                       self.average(a=self.stress(ke=3, tacka=kt.epsilon3(), q=q, napon=1),
                                    b=self.stress(ke=4, tacka=kt.epsilon4(), q=q, napon=1)),
                       self.stress(ke=4, tacka=kt.epsilon7(), q=q, napon=1),
                       self.stress(ke=4, tacka=kt.epsilon3(), q=q, napon=1)
                      ])

```

---

```

        b=self.stress(ke=3, tacka=kt.epsilon4(), q=q, napon=1)),
        self.stress(ke=3, tacka=kt.epsilon7(), q=q, napon=1),
        self.average(a=self.stress(ke=3, tacka=kt.epsilon3(), q=q, napon=1),
                      b=self.stress(ke=4, tacka=kt.epsilon4(), q=q, napon=1)),
        self.stress(ke=4, tacka=kt.epsilon7(), q=q, napon=1),
        self.stress(ke=4, tacka=kt.epsilon3(), q=q, napon=1)
    ])

tau_xy = np.array([self.stress(ke=1, tacka=kt.epsilon4(), q=q, napon=2),
                    self.stress(ke=1, tacka=kt.epsilon7(), q=q, napon=2),
                    self.average(a=self.stress(ke=1, tacka=kt.epsilon3(), q=q, napon=2),
                                  b=self.stress(ke=2, tacka=kt.epsilon4(), q=q, napon=2)),
                    self.stress(ke=2, tacka=kt.epsilon7(), q=q, napon=2),
                    self.average(a=self.stress(ke=2, tacka=kt.epsilon3(), q=q, napon=2),
                                  b=self.stress(ke=3, tacka=kt.epsilon4(), q=q, napon=2)),
                    self.stress(ke=3, tacka=kt.epsilon7(), q=q, napon=2),
                    self.average(a=self.stress(ke=3, tacka=kt.epsilon3(), q=q, napon=2),
                                  b=self.stress(ke=4, tacka=kt.epsilon4(), q=q, napon=2)),
                    self.stress(ke=4, tacka=kt.epsilon7(), q=q, napon=2),
                    self.stress(ke=4, tacka=kt.epsilon3(), q=q, napon=2)
    ])

x_osa = np.linspace(0, 5, num=9) # 4 KE
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x_osa, ugib)
axs[0, 0].set_title('Ugib duz grede [mm]')
axs[0, 0].set_xlabel('L [m]')
axs[0, 0].set_xticks(np.arange(0, 6))
axs[0, 1].plot(x_osa, sigma_x)
axs[0, 1].set_title('\u03c3x [KPa]')
axs[0, 1].set_xlabel('L [m]')
axs[0, 1].set_xticks(np.arange(0, 6))
axs[1, 0].plot(x_osa, sigma_y)
