UNIVERZA V MARIBORU
FAKULTETA ZA STROJNIŠTVO

Tadej NOVAK

PROGRAMIRANJE IN ANALIZA RAVNINSKEGA KONČNEGA ELEMENTA ZA DINAMIKO KONSTRUKCIJ

Diplomsko delo
univerzitetnega študijskega programa 1. stopnje
Strojništvo

Maribor, september 2011

Univerza v Mariboru

Fakulteta za strojništvo

PROGRAMIRANJE IN ANALIZA RAVNINSKEGA KONČNEGA ELEMENTA ZA DINAMIKO KONSTRUKCIJ

Diplomsko delo

Študent(ka):	Tadej NOVAK
Študijski program:	Univerzitetni študijski program 1. stopnje Strojništvo
Smer:	Energetsko, procesno in okoljsko strojništvo
Mentor:	izr. prof. dr. Marko Kegl
Somentor:	red. prof. dr. Matjaž Hriberšek

Maribor, september 2011

Vložen original sklepa
o potrjeni temi
diplomskega dela

I Z J A V A

Podpisani Tadej NOVAK izjavljam, da:

- je bilo predloženo diplomsko delo opravljeno samostojno pod mentorstvom izr. prof. dr. Marka Kegla in somentorstvom red. prof. dr. Matjaža Hriberška ;
- predloženo diplomsko delo v celoti ali v delih ni bilo predloženo za pridobitev kakršnekoli izobrazbe na drugi fakulteti ali univerzi;
- soglašam z javno dostopnostjo diplomskega dela v Knjižnici tehniških fakultet Univerze v Mariboru.

Maribor, 26.8.2011 Podpis: _____

ZAHVALA

Zahvaljujem se mentorju izr. prof. dr. Marku Keglu in somentorju red. prof. dr. Matjažu Hriberšku za pomoč in vodenje pri opravljanju diplomskega dela. Zahvaljujem se tudi Mateji.

Posebna zahvala velja staršem, ki so mi omogočili študij. Očetu za vse praktične izkušnje iz tehnike in mami za vso skrb v času mojega šolanja.

PROGRAMIRANJE IN ANALIZA RAVNINSKEGA KONČNEGA ELEMENTA ZA DINAMIKO KONSTRUKCIJ

Ključne besede: konstrukcije, dinamika, končni element, programiranje

UDK: 621.8.02:624.046.4(043.2)

POVZETEK

Diplomsko delo prikazuje način izpeljave, programiranja in numeričnega testiranja kinematično nelinearnega ravninskega končnega elementa (KE) za računanje prehodnega dinamičnega odziva. V ta namen so na kratko predstavljene potrebne teoretične enačbe KE. Temu sledi kodiranje podprogramov končnega elementa z uporabo naprednega orodja AceGen za simbolično računanje in programiranje. Na koncu so podprogrami v FORTRANU vgrajeni v raziskovalni MKE program EMS in testirani z različnimi integracijskimi shemami. Temu sledi kratka predstavitev rezultatov z razpravo.

PROGRAMMING AND ANALYSIS OF A PLANE FINITE ELEMENT FOR STRUCTURAL DYNAMICS

Key words: structural, dynamics, finite element, programming

UDK: 621.8.02:624.046.4(043.2)

ABSTRACT

This diploma work focuses on development, coding and numerical testing of a kinematically nonlinear plane finite element (FE) for dynamic transient response computations. For this purpose the underlying FE equations are discussed briefly. After that the finite element routines are codied by using the advanced symbolic programming tool AceGen. Finally, the FE code in FORTRAN is integrated into the research FEA program EMS and tested with several implicit integrations schemes. The results are briefly presented and discussed.

KAZALO

UPORABLJENI SIMBOLI

u_i	-	vozliščni pomik
kPS	-	število prostorskih stopenj elementa
T	-	število vozlišč elementa
α_i	-	koeficient polinoma
K	-	togostna matrika
F_i	-	notranja vozliščne sila
R_i	-	zunanja vozliščna sila
L	-	Lagrangejeva funkcija
N_i	-	interpolacijska funkcija
Θ	-	aproksimacijska funkcija
I	-	identična matrika
J	-	matrika preslikave
X	-	matrika preslikave
G	-	Jacobijeva matrika
F	-	deformacijski gradient
ε	-	tenzor deformacij
σ_{ii}	-	napetost
E	-	modul elastičnosti
υ	-	Poisonov količnik
V	-	volumen elementa
Π_F	-	elastični potencial
W_k	-	kinetična energija
w	-	Gaussova utež

UPORABLJENE KRATICE

KE - Končni element

MKE - Metoda končnih elementov

C# - C sharp

MPR - Mid point rule

NTR - Newmark trapezoidal rule

BAM - Bossak-alpha method with high frequency dissipation

1 UVOD

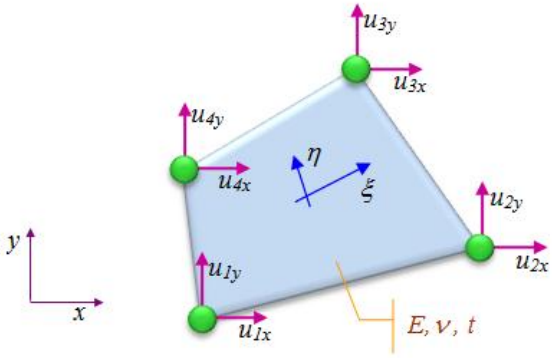
"The mind is sharper and keener in seclusion and uninterrupted solitude. No big laboratory is needed in which to think. Originality thrives in seclusion free of outside influences beating upon us to cripple the creative mind. Be alone, that is the secret of invention; be alone, that is when ideas are born."

Nikola Tesla

1.1 Opis splošnega področja diplomskega dela

Numerične metode so v uporabi že kar nekaj časa. Ena izmed najbolj pomembnih numeričnih metod je metoda končnih elementov (MKE). Preprosto rečeno, metoda končnih elementov pomeni razdelitev obravnavanega območja na majhne elemente, katerih karakteristike so opisane poenostavljeno. V limiti, ko elementi postanejo infinitezimalno majhni, bo postala rešitev zastavljenega problema natančna.

Ključ do pravilne uporabe številnih inženirskih programov je prav detajlno poznavanje končnega elementa. V ta namen je bila napisana pričujoča diplomska naloga, ki zajema preučevanje, programiranje ter numerično testiranje ravninskega 4-vozliščnega končnega elementa za dinamiko konstrukcij. Rdeča nit dela bo prenos ter spreminjanje znanih enačb s področja mehanike končnih elementov v programska jezika FOTRAN ali C#, s pomočjo programskih paketov Mathematica, AceGen ter Microsoft Visual Studio.



Slika 1.1: Ravninski končni element (RNS) s štirimi vozlišči

1.2 Opredelitev diplomskega dela

Diplomsko delo bo med kopico končnih elementov opredeljeno na ravninski 4-vozliščni končni element. Obsegalo bo zapis enačb končnega elementa, programiranje končnega

elementa, vgradnjo elementa v obstoječi MKE program ter študij njegovih lastnosti pri različnih integracijskih shemah.

1.3 Struktura diplomskega dela

Najprej se bomo seznanili z osnovami MKE, nato bomo zapisali teoretične osnove kinematično nelinearnega ravninskega elementa. Naslednje poglavje bo predstavilo vlogo integracijskih shem pri programiranju dinamičnega končnega elementa. V poglavju programiranje v okolju AceGen bo razložen sam potek programiranja končnega elementa (KE). V zadnjem poglavju, imenovanem vgradnja KE v MKE program EMS, pa bomo na konstrukcijskem elementu testirali obravnavan KE.

2 PREGLED STANJA OBRAVNAVANE TEMATIKE

Diplomsko delo obravnava:

a) metodo končnih elementov (MKE)
b) programiranje

a) Metoda končnih elementov je v teoretičnem smislu precej dobro raziskana. Osnovni koraki metode končnih elementov so:

- Korak 1: Diskretizacija računskega območja. Prvi korak je razdeliti področje na končno število poddomen (končnih elementov) preprostih oblik.
- Korak 2: Določitev vozliščnih neznank in interpolacijskih funkcij. Pomike znotraj končnih elementov obravnavanega telesa izrazimo z znanimi interpolacijskimi funkcijami v odvisnosti od pomikov vozlišč.
- Korak 3: Iskanje togostne matrike končnega elementa. Za vsak končni element je potrebno določiti matrike koeficientov, ki predstavljajo togostno karakteristiko končnega elementa. Metoda, ki jo uporabimo za izračun togostne matrike, temelji na minimumu potencialne energije.
- Korak 4: Seštevanje posameznih prispevkov KE. Togostno matriko diskretiziranega telesa dobimo z ustreznim seštevanjem prispevkov posameznih končnih elementov.
- Korak 5: Reševanje sistema enačb. Preden lahko rešimo sistem enačb, moramo določiti še robne pogoje.

b) Programiranje danes poteka še vedno precej ročno, lahko pa si pomagamo z uporabo naprednih orodij (AceGen). Bistvo AceGen je, da iz zelo visokonivojskega zapisa dobimo končno programsko kodo, npr. v FORTRANU, mnoge operacije pa so avtomatične (npr. odvajanje).

3 OSNOVE MKE

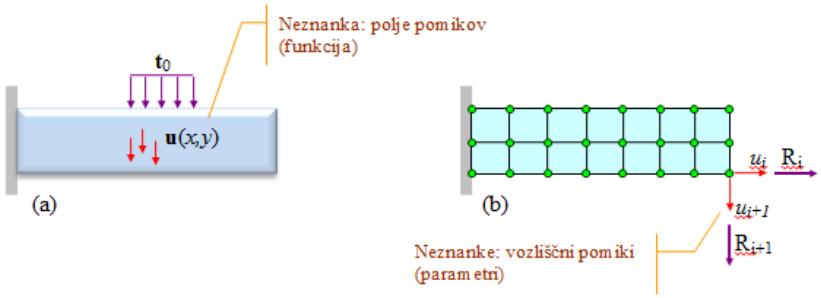
Metoda končnih elementov je metoda reševanje parcialnih diferencialnih enačb z uporabo variacijskih metod (princip virtualnega dela, princip minimuma potencialne energije). Pri reševanju problemov statike in dinamike konstrukcij običajno uporabljamo metodo pomikov, kjer so neznanke pomiki v vozliščih končnih elementov, katere zapišemo v skupnem vektorju pomikov, u, obravnavanega problema. Z uporabo KE funkcijo pomikov $u = u(x, y, z)$, ki ima neskončno število prostorskih stopenj, diskretiziramo v funkcijo s končnim številom prostorskih stopenj. Diskretizacija se izvede lokalno na zelo majhnih, preprostih območjih – končnih elementih. Zaradi diskretizacije lahko reševanje parcialnih diferencialnih enačb prevedemo na reševanje algebraičnih enačb v odvisnosti od notranjih ter zunanjih sil, v določenih točkah končnega elementa, to je vozliščih. Pomike znotraj končnega elementa obravnavanega telesa torej izrazimo z znanimi interpolacijskimi funkcijami v odvisnosti od pomikov vozlišč.

3.1 Statični problemi

Ko imamo problem enkrat diskretiziran, lahko ravnovesne enačbe za statičen primer zapišemo kot

$$F_i - R_i = 0 \ , \ i = 1, \dots, N \ , \tag{3.1}$$

kjer sta F_i in R_i notranja in zunanja vozliščna sila, ki delujeta v smeri vozliščnega pomika u_i, N pa je število vseh prostorskih stopenj (vozliščnih pomikov) telesa (Slika 3.1). Zunanja sila R_i je zunanja obremenitev telesa in je pogosto konstantna (neodvisna od pomikov). Notranja sila F_i pa je izračunana na osnovi napetostnega stanja v telesu. To pomeni, da je odvisna od vozliščnih pomikov u_i.



