UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
U NOVOM SADU

Hibridna softverska arhitektura kao podrška primeni harmonijski spojenog metoda konačnih traka

Petar Marić

doktorska disertacija

Novi Sad, 2015.

Zahvaljujem se svim članovima Komisije koji su svojim korisnim sugestijama doprineli da disertacija bude jasnija i preglednija. Posebno se zahvaljujem mentoru doc. dr Žarku Živanovu za nesebičnu podršku tokom izrade disertacije.

Najveću zahvalnost dugujem mojoj porodici na njihovom razumevanju i podršci.

Novi Sad, 2015.

Petar Marić

Sadržaj

Glava 1

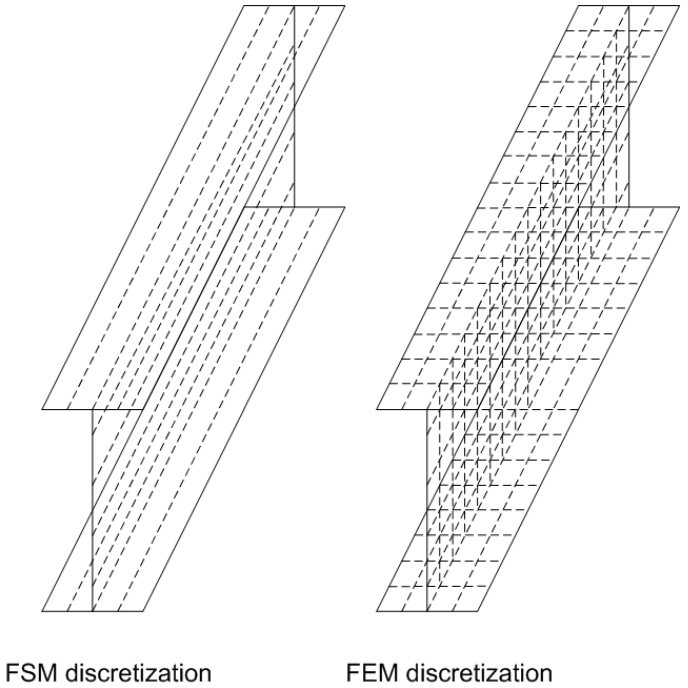
Uvod

1.1 Finite Strip Method

Metod konačnih traka (eng. *Finite Strip Method*, skr. *FSM*) predstavio je Y.K. Cheung 1968. godine [1, 2] kao specijalizaciju metoda konačnih elemenata (eng. *Finite Element Method* [3], skr. *FEM*).

FSM je u osnovi analitičko-numerički metod koji je prilagođen analizi građevinskih struktura kod kojih su svojstva materijala i geometrije konstantni duž jedne ose (longitudalne). Suštinska razlika između FSM i FEM je u načinu diskretizacije mreže, što je prikazano na slici 1.1. FEM diskretizuje mrežu i transverzalno i longitudinalno, dok je za FSM diskretizacija potrebna samo duž transverzalne ose. Za razliku od FEM-a, FSM može da svede dvodimenzionalni problem pravougaonih ploča na jednodimenzionalni problem, kroz izbor odgovarajućih aproksimativnih funkcija pomeranja. Usled smanjenja dimenzionalnosti problema upotreba FSM može dovesti do ušteda računarskih resursa i za red veličine u odnosu na FEM [5, 6].

Na početku je FSM razvijan za analizu konstrukcija sa pravilnim geometrijskim pločastim oblicima i jednostavnim graničnim uslovima. Kasnije je FSM proširen na analizu kružnih ploča [7], kosih ploča, poliedarskih konstrukcija i sandučastih mostova. Formulacijom problema svojstvenih vrednosti FSM može biti primenjen u problemima vibracija i stabilnosti ploča i ljuski [8].



Slika 1.1: Primer razlika u diskretizaciji mreže između FSM i FEM [4]

1.2 Harmonic Coupled Finite Strip Method

Harmonijski spojen metod konačnih traka (eng. *Harmonic Coupled Finite Strip Method*, skr. *HCFSM*) je proširenje FSM-a čije je osnove predstavio Milašinović 1997. godine [8], a formalno ga je imenovao 2011. godine [6] kao zaseban metod. U kasnijim radovima [4,9–17] analizirana su svojstva HCFSM i predstavljene njegove prednosti u odnosu na FSM.

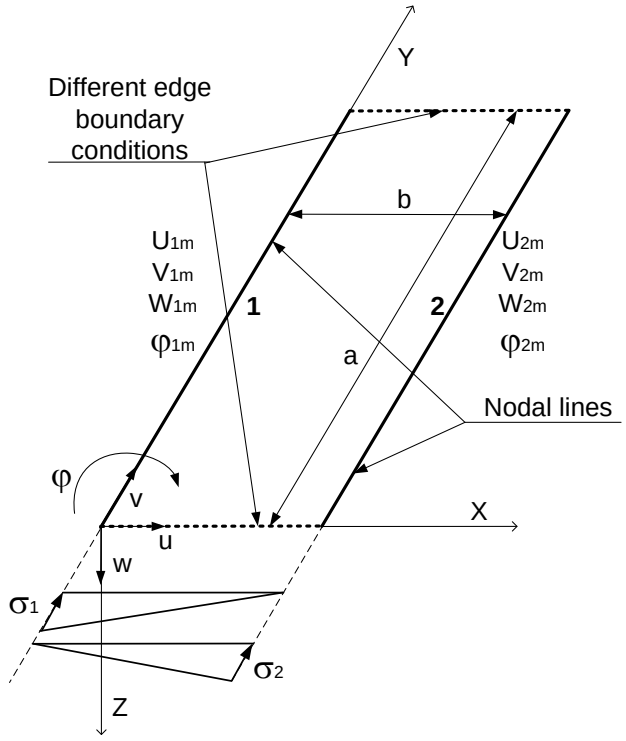
HCFSM tokom kreiranja geometrijske matrice krutosti uzima u obzir i značaj međusobne spojenosti harmonika. Konvencionalni FSM ignoriše ovu interakciju među harmonicima i ne može se koristiti za analizu građevinskih struktura kod kojih su granični uslovi složeniji, jer tada u integralima koji čine matricu krutosti više ne važe svojstva ortogonalnosti [4, 14].

1.2.1 Aproksimativne funkcije

Na FSM/HCFSM se može gledati i kao na spoj klasičnog Ritz-ovog metoda i metoda konačnih elemenata [5, 8], tako da se opšta forma funkcije pomeranja f može napisati kao proizvod polinoma i trigonometrijskih funkcija [6, 8, 10]:

$$f = \mathbf{Cq} = \sum_{m=1}^{r} Y_m(y) \sum_{k=1}^{c} \mathbf{N}_k(x)\mathbf{q}_{km} \tag{1.1}$$

gde $Y_m(y)$ predstavljaju funkcije iz Ritz-ovog metoda, a $\mathbf{N}_k(x)$ su polinomi u svojstvu interpolacionih funkcija iz FEM.



Slika 1.2: Primer konačne trake koja se koristi za rešavanje problema stabilnosti i savijanja ploča u FSM i HCFSM

Za konačnu traku sa slike 1.2 koriste se sledeće aproksimativne

funkcije pomeranja [6, 8, 10]:

$$\begin{aligned} u_0 &= \sum_{m=1}^{r} Y_{um}^{u} \mathbf{N}_u^u \mathbf{q}_{um}^u \\ &= \sum_{m=1}^{r} Y_{um}^{u} \begin{bmatrix} 1 - \frac{x}{b} & \frac{x}{b} \end{bmatrix} \mathbf{q}_{um}^u \\ v_0 &= \sum_{m=1}^{r} \frac{a}{m\pi} Y_{um}^{v} \mathbf{N}_u^v \mathbf{q}_{um}^v \\ &= \sum_{m=1}^{r} \frac{a}{m\pi} Y_{um}^{v} \begin{bmatrix} 1 - \frac{x}{b} & \frac{x}{b} \end{bmatrix} \mathbf{q}_{um}^v \\ w &= \sum_{m=1}^{r} Y_{wm} \mathbf{N}_w \mathbf{q}_{wm} \\ &= \sum_{m=1}^{r} Y_{wm} \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} \mathbf{q}_{wm} \end{aligned} \tag{1.2}$$

gde su polinomi interpolacione funkcije $\mathbf{N}_k(x)$:

$$\begin{aligned} N_1(x) &= 1 - 3\left(\frac{x}{b}\right)^2 + 2\left(\frac{x}{b}\right)^3 \\ N_2(x) &= b\left[\frac{x}{b} - 2\left(\frac{x}{b}\right)^2 + \left(\frac{x}{b}\right)^3\right] \\ N_3(x) &= 3\left(\frac{x}{b}\right)^2 - 2\left(\frac{x}{b}\right)^3 \\ N_4(x) &= b\left[-\left(\frac{x}{b}\right)^2 + \left(\frac{x}{b}\right)^3\right] \end{aligned} \tag{1.3}$$

dok su vektori pomeranja u čvornim linijama trake:

$$\begin{aligned} \mathbf{q}_{wm} &= \begin{bmatrix} w_{1m} \\ \phi_{1m} \\ w_{2m} \\ \phi_{2m} \end{bmatrix} \\ \mathbf{q}_{um}^u &= \begin{bmatrix} u_{1m} \\ u_{2m} \end{bmatrix} \\ \mathbf{q}_{um}^v &= \begin{bmatrix} v_{1m} \\ v_{2m} \end{bmatrix} \end{aligned} \tag{1.4}$$

Prilikom rešavanja problema stabilnosti, jednačine svojstvenih oblika $Y_m(y)$ izvode se iz rešenja sledeće diferencijalne jednačine [8, 18–20]:

$$\frac{d^4 Y_m(y)}{dy^4} - Y_m(y) \left(\frac{\mu_m^4}{a^4} \right) = 0 \tag{1.5}$$

Tada partikularno rešenje diferencijalne jednačine 1.5 koje čini jednačinu svojstvenih oblika zavisi od odabranog graničnog uslova, koji su navedeni u daljem tekstu.

1.2.1.1 Oba kraja slobodno oslonjena (LJ=1)

Jednačina svojstvenih oblika se za ovaj granični uslov definiše kao [8, 18–20]:

$$Y_m(y) = \sin\left(\frac{\mu_m y}{a}\right) \tag{1.6}$$

u kojoj se μ_m određuje rešavanjem karakteristične jednačine:

$$\sin(\mu_m) = 0 \tag{1.7}$$

pri čemu su korenovi karakteristične jednačine:

$$\mu_m = m\pi \tag{1.8}$$

1.2.1.2 Oba kraja uklještena (LJ=2)

Jednačina svojstvenih oblika je [8, 18–20]:

$$\begin{aligned} Y_m(y) &= \sin\left(\frac{\mu_m y}{a}\right) - \sinh\left(\frac{\mu_m y}{a}\right) \\ &\quad - \frac{\sin(\mu_m) - \sinh(\mu_m)}{\cos(\mu_m) - \cosh(\mu_m)} \left[\cos\left(\frac{\mu_m y}{a}\right) - \cosh\left(\frac{\mu_m y}{a}\right)\right] \\ &= \frac{\left[\begin{array}{l} \sin\left(\frac{\mu_m y}{a}\right)\cos(\mu_m) - \sin\left(\frac{\mu_m y}{a}\right)\cosh(\mu_m) \\ -\sinh\left(\frac{\mu_m y}{a}\right)\cos(\mu_m) + \sinh\left(\frac{\mu_m y}{a}\right)\cosh(\mu_m) \\ -\cos\left(\frac{\mu_m y}{a}\right)\sin(\mu_m) + \cosh\left(\frac{\mu_m y}{a}\right)\sin(\mu_m) \\ +\cos\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) - \cosh\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) \end{array}\right]}{\cos(\mu_m) - \cosh(\mu_m)} \end{aligned} \tag{1.9}$$

u kojoj se μ_m određuje rešavanjem karakteristične jednačine:

$$\cos(\mu_m)\cosh(\mu_m) = 1 \tag{1.10}$$

pri čemu su korenovi karakteristične jednačine u literaturi obično dati ili u obliku matematičkog niza [8, 19]:

$$\mu_m = 4.7300, 7.8532, \ldots, (2m+1)\frac{\pi}{2} \tag{1.11}$$

ili dati samo kao aproksimativno rešenje u domenu, kroz analitički oblik opšteg člana niza, tj. korena [20]:

$$\mu_m = (2m+1)\frac{\pi}{2} \tag{1.12}$$

Međutim, iz analitičkog oblika opšteg člana niza 1.12 dobijaju se sledeće vrednosti korenova (zaokružene na 4 decimalna mesta):

$$\hat{\mu}_1 = 4.7124,\ \hat{\mu}_2 = 7.8540 \tag{1.13}$$

koje odudaraju od numeričkih vrednosti datih u literaturi (1.11). U okviru sekcije 2.1 analiziraće se pomenute razlike, posledice korišćenja analitičkog oblika opšteg člana niza (u daljem tekstu „aproksimativno analitičko rešenje karakteristične jednačine") i biće dati predlozi rešenja uočenih problema.

1.2.1.3 Jedan kraj uklješten, drugi slobodan (LJ=3)

Jednačina svojstvenih oblika je [8, 18–20]:

$$\begin{aligned} Y_m(y) &= \sin\left(\frac{\mu_m y}{a}\right) - \sinh\left(\frac{\mu_m y}{a}\right) \\ &\quad - \frac{\sin(\mu_m)+\sinh(\mu_m)}{\cos(\mu_m)+\cosh(\mu_m)}\left[\cos\left(\frac{\mu_m y}{a}\right) - \cosh\left(\frac{\mu_m y}{a}\right)\right] \\ &= \frac{\left[\begin{array}{c} \sin\left(\frac{\mu_m y}{a}\right)\cos(\mu_m) + \sin\left(\frac{\mu_m y}{a}\right)\cosh(\mu_m) \\ -\sinh\left(\frac{\mu_m y}{a}\right)\cos(\mu_m) - \sinh\left(\frac{\mu_m y}{a}\right)\cosh(\mu_m) \\ -\cos\left(\frac{\mu_m y}{a}\right)\sin(\mu_m) + \cosh\left(\frac{\mu_m y}{a}\right)\sin(\mu_m) \\ -\cos\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) + \cosh\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) \end{array}\right]}{\cos(\mu_m)+\cosh(\mu_m)} \end{aligned} \tag{1.14}$$

u kojoj se μ_m određuje rešavanjem karakteristične jednačine:

$$\cos(\mu_m)\cosh(\mu_m) = -1 \tag{1.15}$$

pri čemu je aproksimativno analitičko rešenje karakteristične jednačine:

$$\mu_m = (2m-1)\frac{\pi}{2} \tag{1.16}$$

U literaturi obično su date i numeričke vrednosti prvih nekoliko korenova karakteristične jednačine:

$$\mu_1 = 1.8750,\ \mu_2 = 4.6940 \tag{1.17}$$

međutim iz aproksimativno analitičkog rešenja 1.16 dobijaju se sledeće vrednosti korenova (zaokružene na 4 decimalna mesta):

$$\hat{\mu}_1 = 1.5708,\ \hat{\mu}_2 = 4.7124 \tag{1.18}$$

koje odudaraju od numeričkih vrednosti datih u literaturi (1.17), kao što je predočeno u sekciji 1.2.1.2.