axs[1, 0].set_title('\u03c3y [KPa]')
axs[1, 0].set_xlabel('L [m]')
axs[1, 0].set_xticks(np.arange(0, 6))
axs[1, 1].plot(x_osa, tau_xy)
axs[1, 1].set_title('\u03c4xy [KPa]')
axs[1, 1].set_xlabel('L [m]')
axs[1, 1].set_xticks(np.arange(0, 6))
fig.tight_layout()
plt.show()

```

### 3.3 Анализа резултата скрипти и поређење са аналитичким решењем и резултатима SAP2000 комерцијалног програма

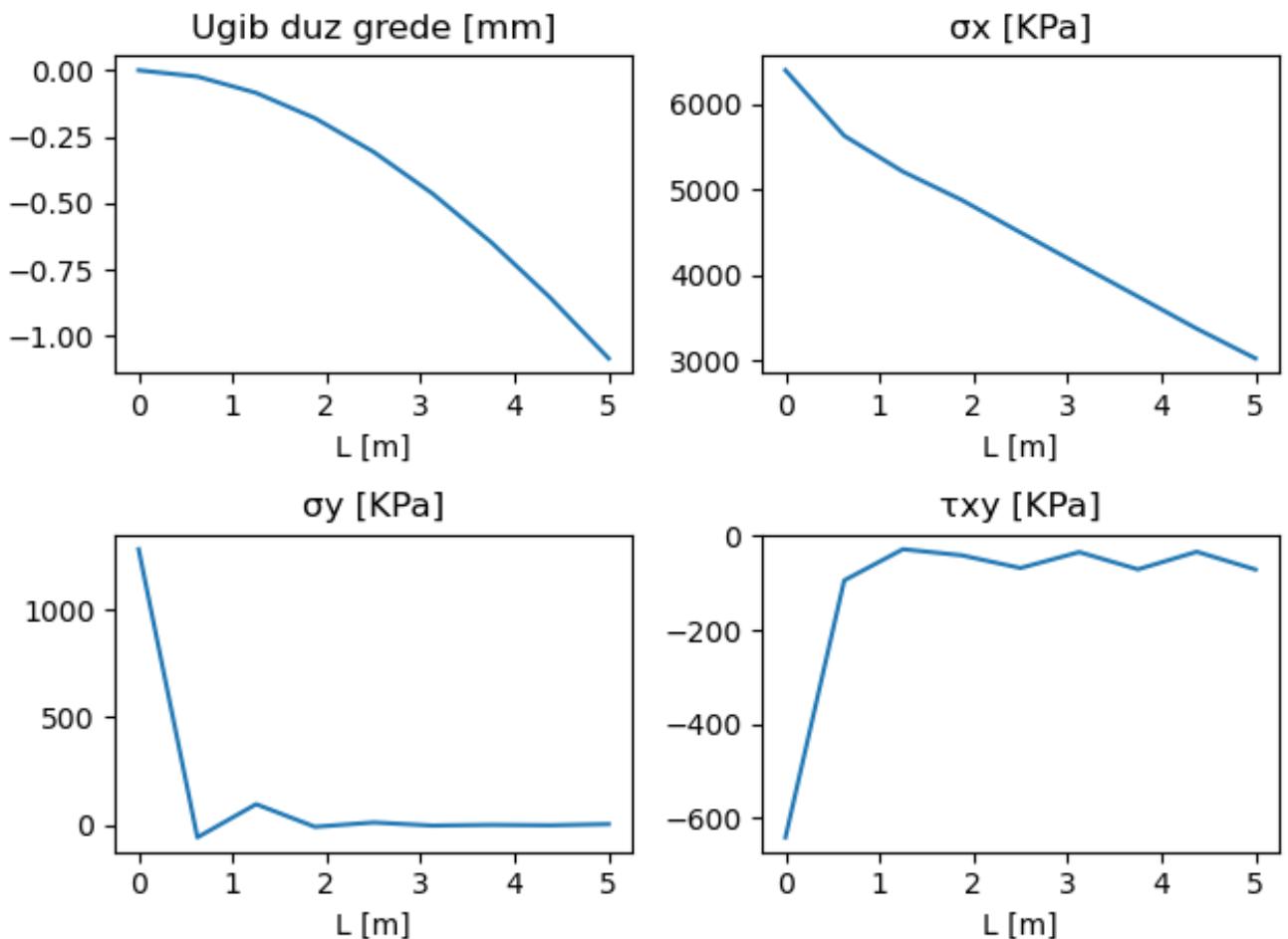
Аналитичко решење максималног угиба за гредну конзолу дужине  $5m$ , димензија правоугаоног попречног пресека  $b/h = 0.2/1m$  и оптерећеном на слободном крају концентрисаном силом од  $P = 40kN$  је:

$$q_{max} = \frac{Pl^3}{3EI} = \frac{12Pl^3}{3Eb h^3} = \frac{4 \cdot 40 \cdot 5^3}{30 \cdot 10^6 \cdot 0.2 \cdot 1^3} = 0.00333m = 3.33mm$$

Аналитичка вредност напона горњег влакна у укљештењу је:

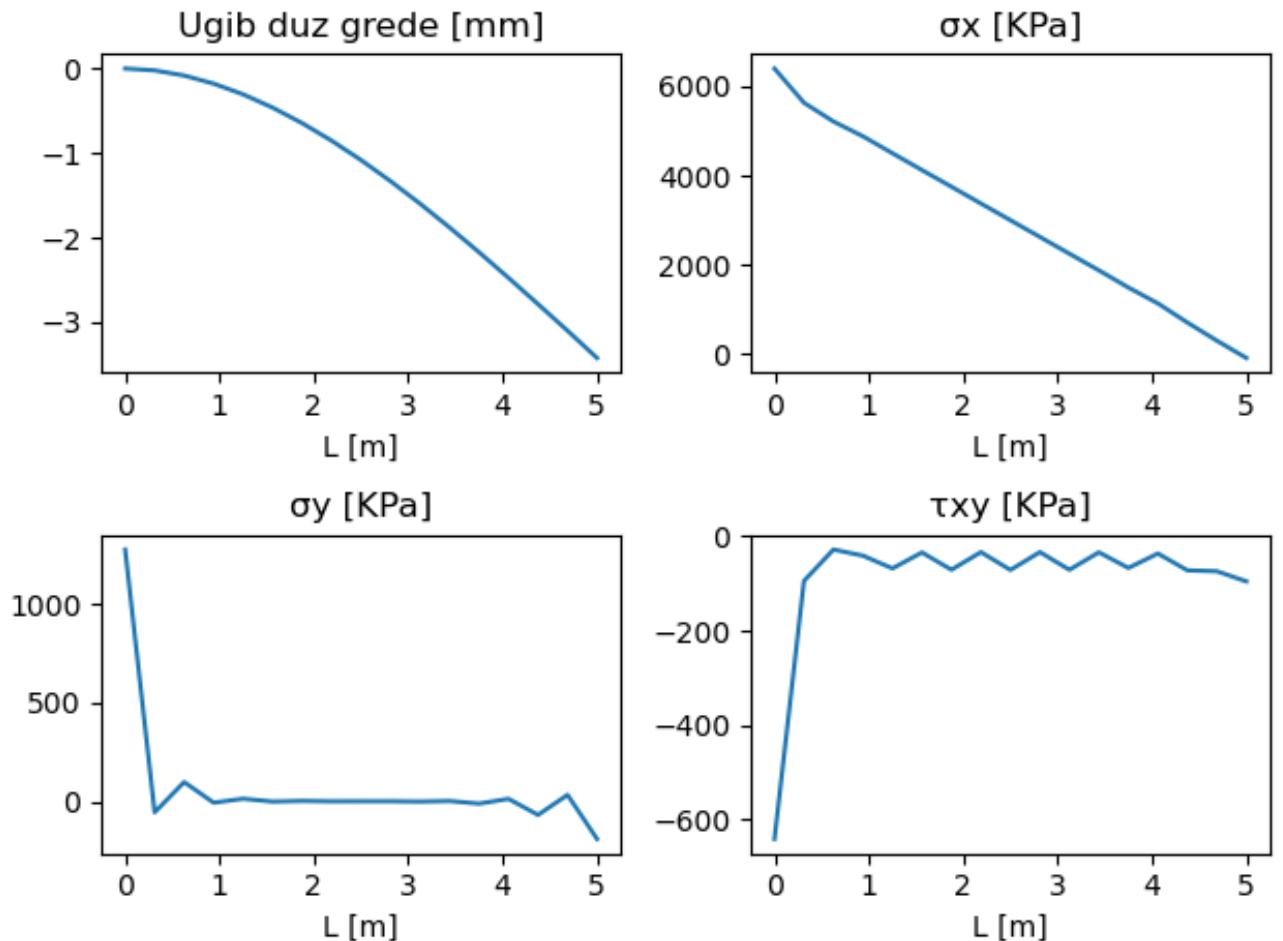
$$\begin{aligned}\sigma_x &= \frac{6M}{bh^2} = \frac{6 \cdot 40 \cdot 5}{0.2 \cdot 1^2} = 6000kPa = 6MPa \\ \sigma_y &= 0 \\ \tau_{xy} &= 0\end{aligned}$$

Резултат скрипте за мрежу од 4 коначна елемента:



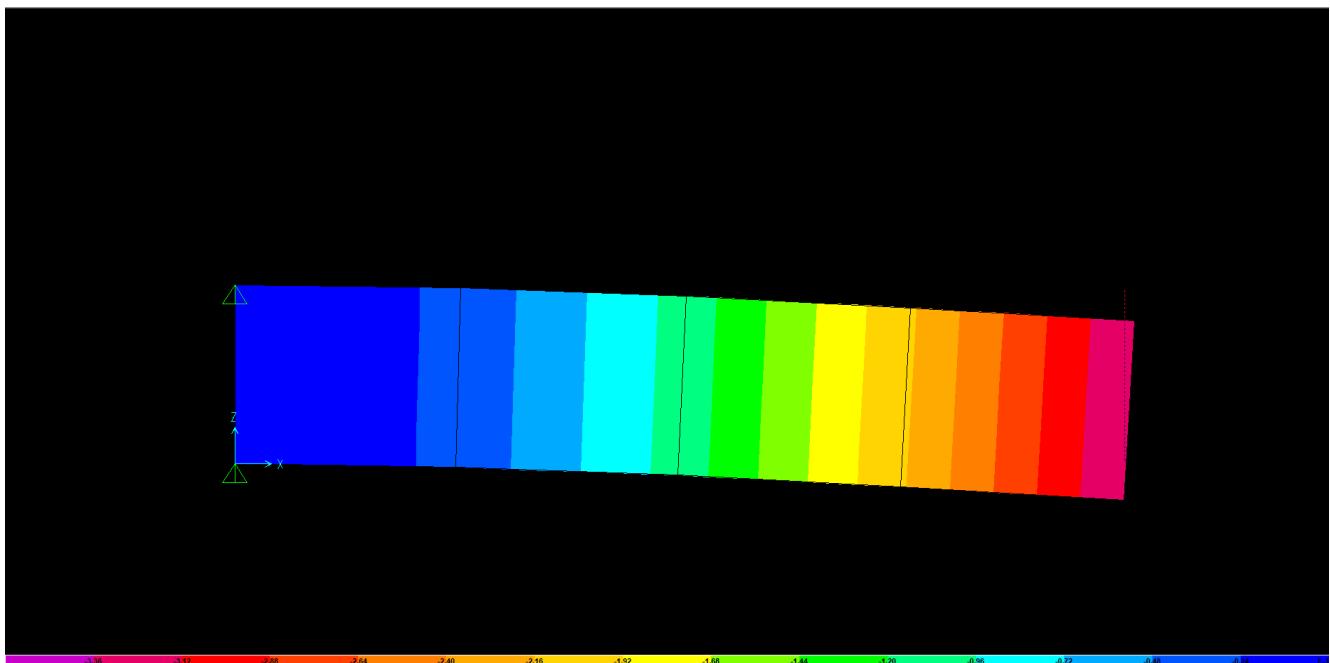
Слика 8: Python - 4 KE

Резултат скрипте за мрежу од 16 коначних елемента:

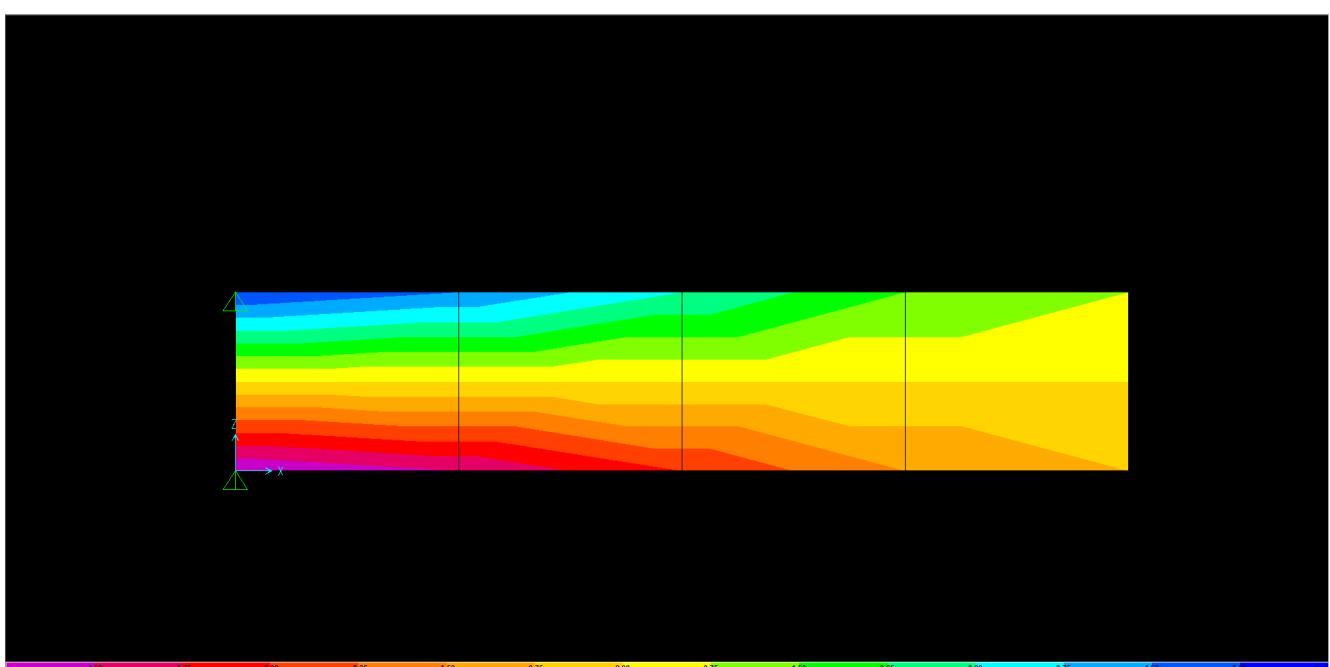


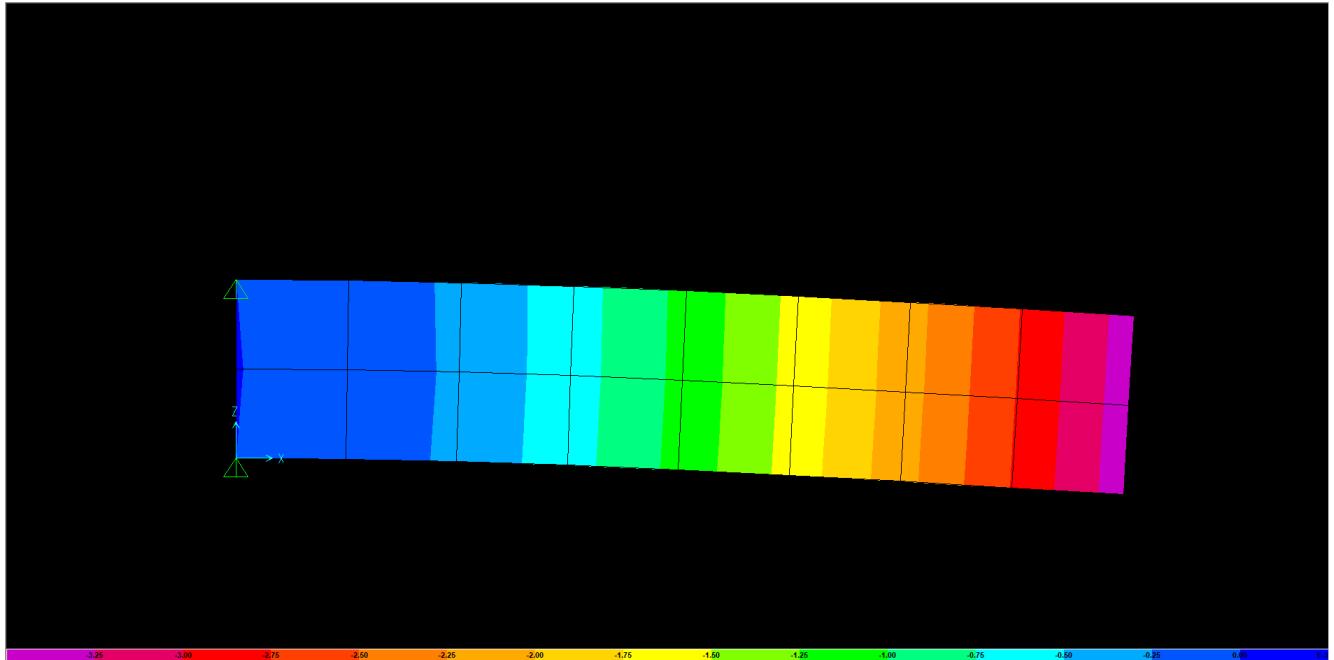
Слика 9: Python - 16 KE

За греду из SAP-а која је моделирана као танка плоча напрегнута у својој равни (plate-thin) резултати су следећи:

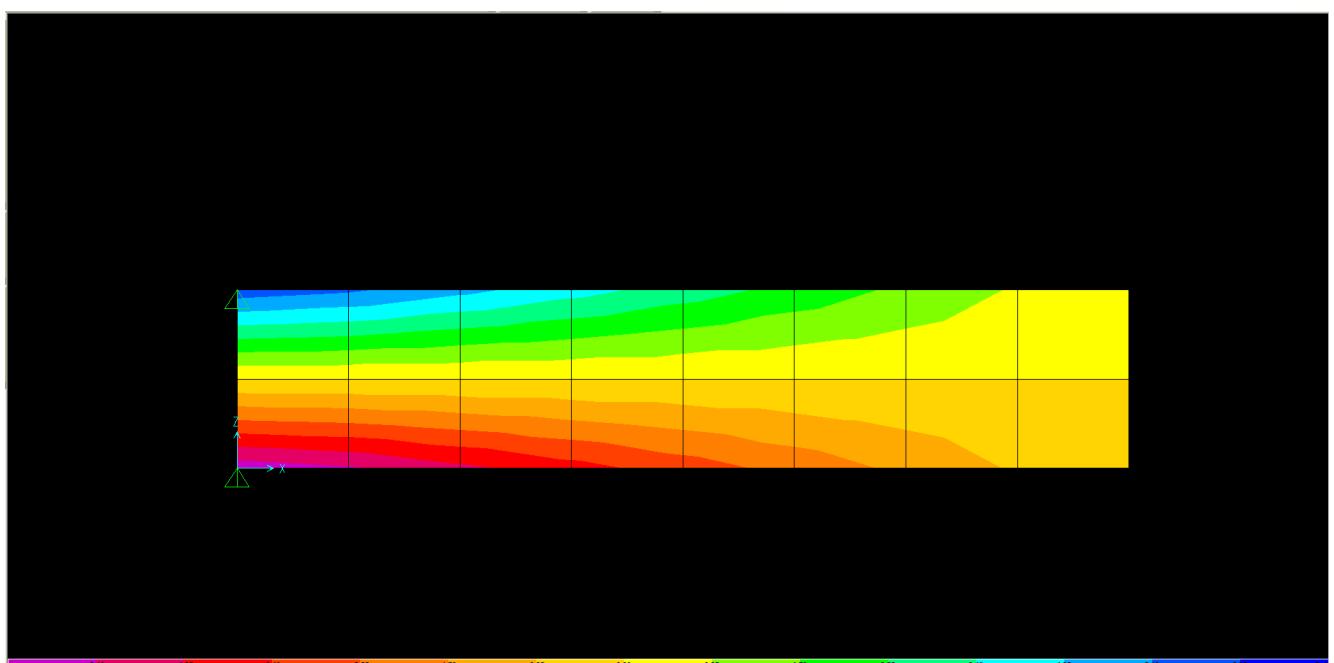


Слика 10: Ugib - 4 KE

Слика 11: Напон  $\sigma_x$  - 4 KE



Слика 12: Ugib - 16 KE

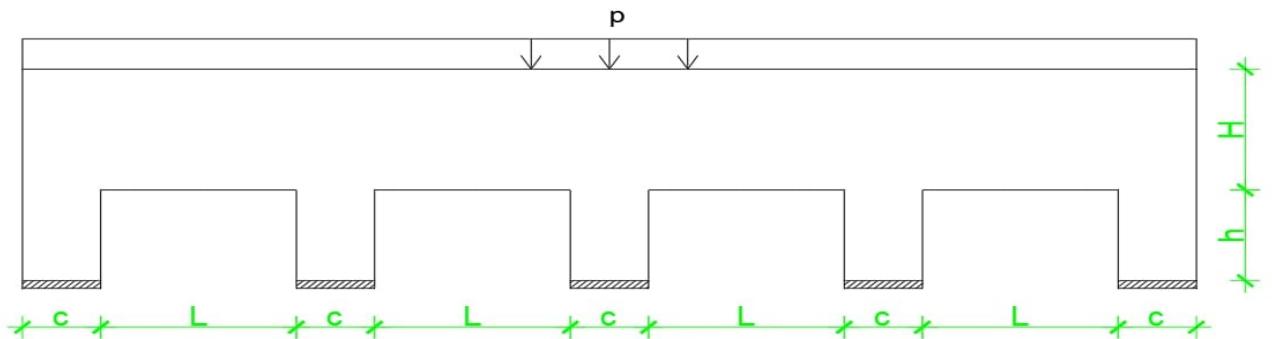
Слика 13: Нарон  $\sigma_x$  - 16 KE

На основу приказаних резултата се може приметити да модел са Q9 коначним елементима има веће деформације од модела из SAP-а који примењује Q4 коначне елементе из чега се може закључити да је матрица крутости Q9 коначног елемента нешто флексибилнија од Q4 коначног елемента. Резултати су врло блиски са аналитичким решењем уколико се примени гушћа мрежа, мрежа са 4 коначна елемента даје приближне вредности напона али далеко непрецизније вредности угиба дуж греде па је потребно прогустити мрежу приликом примене Q9 коначних елемената.