Slika 3.1:Originalen ravninski problem (a) in njegova diskretizacija (b) z 2x7 končnimi elementi, 3x8 vozlišči ter 3x8x2 prostostnimi stopnjami (neznanimi pomiki)

Izračun notranjih sil največkrat opravimo z odvajanjem elastičnega potenciala Π_F deformirane konstrukcije po pomiku u_i, oziroma

$$F_i = \frac{\partial M_F}{\partial u_i}. \tag{3.2}$$

3.2 Dinamični problemi

Ko imamo problem enkrat diskretiziran, lahko enačbe dinamičnega sistema zapišemo podobno kot pri statiki, t.j.

$$F_i - R_i = 0 \ , \tag{3.3}$$

le da v tem primeru simbol F_i pomeni vstoto vztrajnostnih in notranjih elastičnih sil. Razen tega sta pri dinamiki F_i in R_i odvisni tudi od časa t. Izračun sil F_i lahko opravimo s pomočjo Lagrangeevih enačb, t.j.

$$Fi = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{u}_i}\right) - \frac{\partial L}{\partial u_i}, \tag{3.4}$$

pri čemer smo z L označili tako imenovano Lagrangevo funkcijo.

$$L = W_k - \Pi_F \tag{3.5}$$

Kot izhaja iz zapisa Lagrangejeve funkcije, moramo za reševanje nelinearnih dinamičnih problemov znati zapisati kinetično energijo W_k končnega elementa in elastični potencial Π_F končnega elementa. Prav zapis teh dveh veličin bo naša glavna naloga v nadaljevanju.

4 KINEMATIČNO NELINEAREN RAVNINSKI ELEMENT

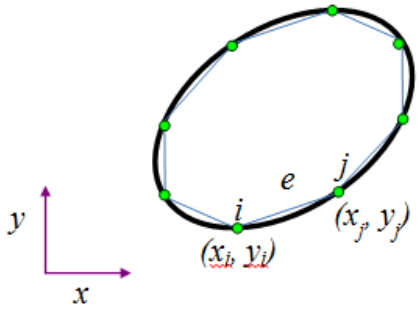
Preden pričnemo programirati končni element, moramo iti skozi njegovo teoretično izpeljavo. Namreč le tako lahko dobro spoznamo veličine, na katerih temelji delovanje končnega elementa.

V nadaljevanju so v logičnem zaporedju prikazani koraki izpeljave štiristranega 4-vozliščnega končnega elementa. Za dobro sledenje enačbam so potrebna osnovna znanja elastoplasto mehanike.

4.1 Zapis interpolacijskih funkcij

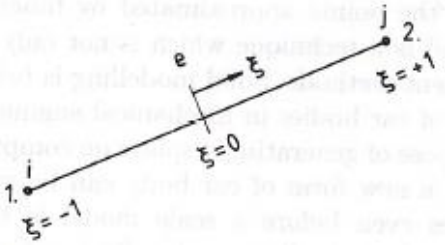
Zapis interpolacijskih funkcij je prvo ključno dejanje. Za lažjo predstavo interpolacijskih funkcij poljubnega štiristranega 4-vozliščnega elementa si najprej poglejmo, kako izgledajo interpolacijske funkcije enodimenzionalnega 2-vozliščnega ter dvodimenzionalnega pravokotnega 4-vozliščnega elementa.

4.1.1 Enodimenzionalen 2-vozliščni element



Slika 4.1: Aproksimacija roba ravninskega lika z linearnimi elementi

Enodimenzionalni končni elementi imajo dva vozlišča, *i* in *j*, definirana s kartezijevimi koordinatami (Slika 4.1). Za vsak element nato v smeri le tega definiramo lokalni koordinatni sistem, ki v sredini doseže vrednost nič, na koncih pa ±1 (Slika 4.2).



Slika 4.2: Linearni element z lokalnim koordinatnim sistemom

Koordinate točke na elementu so določene z:

$$\begin{aligned} x(\xi) &= \frac{x_i + x_j}{2} + \frac{x_j - x_i}{2} * \xi \\ y(\xi) &= \frac{y_i + y_j}{2} + \frac{y_j - y_i}{2} * \xi. \end{aligned} \tag{4.1}$$

Lokalne koordinate je primerno zamenjati z globalnimi

$$\begin{aligned} x_i &= {x_1}^e \\ y_i &= {y_1}^e \end{aligned} \Big\} \text{prvo vozlišče elementa e}$$
$$\begin{aligned} x_j &= {x_2}^e \\ y_j &= {y_2}^e \end{aligned} \Big\} \text{drugo vozlišče elementa e.} \tag{4.2}$$

Na ta način vzpostavimo povezavo med lokalnim in globalnim številčenjem vozlišč. Prvo vozlišče ima globalno številko vozlišča i, drugo pa j.

Ponovno zapišimo enačbo (4.1) kot

$$\begin{aligned} x(\xi) &= \frac{1}{2}(1-\xi){x_1}^e + \frac{1}{2}(1+\xi){x_2}^e \\ y(\xi) &= \tfrac{1}{2}(1-\xi){y_1}^e + \tfrac{1}{2}(1+\xi){y_2}^e, \end{aligned} \tag{4.3}$$

kar lahko zapišemo krajše

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \sum_{n=1}^{T} N_n(\xi) \begin{bmatrix} x_n^e \\ y_n^e \end{bmatrix}, \tag{4.4}$$

kjer je T število vozlišč elementa, N_n pa so interpolacijske funkcije elementa.

Za 2-vozliščni element sta interpolacijski funkciji

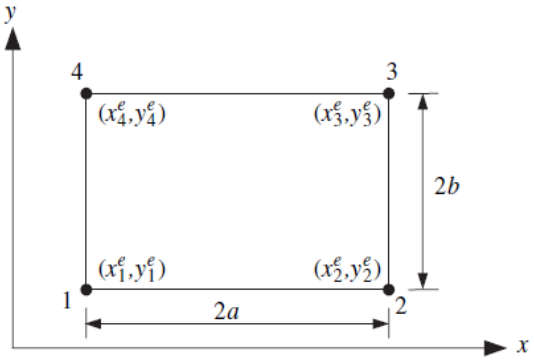
$$\begin{aligned} N_1 &= \frac{1}{2}(1-\xi) \\ N_2 &= \tfrac{1}{2}(1+\xi). \end{aligned} \tag{4.5}$$

4.1.2 Dvodimenzionalen pravokoten 4-vozliščni element

Pred izpeljavo poljubnega štiristranega 4-vozliščnega elementa preučimo še pravokoten 4-vozliščni element. Ker ima element štiri vozlišča, lahko za njegovo aproksimacijo uporabimo polinomom s štirimi parametri. To pomeni, da moramo iz tretje vrste Pascalovega trikotnika izbrati dodaten člen. Potrebujemo samo še en člen, kajti prve tri imamo že iz linearnega polja. Torej moramo izbrati med tremi členi tretje vrste Pascalovega trikotnika. Člen x^2 bi se med vozlišči 1 in 2 ter 3 in 4 obnašal kot kvadratna funkcija, medtem ko bi se člen y^2 med vozlišči 2 in 4 ter 4 in 1 tudi kot kvadratna funkcija. Člen xy bi bil linearen po vseh robovih, dokler sta x in y konstantna po robovih. Člen xy se torej ujema s položajem vozlišč na sliki 4.3. Člen xy je bilinearen člen.

Aproksimacija elementa izgleda sedaj kot

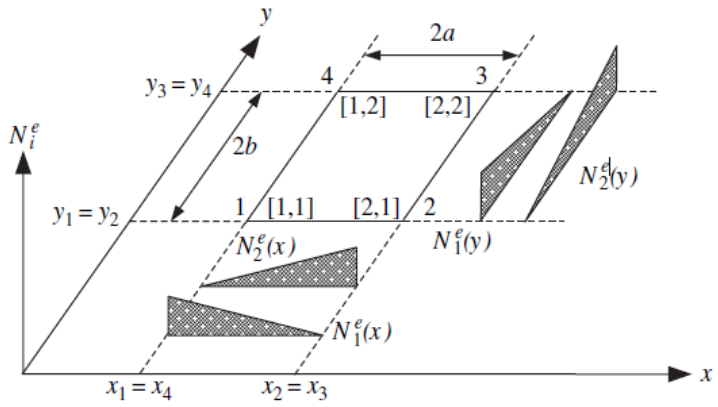
$$\Theta^e(x,y) = \alpha_0^e + \alpha_1^e x + \alpha_2^e y + \alpha_3^e xy\ . \tag{4.6}$$



Slika 4.3: Pravokotni 4-vozliščni element

Izkaže se, da lahko interpolacijske funkcije za pravokoten ravninski element dobimo z množenjem linearnih enodimenzionalnih interpolacijskih funkcij. Na primer, interpolacijsko funkcijo $N_2^e(x,y)$ dobimo z množenjem enodimenzionalnih interpolacijskih funkcij $N_2^e(x)$ in $N_1^e(y)$.

Slika 4.4 prikazuje, da produkt teh dveh interpolacijskih funkcij postane nič v vozliščih 1 in 4, ker gre tam vrednost enodimenzionalne interpolacijske funkcije $N_2^e(x)$ proti nič. Produkt obeh funkcij je nič tudi v vozlišču 3, ker gre tam vrednost interpolacijske funkcije $N_1^e(y)$ proti nič. V vozlišču 2 imata obe interpolacijski funkciji polno vrednost, zato je produkt enak ena.



Slika 4.4: Konstrukcija dvodimenzionalnih interpolacijskih funkcij

Dvodimenzionalne interpolacijske funkcije v tenzorskem zapisu dobimo torej po enačbi (4.7).

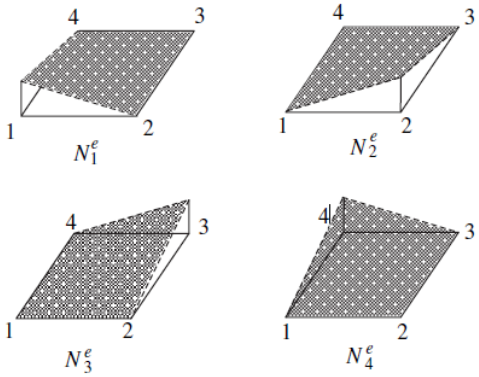
$$N_{[I,J]}^e(x,x) = N_I^e(x) N_J^e(y)\ kjer\ je\ I = 1{,}2\ in\ J = 1{,}2 \tag{4.7}$$

Preglednica 4.1 prikazuje, kako pridemo do interpolacijskih funkcij pravokotnega 4-vozliščnega elementa.

Preglednica 4.1

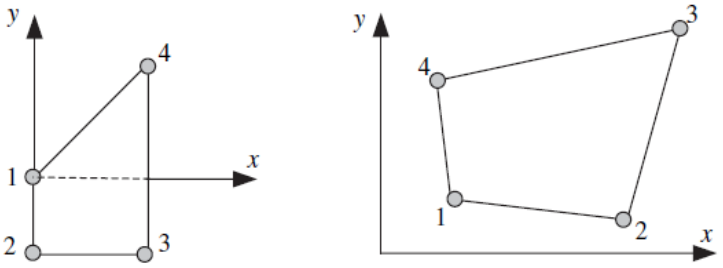
K	I	J	$N_1^e(x)$	$N_2^e(x)$	$N_1^e(y)$	$N_2^e(y)$	2D:$N_K^e(x,y)$
1	1	1	1	0	1	0	$N_1^e(x)N_1^e(y)$
2	2	1	0	1	1	0	$N_2^e(x)N_1^e(y)$
3	2	2	0	1	0	1	$N_2^e(x)N_2^e(y)$
4	1	2	1	0	0	1	$N_1^e(x)N_2^e(y)$

Slika 4.5 prikazuje oblike interpolacijskih funkcij.