1.2.1.4 Jedan kraj slobodno oslonjen, drugi uklješten (LJ=4)

Jednačina svojstvenih oblika je [8, 18–20]:

$$\begin{aligned} Y_m(y) &= \sin\left(\frac{\mu_m y}{a}\right) - \frac{\sin(\mu_m)}{\sinh(\mu_m)}\sinh\left(\frac{\mu_m y}{a}\right) \\ &= \frac{\sin\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) - \sinh\left(\frac{\mu_m y}{a}\right)\sin(\mu_m)}{\sinh(\mu_m)} \end{aligned} \tag{1.19}$$

u kojoj se μ_m određuje rešavanjem karakteristične jednačine:

$$\tan(\mu_m) - \tanh(\mu_m) = 0 \tag{1.20}$$

pri čemu je aproksimativno analitičko rešenje karakteristične jednačine:

$$\mu_m = (4m+1)\frac{\pi}{4} \tag{1.21}$$

U literaturi obično su date i numeričke vrednosti prvih nekoliko korenova karakteristične jednačine:

$$\mu_1 = 3.9266,\ \mu_2 = 7.0685 \tag{1.22}$$

međutim iz aproksimativno analitičkog rešenja 1.21 dobijaju se sledeće vrednosti korenova (zaokružene na 4 decimalna mesta):

$$\hat{\mu}_1 = 3.9270,\ \hat{\mu}_2 = 7.0686 \tag{1.23}$$

koje odudaraju od numeričkih vrednosti datih u literaturi (1.22), kao što je predočeno u sekciji 1.2.1.2.

1.2.1.5 Jedan kraj slobodno oslonjen, drugi slobodan (LJ=5)

Jednačina svojstvenih oblika je [8, 18–20]:

$$\begin{aligned} Y_1(y) &= \frac{y}{a} \\ Y_{m>1}(y) &= \sin\left(\frac{\mu_m y}{a}\right) + \frac{\sin(\mu_m)}{\sinh(\mu_m)}\sinh\left(\frac{\mu_m y}{a}\right) \\ &= \frac{\sin\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) + \sinh\left(\frac{\mu_m y}{a}\right)\sin(\mu_m)}{\sinh(\mu_m)} \end{aligned} \tag{1.24}$$

u kojoj se μ_m određuje rešavanjem karakteristične jednačine:

$$\mu_m : \begin{cases} \mu_1 = 1 & , m = 1 \\ \tan(\mu_m) - \tanh(\mu_m) = 0 & , m > 1 \end{cases} \tag{1.25}$$

pri čemu je aproksimativno analitičko rešenje karakteristične jednačine:

$$\mu_m : \begin{cases} \mu_1 = 1 & , m = 1 \\ \mu_m = (4m-3)\dfrac{\pi}{4} & , m > 1 \end{cases} \tag{1.26}$$

U literaturi obično su date i numeričke vrednosti prvih nekoliko korenova karakteristične jednačine:

$$\mu_2 = 3.9266,\ \mu_3 = 7.0685, \tag{1.27}$$

međutim iz aproksimativno analitičkog rešenja 1.26 dobijaju se sledeće vrednosti korenova (zaokružene na 4 decimalna mesta):

$$\hat{\mu}_2 = 3.9270,\ \hat{\mu}_3 = 7.0686 \tag{1.28}$$

koje odudaraju od numeričkih vrednosti datih u literaturi (1.27), kao što je predočeno u sekciji 1.2.1.2.

1.2.1.6 Oba kraja slobodna (LJ=6)

Jednačina svojstvenih oblika je [8, 18–20]:

$$
\begin{aligned}
Y_1(y) &= 1 \\
Y_2(y) &= 1 - \frac{2y}{a} \\
Y_{m>2}(y) &= \sin\left(\frac{\mu_m y}{a}\right) + \sinh\left(\frac{\mu_m y}{a}\right) \\
&\quad - \frac{\sin(\mu_m) - \sinh(\mu_m)}{\cos(\mu_m) - \cosh(\mu_m)} \left[\cos\left(\frac{\mu_m y}{a}\right) + \cosh\left(\frac{\mu_m y}{a}\right)\right] \\
&= \frac{\left[\begin{array}{l}
\sin\left(\frac{\mu_m y}{a}\right)\cos(\mu_m) - \sin\left(\frac{\mu_m y}{a}\right)\cosh(\mu_m) \\
+\sinh\left(\frac{\mu_m y}{a}\right)\cos(\mu_m) - \sinh\left(\frac{\mu_m y}{a}\right)\cosh(\mu_m) \\
-\cos\left(\frac{\mu_m y}{a}\right)\sin(\mu_m) - \cosh\left(\frac{\mu_m y}{a}\right)\sin(\mu_m) \\
+\cos\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m) + \cosh\left(\frac{\mu_m y}{a}\right)\sinh(\mu_m)
\end{array}\right]}{\cos(\mu_m) - \cosh(\mu_m)}
\end{aligned}
\tag{1.29}
$$

u kojoj se μ_m određuje rešavanjem karakteristične jednačine:

$$
\mu_m : \begin{cases} \mu_1 = 0 & , m = 1 \\ \mu_2 = 1 & , m = 2 \\ \cos(\mu_m)\cosh(\mu_m) = 1 & , m > 2 \end{cases} \tag{1.30}
$$

pri čemu je aproksimativno analitičko rešenje karakteristične jednačine:

$$
\mu_m : \begin{cases} \mu_1 = 0 & , m = 1 \\ \mu_2 = 1 & , m = 2 \\ \mu_m = (2m-3)\dfrac{\pi}{2} & , m > 2 \end{cases} \tag{1.31}
$$

U literaturi obično su date i numeričke vrednosti prvih nekoliko korenova karakteristične jednačine:

$$
\mu_3 = 4.7300, \;\; \mu_4 = 7.8532, \tag{1.32}
$$

međutim iz aproksimativno analitičkog rešenja 1.31 dobijaju se sledeće vrednosti korenova (zaokružene na 4 decimalna mesta):

$$\hat{\mu}_3 = 4.7124,\ \hat{\mu}_4 = 7.8540 \tag{1.33}$$

koje odudaraju od numeričkih vrednosti datih u literaturi (1.32), kao što je predočeno u sekciji 1.2.1.2.

1.2.2 Integrali

Opšti oblik integrala korišćenih u HCFSM [8] je:

$$I_\alpha = \int_0^a F\Big(Y_m(y),\ Y_n(y),\ Y_t(y),\ Y_v(y)\Big)dy \tag{1.34}$$

HCFSM definiše i koristi ukupno 113 integrala [8], pri čemu će u ovoj tezi poseban naglasak biti na integralima potrebnim za rešavanje problema stabilnosti u HCFSM:

$$I_1 = \int_0^a Y_m(y)Y_n(y)dy \tag{1.35}$$

$$I_2 = \int_0^a Y'_m(y)Y'_n(y)dy \tag{1.36}$$

$$I_3 = \int_0^a Y''_m(y)Y_n(y)dy \tag{1.37}$$

$$I_4 = \int_0^a Y'_m(y)Y'_n(y)dy \tag{1.38}$$

$$I_5 = \int_0^a Y_m(y)Y''_n(y)dy \tag{1.39}$$

$$I_6 = \int_0^a Y'_m(y)Y'_n(y)dy \tag{1.40}$$

$$I_7 = \int_0^a Y_m''(y) Y_n''(y) dy \tag{1.41}$$

$$I_8 = \int_0^a Y_m'(y) Y_n'(y) dy \tag{1.42}$$

$$I_{21} = \int_0^a Y_m(y) Y_n(y) dy \tag{1.43}$$

$$I_{22} = \int_0^a Y_m''(y) Y_n(y) dy \tag{1.44}$$

$$I_{23} = \int_0^a Y_m(y) Y_n''(y) dy \tag{1.45}$$

$$I_{24} = \int_0^a Y_m''(y) Y_n''(y) dy \tag{1.46}$$

$$I_{25} = \int_0^a Y_m'(y) Y_n'(y) dy \tag{1.47}$$

1.3 Tema istraživanja

1.3.1 Predmet i problem

Karakteristične jednačine složenijih graničnih uslova se na računaru ne mogu dovoljno tačno rešiti putem direktnog uvrštavanja aproksimativno analitičkog rešenja u odgovarajuću karakterističnu jednačinu, uz korišćenje standardne IEEE 754 binary64 radne preciznosti.

Tako dobijeni korenovi karakteristične jednačine značajno i nepovoljno utiču na tačnost rešavanja jednačina svojstvenih oblika, izračunavanja određenih integrala, kao i celokupnih proračuna.

1.3.2 Cilj

Pronaći ili osmisliti metod koji će na pouzdan način odrediti korenove karakterističnih jednačina odgovarajućeg stepena tačnosti za sve granične uslove date u sekciji 1.2.1. Uporednom analizom jednoznačno ukazati na neophodnost korišćenja odabranog metoda.

Razviti prototip softverskog rešenja koje na računaru uspešno rešava opisani problem i predstavlja referentnu implementaciju odabranog metoda. Dekomponovati razvijeno rešenje u veći broj međusobno nezavisnih Open Source [21] projekata koji se potom slobodno (eng. *Free Software*) mogu koristiti i u druge svrhe.

Razviti podsistem za automatsku verifikaciju koji će rigorozno verifikovati karakteristike odabranog metoda i njegove referentne implementacije, za sve opisane granične uslove.

Glava 2

Problem nedovoljne tačnosti

2.1 Rešavanje karakterističnih jednačina

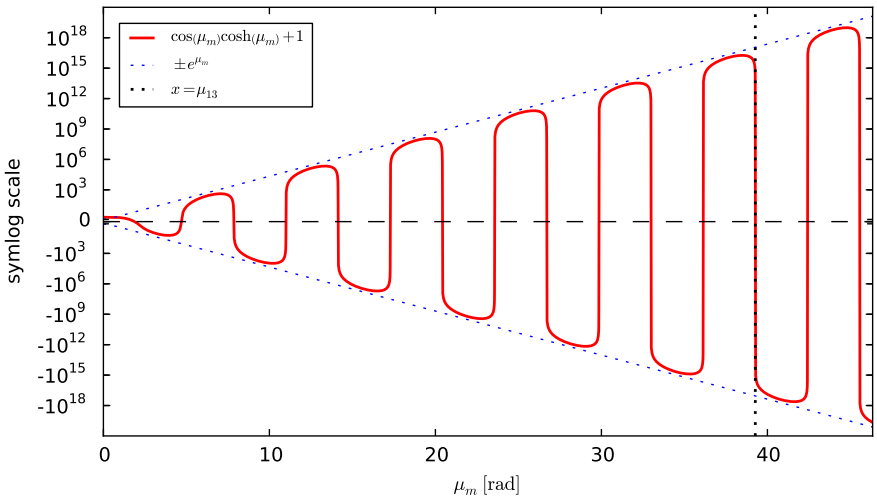
Za sve granične uslove date u sekciji 1.2.1 neophodno je tačno rešiti odgovarajuće karakteristične jednačine za svako μ_m gde je $1 \leq m \leq 100$, pošto se tako dobijene vrednosti μ_m koriste tokom rešavanja jednačina svojstvenih oblika $Y_m(y)$. Bazne funkcije $Y_m(y)$ se potom, kroz integrale opisane u sekciji 1.2.2, koriste za rešavanje problema slobodne vibracije. Kod složenijih graničnih uslova, gde cosh direktno figuriše u karakterističnim jednačinama, bazne funkcije $Y_m(y)$ su (usled svog oblika) izrazito osetljive na pertubacije u vrednostima μ_m. Tada tačno određeni μ_m imaju izuzetan značaj na tačnost celokupnih proračuna.

Granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3, opisan u sekciji 1.2.1.3) biće osnova za dalju diskusiju o problemima tačnosti koji proizilaze tokom rešavanja karakterističnih jednačina. Karakteristična jednačina odabranog graničnog uslova veoma pogodno ukazuje na ozbiljnost problema nedovoljne tačnosti i prikaz prednosti hibridnog metoda predstavljenog u ovoj tezi, pri čemu je u okviru naučne zajednice [8,18] ovaj granični uslov već prepoznat kao zahtevan za rešavanje.

Ako se karakteristična jednačina odabranog graničnog uslova, definisana u jednačini 1.15, transformiše u odgovarajuću kanonsku formu:

$$f(\mu_m) = \cos(\mu_m)\cosh(\mu_m) + 1\,, \quad f(\mu_m) = 0\,. \tag{2.1}$$

tada tako dobijene vrednosti μ_m postaju korenovi karakteristične funkcije f. Time se problem rešavanja karakterističnih jednačina zamenjuje problemom određivanja pozitivnih realnih korenova za transformisanu funkciju f.



Slika 2.1: Karakteristična funkcija za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3), pri čemu je funkcija sračunata u IEEE 754 binary64 preciznosti. Zarad preglednosti prikazani su korenovi za samo prvih 15 modova

Analizom *symlog*[1] grafika karakteristične funkcije prikazanog na slici 2.1 uočavaju se neka njena interesantna svojstva. Naime, karakteristična funkcija u okolini svojih korenova ($\mu_m \pm \varepsilon$ oblast) naglo menja svoj znak uz izrazito visok prvi izvod i gotovo vertikalan pad/rast funkcije, čime verno oponaša *signum* funkciju (mat. sgn) u neposrednoj okolini svojih korenova – što se može i vizuelno potvrditi njenim bliskim podudaranjem sa $x = \mu_{13}$

[1] *symlog* skala [22, 23] je simetrična logaritamska skala koja može da prikaže i pozitivne i negativne vrednosti, dok u okolini 0 oponaša linearnu skalu

vertikalom u neposrednoj okolini njenog 13.-og korena. Uz to lokalni ekstremi karakteristične funkcije u okolini njenih korenova se asimptotski približavaju $\pm e^{\mu_m}$, značajno uvećavajući greške prilikom određivanja korenova karakteristične funkcije - što se može vizuelno potvrditi „visinom" sa koje funkcija pada tokom prilaska svom 13.-om korenu. Usled toga, sa svakim narednim modom greška određivanja korenova karakteristične funkcije raste gotovo eksponencijalno i dostiže $\approx 10^{121}$ za $m = 100$, čineći problem rešavanja karakterističnih jednačina numerički nestabilnim.

U daljem tekstu biće razmatrani različiti pristupi u rešavanju karakterističnih jednačina i biće pokazano da se problem ne može rešiti direktnim uvrštavanjem aproksimativno analitičkog rešenja u karakterističnu jednačinu uz korišćenje IEEE 754 binary64 radne preciznosti. Nakon toga biće predstavljena *hibridna metoda* i analiziran njen učinak.

2.1.1 Pristupi u rešavanju karakterističnih jednačina

Kada se diskutuje o greškama čiji je uzrok neka vrsta aproksimacije obično se u razmatranje uzimaju apsolutna i relativna greška [24]. Za datu tačnu vrednost v i njenu aproksimaciju v_{approx} apsolutna greška se definiše kao:

$$\epsilon = |v - v_{\text{approx}}| \tag{2.2}$$

Ako je $v \neq 0$ relativna greška se definiše kao:

$$\eta = \frac{\epsilon}{|v|} = \left|\frac{v - v_{\text{approx}}}{v}\right| = \left|1 - \frac{v_{\text{approx}}}{v}\right| \tag{2.3}$$

u suprotnom je relativna greška nedefinisana.