## 4 Трећи задатак

### 4.1 Диспозиција и информације о параметрима

За континуални носач приказан на следећој слици је потребно варирати односе  $H/L$  и  $c/L$  и на основу добијених резултата донети закључке:

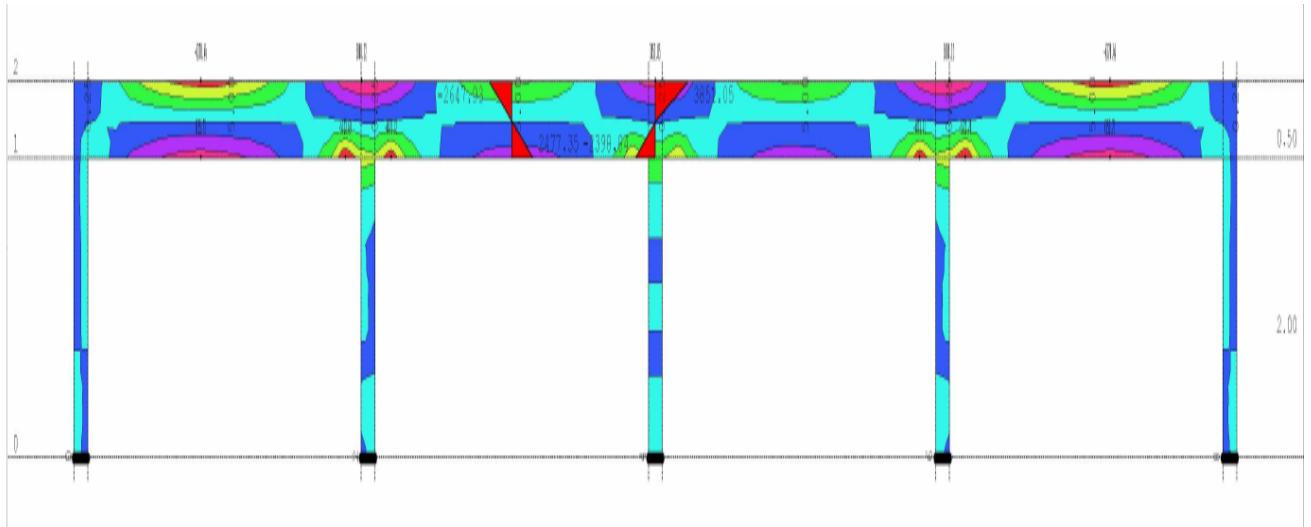
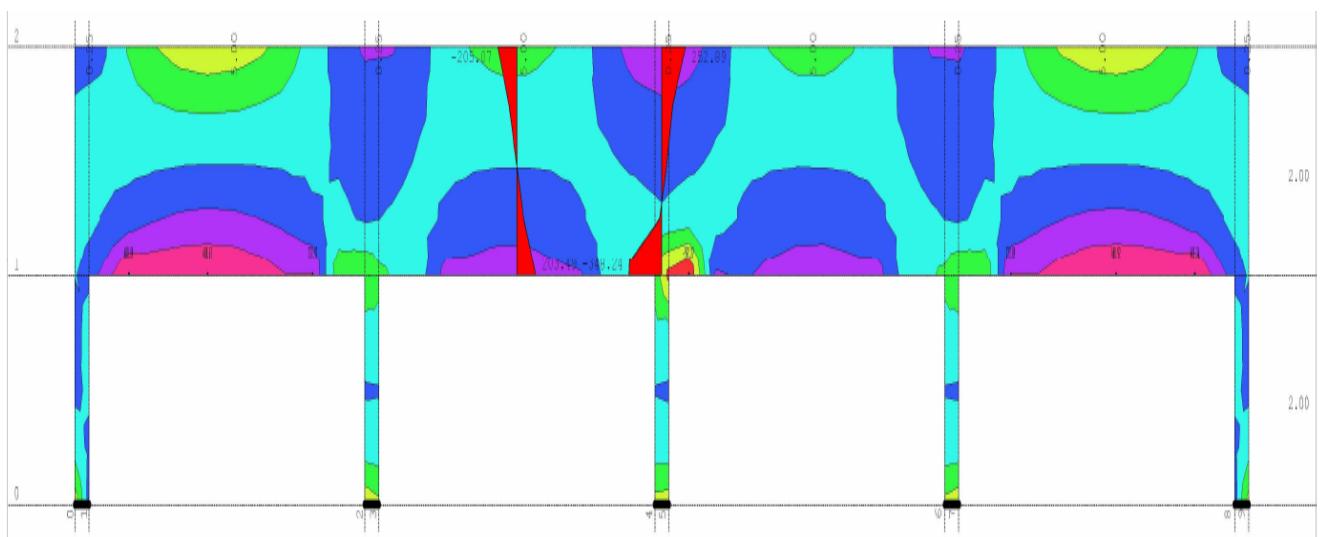
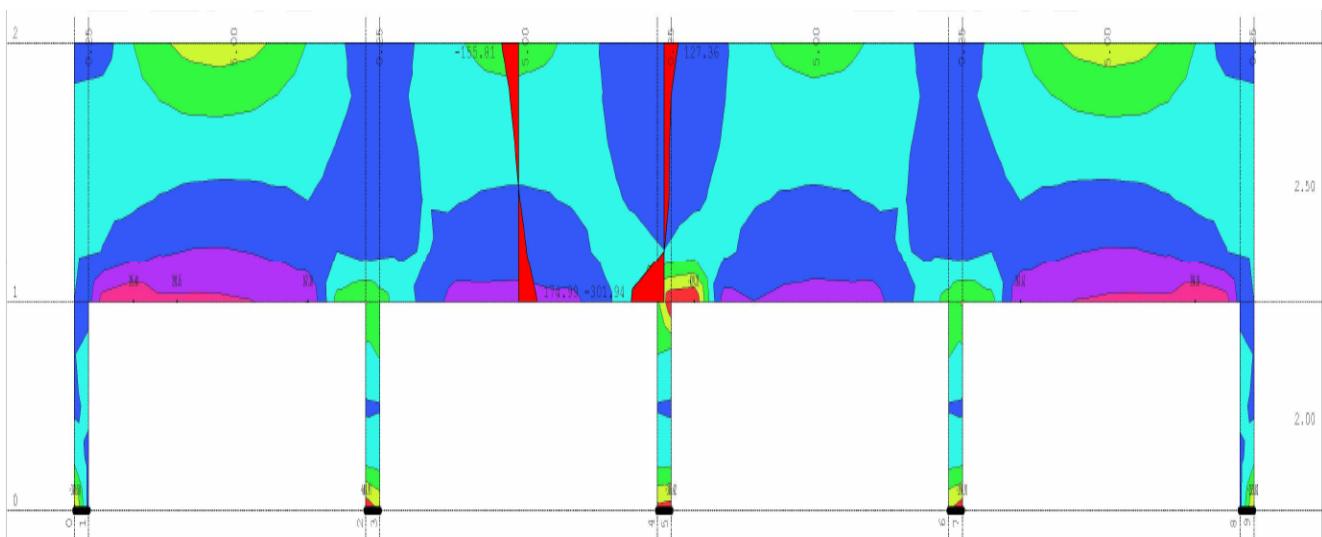


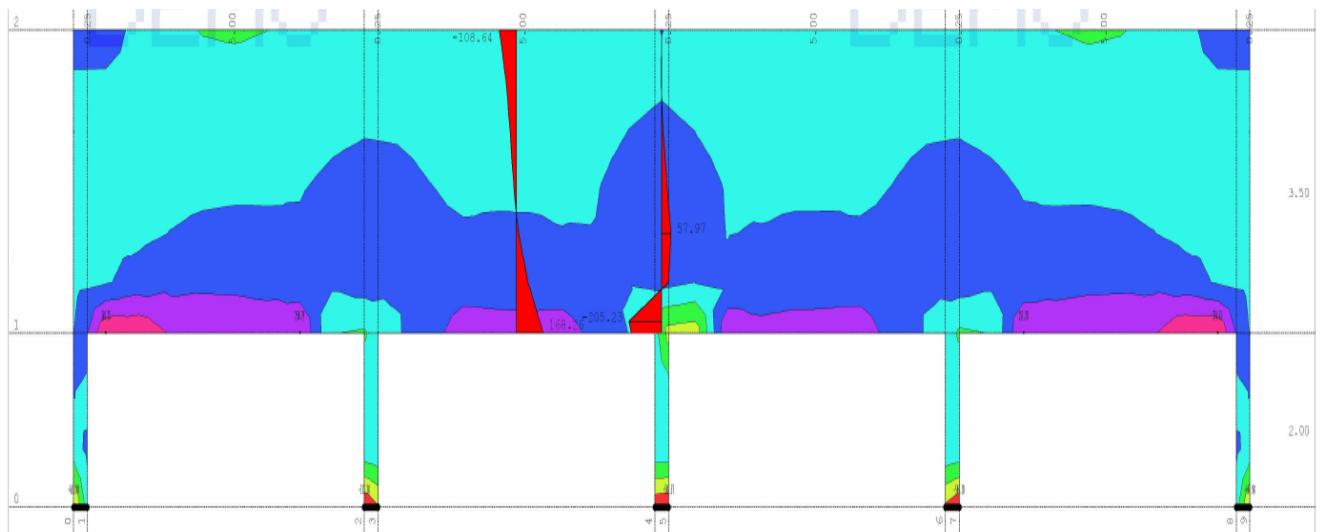