Slika 4.5: Grafična podoba interpolacijskih funkcij pravokotnega elementa

Čeprav opisan način deluje za pravokotne elemente, ni primeren za poljuben štiristrani element. To lahko vidimo s preučitvijo štiristranega elementa na sliki 4.6. Obravnavajmo rob, ki povezuje vozlišči 1 in 4, kjer je $y = x$.



Slika 4.6: Štiristrani 4-vozliščni element

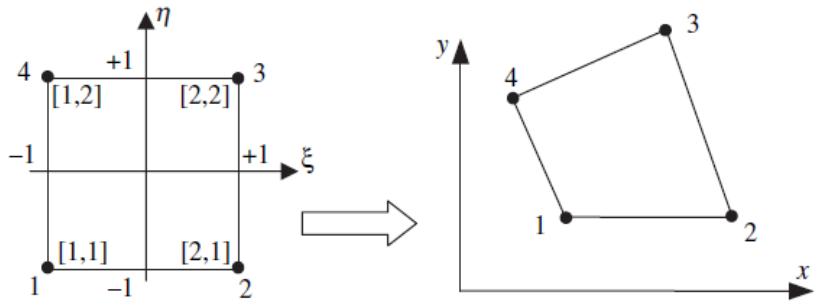
Če $y = x$ vstavimo v enačbo za aproksimacijo, vidimo, da postane aproksimacija kvadratna funkcija po tem robu. To nam da vedeti, da dva vozlišča ne zagotavljata več zveznosti med tem elementom in ostalimi elementi. Zato moramo za konstrukcijo interpolacijske funkcije poljubnega štiristranega elementa uporabiti bolj napredno metodo.

4.1.3 Dvodimenzionalen štiristran 4-vozliščni element

Kot smo do sedaj videli, bi lahko pravokoten 4-vozliščni element opisali z bilinearno funkcijo v odvisnosti od x in y, ampak te interpolacijske funkcije niso linearne po robovih poljubnega štiristranega elementa. Torej dva običajna vozlišča ne zagotavljata zveznosti med elementi. Reševanje te težave je privedlo do enega izmed največjih razvojev na področju končnih elementov, nastanek izoparametričnega končnega elementa. Izometričen koncept nam omogoča oblikovanje končnih elementov s krivimi robovi, ki so zelo uporabni pri modeliranju kompleksnih inženirskih problemov.

Najpomembnejša lastnost izoparametričnih elementov je, da so fizične oz. geometrijske koordinate ter pomiki elementa določene z istimi interpolacijskimi funkcijami kot aproksimacijske funkcije znotraj elementa.

Za izpeljavo štiristranega 4-vozliščnega elementa bomo uporabili kvadrat, prikazan na sliki 4.7. Za zapis interpolacijskih funkcij poljubnega štiristranega 4-vozliščnega elementa, za razliko od enodimenzionalnega elementa, potrebujemo dve lokalni koordinati ζ in η.



Slika 4.7: Preslikava iz lokalnega v globalni koordinatni sistem

Sedaj pridemo od lokalnega elementa do dejanskega elementa preko štirih vozliščnih funkcij

$$x(\xi,\eta) = N^{4Q}(\xi,\eta)x^e \ , \ \ y(\xi,\eta) = N^{4Q}(\xi,\eta)y^e, \tag{4.8}$$

pri čemer $N^{4Q}(\xi,\eta)$ predstavlja interpolacijske funkcije v lokalnem koordinatnem sistemu, x^e in y^e pa sta matriki koordinat vozlišč

$$x^e = [x_1^e x_2^e x_3^e x_4^e]^T \quad , \quad y^e = [y_1^e y_2^e y_3^e y_4^e]^T. \tag{4.9}$$

V enačbi (4.8) smo N^e zamenjali z N^{4Q}, da bi pokazali, da interpolacijske funkcije niso več funkcije koordinat elementa, pač pa so identične za poljuben štiristrani element. Ker je lokalni element kvadrat, so njegove interpolacijske funkcije identične tistim od pravokotnika, le da so izražene z lokalnimi koordinatami. Dobimo jih tako, da zamenjamo x in y z ξ in η ter vozliščne koordinate dejanske domene x_I in x_J z vozliščnimi koordinatami lokalne domene ξ_I in η_J. Rezultat je podan kot

$$N_I^{4Q}(\xi,\eta) = \frac{1}{4}(1+\xi_I\xi)(1+\eta_I\eta)\ , \tag{4.10}$$

kjer sta ξ_I in η_I vozliščni koordinati lokalnega elementa.

Interpolacijske funkcije v enačbi (4.10) vsebujejo konstantni člen, linearna člena ξ in η ter bilinearen člen $\xi\eta$. Tem funkcijam pravimo bilinearne interpolacijske funkcije. Če zapišemo člene še v obliki polinoma, dobimo

$$\Theta^e(\xi,\eta) = \alpha_0^e + \alpha_1^e\xi + \alpha_2^e\eta + \alpha_3^e\xi\eta \tag{4.11}$$

Še enkrat v razviti obliki zapišimo vse štiri interpolacijske funkcije poljubnega štiristranega 4-vozliščnega elementa.

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\ N_1 &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_1 &= \frac{1}{4}(1+\xi)(1+\eta) \\ N_1 &= \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{4.12}$$

Z uporabo teh interpolacijskih funkcij lahko koordinati poljubne točke elementa zapišemo kot

$$x = \sum_{i=1}^{4} N_i x_i \quad , \quad y = \sum_{i=1}^{4} N_i y_i. \tag{4.13}$$

Na enak način interpoliramo tudi pomike v polju elementa. Ti se torej zapišejo kot

$$u_x = \sum_{i=1}^{4} N_i u_x \quad , \quad u_y = \sum_{i=1}^{4} N_i u_y. \tag{4.14}$$

4.2 Zapis deformacij

Pri ravninskem napetostnem stanju, ki ga bomo imeli v našem KE, so za računanje zanimive le deformacije $\varepsilon_{xx}, \varepsilon_{yy}$ in ε_{xy}. Ker imamo pomike izražene v odvisnosti od ξ in η, je pri izračunu deformacij to seveda treba upoštevati. Koordinate ξ in η ter x in y so med seboj povezane preko interpolacijske geometrije. V matrični obliki lahko te relacije izrazimo kot

$$\mathbf{X} = \mathbf{G}\mathbf{J}\ , \tag{4.15}$$

kjer so matrike **X**, **G** in **J** enake

$$\mathbf{X} = \begin{bmatrix} \frac{\partial u_x}{\partial \xi} & \frac{\partial u_x}{\partial \eta} \\ \frac{\partial u_y}{\partial \xi} & \frac{\partial u_y}{\partial \eta} \end{bmatrix}\ ,\ \mathbf{G} = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix}\ ,\ \mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}. \tag{4.16}$$

Matriko **J** imenujemo Jacobijeva matrika preslikave med x in y ter ξ in η. Če upoštevamo zgoraj vpeljane interpolacije geometrije in pomikov, vidimo, da sta matriki **X** in **J** zelo enostavno izračunljivi. Torej ni težko izračunati matrike

$$\mathbf{G} = \mathbf{X}\mathbf{J}^{-1} \tag{4.17}$$

in z njo izraziti tenzor deformacij kot

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\mathbf{G} + \mathbf{G}^{\mathrm{T}} + \mathbf{G}^{\mathrm{T}}\mathbf{G}\right). \tag{4.18}$$

Tenzor deformacij lahko zapišemo tudi v odvisnosti od deformacijskega gradienta **F**

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\mathbf{F}^{\mathrm{T}}\mathbf{F} - \mathbf{I}\right), \tag{4.19}$$

pri čemer velja

$$\mathbf{F} = \mathbf{G} + \mathbf{I} \tag{4.20}$$

in je **I** identična matrika.

4.3 Zapis napetosti

Uporabimo Hookov zakon za ravninsko napetostno stanje, od koder sledi

$$\sigma_{xx} = C\varepsilon_{xx} + C\nu\varepsilon_{yy}\ ,\ \sigma_{yy} = C\varepsilon_{yy} + C\nu\varepsilon_{xx}\ ,\ \sigma_{xy} = 2\mu\varepsilon_{xy}, \tag{4.21}$$

kjer je

$$C = \frac{E}{1-\nu^2}\ , \qquad \mu = \frac{E}{2(1+\nu)} \tag{4.22}$$

4.4 Zapis elastičnega potenciala

Ker imamo ravninsko napetostno stanje, je potencialna energija elementa enaka

$$\Pi_{\mathrm{F}} = \frac{1}{2}\int_{\mathrm{V}} \left(\sigma_{\mathrm{xx}}\varepsilon_{\mathrm{xx}} + \sigma_{\mathrm{xy}}\varepsilon_{\mathrm{xy}} + \sigma_{\mathrm{yx}}\varepsilon_{\mathrm{yx}} + \sigma_{\mathrm{yy}}\varepsilon_{\mathrm{yy}}\right)\mathrm{dV}\ . \tag{4.23}$$

Če upoštevamo, da je za naš element $dV = \psi\, dxdy = U\psi\, d\xi d\eta$, kjer smo z U označili determinanto Jacobijeve matrike **J**, s ψ pa debelino elementa, dobimo

$$\Pi_{\mathrm{F}} = \frac{1}{2}\int_{-1}^{+1}\int_{-1}^{+1}\left(\sigma_{\mathrm{xx}}\varepsilon_{\mathrm{xx}} + \sigma_{\mathrm{xy}}\varepsilon_{\mathrm{xy}} + \sigma_{\mathrm{yx}}\varepsilon_{\mathrm{yx}} + \sigma_{\mathrm{yy}}\varepsilon_{\mathrm{yy}}\right)\mathrm{U}\psi\ \mathrm{d}\xi\mathrm{d}\eta\ . \tag{4.24}$$

Izraz v zgornjem integralu sicer poznamo, vendar je preveč zapleten, da bi integracijo izvedli analitično. Zaradi tega je običajna praksa ta, da integracijo izvedemo numerično. V ta namen je koristno zapisati potencial v nekoliko drugačni obliki, in sicer kot

$$\Pi_{\mathrm{F}} = \int_{-1}^{+1}\int_{-1}^{+1} \Gamma\mathrm{d}\xi\mathrm{d}\eta\ , \tag{4.25}$$

kjer je

$$\Gamma = \frac{1}{2}\left(\sigma_{\mathrm{xx}}\varepsilon_{\mathrm{xx}} + \sigma_{\mathrm{xy}}\varepsilon_{\mathrm{xy}} + \sigma_{\mathrm{yx}}\varepsilon_{\mathrm{yx}} + \sigma_{\mathrm{yy}}\varepsilon_{\mathrm{yy}}\right)\mathrm{U}\psi\ . \tag{4.26}$$

Tako obliko zapisa lahko direktno uporabimo v Gaussovi integracijski shemi.

4.5 Izračun notranjih sil in togostne matrike pri statiki

Posamezne člene togostne matrike dobimo z dvakratnim odvajanjem potenciala po prostorskih stopnjah, in sicer

$$K_{e,1x1x} = \frac{\partial^2 \Pi_F}{\partial u_{1x} \partial u_{1x}} = \int_{-1}^{+1} \int_{-1}^{+1} \frac{\partial^2 \Gamma}{\partial u_{1x} \partial u_{1x}} d\xi d\eta = \int_{-1}^{+1} \int_{-1}^{+1} k_{e,1x1x} d\xi d\eta \,, \tag{4.27}$$

kjer smo vpeljali oznako

$$k_{e,1x1x} = \frac{\partial^2 \Gamma}{\partial u_{1x} \partial u_{1x}} \,. \tag{4.28}$$

Enako velja za ostalih sedem prostorskih stopenj.