Tačna vrednost v se, za problem određivanja korenova karakteristične funkcije, dobija uvrštavanjem tačne vrednosti korena μ_m u jednačinu 2.1, čime važi sledeća jednakost:

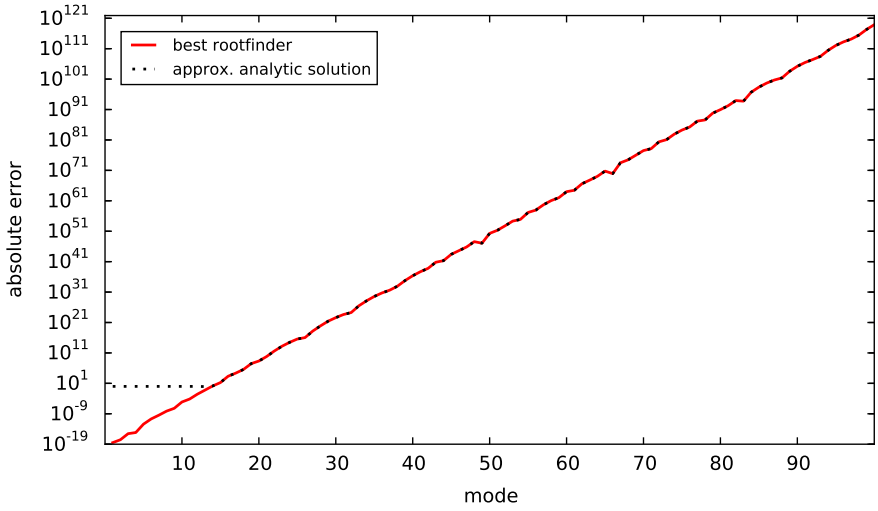
$$v = f(\mu_m) = 0 \tag{2.4}$$

Usled toga relativna greška je nedefinisana i ne može se koristiti za diskusiju o greškama aproksimacije prilikom određivanja korenova karakteristične funkcije, već će se za te potrebe koristiti isključivo apsolutna greška (u daljem tekstu „greška").

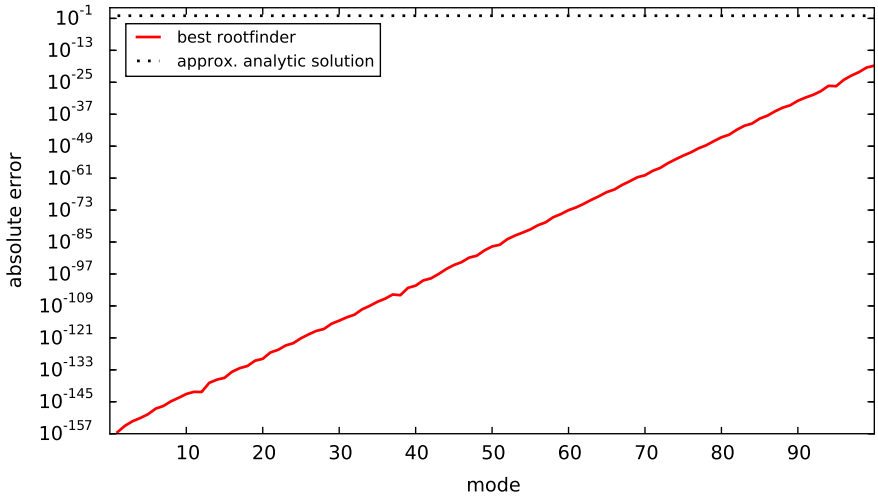
U slučaju graničnog uslova gde je jedan kraj uklješten a drugi slobodan, uvrštavanjem aproksimativno analitičkog rešenja 1.16 u pripadajuću karakterističnu jednačinu 1.15 ne dobijaju se tačni korenovi, što i je predočeno na slici 2.2. Analitičko rešenje propušta da daje tačna rešenja i u režimu visoke preciznosti, ostvareno kroz uvođenje proizvoljno-precizne aritmetike u pokretnom zarezu (eng. *arbitrary-precision floating point arithmetic*), što se može i vizuelno potvrditi na slici 2.3. Prikazane greške tokom rešavanja karakterističnih jednačina su se pokazale previsokim da bi aproksimativno analitičko rešenje bilo prihvatljivo za upotrebu, a naročito u slučaju viših modova. Uz to *Computer Algebra System* programi kao što su SymPy [25] i Mathematica nisu bili u mogućnosti da simbolički reše datu karakterističnu jednačinu i proizvedu njeno tačno analitičko rešenje.

Još je interesantnije ponašanje karakteristične funkcije za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4, opisan u sekciji 1.2.1.4), prikazano na slici 2.4. U odnosu na granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) karakteristična funkcija ponaša se upravo suprotno: korenovi dobijeni iz aproksimativno analitičkog rešenja se stabilizuju u višim modovima, dok su za niže modove vidno netačni (u odnosu na numerički optimizovano rešenje predstavljeno u daljem tekstu). Moguće je da se iz ovih razloga u literaturi obično i daju numeričke vrednosti korenova za prvih nekoliko modova. U literaturi često nisu date numeričke vrednosti korenova za više modove, pošto se smatralo da će korenovi dobijeni iz aproksimativno analitičkog rešenja konvergirati u tačna rešenja sa višim modovima, čime bi numerička optimizacija korenova bila bespotrebna.

U ovoj tezi je pokazano da, za više modove nekih od složenijih graničnih uslova, korenovi dobijeni iz aproksimativno analitičkog rešenja neće konvergirati ka tačnom rešenju, bez obzira na dodatno povećanje preciznosti u kojoj su vršeni proračuni. Primera radi, za prethodno odabrani granični uslov gde je jedan kraj

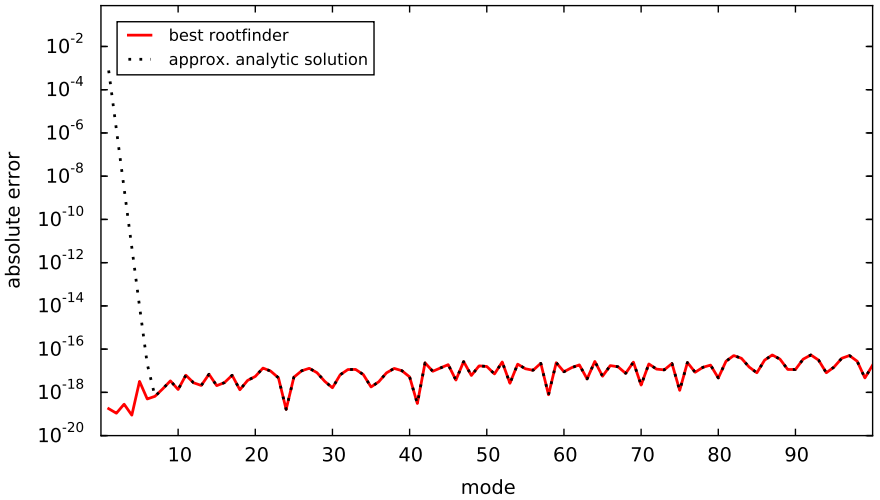


Slika 2.2: Greške prilikom određivanja korenova karakteristične funkcije za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3), pri čemu su proračuni vršeni u IEEE 754 binary64 preciznosti. Na krivoj „best rootfinder" prikazane su greške turnirski baziranog takmičenja između numeričkih određivača korenova karakteristične jednačine, opisanog na strani 20



Slika 2.3: Greške prilikom određivanja korenova karakteristične funkcije za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3), pri čemu su proračuni vršeni u režimu visoke preciznosti na 155 decimalnih mesta (podrazumevana radna preciznost za `beam_integrals` projekat, detaljnije na strani 26)

uklješten a drugi slobodan (LJ=3) kroz slike 2.2 i 2.3 se vizuelno mogu uporediti greške korenova dobijenih iz aproksimativno analitičkog rešenja, te potvrditi da oni neće konvergirati ka tačnom rešenju ni sa povećanjem preciznosti u kojoj su vršeni proračuni, pošto i tada greška konvergira ka konstanti $E \neq 0$.



Slika 2.4: Greške prilikom određivanja korenova karakteristične funkcije za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4), pri čemu su proračuni vršeni u IEEE 754 binary64 preciznosti

Na osnovu ove analize utvrđeno je da će numerički određivači korenova (eng. *numerical root-finding solvers*) pružiti značajno bolje rezultate u odnosu na aproksimativno analitičko rešenje, pri čemu će u predstavljenom hibridnom metodu numerički određivači koristiti aproksimativno analitičko rešenje kao osnovu svoje početne pretpostavke približne lokacije traženog korena.

Postoji veći broj numeričkih određivača korenova (i srodnih im algoritama), gde svaki od njih poseduje određene karakteristike iz kojih proizilaze njihove prednosti i mane prilikom rešavanja konkretnih problema. U opštem slučaju nije lako unapred utvrditi koji će od numeričkih određivača korenova dati optimalan rezultat, uz razuman utrošak računarskih resursa. Umesto toga, rešenje predstavljeno u ovoj tezi će za svaki traženi koren organizovati turnir između svih podržanih numeričkih određivača, pri čemu će pobednik turnira biti određivač čiji numerički

optimizovan koren ima najmanju grešku. Zarad veće sigurnosti u odabiru optimalnog korena i boljih performansi sistem će sve turnire uvek organizovati međusobno nezavisno, za svaku zasebnu kombinaciju graničnog uslova i moda.

Nažalost, na osnovu slike 2.2 se može vizuelno utvrditi da numerički određivači korenova nisu u mogućnosti da proizvedu tačne korenove za više modove u slučaju složenijih graničnih uslova, čak ni kada se međusobno takmiče kroz organizovane turnire. U sledećem delu analiziraće se uzroci ove pojave i opisati odabran način rešavanja.

2.1.2 Zašto IEEE 754 standard ne pruža dovoljnu radnu preciznost za problem rešavanja karakterističnih jednačina

Poznato je da korišćenje aritmetike u pokretnom zarezu može dovesti do smanjenja tačnosti rezultata proračuna, usled njene konačne radne preciznosti. Ako se određivanje korenova karakteristične funkcije vrši u IEEE 754 binary64 [26,27] preciznosti tada će ε okolina oko korena μ_m biti veoma mala (mašinsko epsilon je $\approx 1.11 \times 10^{-16}$ [28–30]).

Međutim i tako malo ε će, usled prirode same karakteristične funkcije, uzrokovati da greška određivanja korenova karakteristične funkcije raste gotovo eksponencijalno sa svakim narednim modom i dostigne $\approx 10^{121}$ za $m = 100$, što se može i vizuelno potvrditi na slici 2.2. Uz to, usled nedovoljne radne preciznosti proračuna ni analitički određeni korenovi niti numerički određivači korenova ne uspevaju da proizvedu tačne korenove za više modove.

Definišimo maksimalnu grešku (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije kao:

$$\text{Error}_{\max} = \max\{|f(\hat{\mu}_m)| : m \in (1, \ldots, 100)\} \tag{2.5}$$

gde je $\hat{\mu}_m$ dobijena aproksimirana vrednost korena μ_m. Tada se na osnovu slike 2.7 može vizuelno utvrditi da se ni uz korišćenje viših preciznosti definisanih IEEE 754 standardom (IEEE 754

binary80 i IEEE 754 binary128) neće dobiti tačni korenovi, usled nedovoljne radne preciznosti proračuna. Ovaj problem je dobro poznat u literaturi [31] i obično se rešava korišćenjem režima visoke preciznosti, kroz uvođenje proizvoljno-precizne aritmetike u pokretnom zarezu.

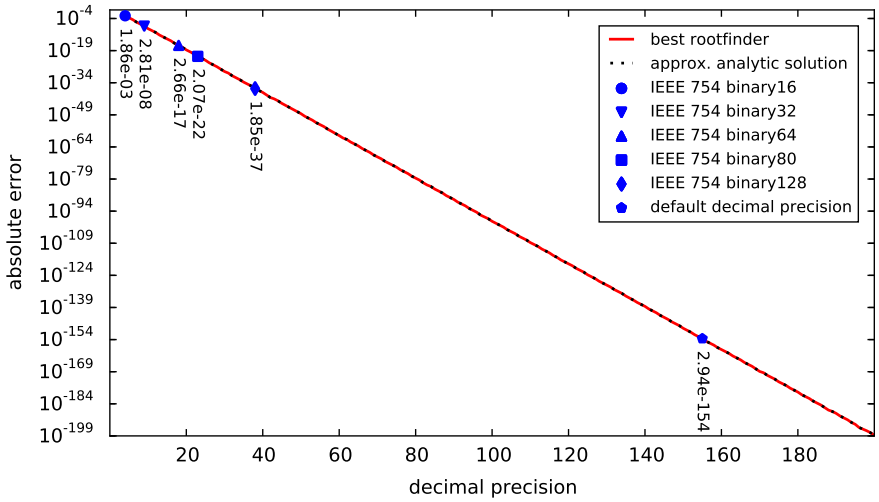
Međutim, na osnovu slike 2.7 može se potvrditi da aproksimativno analitičko rešenje propušta da daje tačna rešenja bez obzira na značajno povećanje radne preciznosti, ostvareno kroz uvođenje proizvoljno-precizne aritmetike u pokretnom zarezu. U sledećem delu biće predstavljena hibridna metoda koja će rešiti problem tačnosti koji proizilazi iz određivanja korenova karakterističnih funkcija, tj. iz originalnog problema rešavanja karakterističnih jednačina.

2.1.3 Hibridna metoda

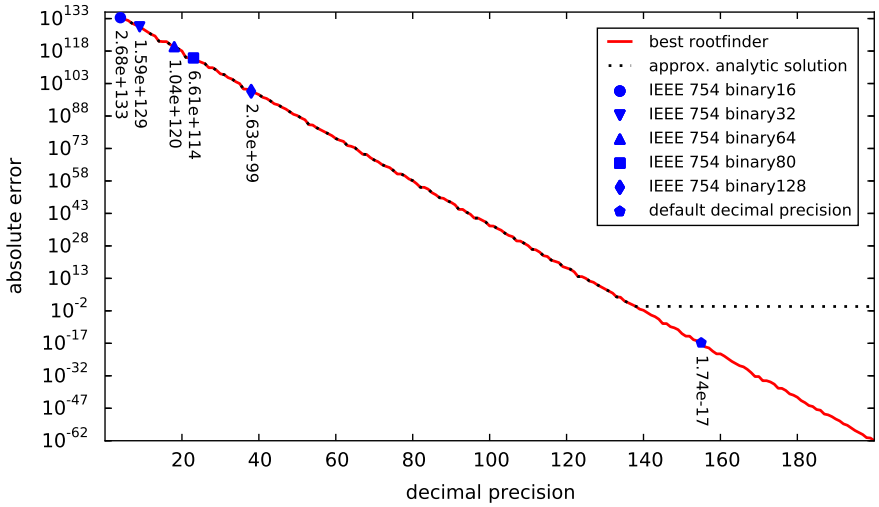
U sekciji 2.1.1 predstavljeni su različiti pristupi u rešavanju karakterističnih jednačina:

- direktno uvrštavanje aproksimativno analitičkog rešenja u karakterističnu jednačinu,
- simboličko rešavanje karakteristične jednačine,
- turnirski bazirano takmičenje između numeričkih određivača korenova karakteristične jednačine.