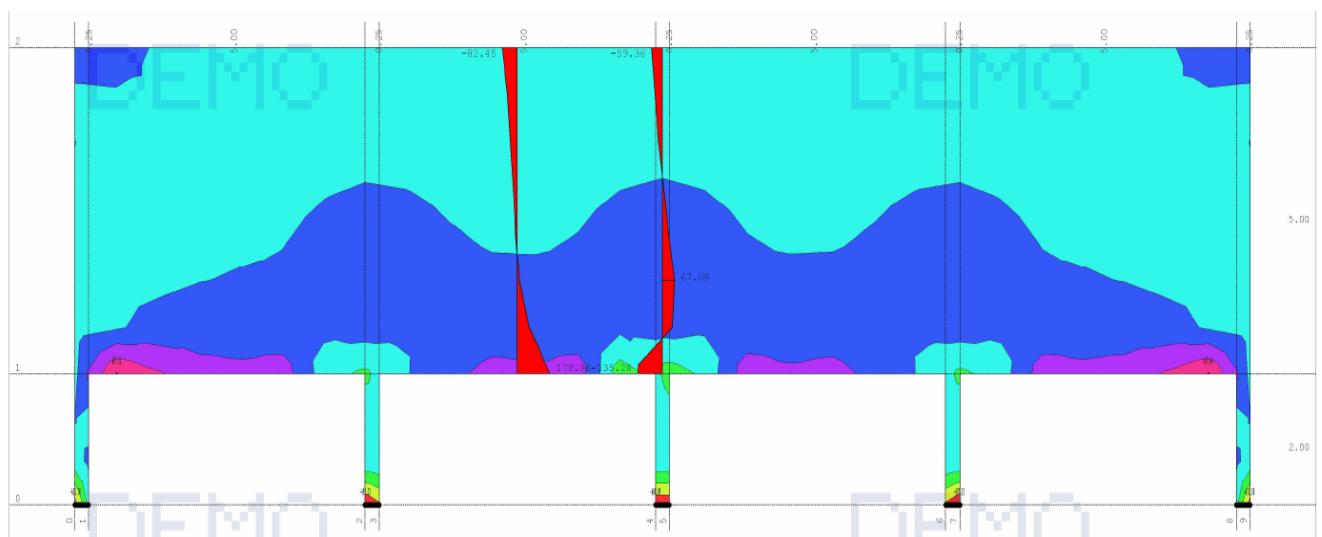
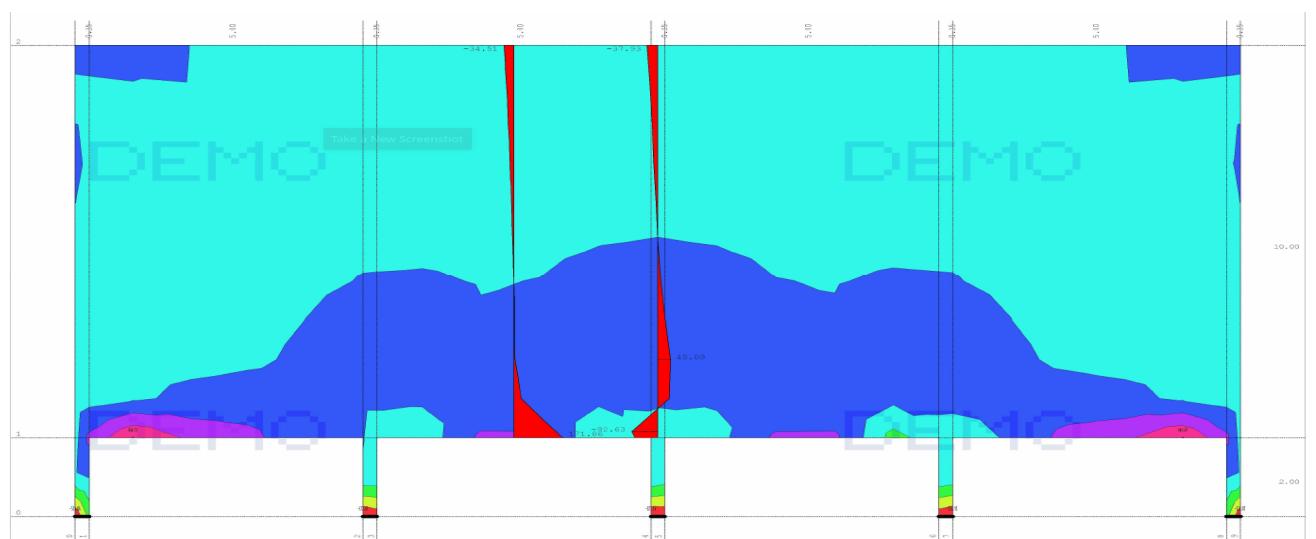
Слика 14: Диспозиција континуалног носача

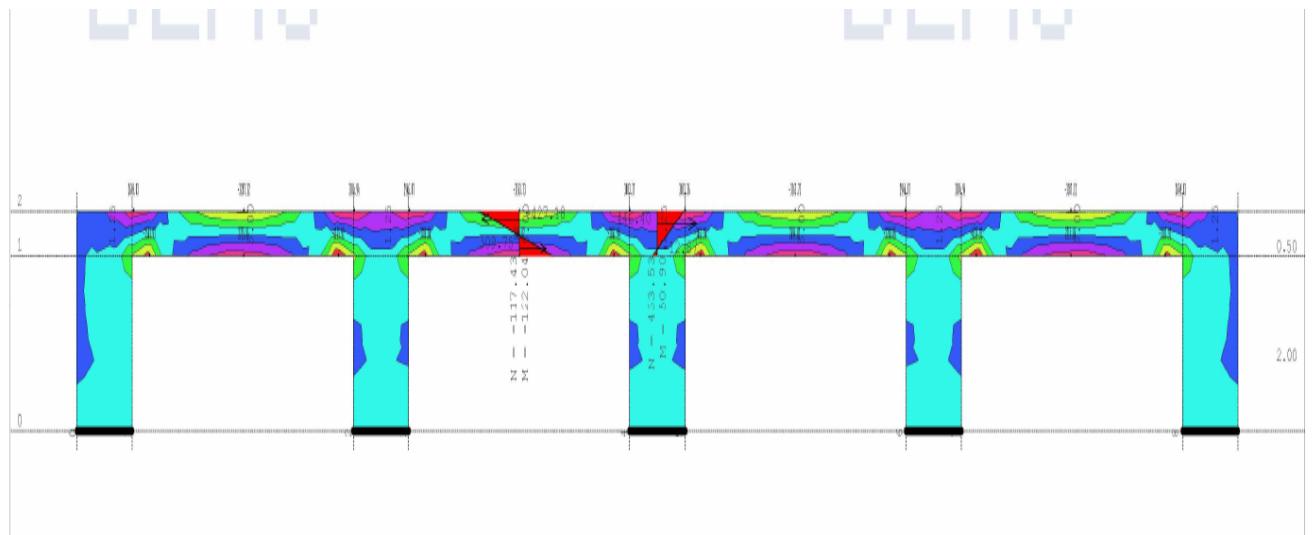
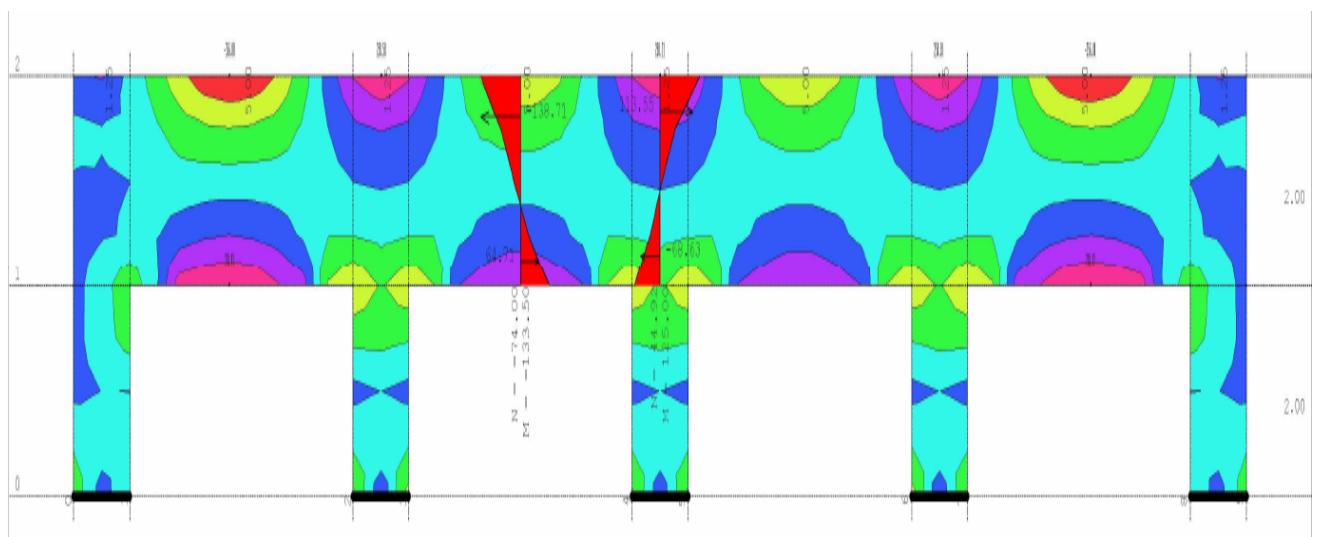
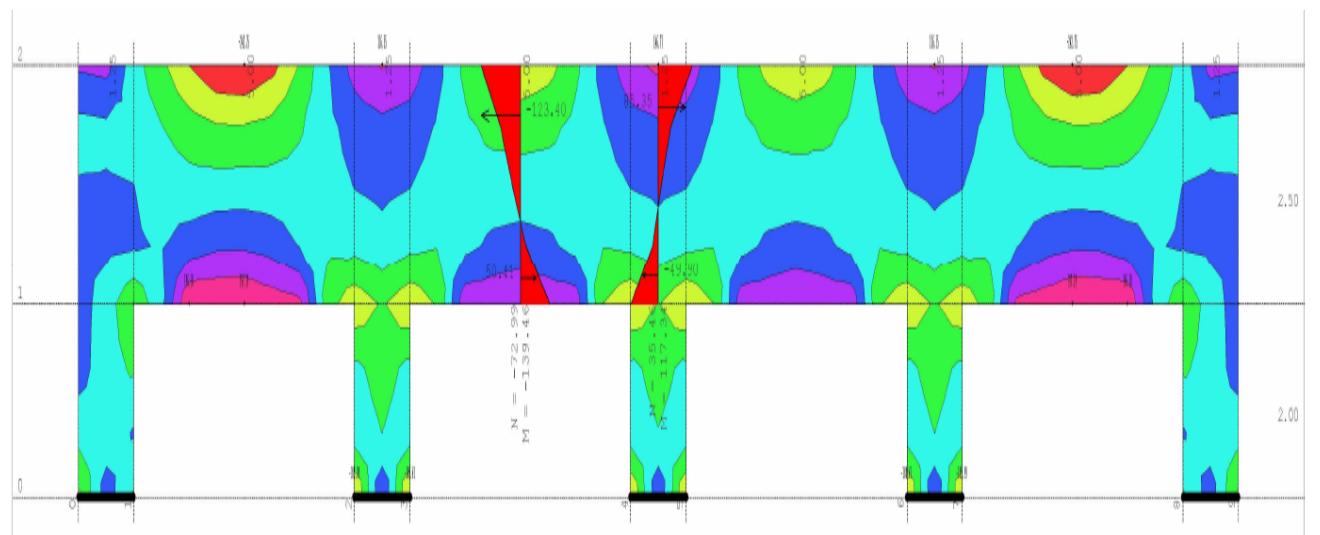
Параметри који су непроменљиви за све моделе су оптерећење  $p = 100kN/m$ , дебљина плоче (греде)  $d_{pl} = 30cm$  и висина стубова  $h = 2m$ .