4.6 Integracija štiristranega elementa – Gaussova kvadratura

Kot smo že dejali, integracijo po elementu izvedemo numerično. V ta namen uporabimo Gaussovo kvadraturo. V tem primeru lahko integracijo členov togostne matrike izvedemo kot

$$K_{e,1x1x} = \sum_{n=1}^{N} \sum_{m=1}^{M} w^{(n,m)} k_{e1x1x}^{(n,m)} \,, \tag{4.29}$$

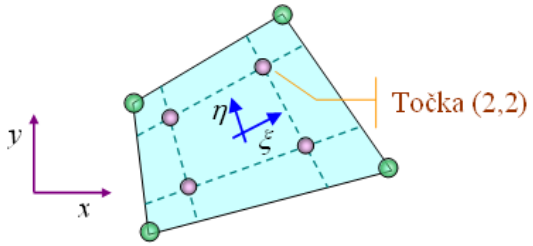
kjer N in M označujeta število Gaussovih integracijskih točk v smereh ξ in η, $w^{(n,m)}$ je Gaussova utež (konstanta) v integracijski točki (n,m), $k_{e,1x1x}^{(n,m)}$ pa je vrednost $k_{e,1x1x}$, izračunana v integracijski točki (n,m).

Za obravnavan element vzamemo običajno $2x2 = 4$ integracijskih točk $(N = M = 2)$. Za ta primer so Gaussove uteži in približne koordinate Gaussovih točk podane v preglednici 4.2.

Preglednica 4.2

Točka	Utež w	Koordinata ξ	Koordinata η
(1,1)	1.0	-0.57735	-0.57735
(2,1)	1.0	+0.57735	-0.57735
(1,2)	1.0	-0.57735	+0.57735
(2,2)	1.0	+0.57735	+0.57735

Približna lega teh Gaussovih točk je prikazana na sliki 4.8.



Slika 4.8: Gaussove integracijske točke (2x2) ravninskega končnega elementa

5 ČASOVNE INTEGRACIJSKE SHEME

Za zapis dinamične enačbe končnega elementa moramo nujno poznati pomen časovnih integracijskih shem. Naš končni element bo napisan za poljubno integracijsko shemo. Poskusimo razložiti pomen integracijskih shem na primeru linearne diferencialne enačbe s pomočjo *midpoint* integracijske sheme. Zamislimo si linearno diferencialno enačbo, na primer

$$m\ddot{x} + c\dot{x} + kx = aCos(t)\ , \tag{5.1}$$

kjer so m, c, k, a poljubne konstante in t čas.

Enačba je torej funkcija časa. Recimo sedaj, da v času $t = t_0$ poznamo lego in hitrost točke. Označimo ju z $x_{oj}\ in\ \dot{x}_{oj}$. Simbola $x_{1j}\ in\ \dot{x}_{1j}$ pa naj bosta isti a neznani veličini izračunani v času $t = t_1$, pri čemer je $t_1 = t_0\ +\ \Delta t$. Stanja v točki 1 ne poznamo, poznamo pa stanje v točki 0. *Midpoint* integracijska shema temelji na točno tem poznavanju. Privzamemo, da so količine v srednji točki med točko 0 in 1 nekako povprečje. Ob *midpoint* času $t_m = t_0\ +\ \Delta t/2$ lahko zapišemo približne enačbe lege, hitrosti in pospeška kot

$$x_{mj} = \frac{x_{1j} + x_{0j}}{2} \tag{5.2}$$

$$\dot{x}_{mj} = \frac{(x_{1j} - x_{0j})}{\Delta t} \tag{5.3}$$

$$\ddot{x}_{mj} = \frac{(x_{1j} - x_{0j} - \Delta t\ \dot{x}_{0j})}{\Delta t^2/2}. \tag{5.4}$$

Če te količine sedaj vstavimo v zgornjo diferencialno enačbo, dobimo algebraično enačbo z eno neznanko x_{1j}. Torej lahko izrazimo in izračunamo položaj točke 1, x_{1j}. Ko poznamo x_{1j}, ni težko izračunati $\dot{x}_{1j}\ in\ \ddot{x}_{1j}$ iz enačb (5.3) in (5.4). Paziti moramo le, da je x_{1j} funkcijsko odvisna tudi od $\dot{x}_{oj}$, kar pomeni, da moramo hkrati reševati še enačbo za hitrost

$$\dot{x}_{1j} = \frac{2}{\Delta t}\left(x_{1j} - x_{0j}\right) - \dot{x}_{oj}\ . \tag{5.5}$$

Za poljubne koeficiente m, c, k, a in Δt preverimo, kakšen je odziv sistema s pomočjo diferencialne enačbe.

$m = 2\ , c = 3\ , k = 4\ , a = 2\ , \Delta t = 0.1$

Prikažimo diferencialno enačbo, sprogramirano v Microsoft Visual Studio, programski jezik C# (Slika 5.1).

```
private void button2_Click(object sender, EventArgs e)  // Izračun 2
{

    double a = 2;
    double m = 2;
    double c = 3;
    double k = 4;
    double[] x;                         //položaj
    double[] xc;                        // hitrost
    double dt = 0.1;                     // delta t
    double t0 = 0;                      // začetni čas
    double tm = t0 + dt / 2;            // midpoint čas
    double t;
    x = new double[100];
    x[0] = 0;                          // zač. vrednost položaj
    xc = new double[100];
    xc[0] = 0;                        // zač vrednost hitrost


    for (int i = 0; i < 99; i++)
    {
        x[i + 1] = ((a * Math.Cos(t = tm + i*0.1) + x[i] * (2 * m / (dt * dt) + c / dt - k / 2)
            + m * xc[i] * 2 / dt) / (2 * m / (dt * dt) + c / dt + k / 2));    // izračun pozicije

        xc[i + 1] = 2 / dt * (x[i + 1] - x[i]) - xc[i];                 // izračun hitrosti


        DataSet1.VrednostiRow rw = vre.Vrednosti.NewVrednostiRow();     // zapis v tabelo
        rw.t = Math.Round(t, 5);
        rw.x = Math.Round(x[i + 1], 5);
        rw.xc = Math.Round(xc[i + 1], 5);
        vre.Vrednosti.AddVrednostiRow(rw);

    }
```

Slika 5.1: C# koda

Z ukazom *double* deklariramo poljubno realno število. Z ukazom *double[]* deklariramo eno razsežno polje z realnimi števili. Z ukazom *new double[100]* kreiramo polje velikosti 100.

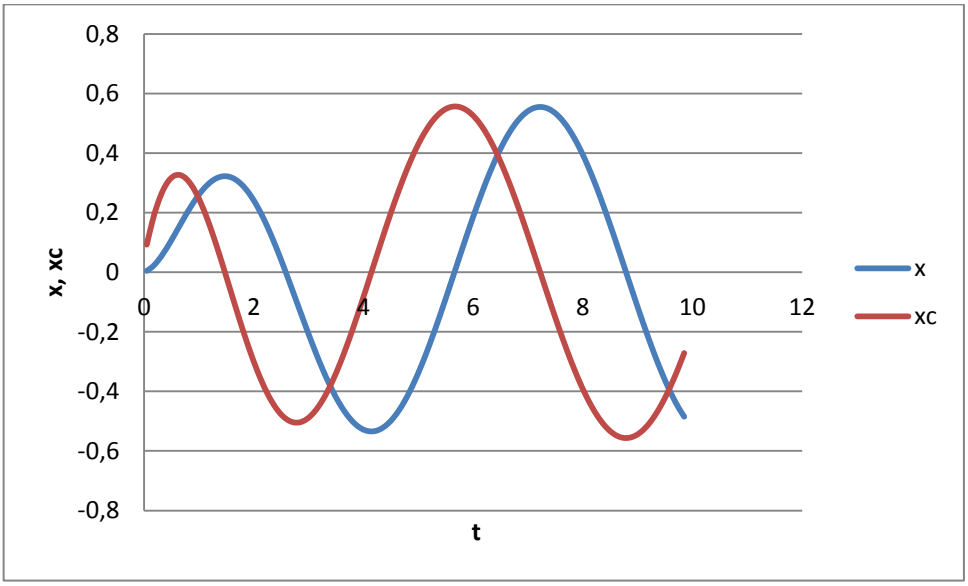
Kreiramo *for* zanko, ki naj ima malo majn obhodov, kot je velikost polja, kamor zapisujemo. Iz diferencialne enačbe smo eksplicitno izrazili x_{1j}. Hkrati rešujemo še enačbo za hitrost $\dot{x}_{1j}$. Pri vsakem obhodu zanke izračunamo i-ti položaj $x[i\ +1]$ in i-to hitrost $xc[i\ +1]$.

Parametre, skupen čas t, položaj v točki 1,x, ter hitrost v točki 1,xc, sproti zapisujemo v preglednico 5.1.

Preglednica5.1

t	x	xc	t	x	xc	t	x	xc
0,05	0,00462	0,09248	3,35	-0,36448	-0,39805	6,65	0,46498	0,30442
0,15	0,01772	0,16947	3,45	-0,40237	-0,3598	6,75	0,493	0,2559
0,25	0,03773	0,2308	3,55	-0,43621	-0,31696	6,85	0,51604	0,20481
0,35	0,0631	0,2766	3,65	-0,46556	-0,27011	6,95	0,53386	0,15166
0,45	0,0923	0,30731	3,75	-0,49006	-0,21985	7,05	0,54629	0,09701
0,55	0,12385	0,32363	3,85	-0,5094	-0,16684	7,15	0,55321	0,0414
0,65	0,15635	0,32646	3,95	-0,52333	-0,11174	7,25	0,55455	-0,01459
0,75	0,18852	0,31689	4,05	-0,53168	-0,05522	7,35	0,5503	-0,07041
0,85	0,21917	0,29614	4,15	-0,53433	0,00205	7,45	0,54051	-0,12549
0,95	0,24725	0,26554	4,25	-0,53126	0,05939	7,55	0,52527	-0,17928
1,05	0,27186	0,22649	4,35	-0,52248	0,11615	7,65	0,50474	-0,23124
1,15	0,2922	0,18041	4,45	-0,50809	0,17168	7,75	0,47914	-0,28084
1,25	0,30766	0,12877	4,55	-0,48824	0,22537	7,85	0,44872	-0,32759
1,35	0,31775	0,073	4,65	-0,46314	0,27662	7,95	0,41379	-0,37103
1,45	0,32212	0,0145	4,75	-0,43307	0,32489	8,05	0,3747	-0,41073
1,55	0,32058	-0,04538	4,85	-0,39834	0,36966	8,15	0,33185	-0,44628
1,65	0,31304	-0,10535	4,95	-0,35933	0,41045	8,25	0,28567	-0,47733
1,75	0,29956	-0,16421	5,05	-0,31647	0,44686	8,35	0,23662	-0,50359
1,85	0,28031	-0,22088	5,15	-0,2702	0,4785	8,45	0,1852	-0,52478
1,95	0,25555	-0,27434	5,25	-0,22102	0,50506	8,55	0,13193	-0,5407
2,05	0,22565	-0,32371	5,35	-0,16945	0,52629	8,65	0,07734	-0,55119
2,15	0,19105	-0,36822	5,45	-0,11604	0,54197	8,75	0,02197	-0,55615
2,25	0,15228	-0,40722	5,55	-0,06134	0,55197	8,85	-0,03362	-0,55554
2,35	0,10991	-0,44015	5,65	-0,00594	0,5562	8,95	-0,08886	-0,54937
2,45	0,06457	-0,46662	5,75	0,04961	0,55464	9,05	-0,14321	-0,53769
2,55	0,01692	-0,4863	5,85	0,1047	0,54732	9,15	-0,19613	-0,52063
2,65	-0,03234	-0,49902	5,95	0,15879	0,53435	9,25	-0,24708	-0,49836
2,75	-0,08253	-0,50468	6,05	0,2113	0,51587	9,35	-0,29555	-0,47111
2,85	-0,13293	-0,50332	6,15	0,2617	0,49209	9,45	-0,34107	-0,43914
2,95	-0,18285	-0,49503	6,25	0,30946	0,46326	9,55	-0,38316	-0,40279
3,05	-0,2316	-0,48004	6,35	0,35411	0,42971	9,65	-0,42142	-0,36242
3,15	-0,27853	-0,45862	6,45	0,39519	0,39178	9,75	-0,45547	-0,31842
3,25	-0,32302	-0,43115	6,55	0,43227	0,34987	9,85	-0,48495	-0,27125

Prikažimo rezultate še z diagramom, slika 5.2.