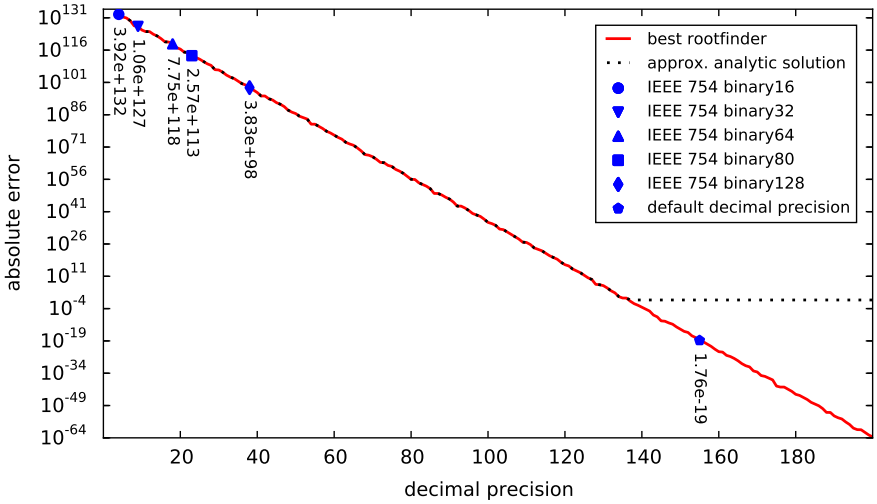
Pojedinačno je analiziran svaki predstavljeni pristup i ukazane su njihove prednosti i mane. Kao najpodobniji pristup je odabrano turnirski bazirano takmičenje između numeričkih određivača korenova karakteristične jednačine – pošto se za složenije granične uslove karakteristične jednačine ne mogu simbolički rešiti, dok korenovi dobijeni iz aproksimativno analitičkog rešenja neće konvergirati ka tačnom rešenju za složenije granične uslove (slike 2.2 i 2.3). Pokazano je da ni odabrani pristup ne može samostalno proizvesti tačne korenove za više modove u slučaju složenijih graničnih uslova.



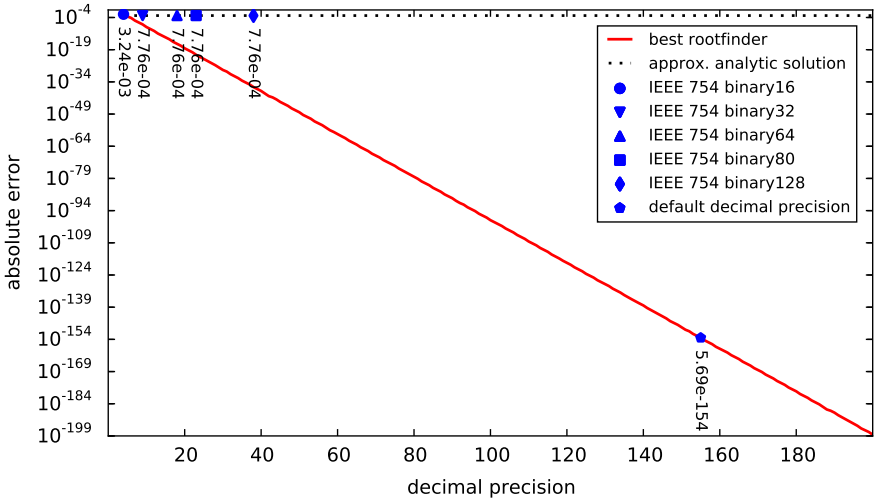
Slika 2.5: Maksimalne greške (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije za granični uslov gde su oba kraja slobodno oslonjena (LJ=1). Oznaka „default decimal precision" je postavljena na 155 decimalnih mesta, što je podrazumevana radna preciznost za `beam_integrals` projekat



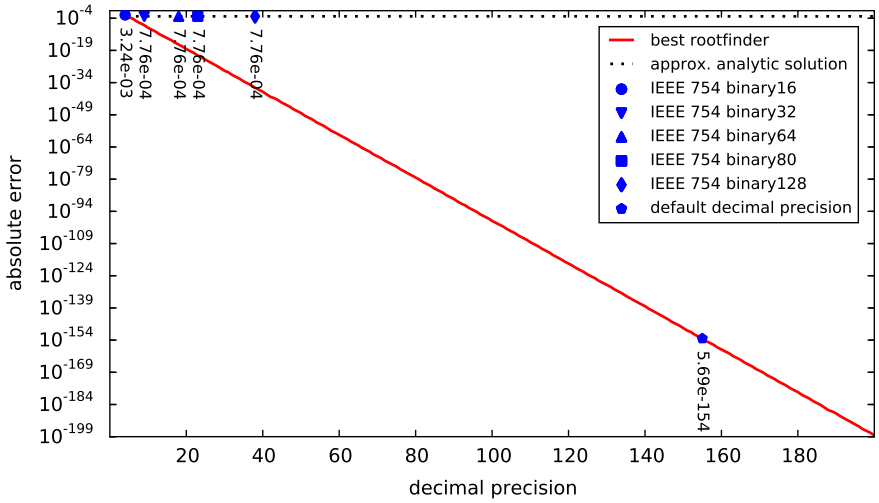
Slika 2.6: Maksimalne greške (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije za granični uslov gde oba kraja uklještena (LJ=2)



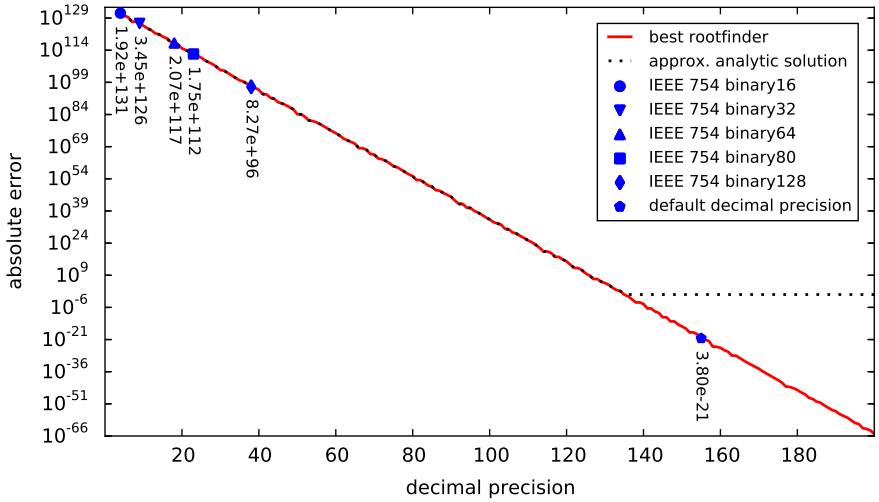
Slika 2.7: Maksimalne greške (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



Slika 2.8: Maksimalne greške (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4)



Slika 2.9: Maksimalne greške (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5)



Slika 2.10: Maksimalne greške (kroz svih 100 modova) prilikom određivanja korenova karakteristične funkcije za granični uslov gde oba kraja slobodna (LJ=6)

U sekciji 2.1.2 analiziran je problem nedostatka radne preciznosti tokom rešavanja karakterističnih jednačina, uz nemogućnost dobijanja tačnih korenova ni uz korišćenje viših preciznosti definisanih IEEE 754 standardom. Usled toga uočena je potreba da se karakteristične jednačine rešavaju korišćenjem režima visoke preciznosti, kroz uvođenje proizvoljno-precizne aritmetike u pokretnom zarezu. Pokazano je da bez obzira na značajno povećanje radne preciznosti aproksimativno analitičko rešenje propušta da daje tačne korenove za složenije granične uslove (slika 2.7), pošto i tada greška konvergira ka konstanti $E \neq 0$.

Hibridna metoda za rešavanje karakterističnih jednačina dobija se kombinacijom:

1. turnirski baziranog takmičenja između numeričkih određivača korenova karakteristične jednačine i

2. korišćenjem režima visoke preciznosti, kroz uvođenje proizvoljno precizne aritmetike u pokretnom zarezu.

Projekat `beam_integrals` (opisan u sekciji 4.3) pruža referentnu implementaciju hibridne metode. Za potrebe automatske verifikacije rezultata projekat `beam_integrals` definiše 10^{-16} za gornji prag tolerancije apsolutne greške prilikom provere tačnosti svih podržanih proračuna – uključujući tu i rešavanje karakterističnih jednačina. Na osnovu zadate tolerancije hibridni metod je algoritamski potvrđen zadovoljenjem svih ≈ 4400 međusobno nezavisnih testova podsistema za automatsku verifikaciju (opisanog u sekciji 4.3.7).

Optimalna (tj. minimalna potrebna) radna preciznost hibridnog metoda određena je eksperimentalno na 155 decimalnih mesta, što je ujedno i podrazumevana radna preciznost za projekat `beam_integrals`. Na početku razvoja projekta `beam_integrals` podrazumevana decimalna preciznost je pretpostavljena na 300 decimalnih mesta da bi kasnije, zarad optimizacije vremena proračuna, bila iterativno snižavana sve do minimalne vrednosti (155 decimalnih mesta) potrebne da se zadovolje svi testovi podsistema za automatsku verifikaciju.

Koristeći projekat `beam_integrals` na listingu 2.1 se, kroz interaktivnu Python [32] sesiju, porede:

- hibridna metoda, pri čemu su proračuni vršeni u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat
- turnirsko bazirano takmičenje između numeričkih određivača korenova karakteristične jednačine, pri čemu su proračuni vršeni u IEEE 754 binary64 radnoj preciznosti

za određivanje 100.-og korena u slučaju graničnog uslova gde je jedan kraj uklješten a drugi slobodan (LJ=3).

```
>>> from beam_integrals.beam_types import BaseBeamType
>>> from beam_integrals.characteristic_equation_solvers \
... import find_best_root

# clamped-free beam (LJ=3), 100th mode
>>> beam_type = BaseBeamType.coerce(3)
>>> mode = 100

# computes (root, error) at the default decimal precision
>>> r1, e1 = find_best_root(beam_type, mode, include_error=True)

# computes (root, error) at IEEE 754 binary64 precision
>>> r2, e2 = find_best_root(
...     beam_type, mode, decimal_precision=15,
...     include_error=True
... )

>>> print "%e\terror(high arbitrary-precision)\n" \
...       "%e\terror(IEEE 754 binary64 precision)" % (e1, e2)
1.761524e-19    error(high arbitrary-precision)
2.094024e+121    error(IEEE 754 binary64 precision)
```

Listing 2.1: Demonstracija prednosti hibridne metode u odnosu na turnirsko bazirano takmičenje između numeričkih određivača korenova karakteristične jednačine

Sa listinga 2.1 mogu se uporediti greške prilikom određivanja korenova karakteristične jednačine:

- 1.761524×10^{-19} za hibridnu metodu
- 2.094024×10^{121} za turnirsko bazirano takmičenje između numeričkih određivača korenova karakteristične jednačine

i uvideti značajne prednosti hibridne metode u odnosu na turnirsko bazirano takmičenje između numeričkih određivača korenova karakteristične jednačine.

Koristeći projekat `beam_integrals` na listingu 2.2 se, kroz interaktivnu Python sesiju, porede:

- hibridna metoda i
- aproksimativno analitičko rešenje

za određivanje 100.-og korena u slučaju graničnog uslova gde je jedan kraj uklješten a drugi slobodan, pri čemu su oba proračuna vršena u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat.

```
>>> from beam_integrals import DEFAULT_DECIMAL_PRECISION
>>> from beam_integrals.beam_types import BaseBeamType
>>> from beam_integrals.characteristic_equation_solvers \
... import find_best_root

# clamped-free beam (LJ=3), 100th mode
>>> beam_type = BaseBeamType.coerce(3)
>>> mode = 100

# computes (root, error) using numerical optimization
>>> r1, e1 = find_best_root(beam_type, mode, include_error=True)

# computes (root, error) using approx. analytic solution
>>> r2 = beam_type.mu_m_initial_guess(mode)
>>> e2 = beam_type.characteristic_function.evalf(
...     n=DEFAULT_DECIMAL_PRECISION, subs={'mu_m': r2}
... )

>>> print "%e\terror(numerical optimization)\n" \
...       "%e\terror(analytic approach)" % (e1, e2)
1.761524e-19    error(numerical optimization)
1.000000e+00    error(analytic approach)
```

Listing 2.2: Demonstracija prednosti hibridne metode u odnosu na aproksimativno analitičko rešenje

Sa listinga 2.2 mogu se uporediti greške prilikom određivanja korenova karakteristične jednačine:

- 1.761524×10^{-19} za hibridnu metodu
- 1.000000×10^{0} za aproksimativno analitičko rešenje

i uvideti značajne prednosti hibridne metode u odnosu na aproksimativno analitičko rešenje.

Učinak hibridne metode u rešavanju karakterističnih jednačina može se i vizuelno potvrditi na slikama 2.5–2.10 („best rootfinder" kriva uz „default decimal precision" radnu preciznost).

Na slikama se može videti da, korišćenjem hibridnog metoda, maksimalna greška ne prelazi prag tolerancije apsolutne greške od 10^{-16}, dok to ne važi za druge načine rešavanja karakterističnih jednačina, sem u trivijalnom slučaju kada su oba kraja slobodno oslonjena (sekcija 1.2.1.1).

2.2 Rešavanje jednačina svojstvenih oblika

Za sve granične uslove date u sekciji 1.2.1 rešavanjem odgovarajuće karakteristične jednačine dobijaju se njeni korenovi μ_m, gde je $1 \leq m \leq 100$. Tako dobijene vrednosti μ_m koriste se tokom rešavanja pripadajućih jednačina svojstvenih oblika $Y_m(y)$.