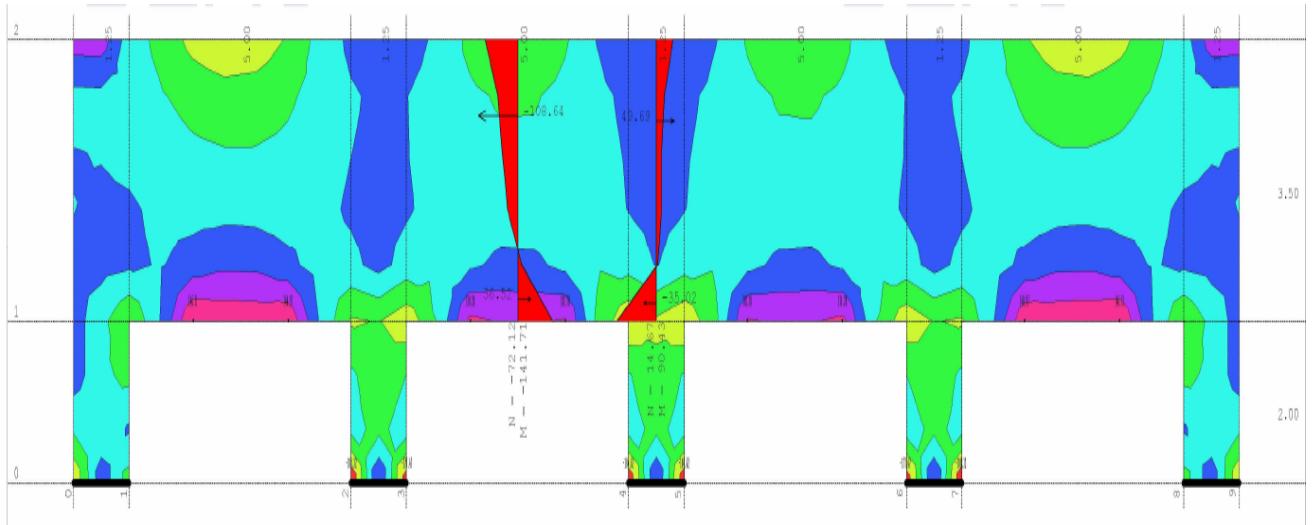
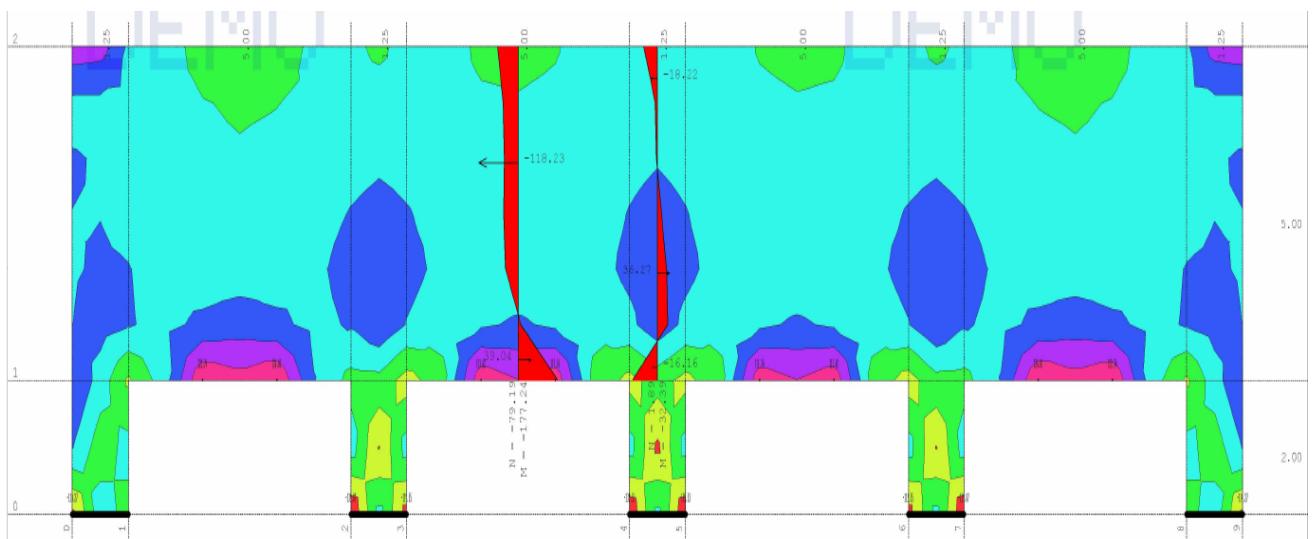
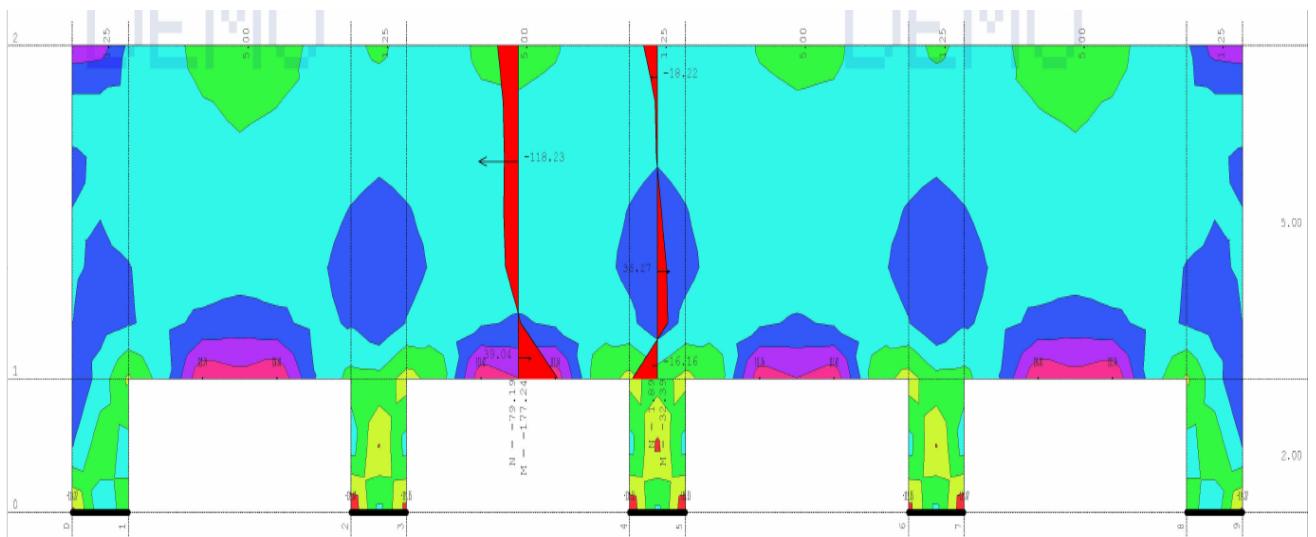
### 4.2 Дијаграми пресечних сила Nx у зависности од односа димензија

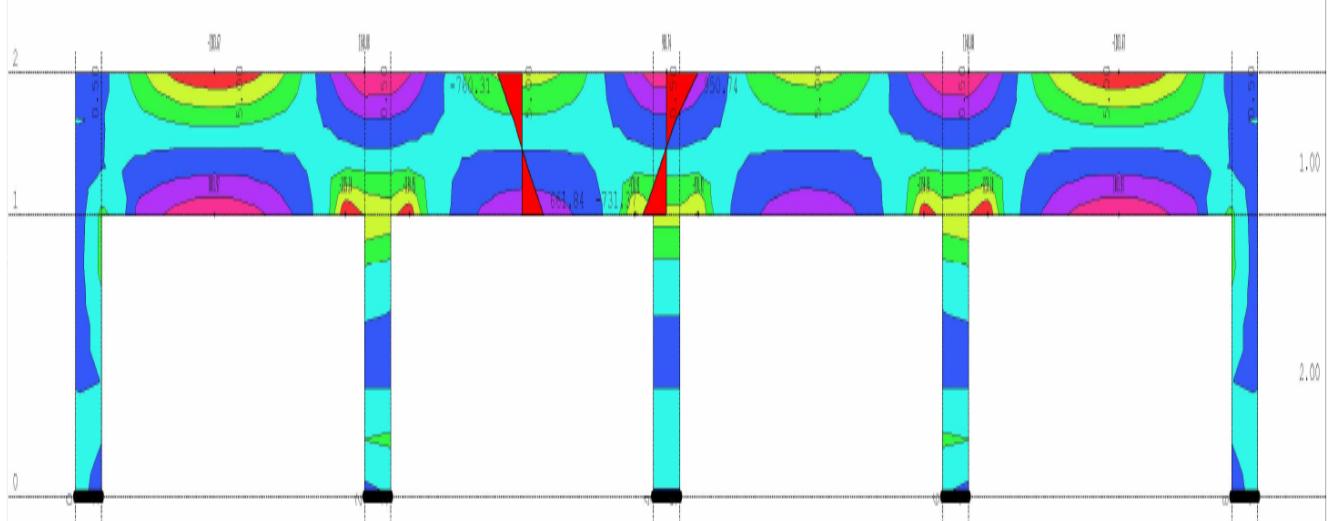
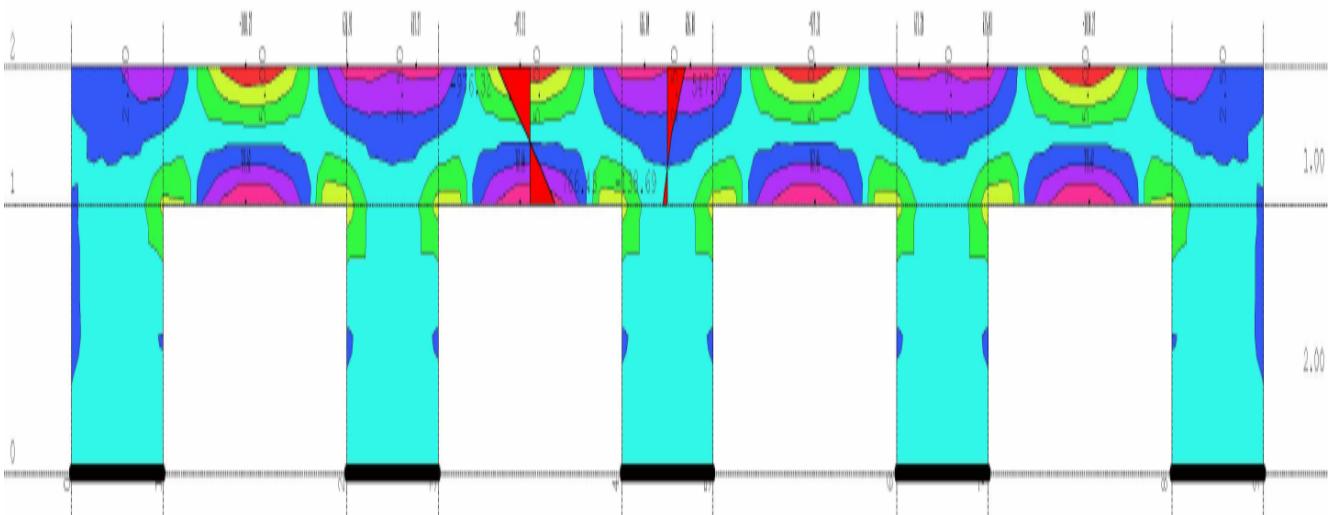
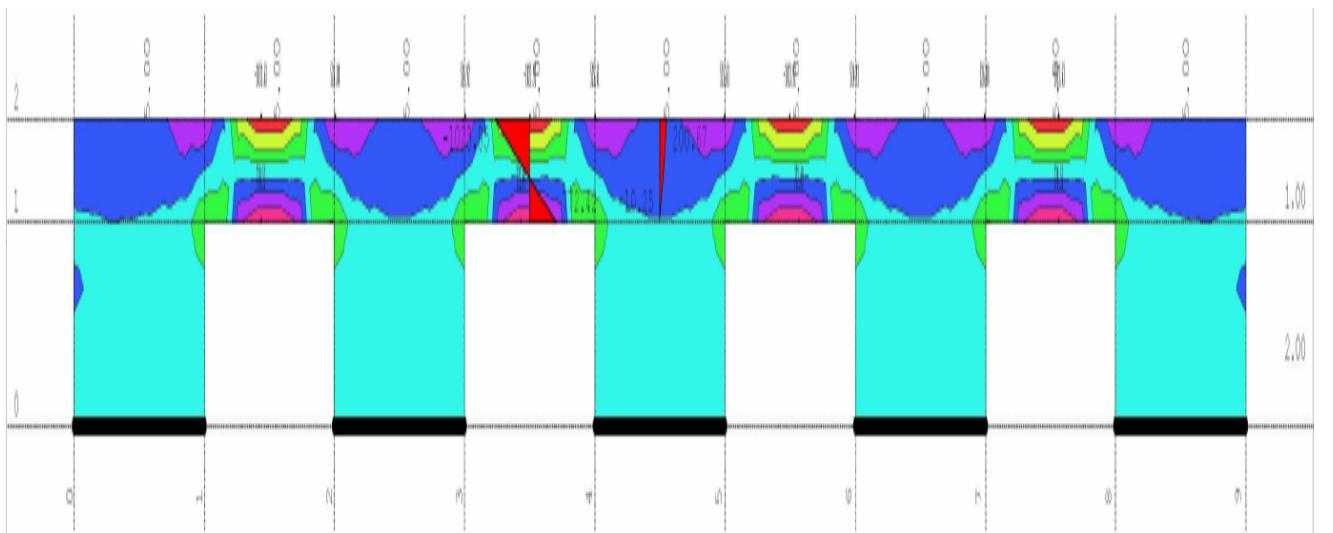
Резултати су добијени у софтверском пакету Tower demo. Варирањем односа  $c/L$  и  $H/L$  су приказани следећи резултати:

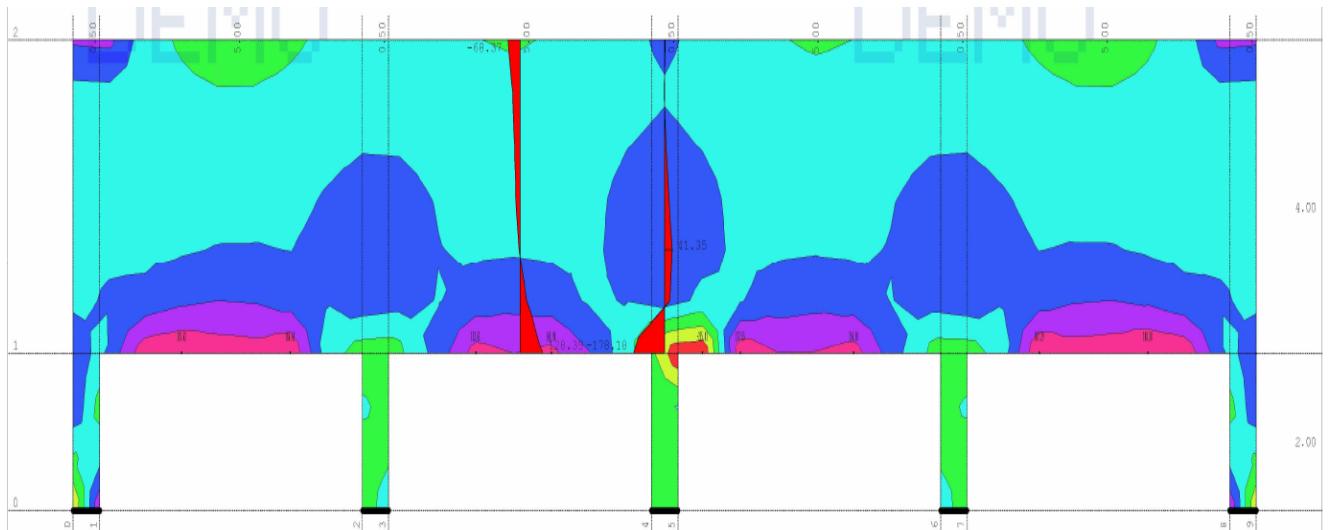
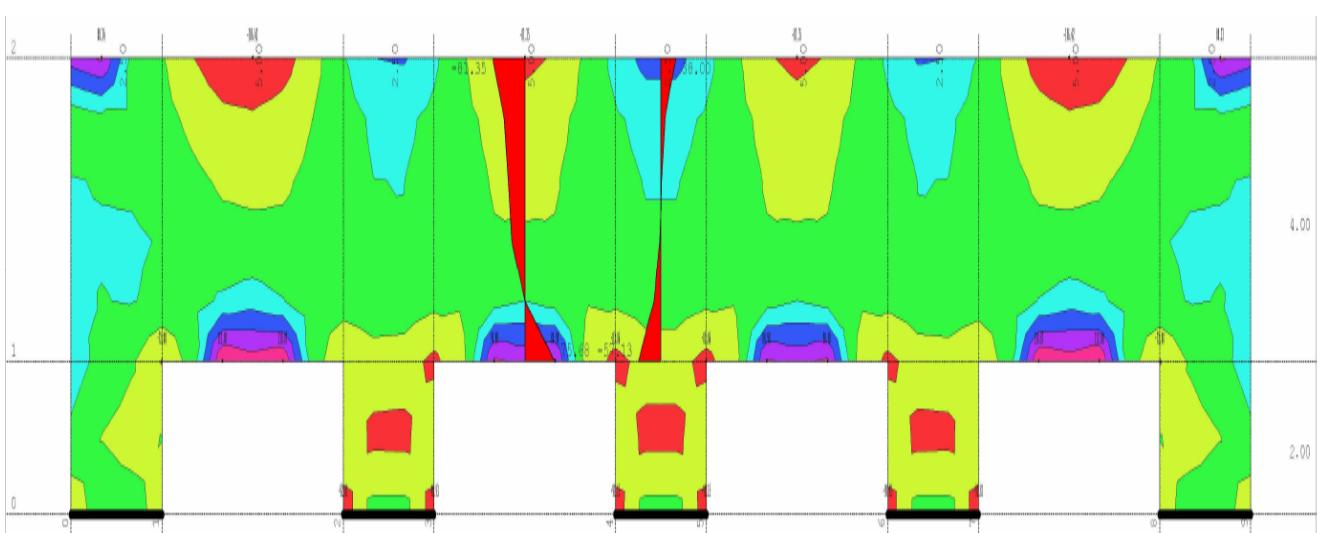
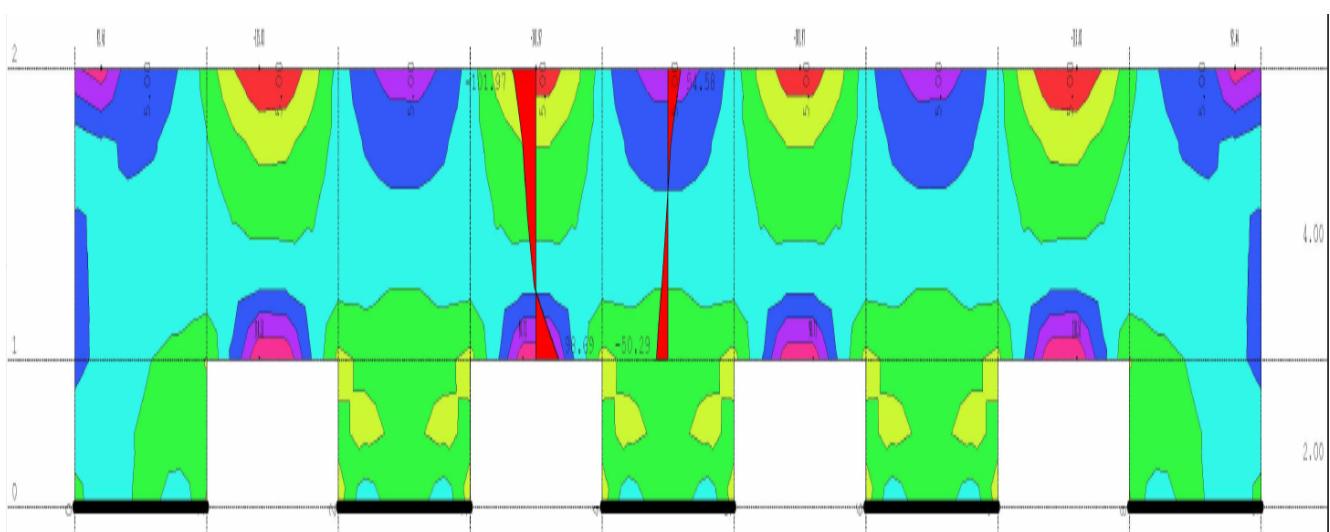
Слика 15:  $c/L=0.05, H/L=0.1$ Слика 16:  $c/L=0.05, H/L=0.4$ Слика 17:  $c/L=0.05, H/L=0.5$

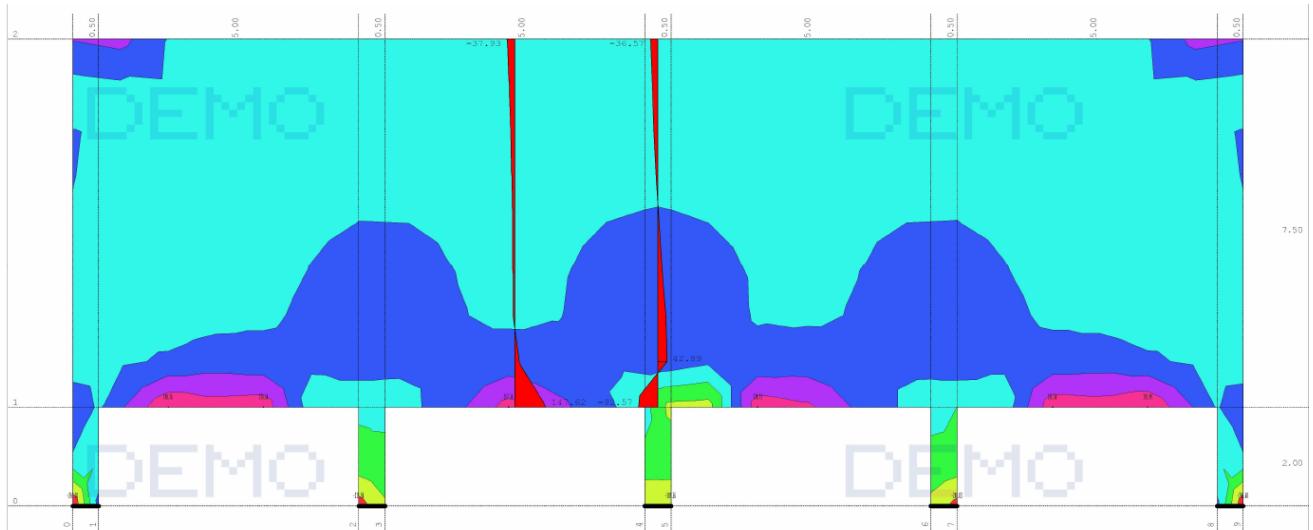
Слика 18:  $c/L=0.05$ ,  $H/L=0.7$ Слика 19:  $c/L=0.05$ ,  $H/L=1$ Слика 20:  $c/L=0.05$ ,  $H/L=2$

Слика 21:  $c/L=0.25$ ,  $H/L=0.1$ Слика 22:  $c/L=0.25$ ,  $H/L=0.4$ Слика 23:  $c/L=0.25$ ,  $H/L=0.5$

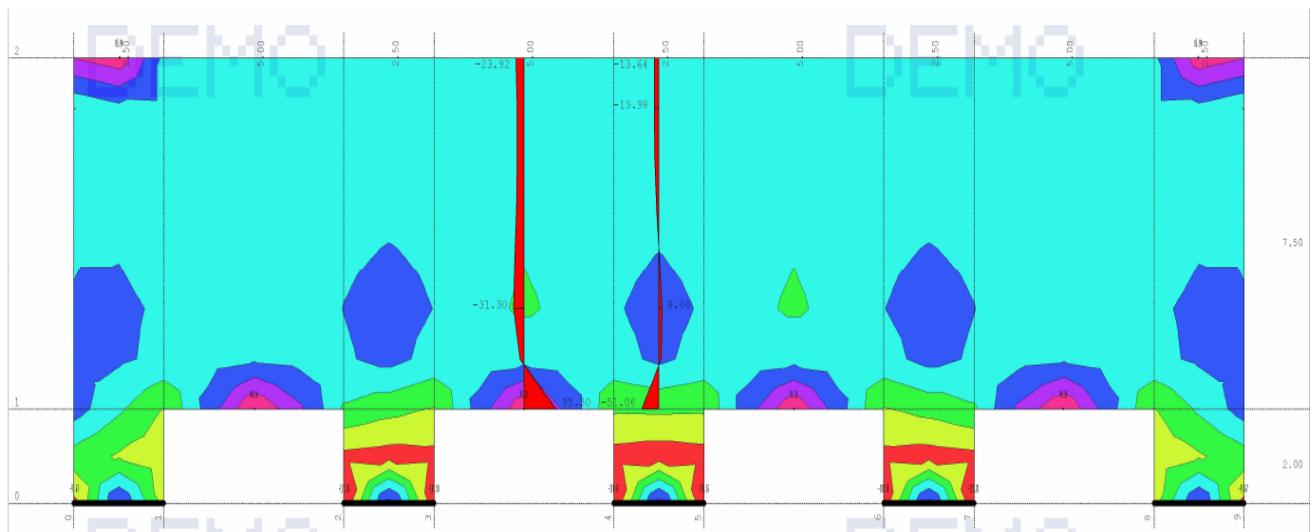
Слика 24:  $c/L=0.25$ ,  $H/L=0.7$ Слика 25:  $c/L=0.25$ ,  $H/L=1$ Слика 26:  $c/L=0.25$ ,  $H/L=2$

Слика 27:  $H/L=0.2$ ,  $c/L=0.1$ Слика 28:  $H/L=0.2$ ,  $c/L=0.5$ Слика 29:  $H/L=0.2$ ,  $c/L=1$

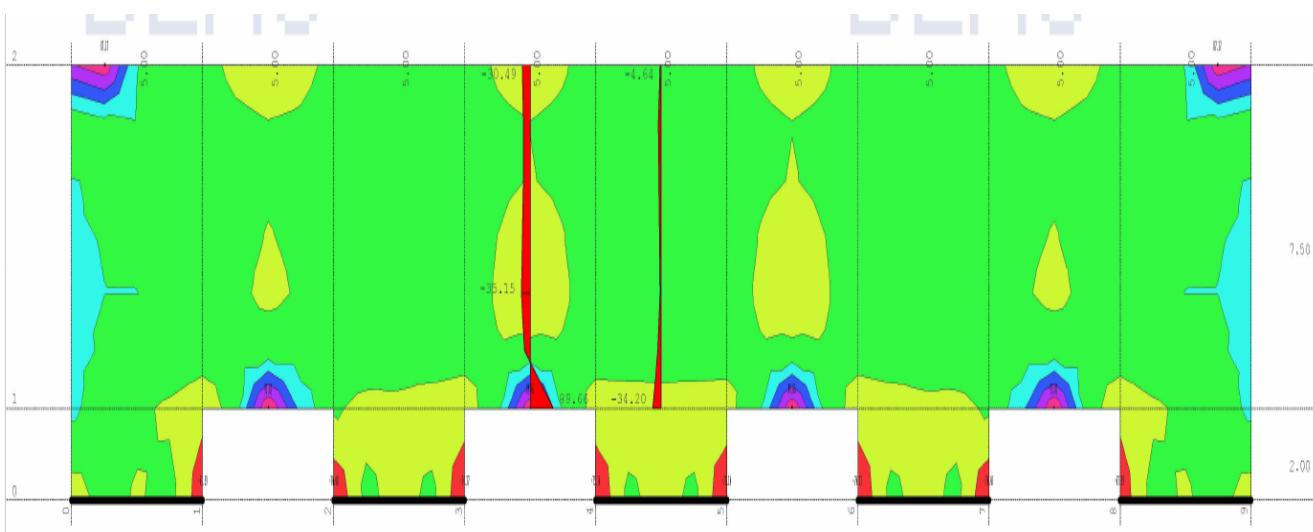
Слика 30:  $H/L=0.8$ ,  $c/L=0.1$ Слика 31:  $H/L=0.8$ ,  $c/L=0.5$ Слика 32:  $H/L=0.8$ ,  $c/L=1$



Слика 33:  $H/L=1.5$ ,  $c/L=0.1$



Слика 34:  $H/L=1.5$ ,  $c/L=0.5$



Слика 35:  $H/L=1.5$ ,  $c/L=1$

## Табеларни приказ пресечних сила у пољу и над ослонцем за горње и доње

влакно са графицима за различите фиксне односе  $c/L$  и варирањем  $H/L$ :

C/L = 0.05				