Slika 5.2: Odziv sistema, dobljen z uporabo diferencialne enačbe

Mitpoint integracijska shema je med vsemi integracijskimi shemami najpreprostejša. Kljub temu nam je bila za zgled, kakšen je pomen integracijskih shem. Bolj napredne sheme se od te razlikujejo po tem, da npr. neznani količini $\dot{x}_{1j}$ in $\ddot{x}_{1j}$ zapišemo kot aproksimacijo s produkti parametrov in začetnih stanj v točki 0.

Sedaj, ko vemo, kako uporabiti integracijske sheme pri reševanju dinamične enačbe, se podajmo na programiranje dinamičnega končnega elementa v okolju AceGen.

6 PROGRAMIRANJE V OKOLJU ACEGEN

AceGen je paket v Mathematici, ki se uporablja za avtomatsko generiranje in izpeljavo formul in programske kode, potrebnih v numeričnih postopkih. AceGen uporabniku generira zelo optimizirano kodo v programskem jeziku C# ali FORTRAN.

Za bralca, ki mu Mathematica ni tuja, bo razumevanje AceGena morda lažje. Kakorkoli, razlage se bodo nanašale samo na sintakso, povezano z AceGen, razumevanje sintakse v Mathematici pa bo veljalo za samoumevno.

V principu lahko input V AceGenu dobimo tako, da operatorje $=$ v standardni Mathematica sintaksi zamenjamo s primernimi v AceGenu ($\vDash$, $\exists$, $\dashv$), pogojne stavke If, Which, Switch s SMSIf, SMSWhich, SMSSwitch in standardno Do zanko s SMSDo.

V nadaljevanju bo v prvem podpoglavju prikazano programiranje statičnega elementa. Vsa neznana simbolika in sintaksa bo pojasnjena vzporedno s samim postopkom programiranja. Določene stvari se bodo sklicevale na že predhodno razloženo teorijo. Ker pa nas zanima dinamično obnašanja končnega elementa, bodo v drugem podpoglavju dodane enačbe še za dinamični del, ter popravljene tiste, ki se, v nasprotju s statičnim elementom, tu spremenijo.

6.1 Programiranje statičnega 4-vozliščnega ravninskega elementa

Preden začnemo delati z paketom AceGen, ga moramo zagnati. V Mathematici to storimo z ukazom

```
<< AceGen`
kPS = 8; (* št. PS*)
modul = "RNS";
```

Ko ukaz izvršimo, se nam pojavi AceGen okence, kjer imamo na voljo vse operatorje, pomoč in podporo. Običajno na začetku definiramo še število prostorskih stopenj elementa kPS in ime modula. Ta korak ni potreben na začetku, je pa koristen.

Sedaj lahko začnemo pisati enačbe našega končnega elementa. Tako kot moramo pri teoretični izpeljavi iti korak za korakom, tako moramo pri programiranju še bolj paziti na doslednost. Če vemo, da program bere celice po vrsti od zgoraj navzdol, potem je hitro jasno, da ne more izračunati funkcije $y(x)$, dokler ne pozna x-a.

6.1.1 Osnovni input

Kot že rečeno, je na začetku najbolje zapisati vse vhodne podatke, ki jih bomo potrebovali pri oblikovanju formul. Najbolje jih je zapisati v svojo celico in jo poimenovati na primer »osnovniInput«. Če smo natančni, lahko vhodne parametre zaradi večje jasnosti delimo na čiste konstante, poljubno spremenljive konstante ter neznanke pomikov. Zapis izgleda takole:

OsnovniInput

```
(* Čiste konstante *)
c = Array[SMSReal[c$$[#1]] &, {2}];
{Emodul, v} = c;

(* Poljubno spremenljive konstante *)
p = Array[SMSReal[p$$[#1]] &, {9}];
xi = Take[p, {1, kPS - 1, 2}];
yi = Take[p, {2, kPS, 2}];
t = p[[kPS + 1]];

(* Neznanke pomiki *)
u = Array[SMSReal[u$$[#1]] &, {kPS}];
uxi = Take[u, {1, kPS - 1, 2}];
uyi = Take[u, {2, kPS, 2}];
```

Iz zapisa vidimo, da poljubno polje definiramo s sintakso *Array*, kjer v oglati oklepaj po vrsti zapišemo vrsto zunanje spremenljivke *SMSReal* (realno število), s simbolom pred $$ označimo, kako bo objekt poimenovan v naši izstopni C# kodi, v zavit oklepaj {} pa zapišemo razsežnost našega polja.

Na ta način zapišemo polje c, kjer se skrivata dva parametra, ki sta konstantna, elastični modul $Emodul$ ter Poissonov količnik v.

Za polje p rečemo, da naj predstavlja koordinate končnega elementa in naj vsebuje še debelino elementa. Ker je končni element 4-vozliščni, imamo torej 8 koordinat, kar predstavlja osem mest v polju. Rekli smo, da v polje p zapišemo še debelino elementa deb, kar pomeni še dodatno mesto v polju. Skupaj imamo torej polje velikosti 9. Sedaj moramo v polju določiti, katero meso v polju predstavlja katero koordinato elementa. To naredimo s funkcijo *Take*, kjer za x_i koordinate elementa iz polja poberemo lihe pozicije od 1 do kPS. Za y_i koordinate elementa pa iz polja poberemo sode pozicije od 2 do kPS. Za debelino elementa deb tako ostane še 9. pozicija v polju.

V polje u po istem postopku zapišemo še neznanke pomikov. Velikost polja je enaka kPS. S funkcijo *Take* zopet ločimo pomike po smereh x in y.

Tako smo zapisali vse potrebne vhodne parametre.

6.1.2 Konstante

Naslednja celica, ki jo je pametno oblikovati, je skupina konstant, ki so v glavnem kombinacije vhodnih konstant, ki smo jih že zapisali, in nam bodo kasneje prišle prav pri zapisu deformacijskega tenzorja. To so konstante $c1, c2, \mu$, ki jih zaradi hitrejšega računanja zapišemo v materialno matriko **Cmat**.

Konstante

```
(* Pomožne konstante *)
c1 = Emodul / (1 - ν^2);
c2 = c1 * ν;
μ = Emodul / (2 * (1 + ν));
Cmat = { {c1, c2, 0}, {c2, c1, 0}, {0, 0, 2 * μ} };
```

6.1.3 Geometrija

V naslednjo celico zapišemo enačbe, ki so povezane z geometrijo elementa. Kot prvo so to interpolacijske funkcije

Geometrija

```
(* Interpolacijske funkcije *)
N1 = 1/4 * (1 - ξ) * (1 - η);
N2 = 1/4 * (1 + ξ) * (1 - η);
N3 = 1/4 * (1 + ξ) * (1 + η);
N4 = 1/4 * (1 - ξ) * (1 + η);
Ni = {N1, N2, N3, N4};
```

,

kot drugo izračun koordinat poljubne točke na elementu

```
(* Koordinate poljubne točke *)
x = SMSFreeze[Ni.xi];
y = SMSFreeze[Ni.yi];
```

,

kot tretje izračun pomikov v polju elementa

```
(* Pomiki v polju elementa *)
ux = Ni.uxi;
uy = Ni.uyi;
```

in kot zadnje zapis Jacobijeve matrike elementa in njene determinante

```
(* Izračun jakobijeve matrike *)
Jg = SMSD[{x, y}, {ξ, η}];
Jac = SMSDet[Jg];
```

Nove funkcije, ki smo jih uporabili so *SMSFreeze*, *SMSD*, *SMSDet*.

Funkcijo *SMSFreeze* uporabimo, kadar hočemo zamrzniti odvisnost neke spremenljivke. Na primer, če želimo po uporabi ukaza *SMSFreeze* odvajati spremenljivko x, program ne bo upošteval njene predhodne odvisnosti od drugih veličin. *SMSD* je izraz za odvod. Pri kasnejšem računanju potrebujemo Jacobijevo matriko končnega elementa **Jg**, ki jo dobimo tako, da koordinati x in y odvajamo po ξ in η. **Jg** je torej matrika velikosti 2x2. S funkcijo *SMSDet* izračunamo še determinanto Jac matrike **Jg**.

6.1.4 Elastični potencial

Poglejmo si, kako poteka zapis diferenciala elastičnega potenciala končnega elementa. Veličini, ki nam za zapis potenciala še manjkata,sta tenzorja deformacij in napetosti. Izračunajmo deformacijski tenzor. Do njega pridemo na sledeč način.

Najprej izračunamo deformacijski gradient. Uporabimo enačbo (4.20) iz izpeljave in sicer:

$$\mathbf{F} = \mathbf{G} + \mathbf{I}$$

$$\mathbf{F} = \mathbf{X}\mathbf{J}^{-1} + \mathbf{I}$$

$$\mathbf{F} = \begin{bmatrix} \frac{\partial u_x}{\partial \xi} & \frac{\partial u_x}{\partial \eta} \\ \frac{\partial u_y}{\partial \xi} & \frac{\partial u_y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.1}$$

Zaradi razumevanja zapišimo eno komponento gradienta **F**

$$\frac{\partial u_x}{\partial x} = \frac{\partial u_x}{\partial \xi}\frac{\partial x}{\partial \xi} + \frac{\partial u_x}{\partial \eta}\frac{\partial x}{\partial \eta} + 1. \tag{6.2}$$

Torej, da dobimo deformacijski gradient **F**, moramo parcialno odvajati $x + u_x$ in $y + u_y$ po ξ in η.

Sintaksa izgleda takole:

ElasticniPotencial

```
SMSDefineDerivative[{ξ, η}, {x, y}, SMSInverse[Jg]];

(* Deformacijski gradient Fij in tenzor εij *)
Fij = SMSD[{x + ux, y + uy}, {x, y}];
```

Nova je funkcija *SMSDefineDerivative*, ki pove, da je odvod ξ in η po x in y enak inverzu Jacobijeve matrike **Jg**. To moramo določiti z namenom, da računalnik ve, da mora odvajati po lokalnih koordinatah in ne po globalnih.

Sedaj, ko imamo zapisan deformacijski gradient **Fij**, lahko izračunamo deformacijski tenzor ***εij*** po enačbi (4.19).

```
εij = 1/2 (Transpose[Fij].Fij - IdentityMatrix[2]);
```

Zaradi uporabnosti zložimo tenzor deformacij v vektorsko polje.

```
(* Vektorji - pravi in korigirani za izračun energije*)
εk = {εij[[1, 1]], εij[[2, 2]], εij[[1, 2]]};
εkA = {εij[[1, 1]], εij[[2, 2]], 2 * εij[[1, 2]]};
```

Oglati oklepaji predstavljajo komponento tenzorja, ker ni vseeno, kako jih zložimo v vektor. **εk** predstavlja vektor deformacij, **εka** pa korigiran vektor deformacij, ki upošteva, da imamo dve simetrični komponenti, ki ju moramo pri kasnejšem računanju potenciala upoštevati.

Sedaj izračunajmo napetosti.

```
(* Izračun napetosti *)
Sk = Cmat.εk;
```

Skalarno smo množili materialno matriko **Cmat** z vektorjem deformacij **εk** in dobili vektor napetosti **Sk**.

Zdaj imamo vse potrebne komponente za zapis diferenciala potenciala. Sintaksa v AceGenu je naslednja.

```
(* Zapis potenciala *)
dΠ = 1/2 * εkA . Sk * deb * Jac;
```

Pika predstavlja skalarni produkt vektorjev. Zapis diferenciala potenciala je s tem končan.