Na slikama 2.11–2.22 poredi se uticaj korenova karakteristične jednačine dobijenih:

- hibridnom metodom, pri čemu su proračuni vršeni u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat („best rootfinder @ default decimal precision" kriva)
- sa direktnim uvrštavanjem aproksimativno analitičkog rešenja u karakterističnu jednačinu, pri čemu su proračuni vršeni u IEEE 754 binary64 radnoj preciznosti („approx. analytic solution @ IEEE 754 binary64" kriva)

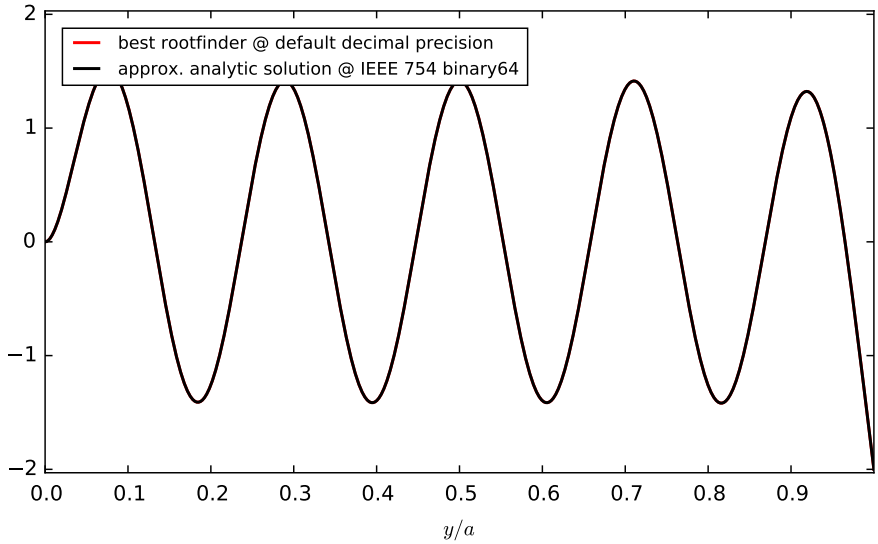
na oblik funkcije $Y_m(y)$ u slučaju graničnog uslova gde je jedan kraj uklješten a drugi slobodan (LJ=3) za:

$$m \in \{10, 11, 12, 13, 14, 15, 20, 40, 60, 80, 99, 100\}$$

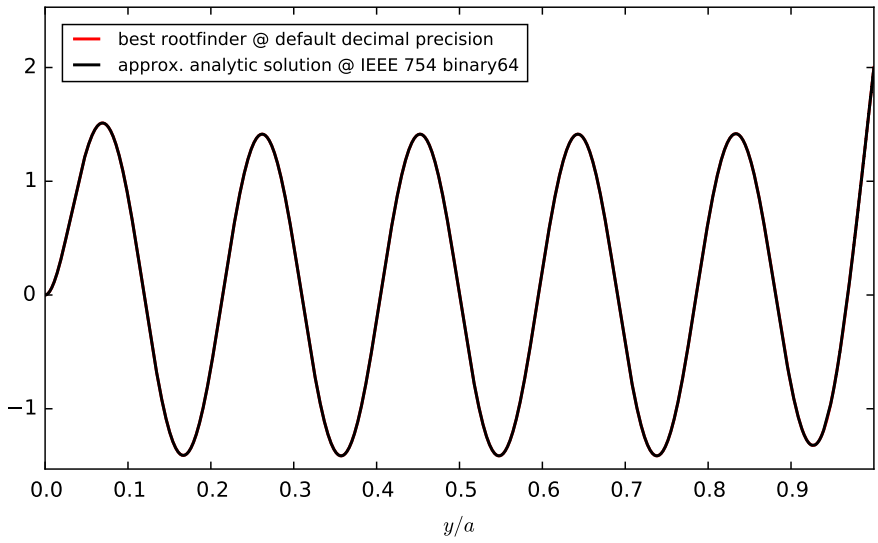
Na osnovu slika 2.11–2.12 može se vizuelno utvrditi da kod nižih modova oblik funkcije $Y_m(y)$ dobijen korišćenjem aproksimativno analitičkog rešenja karakteristične jednačine oponaša oblik funkcije $Y_m(y)$ dobijen korišćenjem hibridne metode. Međutim, sa višim modovima dolazi do značajnog razmimoilaženja u oblicima funkcije $Y_m(y)$, što se može vizuelno potvrditi na slikama 2.14–2.22.

Na slikama 2.23–2.28 uporedno se analiziraju oblici 25.-og moda funkcije $Y_m(y)$ za sve definisane granične uslove. Uočava se da su funkcije $Y_m(y)$ izrazito osetljive na pertubacije za sve složenije granične uslove, što se vidi na osnovu ponašanja oblika moda (eng. *mode shape*) dobijenog korišćenjem aproksimativno analitičkog rešenja karakteristične jednačine.

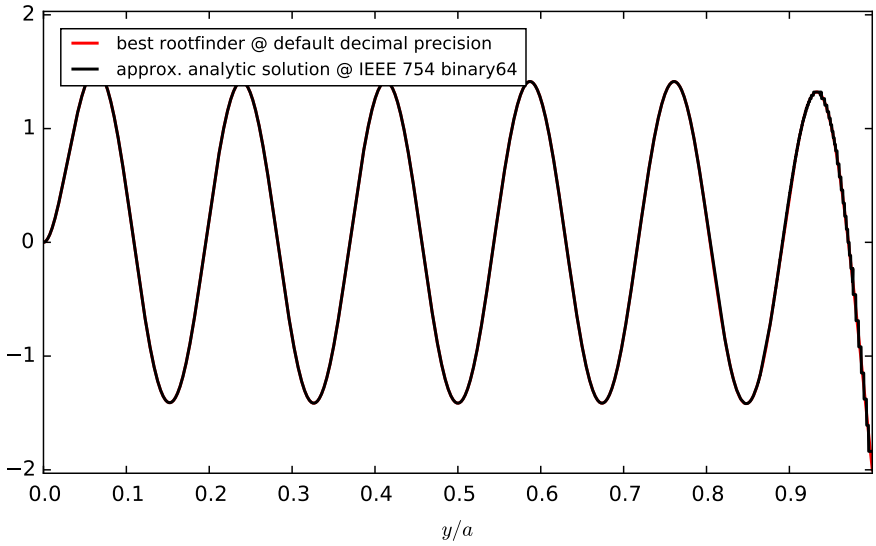
Iz navedenog, a zarad održanja kvaliteta rezultata za sve kombinacije modova i graničnih uslova, može se zaključiti da je upotreba hibridne metode neophodna prilikom rada sa jednačinama svojstvenih oblika.



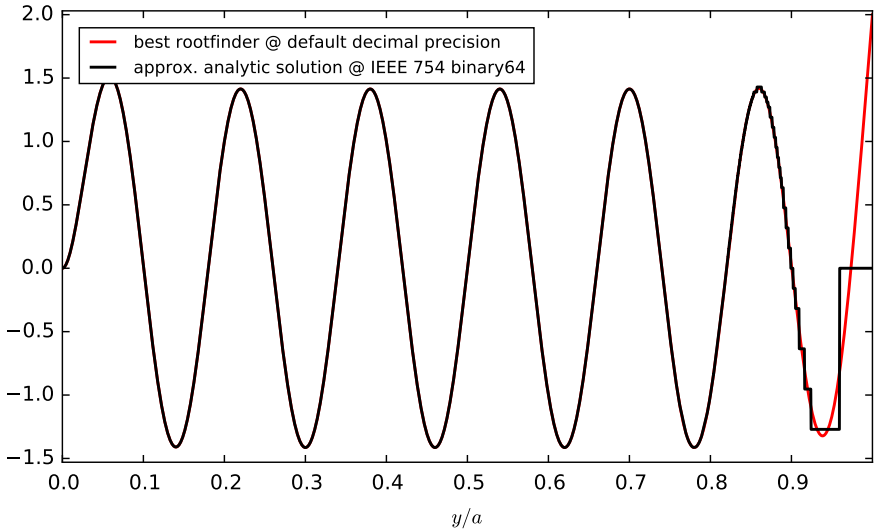
Slika 2.11: Oblik 10.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3). Horizontalna osa je normalizovana sa $y/a \in [0, 1]$, na osnovu opšteg oblika integrala iz jednačine 1.34



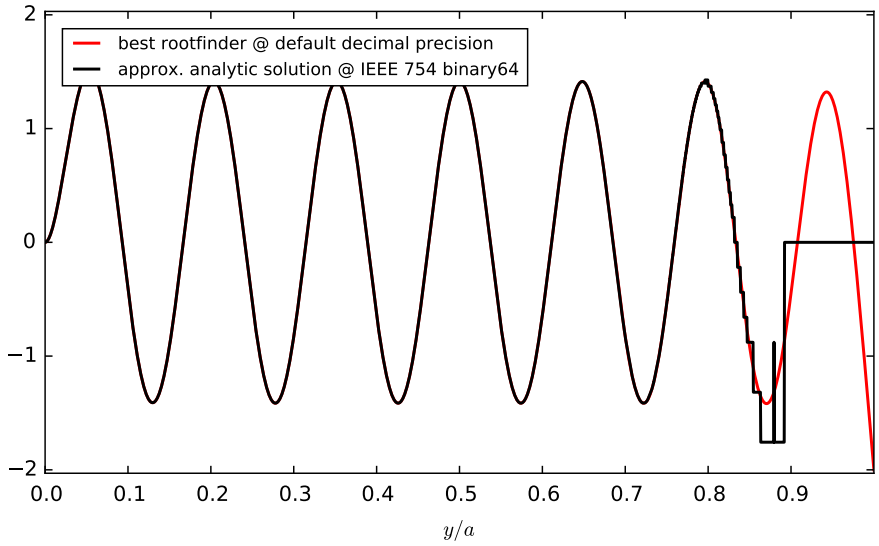
Slika 2.12: Oblik 11.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



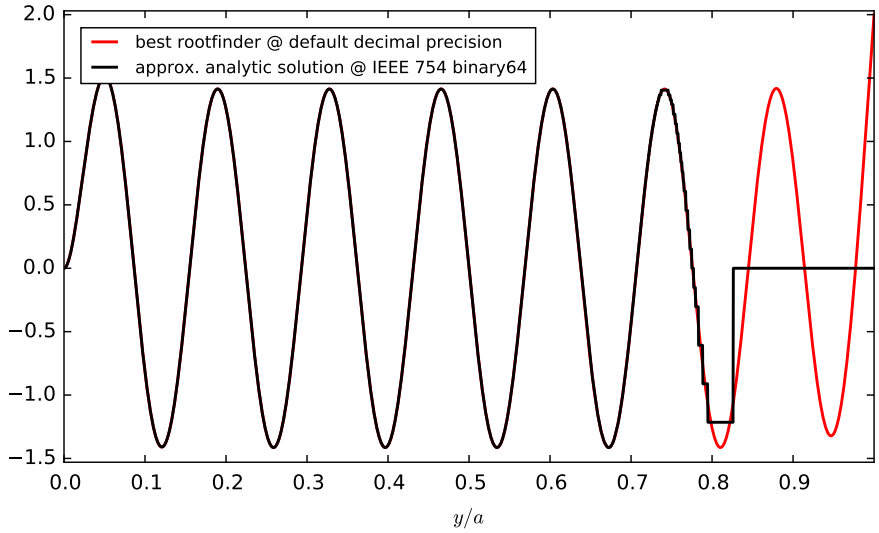
Slika 2.13: Oblik 12.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



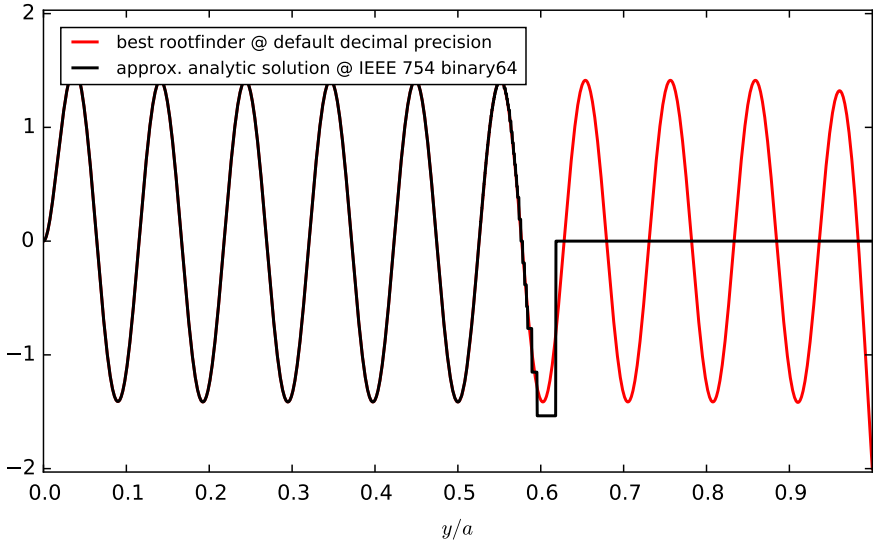
Slika 2.14: Oblik 13.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



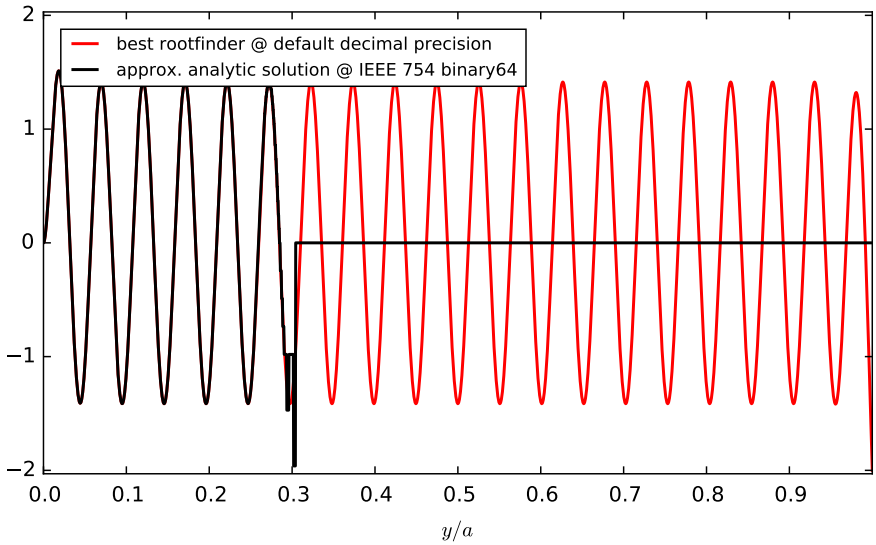
Slika 2.15: Oblik 14.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



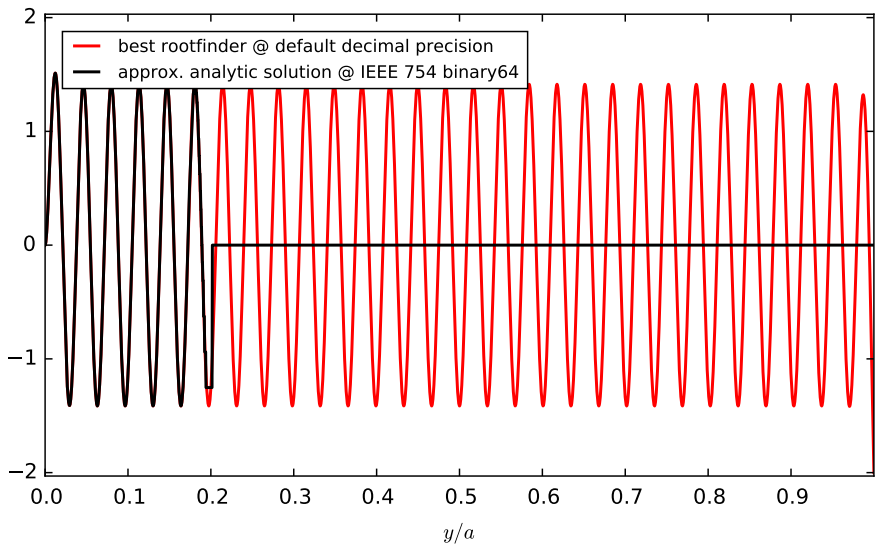
Slika 2.16: Oblik 15.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



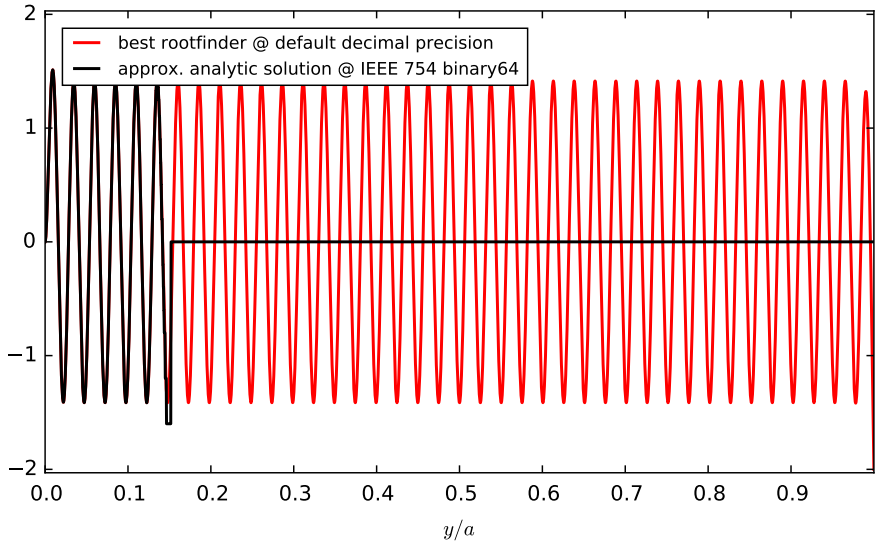
Slika 2.17: Oblik 20.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