H/L	Nx поље(средње)		Nx ослонац(средњи)	
	Доње влакно	Горње влакно	Доње влакно	Горње влакно
0.1	2477.35	-2647.93	-2398.84	3851.05
0.2	616.63	-647.96	-775.1	1024.23
0.3	302.99	-319.22	-436.82	476.67
0.4	203.49	-205.07	-348.24	252.89
0.5	174.99	-155.81	-301.94	127.36
0.6	167.03	-124.97	-223.84	49.51
0.7	168.26	-108.64	-205.23	0.55
0.8	171.36	-97.28	-187.18	-30.53
0.9	173.76	-90.31	-179.14	-48.86
1	179.96	-82.45	-135.28	-59.36

Слика 36: Промена пресечних сила  $N_x$  варирањем односа  $H/L$  за фиксни однос  $c/L = 0.05$

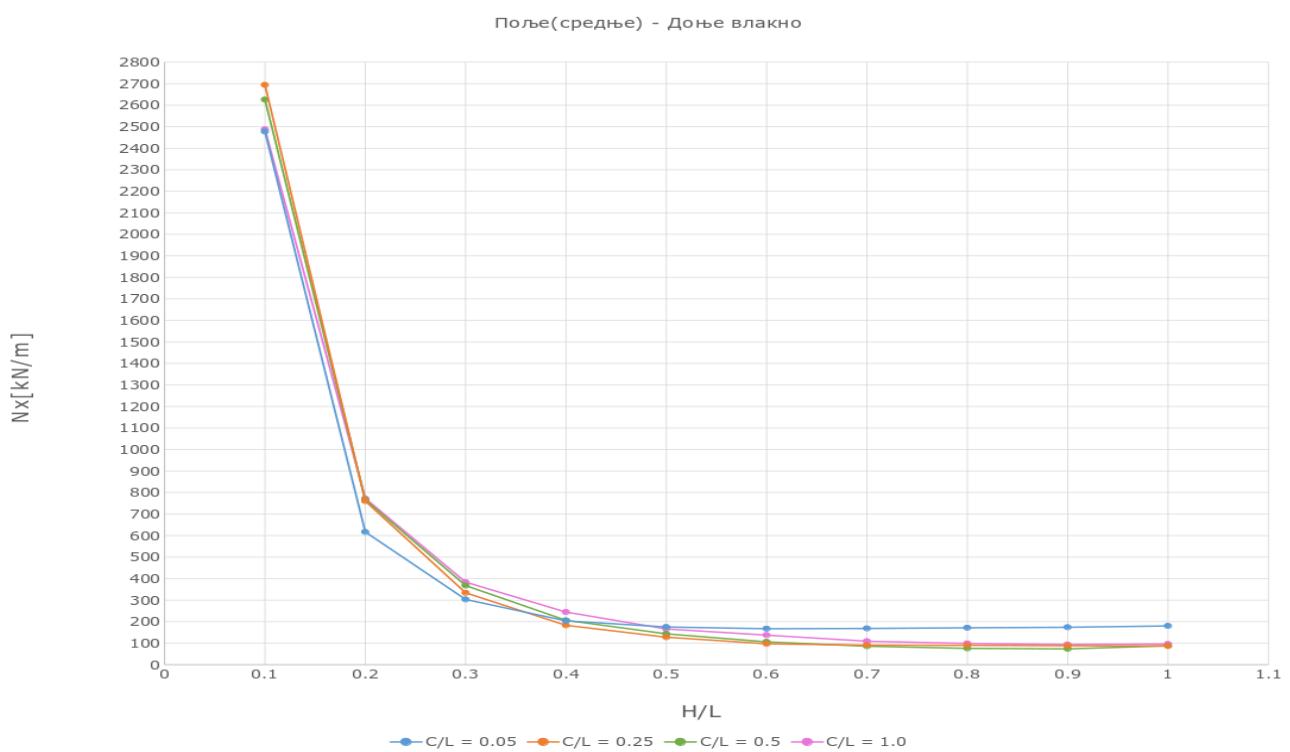
C/L = 0.25				
H/L	Nx поље(средње)		Nx ослонац(средњи)	
	Доње влакно	Горње влакно	Доње влакно	Горње влакно
0.1	2694.02	-3161.73	-354.65	2088.78
0.2	760.09	-951.5	-366.61	899.49
0.3	333.69	-442.5	-216.48	445.72
0.4	182.82	-250.86	-162.26	250.11
0.5	127.55	-170.27	-117.97	146.77
0.6	96.95	-111.79	-104.29	78.55
0.7	90.57	-84.32	-102.73	43.06
0.8	90.12	-66.15	-103.14	18.04
0.9	87.88	-53.58	-99.04	-16.73
1	86.68	-44.71	-55.01	-30.53

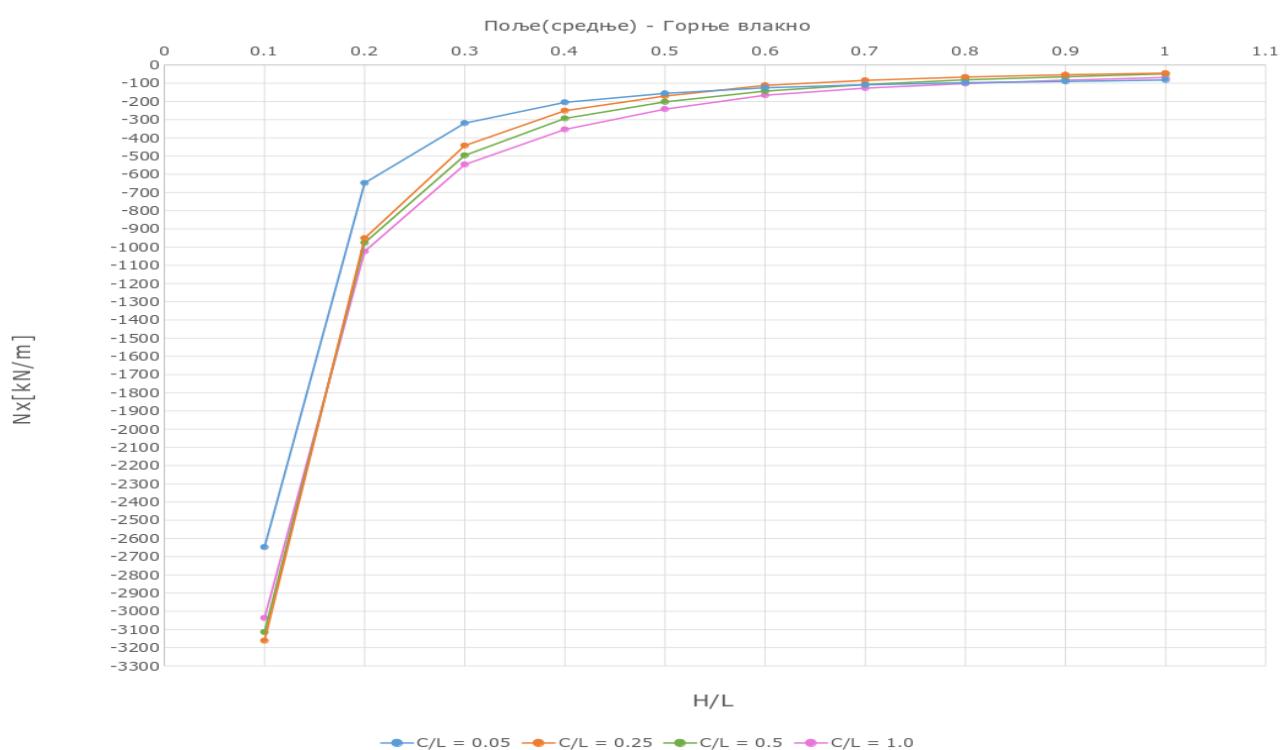
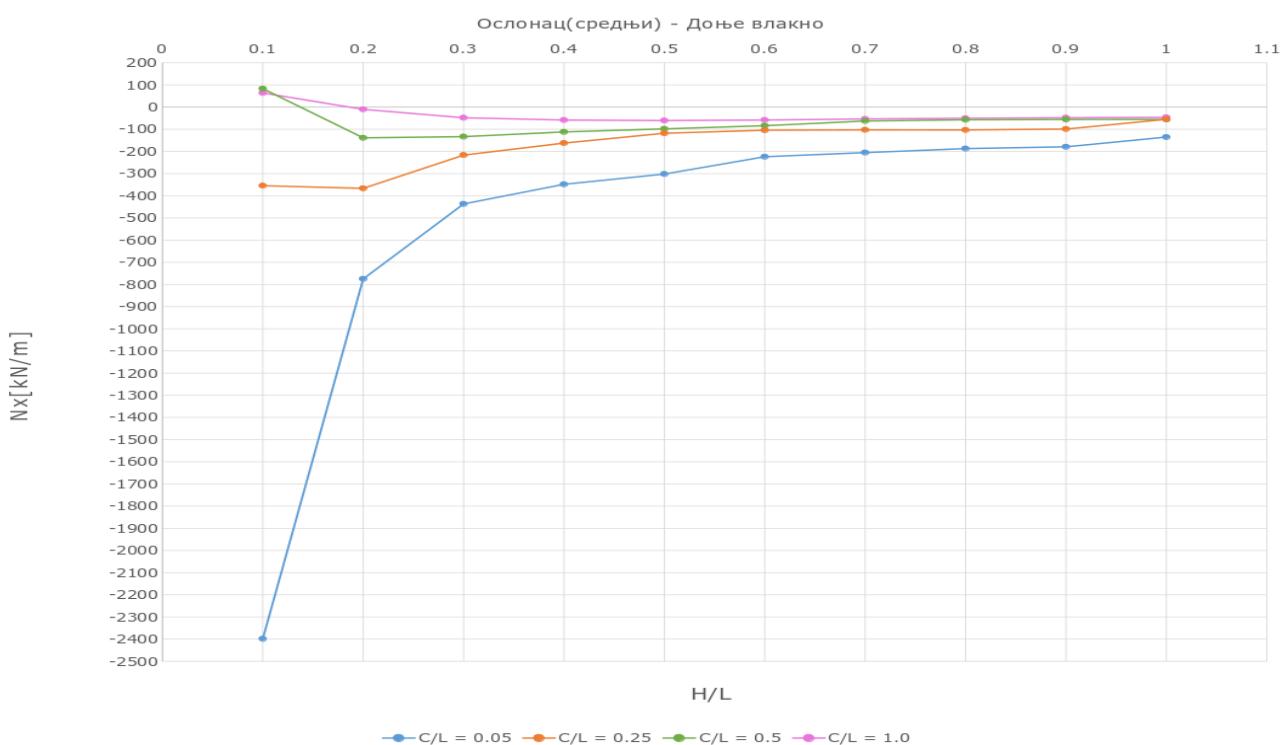
Слика 37: Промена пресечних сила  $N_x$  варирањем односа  $H/L$  за фиксни однос  $c/L = 0.25$

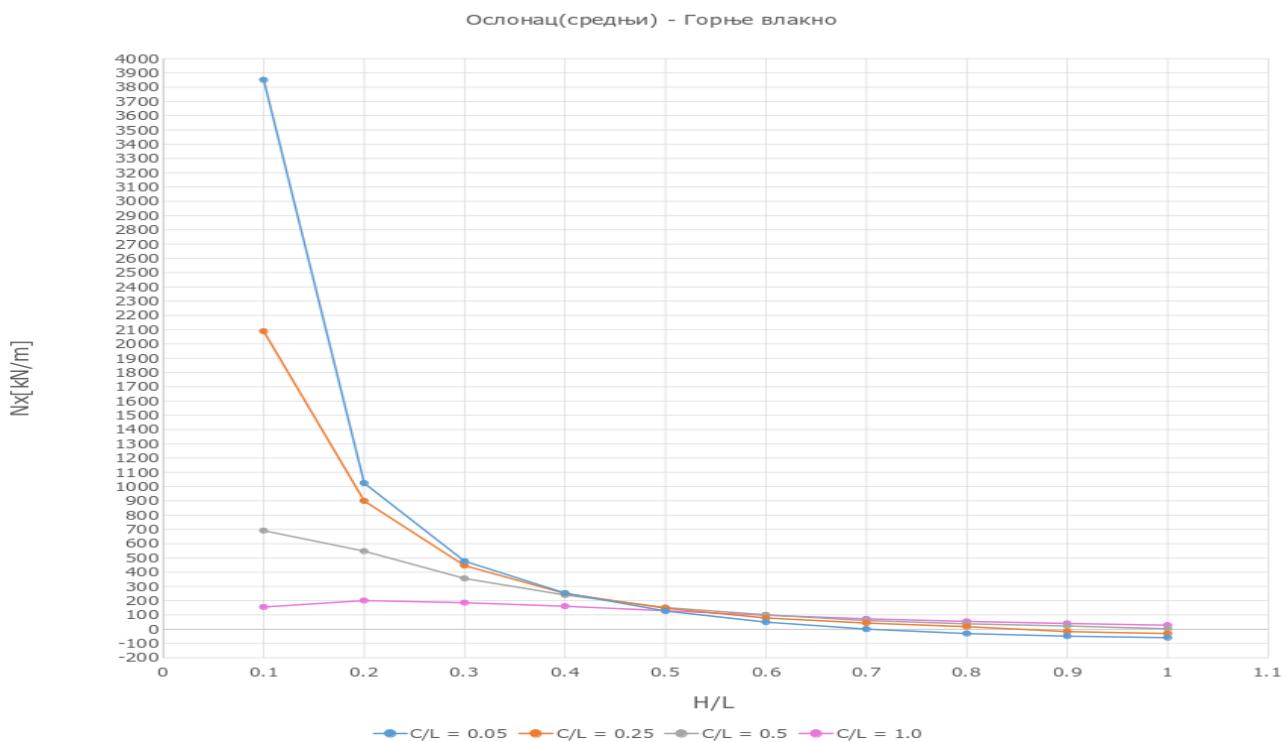
C/L = 0.5				
H/L	Nx поље(средње)		Nx ослонац(средњи)	
	Доње влакно	Горње влакно	Доње влакно	Горње влакно
0.1	2625.7	-3114.62	83.12	690.9
0.2	766.43	-976.32	-138.69	547.09
0.3	367.43	-496.32	-133.3	355.65
0.4	205.91	-293.22	-111.96	239.35
0.5	143.15	-202.34	-98.33	151.82
0.6	105.87	-143.41	-83.97	100.8
0.7	85.62	-106.92	-62.67	60.61