6.1.5 Integracija

Zapisan diferencial potenciala moramo sedaj integrirati v Gaussovih točkah po koordinatah ξ in η. V ta namen v programu zapišemo dve zanki. Poimenujemo ju $ig1$, in $ig2$. Sta tip spremenljivke *SMSInteger* oz. pozitivni celi števili s korakom 1.

```
ZacetekIntegracije
        SMSDo[ig1, 1, SMSInteger[kG$$]];
        SMSDo[ig2, 1, SMSInteger[kG$$]];
```

Ker poteka integracija numerično z metodo Gaussove kvadrature, moramo to zunanjo podrutino, ki izvaja računanje, z znanimi vhodnimi parametri nekako priklicati v program. To storimo z ukazom *SMSCall*.

```
integracija = SMSCall["Gauss_GetAW", Integer[kG$$], ig1, ig2,
   Real[xGau$$, yGau$$, wGauss$$], "System" → False];
```

Ime rutine je integracija, kličemo metodo $Gauss\_GetAW$ z vhodnima parametroma $ig1, ig2$, izhodni parametri pa so koordinati Gaussovih točk $xGau$ in $yGau$ ter Gaussova utež $wGauss$.

Pri integriranju naši lokalni koordinati ξ in η postaneta Gaussovi koordinati $xGau$ in $yGau$. Z ukazom »subordinate« povemo, da so te definirane spremenljivke podrejene rutini integracija.

Koda izgleda tako:

```
{ξ, η, wGauss} ↦ SMSReal[{xGau$$, yGau$$, wGauss$$},
    "Subordinate" → integracija];
```

Za konec integracije moramo samo še zaključiti zanki.

```
KonecIntegracije
        SMSEndDo[];
        SMSEndDo[];
```

6.1.6 Start modula in vhodnih podatkov

Prišli smo tako daleč, da lahko začnemo z izdelavo kode KE. Pred tem moramo še zagnati modul ter prebrati celice z našimi vhodnimi podatki.

Na vrsto prideta dva stavka, ki ju moramo obvezno zapisati na začetku zagona programa.

```
SMSInitialize[modul, "VectorLength" → 350, "Language" → "C#",
  "Mode" → "Optimal"];
SMSModule[modul];
```

Ukaz *SMSInitialize* začne novo AceGen sejo, pri čemer je modul ime datoteke, kamor shranjujemo projekt, »VectroLenght« velikost sistemskih vektorjev, »Language« programski

jezik, v katerem želimo imeti kodo našega končnega elementa, »Optimal« pa pomeni generiranje najhitrejše in najkrajše programske kode.

Ukaz *SMSModule* zažene nov modul z imenom, v našem primeru modul, brez vhodnih ali izhodnih parametrov.

Z ukazom *SMSEvaluateCellsWithTag* pa preberemo celici z vhodnimi podatki »OsnovniInput« ter »Konstante«, ki smo ju pred tem sprogramirali.

```
SMSEvaluateCellsWithTag["OsnovniInput"];
SMSEvaluateCellsWithTag["Konstante"];
```

6.1.7 Račun in export

Sintaksa izračuna togostne matrike izgleda takole:

Račun in export

```
SMSEvaluateCellsWithTag["ZacetekIntegracije"];

SMSEvaluateCellsWithTag["Geometrija"];
SMSEvaluateCellsWithTag["ElasticniPotencial"];

SMSDo[i, 1, kPS];
fi = SMSD[dΠ * wGauss, u, i];
SMSExport[fi, f$$[i], "AddIn" → True];
SMSDo[j, i, kPS];
fij = SMSD[fi, u, j];
SMSExport[fij, fu$$[i, j], "AddIn" → True];
SMSEndDo[];
SMSEndDo[];

SMSEvaluateCellsWithTag["KonecIntegracije"];
```

Računanje togostne matrike se prične s priklicem celice »ZacetekIntegracije« in konča s priklicem celice »KonecIntegracije«. To ponovno naredimo z ukazom *SMSEvaluateCellsWithTag*. Z istim ukazom preberemo še enačbe iz celic »Geometrija« ter »ElasticniPotencial«.

Potem pride na v vrsto zunanja zanka (funkcija *SMSDo*), ki teče od i=1 do i=kPS.

Potencial v določeni integracijski točki množimo z Gaussovo utežjo $wGauss$ in ga nato odvajamo po i-ti komponenti pomika. Vse to se zapisuje pod spremenljivko fi. S funkcijo *SMSExport* povemo, kaj naj program izvozi kot naš rezultat. S funkcijo *AddIn -> True* se rezultat, ki se izvozi, prišteje k $f\$\$[i]$. Izvozno spremenljivko $f\$\$[i]$ imenujemo vektor notranjih sil in ima 8 komponent (kPS = 8).

Nato sledi še notranja zanka, ki teče od j=i do j=kPS. Zdaj fi odvajamo po j-tem pomiku in zapisujemo pod spremenljivko fij. Z funkcijo *SMSExport* izvozimo $fu\$\$[i,j]$. S funkcijo *AddIn ->True* pa se rezultat, ki se izvozi, prišteje k $fu\$\$[i,j]$. Izvozno spremenljivko $fu\$\$[i,j]$ imenujemo togostna matrika in je razsežnosti 8x8 (kPS x kPS).

Programiranje togostne matrike je s tem končano. Sedaj nam ne preostane drugega, kot pa da kodo zapišemo v datoteko. To storimo z ukazom *SMSWrite*.

```
^ SMSWrite[];
```

6.2 Programiranje dinamičnega 4-vozliščnega ravninskega elementa

Poglejmo si programiranje dinamičnega končnega elementa. Poudarimo, kar se v nasprotju s statičnim elementom pri dinamičnem spremeni.

6.2.1 Osnovni input

Prva stvar, ki se pojavi na novo, je čas t, saj je dinamika časovno odvisna. S funkcijo *SMSFictive* povemo, da je spremenljivka t začasna spremenljivka za reševanje algebraičnih operacij, ki pa ne bo prisotna v končnem rezultatu.

```
OsnovniInput
        (* FIKTIVNE *)
        t ▸ SMSFictive[];
```

Kot drugo, se nam poveča input konstant. Zapišemo jih pod polje c. Sedaj povejmo, da je končni element sprogramiran za poljubno integracijsko shemo in ne točno za *midpoint* integracijsko shemo, ki smo si jo spoznali v poglavju 5. Princip razumevanja pa je enak za vse integracijske sheme, razlikujejo se le v tem, kje med vrednostima 0 in 1 leži točka m, v kateri integriramo.

```
(* INPUT KONSTANT *)
c ▸ Array[SMSReal[c$$[#1]] &, {12}];
{ρ, Emodul, ν, Δt, αf, αm, c1, c2, c3, c4, c5, c6} ▸ c;
```

Koeficienta αm in αf sta odvisna od vrste časovne integracijske sheme in določata, kje se nahaja ravnovesna točka u_m:

$$u_m = (\alpha m, \alpha f)u_0 + (\alpha f, \alpha m)u_1$$

Konstante $c1, c2, c3, c4, c5, c6$ so odvisne od αm in αf .

Polje p s koordinatami ostane enako kot pri statičnem elementu, spremeni pa se polje pomikov. Pred tem smo imeli samo ene pomike, sedaj pa imamo pomike v času t_0 in t_1, ter hitrosti in pospeške.

```
(*INPUT ODZIVNIH SPREMENLJIVK*)
u0 = Array[SMSReal[u0$$[#1]] &, {kPS}];
ut0 = Array[SMSReal[ut0$$[#1]] &, {kPS}];
utt0 = Array[SMSReal[utt0$$[#1]] &, {kPS}];
u1 = Array[SMSReal[u1$$[#1]] &, {kPS}];
```

$u0$ predstavlja polje pomikov v točki 0, $ut0$ predstavlja polje hitrosti pomikov v točki 0, $utt0$ predstavlja polje pospeškov pomikov v točki 0. $u1$ predstavlja neznano polje pomikov v točki 1.

Manjkata nam še stanja $ut1$ in $utt1$, ki pa ju lahko izračunamo s kombinacijo ostalih, že definiranih parametrov.Poglejmo:

```
(*STANJE V CASU T1 - ZA POLJUBNO INT SHEMO*)
ut1 = NewFun[c1, c2, c3, u0, ut0, utt0, u1];
utt1 = NewFun[c4, c5, c6, u0, ut0, utt0, u1];
```

Pri tem je *NewFun* funkcija s katero izračunamo neznane hitrosti in pospeške z znanimi količinami. Enaka je za vse časovne integracijske sheme.

```
NewFun[c1_, c2_, c3_, f0_, ft0_, ftt0_, f1_] := c1 * (f1 - f0) + c2 * ft0 + c3 * ftt0;
```

S tem smo zapisali vse vhodne podatke. Sedaj si poglejmo stanje v ravnovesni točki.

6.2.2 Stanje v ravnovesni časovni točki

Stanje zapišemo za poljubno integracijsko shemo.

```
StanjeVRavnovesniCasovniTocki
        (* Vozliščni pomiki za elastični potencial *)
        uf = SMSFreeze[af * u1 + (1 - af) * u0];

        (* Vozliščni pomiki,
        hitrosti in pospeški za kinetično energijo *)
        um = SMSFreeze[am * u1 + (1 - am) * u0];
        umt = SMSFreeze[am * ut1 + (1 - am) * ut0];
        umtt = SMSFreeze[am * utt1 + (1 - am) * utt0];
```

uf je vozliščni pomik za elastičen potencial, $um, umt, umtt$ pa so po vrsti vozliščni pomiki, hitrosti in pospeški za kinetični potencial. Zopet uporabimo funkcijo *SMSFreeze*.

Računalniku je treba povedati še, kakšna je povezava med parametri um, umt in $umtt$. To naredimo tako, da definiramo odvode.

```
SMSDefineDerivative[um, t, umt];
SMSDefineDerivative[umt, t, umtt];
```

S tem povemo, da je odvod um po času t enak umt, odvod umt po t pa je enak $umtt$.

6.2.3 Konstante

Celica s pomožnimi konstantami ostane enaka kot pri statičnem elementu.

6.2.4 Geometrija

Celica geometrija ostane enaka, z razliko, da odstranimo polje pomikov. Definirali ga bomo kasneje, posebej za elastični potencial in kinetično energijo.

6.2.5 Elastični potencial

Tudi elastični potencial moramo malo korigirati. Do sedaj imamo zapisan vektor vseh vozliščnih pomikov uf v integracijski točki. Za izračun deformacijskega tenzorja pa potrebujemo polje pomikov, torej funkcijo, odvisno od ξ in η po celem elementu. To naredimo z interpolacijo.

ElasticniPotencial

```
ufx = Ni.Take[uf, {1, kPS - 1, 2}];
ufy = Ni.Take[uf, {2, kPS, 2}];
```

Posebej smo zapisali x in y komponenti.

Sedaj moramo pri zapisu deformacijskega gradienta spremeniti $x + u_x$ in $y + u_y$ v $x + u_{fx}$ in $y + u_{fy}$.

```
Fij = SMSD[{x + ufx, y + ufy}, {x, y}];
```

Diferencial potenciala nato izračunamo po enakih, že razloženih formulah.

6.2.6 Kinetična energija

Kinetično energijo potenciala mase dm zapišemo kot

$$dWk = \frac{1}{2} v^2 dm\,. \tag{6.3}$$

Do sedaj imamo zapisan vektor hitrosti vseh vozlišč umt, mi pa za izračun skupne hitrosti v potrebujemo polje hitrosti po celotnem elementu po smereh x in y. Do tega zopet pridemo z interpolacijo.