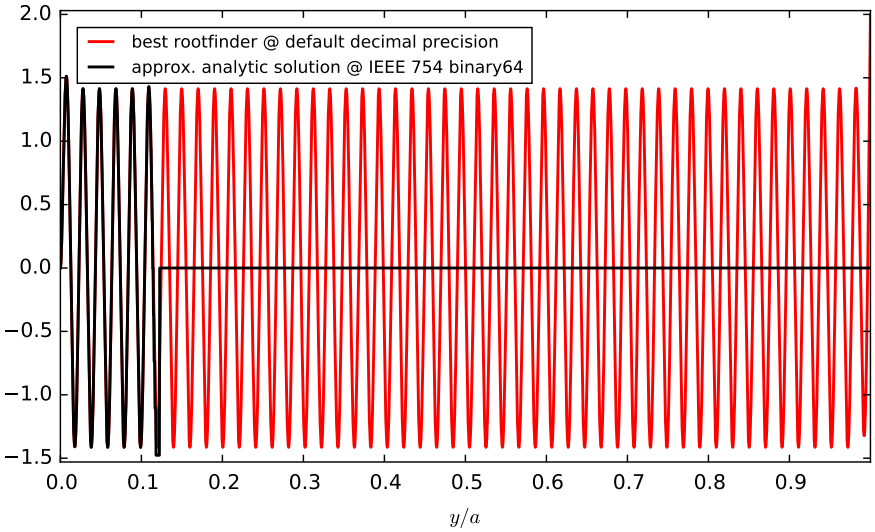
Slika 2.18: Oblik 40.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



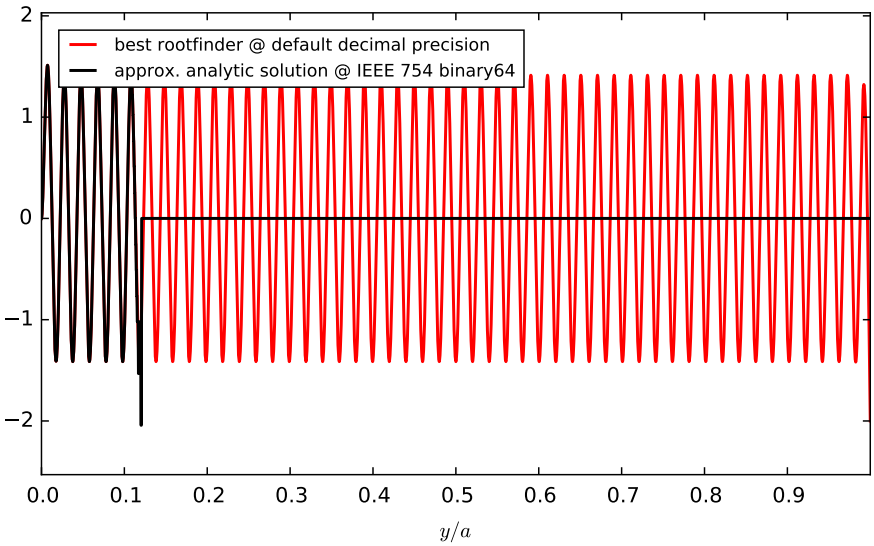
Slika 2.19: Oblik 60.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



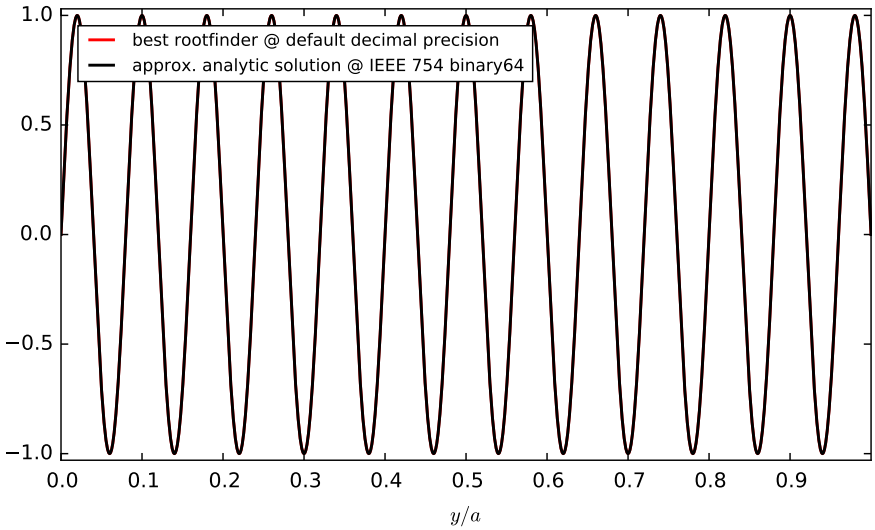
Slika 2.20: Oblik 80.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



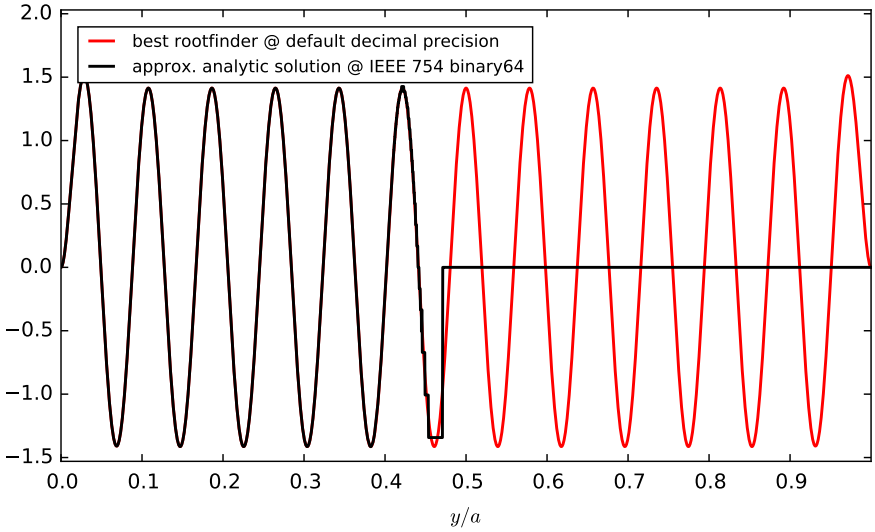
Slika 2.21: Oblik 99.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



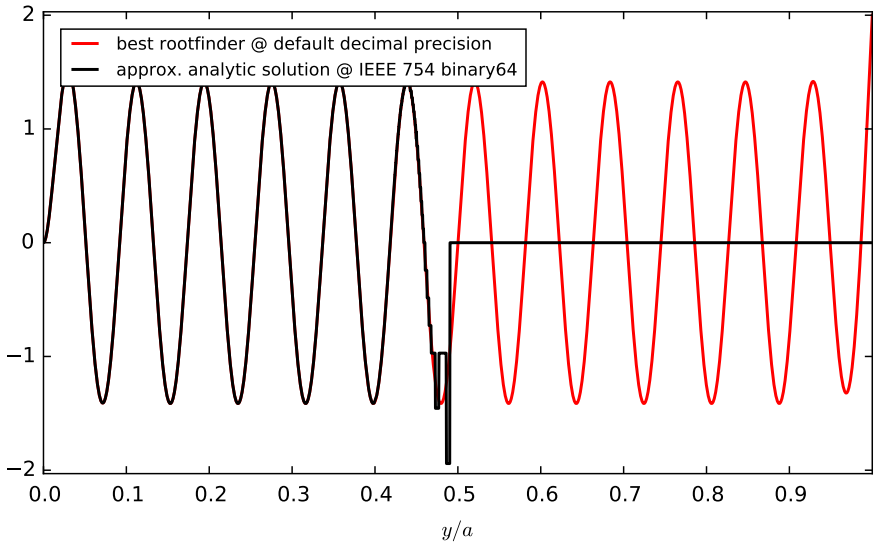
Slika 2.22: Oblik 100.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



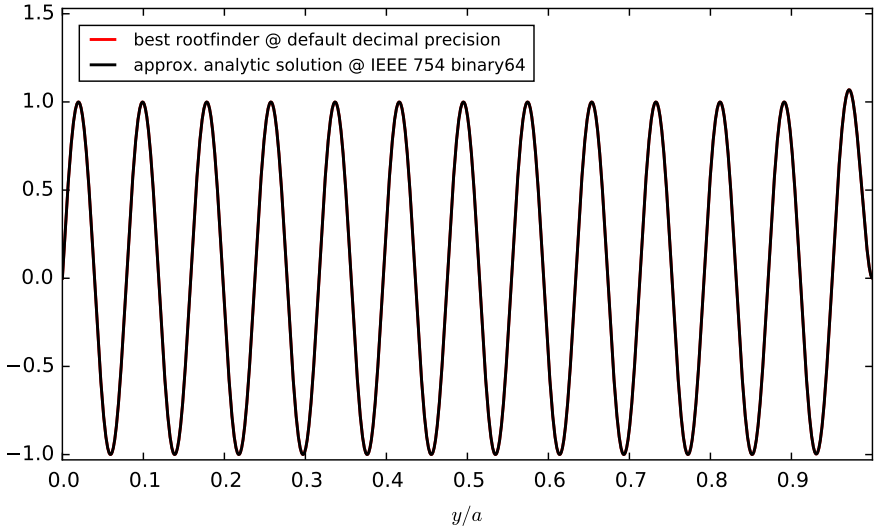
Slika 2.23: Oblik 25.-og moda funkcije $Y_m(y)$ za granični uslov gde su oba kraja slobodno oslonjena (LJ=1)



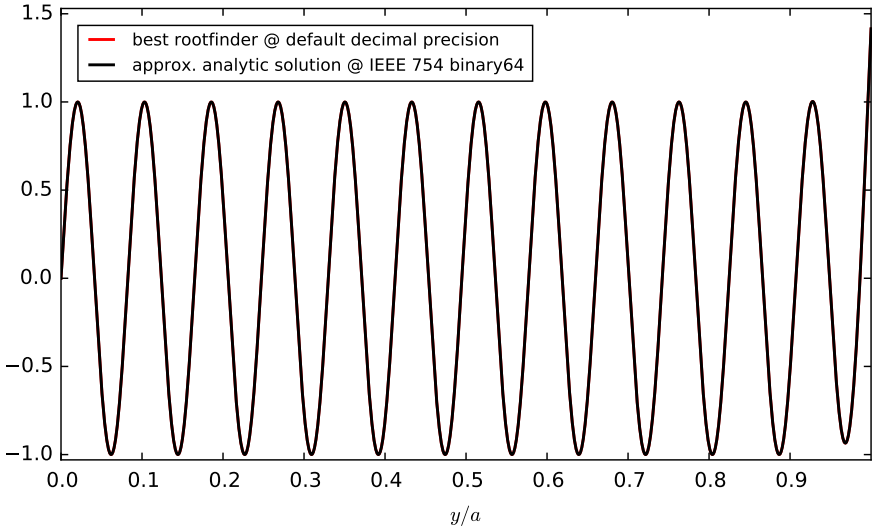
Slika 2.24: Oblik 25.-og moda funkcije $Y_m(y)$ za granični uslov gde su oba kraja uklještena (LJ=2)



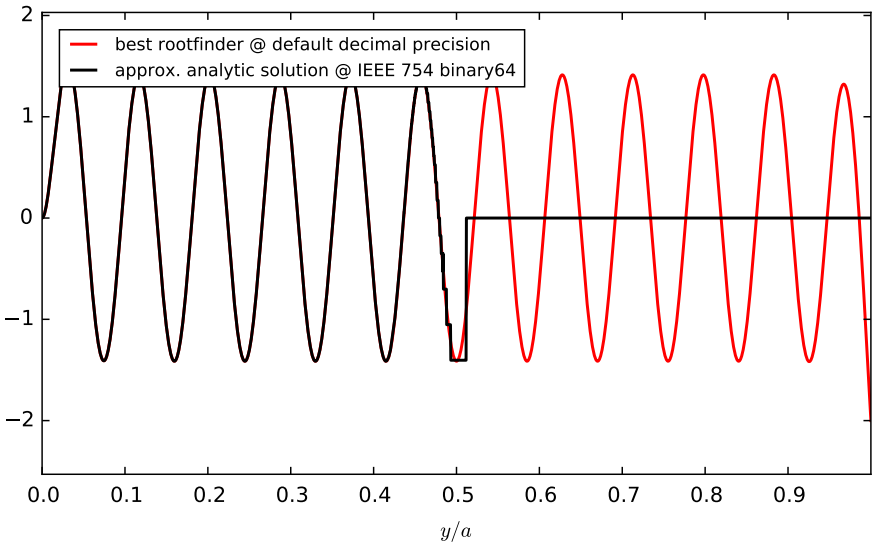
Slika 2.25: Oblik 25.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



Slika 2.26: Oblik 25.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4)



Slika 2.27: Oblik 25.-og moda funkcije $Y_m(y)$ za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5)



Slika 2.28: Oblik 25.-og moda funkcije $Y_m(y)$ za granični uslov gde su oba kraja slobodna (LJ=6)

2.3 Problem izračunavanja određenih integrala

Postoje dva osnovna pristupa za izračunavanje određenog integrala:

- iz neodređenog integrala, primenom Newton-Leibnitzove formule [33] i
- iz podintegralne funkcije (tzv. integrand) putem numeričke integracije.

Izračunavanje određenog integrala iz neodređenog integrala je poželjnije usled:

- potencijalno manje greške integracije i
- značajno boljih performansi tokom integracije.

Poboljšane performanse pri izračunavanju određenog integrala iz neodređenog integrala mogu se ostvariti na sledeći način – za svaku kombinaciju graničnog uslova i integrala `beam_integrals` projekat će, koristeći SymPy biblioteku, na osnovu opšteg oblika integrala iz jednačine 1.34 i zadatog integrala simbolički odrediti analitičku formulaciju njegovog neodređenog integrala, tj. $G(y)$:

$$G(y) = \int F\Big(Y_m(y),\ Y_n(y),\ Y_t(y),\ Y_v(y)\Big)dy \tag{2.6}$$

potom će, primenom Newton-Leibnitzove formule, iz neodređenog integrala $G(y)$ simbolički odrediti analitičku formulaciju određenog integrala, tj. $H(a)$:

$$\begin{aligned} I_\alpha(a) &= \int_0^a F\Big(Y_m(y),\ Y_n(y),\ Y_t(y),\ Y_v(y)\Big)dy \\ &= G(a) - G(0) \\ &= H(a) \end{aligned} \tag{2.7}$$

Funkcija $H(a)$ se može serijalizovati u odgovarajuću platformski nezavisnu programsku funkciju, tj. čist Python kôd u slučaju

`beam_integrals` projekta, pri čemu generisana programska funkcija tačno opisuje traženi određeni integral $I_\alpha(a)$. Tako dobijene Python funkcije se potom mogu i programski keširati na disku u obliku Python modula, čime se izbegava njihovo ponovno i vremenski zahtevno generisanje pri budućim izračunavanjima određenih integrala.