0.8	75.68	-81.35	-57.13	38
0.9	73.3	-64.48	-55.81	22.7
1	87.51	-48.62	-55.52	2.88

Слика 38: Промена пресечних сила  $N_x$  варирањем односа  $H/L$  за фиксни однос  $c/L = 0.5$

C/L = 1.0				
H/L	Nx поље(средње)		Nx ослонац(средњи)	
	Доње влакно	Горње влакно	Доње влакно	Горње влакно
0.1	2487.76	-3037.32	63.65	155.4
0.2	772.42	-1023.85	-10.25	200.67
0.3	383.24	-546.52	-48.17	185.35
0.4	244.38	-353.83	-58.34	160.77
0.5	165.83	-242.21	-60.73	129.28
0.6	137.11	-165.89	-58.21	97.82
0.7	108.84	-126.86	-53.47	72.57
0.8	98.69	-101.97	-50.29	54.58
0.9	93.65	-83.37	-47.98	40.07
1	95.88	-68.47	-46.15	28.09

Слика 39: Промена пресечних сила  $N_x$  варирањем односа  $H/L$  за фиксни однос  $c/L = 1$ Слика 40: График промене пресечних сила  $N_x$  доњег влакна у средњем пољу

Слика 41: График промене пресечних сила  $N_x$  горњег влакна у средњем пољуСлика 42: График промене пресечних сила  $N_x$  доњег влакна над средњим ослонцем



Слика 43: График промене пресечних сила  $N_x$  горњег влакна над средњим ослонцем

### 4.3 Закључак

Због ограничења Tower верзије могућност сагледавања резултата је за мрежу од максимално 300 чвррова коначних елемената.

Пресечне сile у средњем пољу континуалног носача не одступају превише од свог "симетричног" понашања за горње и доње влакно за исту вредност  $c/L$  као ни пресечне сile над ослонцем, међутим повећавањем односа  $c/L$  се вредност пресечне сile  $N_x$  смањује што указује на изражену крутост стубова на савијање који на себе прикупљају утицаје момената па и утицај са тиме опада у два пресека. За