KineticnaEnergija

```
umtx = Ni.Take[umt, {1, kPS - 1, 2}];
umty = Ni.Take[umt, {2, kPS, 2}];
```

Diferencial kinetične energije zapišemo v eni vrstici takole

```
dW = 1/2 * (umtx * umtx + umty * umty) * deb * Jac * ρ;
```

,

pri čemer oklepaj $(umtx\ umtx + umty\ umty)$ predstavlja po Pitagorovem pravilu izračunan kvadrat hitrosti, v^2. Produkt $(deb\ Jac\ \rho)$ pa je ekvivalenten masi diferenciala končnega elementa.

Prišli smo do točke, ko imamo zapisan elastični potencial in kinetično energijo. Na vrsti je numerična integracija. Celici začetek in konec integracije se ne spremenita, zato ju ne bomo še enkrat pisali.

6.2.7 Start modula in input

Sintaksa nam je že poznana. V nasprotju z statičnim elementom moramo prebrati še celico s stanji v integracijski točki.

```
modul = "FEDP4_GetFFu";
SMSInitialize[modul, "VectorLength" → 500,
  "Language" → "Fortran90", "Mode" → "Optimal"];
SMSModule[modul];

SMSEvaluateCellsWithTag["OsnovniInput"];
SMSEvaluateCellsWithTag["Konstante"];
SMSEvaluateCellsWithTag[
  "StanjeVRavnovesniCasovniTocki"];
```

6.2.8 Račun in export

■ **Račun in export**

```
SMSEvaluateCellsWithTag["ZacetekIntegracije"];

SMSEvaluateCellsWithTag["Geometrija"];
SMSEvaluateCellsWithTag["ElasticniPotencial"];
SMSEvaluateCellsWithTag["KineticnaEnergija"];

SMSDo[i, 1, kPS];
fie ⊧ SMSD[dΠ * wGauss, uf, i];
fik ⊧ SMSD[SMSD[dW * wGauss, umt, i], t];
fi ⊧ fie + fik;
SMSExport[fi, f$$[i], "AddIn" → True];
SMSDo[j, i, kPS];
fij ⊧ SMSD[fi, u1, j];
SMSExport[fij, fu$$[i, j], "AddIn" → True];
SMSEndDo[];
SMSEndDo[];

SMSEvaluateCellsWithTag["KonecIntegracije"];
```

Kot že vemo in je razvidno iz kode, se računanje togostne matrike prične s priklicem celice »ZacetekIntegracije« in konča s priklicem celice »KonecIntegracije«. To ponovno naredimo z

ukazom »*SMSEvaluateCellsWithTag*«. Z istim ukazom preberemo še enačbe iz celic »Geometrija«, »ElasticniPotencial« in »KineticnaEnergija«.

Z *SMSDo* zapišemo zunanjo zanko, ki teče od i=1 do i=kPS.

Diferencial elastičnega potenciala $d\Pi$ v določeni integracijski točki množimo z Gaussovo utežjo $wGauss$ in ga nato odvajamo po i-ti komponenti pomika uf. Vse to se zapisuje pod spremenljivko fie, ki predstavlja notranjo silo.

Diferencial kinetične energije dW v določeni integracijski točki množimo z Gaussovo utežjo $wGauss$ in ga nato prvič odvajamo po i-ti komponenti hitrosti umt, drugič pa po času t. S tem izračunamo vztrajnostno silo, ki jo zapisujemo pod fik. Nato obe sili seštevamo v skupno silo $fi = fie + fik$. Vsoto izvozimo kot $f\$\$[i]$ s funkcijo *SMSExport*, z *AddIn* pa prištejemo k $f\$\$[i]$. Seštevamo namreč po Gaussovih točkah.

Sedaj zapišemo notranjo zanko *SMSDo*, ki teče od j=i do j=kPs.

Da bi dobili tangento matriko za iteracijo, moramo fi odvajati po naši neznanki $u1$. Rezultat zapisujemo pod spremenljivko fij, izvozimo ga kot $fu\$\$[i,j]$. Z *AddIn* zopet prištevamo k $fu\$\$[i,j]$ za vsako Gaussovo točko.

Sedaj kodo zapišemo v datoteko. To storimo z ukazom *SMSWrite*.

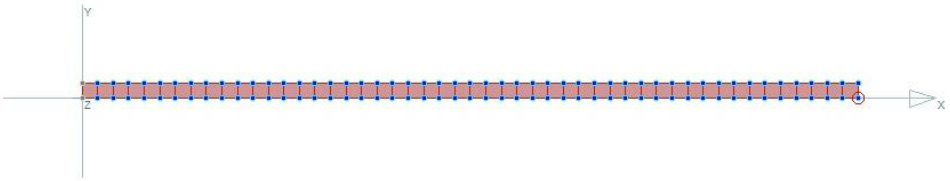
```
SMSWrite[];
FilePrint[modul <> ".f90"];
```

Za ilustracijo si oglejmo še odsek generirane kode, ki je prikazan spodaj.

```
****************** S U B R O U T I N E **********************
SUBROUTINE FEDP4_GetFFu(v)
USE SMSUtility
IMPLICIT NONE
INTEGER i01,i02,kG,i123,i124,i188,i208
DOUBLE PRECISION v(637),wGauss,xGau,yGau
v(2)=c(1)
v(6)=c(5)
v(232)=1d0-v(6)
v(7)=c(6)
v(233)=1d0-v(7)
v(11)=c(10)
v(14)=p(1)
v(15)=p(2)
v(16)=p(3)
v(17)=p(4)
...
```

7 VGRADNJA ELEMENTA V MKE PROGRAM EMS

Dokončan oz. sprogramiran KE vgradimo v raziskovalni MKE program EMS. EMS je MKE program za računanje prehodnega dinamičnega odziva nelinearnih konstrukcij. Slika 7.1 prikazuje konzolo, na kateri bomo preizkušali KE.



Slika 7.1: Konzola dimenzij (1x0,02x0,02)m

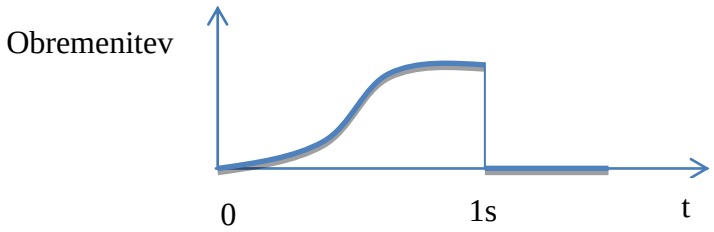
Možnosti analiziranja so:

- spreminjanje vrste obremenitve
- spreminjanje računske mreže
- spreminjanje pozicije obremenitve
- spreminjanje vrste integracijske sheme
- opazovanje pomika poljubnega vozlišča
- opazovanje energije konzole

Rezultate, ki jih bomo dobili pri analizi, bo potrebno primerjati s KE, za katerega vemo, da daje realne rezultate. V ta namen bo za primerjavo uporabljen kvaliteten končni element – nosilec.

7.1 Obremenitev

Za testiranje bomo uporabili počasi naraščajočo sinusno obremenitev na sliki 7.2, ki v času $t = 1s$ doseže maksimalno vrednost $R_{max} = 200N$.



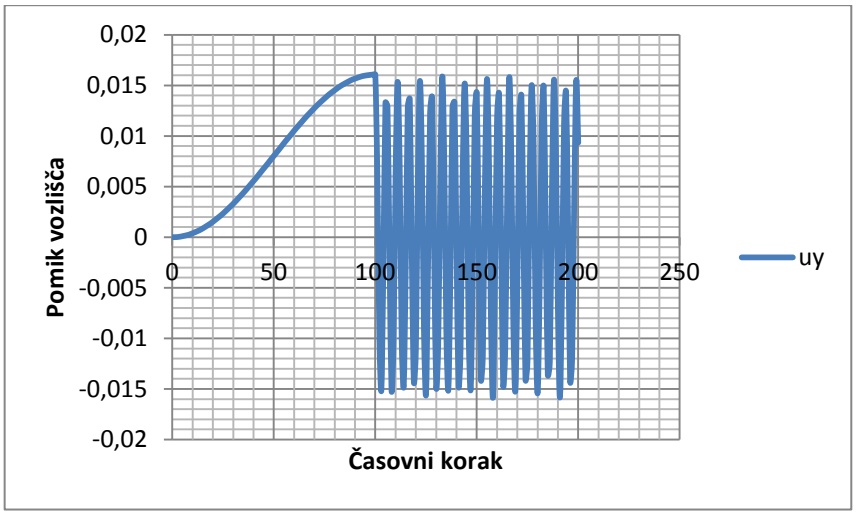
Slika 7.2: Obremenitev konzole

7.2 Zgoščevanje mreže

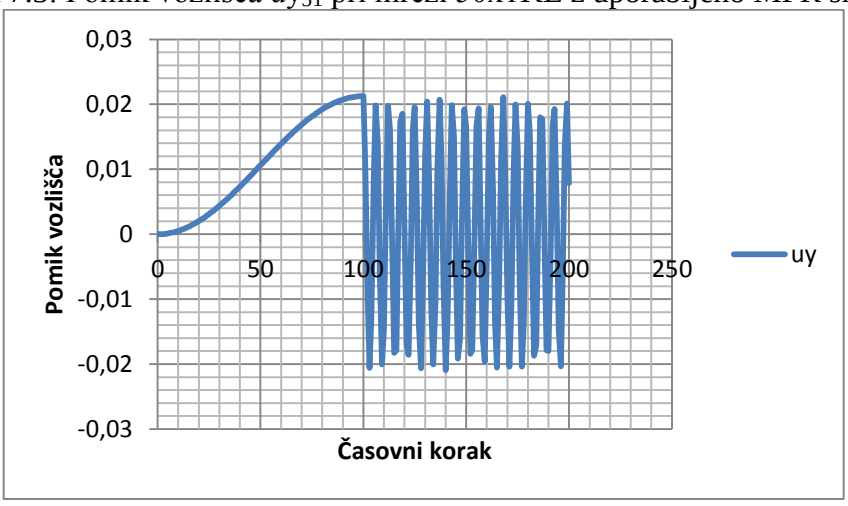
Preučimo, kako spreminjanje velikosti mreže (št. KE) vpliva na vrednost pomika v najbolj krajnem vozlišču konzole. Referenčna vrednost, s katero primerjamo, je $u_y = 0{,}024$.

Z zgoščevanjem računske mreže rezultati konvergirajo k referenčni vrednosti.