Koristeći goreopisani postupak problem izračunavanja određenih integrala (za svaku kombinaciju graničnog uslova i integrala) zamenjuje se jednostavnim pozivima odgovarajuće funkcije iz dinamički generisanog Python modula, čime bi se značajno ubrzao i pojednostavio proces izračunavanja određenih integrala.

Tokom razvoja projekta `beam_integrals` ustanovljeno je da se od ovog postupka nažalost mora odustati pošto je utvrđeno da ni SymPy ni Mathematica nisu u mogućnosti da simbolički odrede analitičku formulaciju neodređenog integrala za sve kombinacije složenijih graničnih uslova i integrala. Stoga je za izračunavanje određenih integrala odabran pristup numeričke integracije.

Na osnovu opšteg oblika integrala iz jednačine 1.34 i pojedinačnih integrala opisanih u sekciji 1.2.2 vidno je da se integrand sastoji od kombinacije $Y_m(y)$ funkcije i njena prva dva izvoda.

U sekciji 2.2 već je pokazano da su funkcije $Y_m(y)$ izrazito osetljive na pertubacije za sve složenije granične uslove, što se i videlo iz prikaza ponašanja oblika modova koji su dobijeni korišćenjem aproksimativno analitičkog rešenja karakteristične jednačine. Došlo se do zaključka da je upotreba hibridne metode neophodna prilikom rada sa jednačinama svojstvenih oblika $Y_m(y)$. Pošto opšti oblik integrala zavisi od kombinacija $Y_m(y)$ funkcija i njenih izvoda može se zaključiti da je upotreba hibridne metode neophodna i prilikom izračunavanja određenih integrala.

Stoga se za potrebe numeričke integracije neće analizirati integrali dobijeni na osnovu direktnog uvrštavanja aproksimativno analitičkog rešenja u karakterističnu jednačinu, već će se koristiti isključivo hibridni metod.

2.3.1 Uticaj hibridne metode na kvalitet rezultata

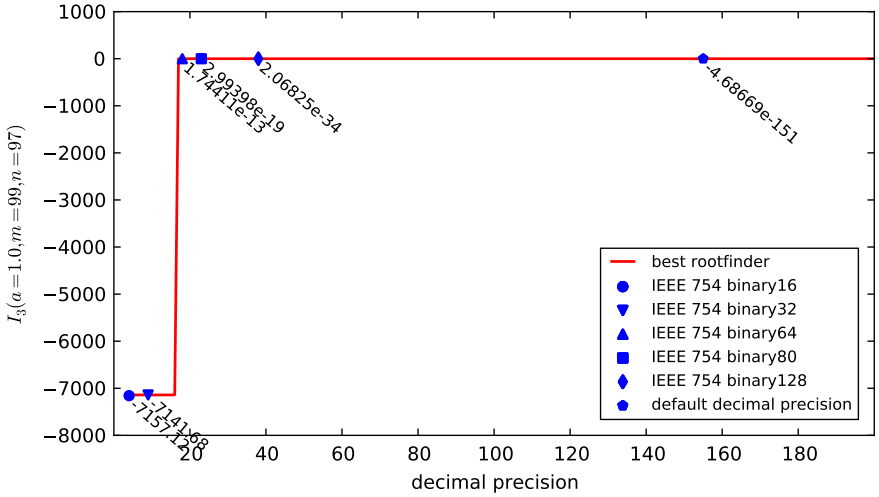
Na slikama 2.29–2.40 prikazana je zavisnost vrednosti određenog integrala I_3 od radne preciznosti proračuna. Na svim prikazanim slikama važi $a = 1.0$, $m = 99$ i $n = 97$ (jednačina 1.37).

Analizom slike 2.30 može se uočiti da je za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) integral $I_3 \approx 0$. Dobijena vrednost slaže se sa teoretskom postavkom da će, za granični uslov gde su oba kraja slobodno oslonjena, vrednost integrala I_3 uvek biti 0 ako je $m \neq n$ [8].

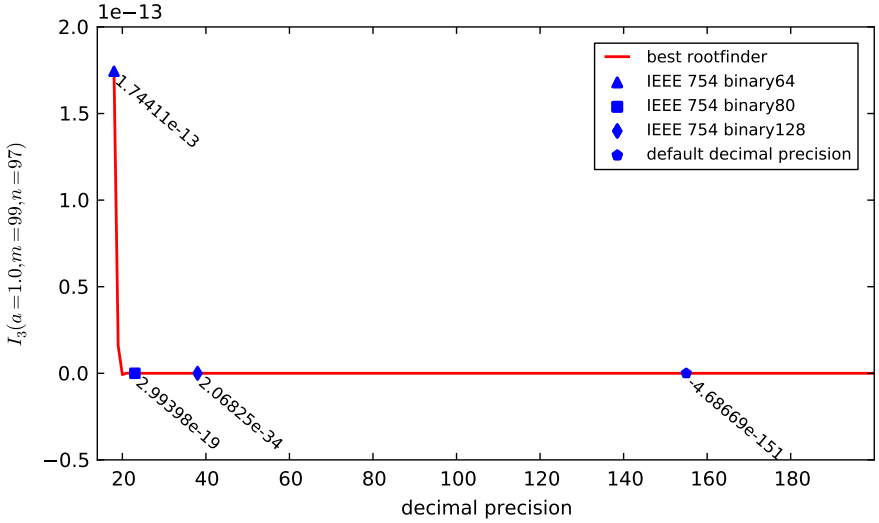
Uporednom analizom slika 2.29, 2.31, 2.33, 2.35, 2.37 i 2.39 može se uočiti da za sve granične uslove dolazi do nagle promene vrednosti integrala (tzv. „koleno") u neposrednoj okolini IEEE 754 binary64 preciznosti. Naime, sa slika se vidi da su vrednosti sa leve strane kolena relativno stabilne, ali da nakon kolena dolazi do naglog skoka (tj. pada) vrednosti integrala. Može se reći da je u oblasti kolena došlo do naglog poboljšanja tačnosti integrala nakon blagog povećanja radne preciznosti.

U slučajevima složenijih graničnih uslova, tj. LJ $\in \{2, 3, 6\}$, vrednosti integrala se interesantno ponašaju i sa desne strane kolena, što se može delimično primetiti uporednom analizom slika 2.31, 2.33 i 2.39. Međutim kada se analiziraju fokusirani prikazi navedenih slika, tj. slike 2.32, 2.34 i 2.40 očevidno je postojanje i drugog kolena u blizini 130.-og mesta decimalne preciznosti. Razlika vrednosti integrala između leve i desne strane drugog kolena nisu toliko drastične kao u slučaju prvog kolena, ali je ipak njihova razlika nezanemarljiva. Može se reći da je i u oblasti drugog kolena došlo do naglog poboljšanja tačnosti integrala nakon blagog povećanja radne preciznosti.

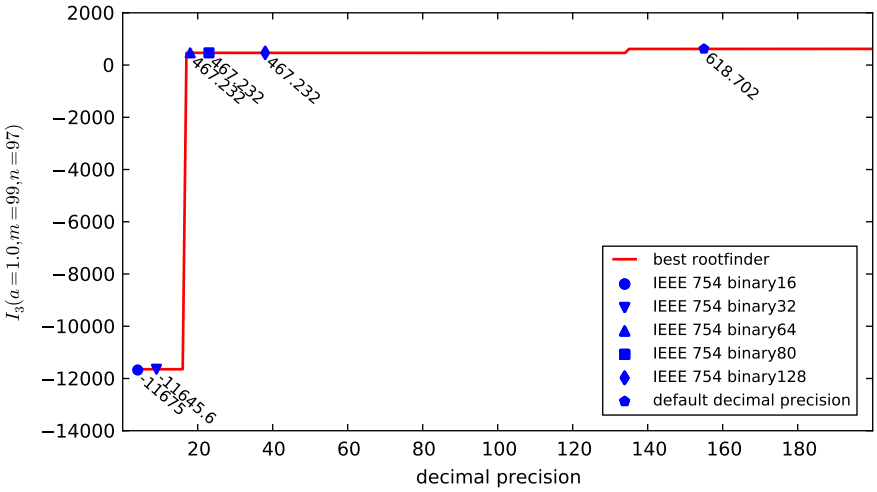
Ovim se na dodatan način potvrđuje pretpostavka da IEEE 754 standard ne pruža dovoljnu preciznost za opisane proračune, već je te proračune neophodno vršiti u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat – što se pokazalo naročito tačnim prilikom izračunavanja određenih integrala.



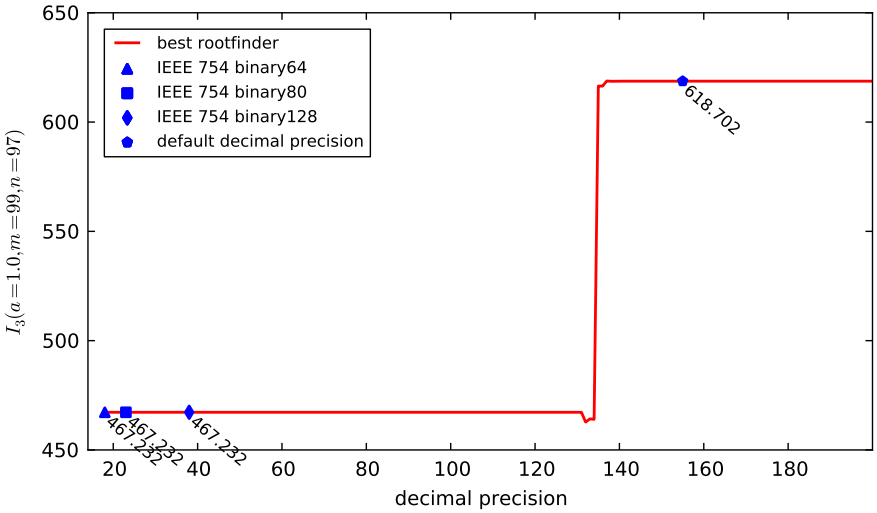
Slika 2.29: Prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde su oba kraja slobodno oslonjena (LJ=1)



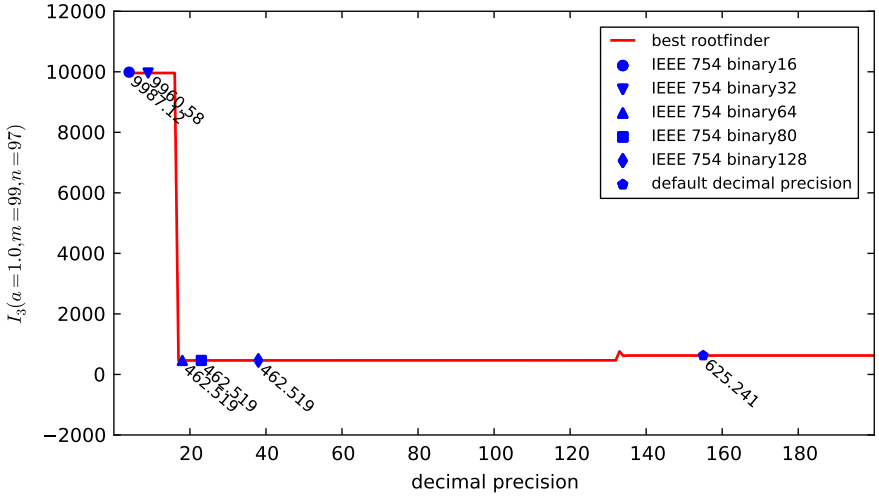
Slika 2.30: *Fokusirani* prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde su oba kraja slobodno oslonjena (LJ=1)



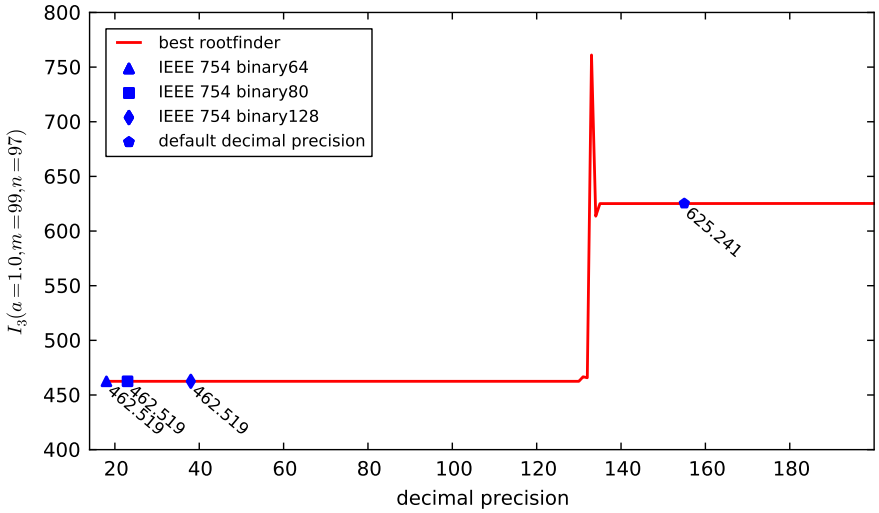
Slika 2.31: Prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde su oba kraja uklještena (LJ=2)



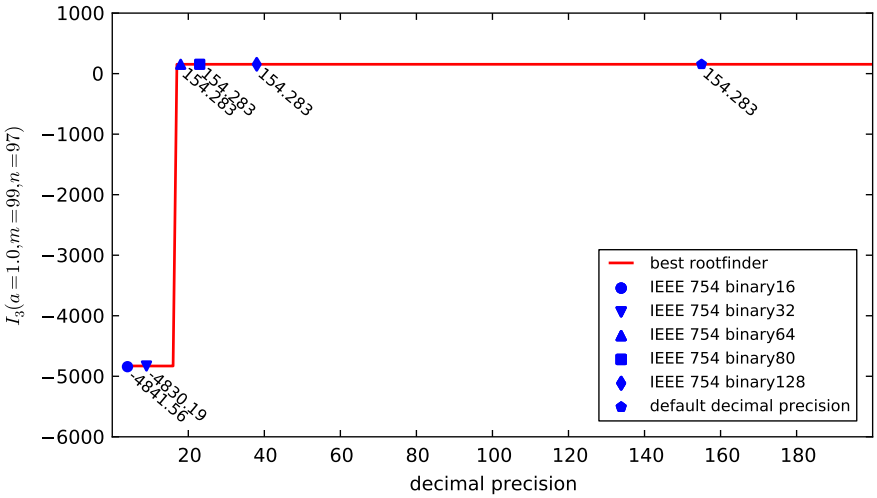
Slika 2.32: *Fokusirani* prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde su oba kraja uklještena (LJ=2)