мали однос  $c/L$  стубови се понашају као прости штапови па нису у могућности да на себе "навуку" додатне утицаје. На дијаграмима се такође види да повећањем односа  $H/L$  носач почиње да се понаша као зидни носач због расподеле пресечних сила по висини пресека која више није линеарна за разлику од мањих односа  $H/L$  који одговара више гредним носачима.

## 5 Прилози

Упутство за инсталирање Python-а и потребних пакета можете пронаћи на:  
<https://codeberg.org/nikolal/VKMKE>

Кратку документацију са описима рада метода можете пронаћи на: <https://codeberg.org/nikolal/VKMKE/wiki>

Код скрипти можете пронаћи на следећим линковима:

zad1.py: <https://codeberg.org/nikolal/VKMKE/src/branch/master/zad1.py>

zad2.py: <https://codeberg.org/nikolal/VKMKE/src/branch/master/zad2.py>

mapiranjeQ9.py: <https://codeberg.org/nikolal/VKMKE/src/branch/master/mapiranjeQ9.py>

## Литература

- [1] John D. Hunter и сарадници. *matplotlib*. 2003. URL: <https://matplotlib.org/>.
- [2] Wes McKinney и сарадници. *pandas*. 2008. URL: <https://pandas.pydata.org/>.
- [3] Travis Oliphant и сарадници. *NumPy*. 2006. URL: <https://numpy.org/>.
- [4] Eugenio Oñate. *Structural Analysis with the Finite Element Method. Volume 1. Basis and Solids*. Springer, 2009.
- [5] SymPy development team и сарадници. *SymPy*. 2007. URL: <https://www.sympy.org/en/index.html>.
- [6] проф.др. Марија Т. Невовска-Даниловић. *Белешке са предавања*. Грађевински факултет Београд, 2020.
- [7] Биљана Деретић-Стојановић и Шериф Дуница. *Отпорност Материјала*. Грађевински факултет, 2014.