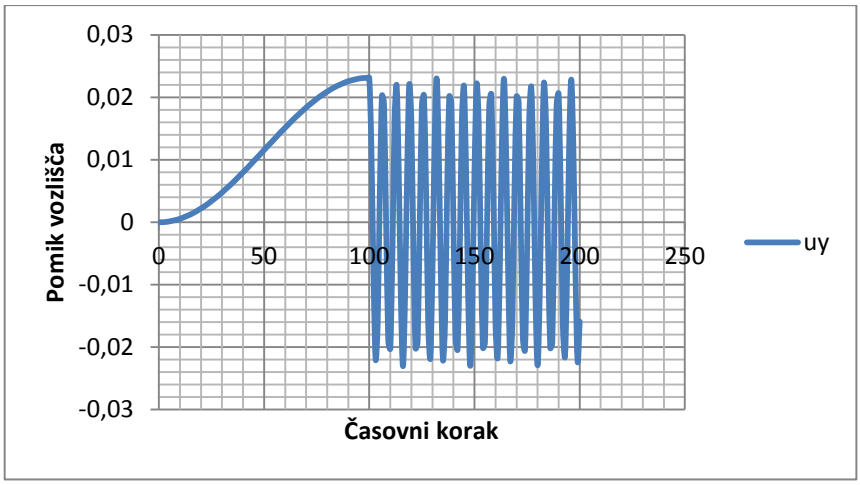
- Maksimalen pomik u_y pri računski mreži 50x1 KE znaša 0,016mm. Slika 7.3
- Maksimalen pomik u_y pri računski mreži 100x2 KE znaša 0,021mm. Slika 7.4
- Maksimalen pomik u_y pri računski mreži 200x4 KE znaša 0,024mm. Slika 7.5



Slika 7.3: Pomik vozlišča uy_{51} pri mreži 50x1KE z uporabljeno MPR shemo



Slika 7.4: Pomik vozlišča uy_{101} pri mreži 100x2KE z uporabljeno MPR shemo

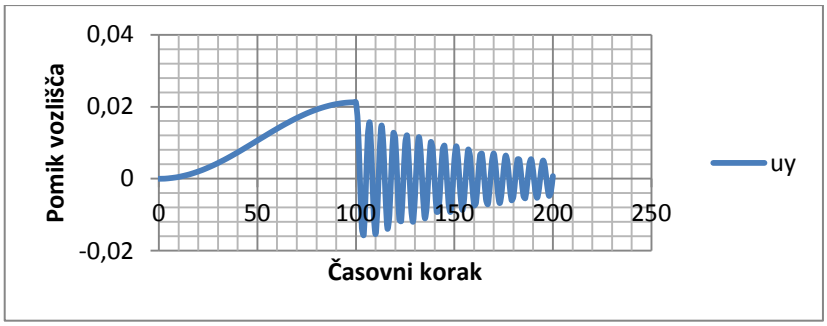


Slika 7.5: Pomik vozlišča uy_{201} pri mreži 200x4 KE z uporabljeno MPR shemo

7.3 Spreminjanje integracijskih shem

Na obnašanje KE vplivamo predvsem z izbiro integracijske sheme. Poglejmo si, kaj se dogaja s pomikom pri Bossakovi metodi z disipacijo visokih frekvenc (BAM). Uporabimo mrežo gostote 100x2 KE.

BAM integracijska shema deluje tako, da numerično duši visoke frekvence. Intenzivnost dušenja določimo z koeficientom $RoInf$, pri čemer je $0 < RoInf < 1$. Manjši $RoInf$ pomeni močnejše dušenje. Primer za $RoInf = 0{,}8$ prikazuje slika 7.6.



Slika 7.6: Pomik vozlišča uy101 pri mreži 100x2 KE z uporabljeno BAM shemo

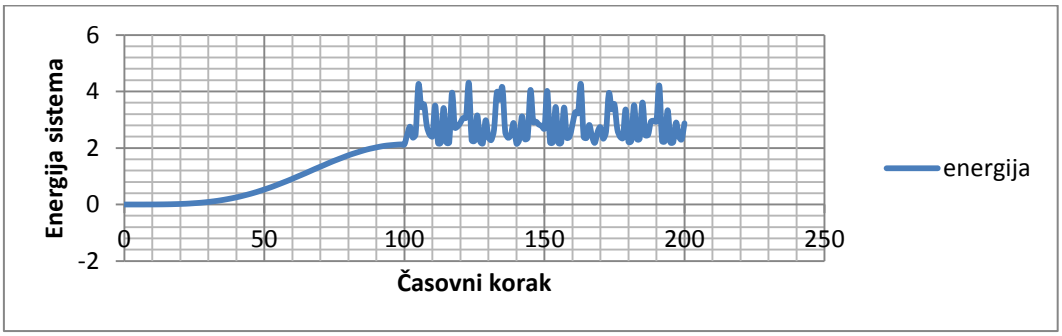
7.4 Spreminjanje časovnega koraka in opazovanje energije konzole

Zadnja stvar, ki jo bomo omenili in vpliva na obnašanje KE, je časovni korak dt. Pri implicitnih metodah, ki jih obravnavamo, je pomembno, da je časovni korak dovolj majhen, z razlogom, da je napaka čim manjša.

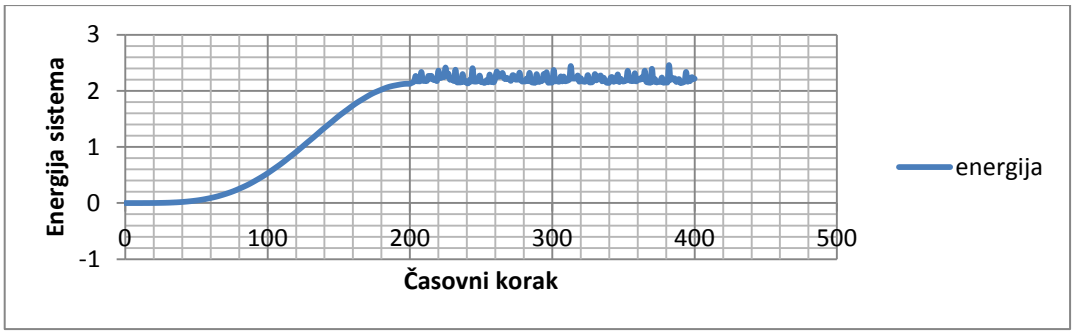
Poglejmo na primeru skupne energije konzole, kakšen doprinos ima sprememba časovnega koraka. Energija je v vsakem trenutku vsota elastične (notranje) in vztrajnostne sile. Energija se ohranja in bi teoretično morala biti konstantna.

Uporabimo mrežo 100x2 KE, MPR integracijsko shemo, sinusno obremenitev $R_{max} = 200N$ in s spreminjanjem časovnega koraka opazujmo, kaj se dogaja z energijo.

1. 200 časovnik korakov. Kot vidimo na sliki 7.7, je dobljen graf nepravilen, energija namreč po razbremenitvi konzole celo naraste. Poleg tega tudi neprestano niha vseskozi nad vrednostjo vložene energije. Sklepajmo, da je krivec zato premajhen časovni korak in ga povečajmo.
2. 400 časovnih korakov. Graf na sliki 7.8 nam da vedeti, da tudi 400 korakov v dveh sekundah ne da čisto pričakovane rešitve. Odgovor se skriva drugje.

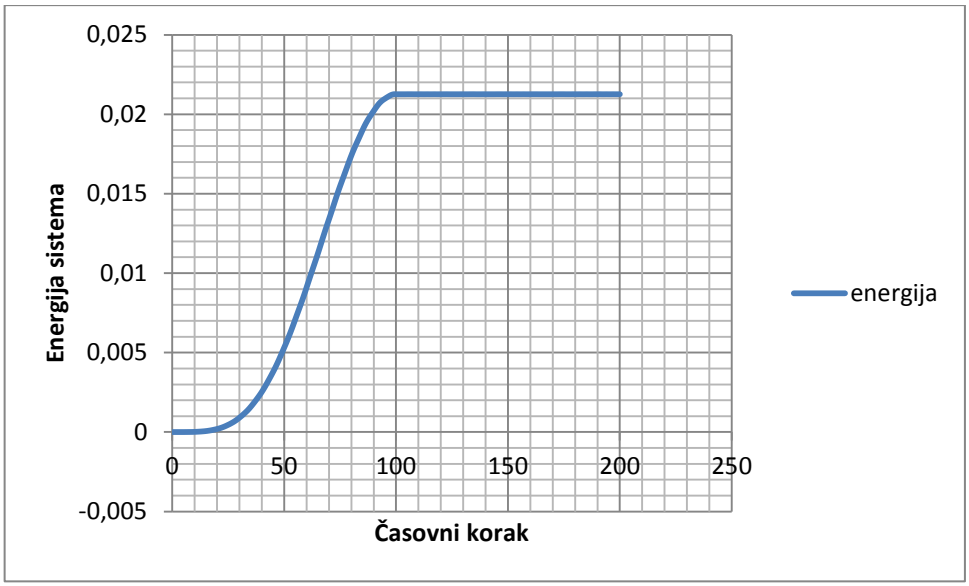


Slika 7.7: Energija sistema pri mreži 100x2 KE z uporabljeno MPR shemo in z 200 časovnimi koraki



Slika 7.8:Energija sistema pri mreži 100x2 KE z uporabljeno MPR shemo in z 400 časovnimi koraki

Problem z energijo ima numeričen izvor. Takšna težava se lahko pojavi pri analiziranju zelo togih teles, takšnih, kot je naša konzola. Problem je v tem, da so vztrajnostne sile, v nasprotju z elastičnimi, skoraj zanemarljive. Poskusimo povečati gostoto našega materiala, ki je jeklo, iz $7800\frac{kg}{m^3}$ na $780000\frac{kg}{m^3}$ z namenom, da povečamo vlogo vztrajnostnih sil. Poglejmo si rezultate na sliki 7.9.



Slika 7.9: Energija sistema pri mreži 100x2 KE z uporabljeno MPR shemo z 200 časovnimi koraki in namerno povečanimi vztrajnostnimi silami

Rezultati so sedaj takšni, kot smo jih pričakovali. Do podobnih rezultatov pridemo tudi pri drugih integracijskih shemah, ki vsebujejo dušenje visokih frekvenc. V normalnem stanju je dušenje premočno in se energija sistema izgublja, kar tudi nima fizikalnega smisla. S povečanjem vztrajnostnih sil, bodisi s povečanjem gostote materiala, bodisi z dodajanjem dodatne obremenitve na konzolo, pa postane energija ravna, konstantna premica.

V tem poglavju smo opozorili na dejstva, ki so ključna za pravilno interpretacijo rezultatov, ki jih dobimo s pomočjo MKE programov. V prvi vrsti se moramo zavedati, kakšen je naš KE, ki ga uporabimo za mreženje našega predmeta. Pomembno je tudi, kakšna je najmanjša gostota računske mreže, ki da še zadovoljive rezultate. Ključna je še pravilna uporaba integracijskih shem in pa zavedanje, da numerične metode ne obravnavajo telesa kot zveznega. Temu stanju se lahko približamo le z dovolj majhnim računskim korakom.

8 DISKUSIJA

Programiranje KE se je izkazalo za dobro izbrano temo. Preučili smo koncept delovanja diskretiziranih (ne zveznih) problemov. Zapisali smo ravnovesno enačbo za posamezen diferencialno majhen KE. Zapisali smo elastičen in kinetičen potencial. Z odvajanjem potenciala po prostorskih stopnjah smo dobili togostno matriko elementa.

Pri tem smo poleg strojniških znanj osvojili tudi mnogo programerskih, ki nam v prvi vrsti olajšajo delo oz. nam omogočajo reševanje problemov, ki bil nam drugače vzeli ogromno časa. Spoznali smo osnovno delovanje numeričnih metod in se naučili pravilno tolmačiti rezultate, ki jih le te dajejo.

9 SKLEP

Sprogramiran 4-vozliščni ravninski KE je bil uspešno testiran. KE daje ob ustreznih nastavitvah zadovoljive rezultate, pri malce slabših pa so lahko rezultati že fizikalno napačni. Že 8-vozliščni ravninski KE bi bil neizmerno bolj natančen. Tukaj pa se pokaže uporabnost takšnega programiranja, saj nas od našega KE pa do 8-vozliščnega KE loči največ pol ure dela. Programska koda ostane skoraj enaka, spremenijo se le interpolacijske funkcije, ker se poveča število vozlišč in prostorskih stopenj.

Takšno univerzalno programiranje ponuja številne možnosti za nadaljnje delo. Zamislimo si lahko KE poljubne oblike in dimenzionalnosti (1D, 2D, 3D) in ga z nekaj dodanega truda sprogramiramo. Natančnost KE imamo možnosti izboljšati z optimizacijskimi metodami.

SEZNAM UPORABLJENIH VIROV

[1] Beer G., Watson J.O.. *Introduction to finite and boundary element methods for engineers*. West Sussex: John Wiley & Sons, 1992

[2] Fish Jacob, Belytschko Ted. *A first course in finite elements*. West Sussex : John Wiley & Sons, 2007

[3] Kegl Marko. *Mehanika III*. Maribor : Fakulteta za strojništvo, 1998.

[4] Smith I. M., Griffiths D. V.. *Programming the finite element method*. West Sussex : John Wiley & Sons, 1988