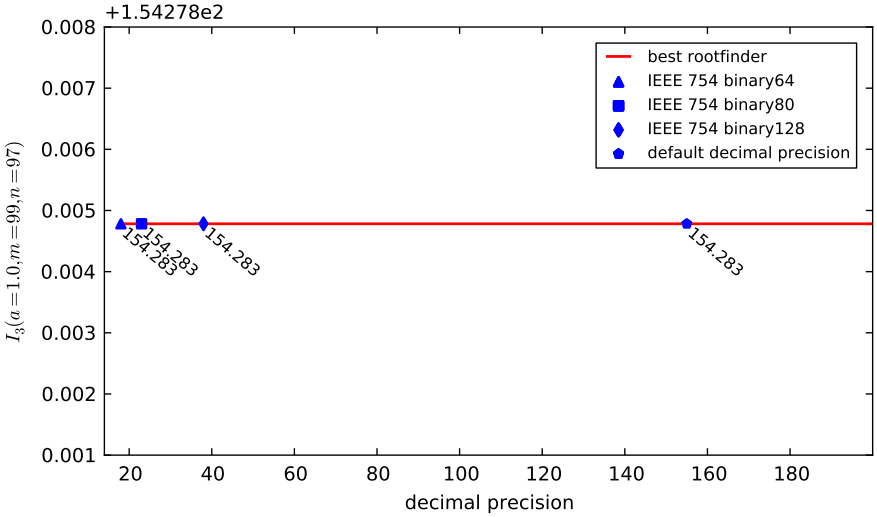
Slika 2.33: Prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



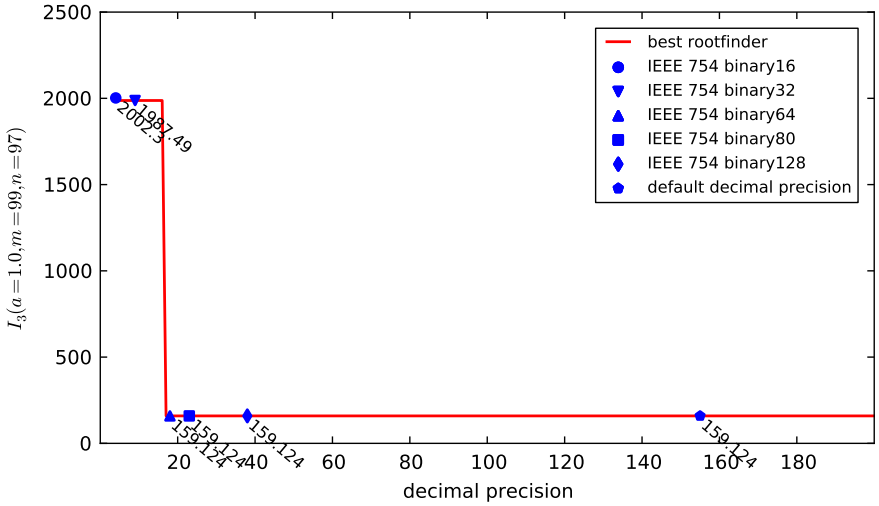
Slika 2.34: *Fokusirani* prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)



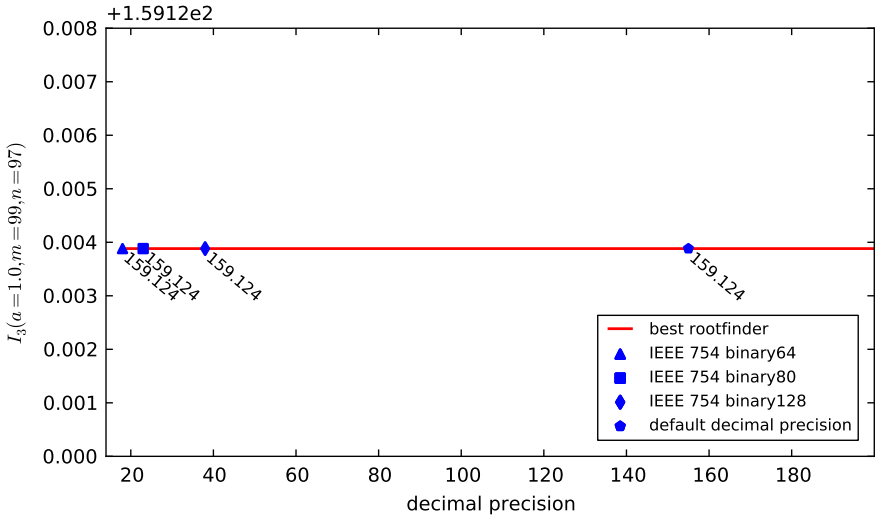
Slika 2.35: Prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4)



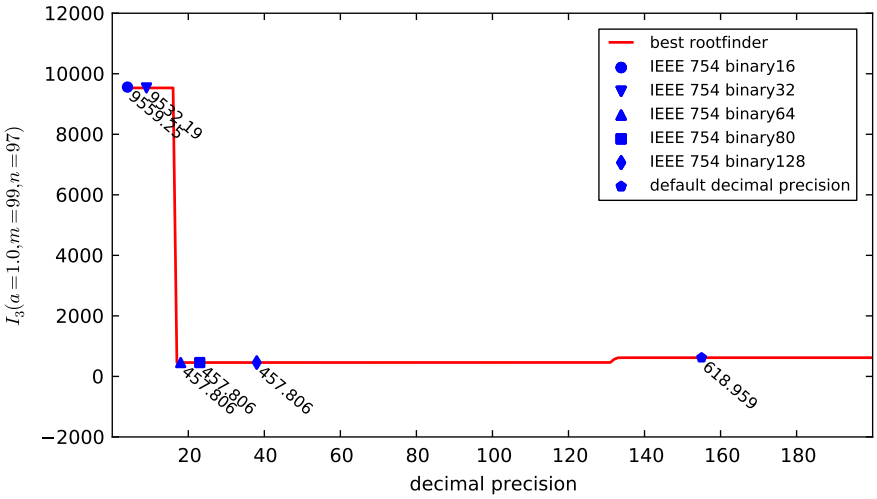
Slika 2.36: *Fokusirani* prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4)



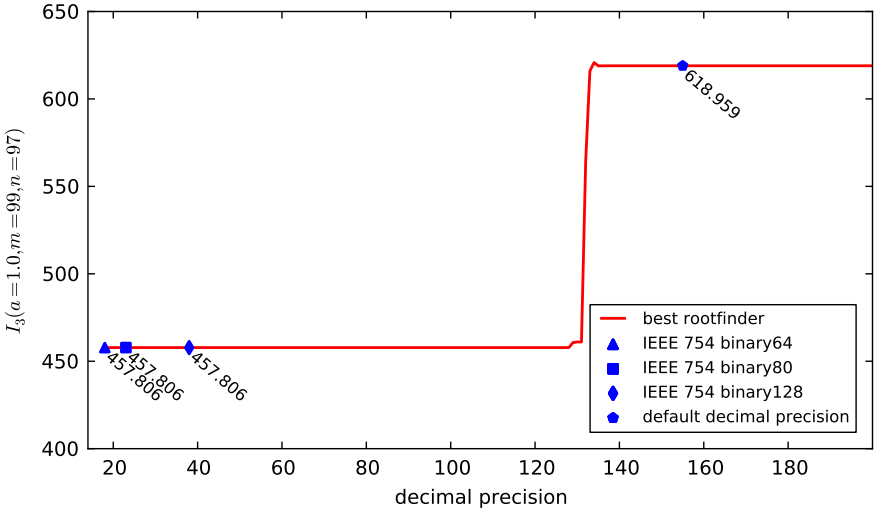
Slika 2.37: Prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5)



Slika 2.38: *Fokusirani* prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5)



Slika 2.39: Prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde su oba kraja slobodna (LJ=6)



Slika 2.40: *Fokusirani* prikaz zavisnosti vrednosti određenog integrala I_3 od radne preciznosti proračuna, za granični uslov gde su oba kraja slobodna (LJ=6)

Glava 3

Kreiranje tabela integrala

U prethodnom poglavlju analiziran je problem nedovoljne tačnosti i ukazano je na neophodnost korišćenja predstavljene hibridne metode kroz celokupni postupak izračunavanja određenih integrala.

Međutim, takav postupak izračunavanja određenih integrala je veoma zahtevan po pitanju utroška računarskih resursa i vremena proračuna. Kreiranjem tabela integrala, za sve kombinacije modova i graničnih uslova, ubuduće se izbegava ponovno i veoma zahtevno izračunavanje određenih integrala.

U daljem tekstu biće razmatrani različiti izazovi koji proističu tokom kreiranja tabela integrala i biće predloženi njihovi načini rešavanja.

3.1 Problemi efikasne organizacije i skladištenja tabela integrala

Kriterijumi za odabir formata skladišta podataka:

1. Skladište podataka treba da bude snimljeno u formatu koji je nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika, zarad maksimalne prenosivosti i upotrebljivosti tabela integrala.

2. Prilikom kreiranja tabela integrala stvara se ogromna koli-

čina podataka. Stoga je poželjno da odabrani format skladišta podataka podržava efikasnu kompresiju, pri čemu je zgodno da mehanizam kompresije skladišta bude transparentan za korisnika i nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika.

3. Skladište podataka treba da bude snimljeno u formatu koji je istovremeno i mašinski i ljudski lako čitljiv.

4. Format skladišta podataka ne bi smeo da zahteva komplikovanu instalaciju njegove programske podrške i biblioteka. Takođe ne bi trebao da zahteva od korisnika specifično znanje i veštine za njegovo održavanje i administraciju.

5. U zavisnosti od tipa problema koji se rešava, podacima se iz odabrane tabele integrala pristupa ili strogo sekvencijalno ili pak u nekom drugom (ne)uređenom redosledu.

   Stoga je poželjno da odabrani format skladišta podataka omogućava, na efikasan način, kako sekvencijalni tako i random pristup pojedinačnim podacima.

6. Prilikom kreiranja tabela integrala generiše se veća količina metapodataka koji podrobnije opisuju svojstva kreiranih tabela integrala - npr. maksimalni mod do kog su generisani podaci, koeficijent skaliranja vrednosti integrala, broj i spisak integracionih promenljivih koje definišu analitičku formulu odabranog integrala...

   Stoga je poželjno da odabrani format skladišta podataka omogućava kontekstualnu ugradnju metapodataka uz same podatke, jer time tabele integrala postaju u potpunosti samo-opisne.

7. U zavisnosti od tipa problema koji se rešava, tokom rada može biti potreban pristup podacima iz nekoliko različitih tabela integrala.

   Zarad bolje organizacije poželjno je da odabrani format skladišta podataka omogućava, na efikasan način, razdvajanje različitih tabela integrala u zasebne fizičke ili logičke

grupe unutar skladišta. Takođe je poželjno da odabrani format skladišta održi efikasnost sekvencijalnog i random pristupa pojedinačnim podacima, bez obzira da li naredno traženi podaci pripadaju istoj tabeli integrala.

8. Pojedini integrali poseduju istovetne kanoničke forme svojih analitičkih formula, što je razmatrano u sekciji 3.3.1. U tim slučajevima i njihove tabele integrala će biti istovetne.

   Stoga je poželjno da odabrani format skladišta podataka podržava efikasnu deduplikaciju istovetnih tabela integrala unutar jednog skladišta, pri čemu je zgodno da mehanizam deduplikacije podataka bude transparentan za korisnika i nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika.

Na osnovu postavljenih kriterijuma odabrano je nekoliko potencijalnih formata skladišta podataka:

1. CSV/TSV
2. JSON
3. YAML
4. HDF5

U daljem tekstu analiziraće se podobnost svakog od predloženih formata skladišta podataka za skladištenje tabela integrala.

3.1.1 CSV/TSV

CSV (eng. *Character-Separated Values*) spada među najstarije strukturirane formate datoteka [34]. Zbog svoje jednostavnosti odavno se koristi za razmenu podataka između programa različitih proizvođača, koji se uz to mogu izvršavati i na različitim operativnim sistemima ili hardverskim platformama.

CSV nije jedinstven, dobro definisani format datoteke pošto ne postoji njegova formalna specifikacija. U praksi se obično smatra da je neka datoteka snimljena u CSV formatu ako:

1. su svi podaci u okviru datoteke snimljeni isključivo u tekstualnom formatu, koristeći *ASCII*, *Unicode*, *EBCDIC* ili sličan skup karaktera

2. se datoteka sastoji od slogova (obično je jedan slog po liniji)

3. se svaki slog sastoji od polja međusobno razdvojenih delimiterima (obično su u pitanju rezervisani karakteri kao što su zarez, tačka-zarez ili tabulator). U okviru delimitera je, u pojedinim slučajevima, dozvoljeno i prisustvo jednog ili više karaktera za razmak.

4. za sve slogove važi da se sastoje od istog broja polja

5. za sve slogove važi da su im polja zapisana u istovetnom redosledu

Suštinska razlika TSV (eng. *Tab-Separated Values*) formata u odnosu na CSV format datoteke je što su polja unutar sloga uvek međusobno razdvojena tabulatorom (specijalni *TAB* karakter).

IETF RFC 4180 [35] pokušava da donekle standardizuje CSV format, ali to čini samo za polja tekstualnog tipa. Tumačenje smisla konkretnog podatka unutar takvog tekstualnog polja prepušteno je aplikacionom nivou. Za potrebe skladištenja tabela integrala veoma je bitno znati da ne postoji jedinstveni standard za tekstualnu reprezentaciju decimalnih brojeva u okviru CSV formata, iako se decimalni brojevi često koriste u CSV datotekama, već pojedine zemlje koriste *tačku* kao decimalni razmak dok druge zemlje koriste *zarez* za te potrebe. Primera radi, dok bi u Sjedinjenim Američkim Državama konstanta π bila zapisana u CSV datoteci kao 3.141593 – u Srbiji, Francuskoj ili Nemačkoj bi mogla da bude zapisana kao 3,141593 (u zavisnosti od konkretnog programa i njegovih regionalnih podešavanja).

Na listingu 3.1 dat je kratak primer jednog zapisa tabele integrala u CSV formatu, dobijen korišćenjem `beam_integrals` projekta u sledećoj konfiguraciji:

- za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)

- za integral I_3
- određeni integral se izračunava do $a = 1.0$
- tabela integrala se kreira za sve kombinacije m i n do 3.-eg moda
- svi proračuni su vršeni u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat

```
m,n,integral,error
1,1,1.5925953511467401,9.999999999999999e-197
1,2,2.506316581931779,9.999999999999999e-190
1,3,2.1328568885828862,9.999999999999999e-170
2,1,-15.706817907681641,1e-188
2,2,-12.816540145876225,9.999999999999999e-169
2,3,3.173235940357649,9.999999999999999e-173
3,1,37.42626772620566,9.999999999999999e-167
3,2,-8.885152517472523,1e-171
3,3,-45.975507298454154,1e-320
```

Listing 3.1: Kratak primer tabele integrala, snimljene u CSV formatu

Analiza podobnosti CSV formata skladišta podataka, na osnovu prethodno postavljenih kriterijuma sa strane 49:

1. CSV format skladišta je generalno nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika.
2. Sâm CSV format skladišta podataka ne podržava transparentnu kompresiju podataka, ali se rezultujuća datoteka skladišta može zasebno kompresovati željenim metodom kompresije.
3. Podaci u okviru CSV datoteke snimaju se u tekstualnom formatu, u obliku koji je istovremeno i mašinski i ljudski lako čitljiv.
4. Mnogi programski jezici poseduju odgovarajuću programsku podršku za rad sa CSV datotekama, bilo u okviru samog programskog jezika ili uz korišćenje dodatne programske biblioteke.

5. Pošto je CSV datoteka sekvencijalnog tipa pristupa, random pristup pojedinačnim podacima je jedino ostvariv skeniranjem kroz celokupnu datoteku, što se ne može smatrati efikasnom operacijom.

6. CSV format skladišta ne pruža mogućnost ugradnje metapodataka uz same podatke.

7. CSV format skladišta ne pruža mogućnost grupisanja podataka u zasebne fizičke ili logičke grupe unutar jedne datoteke. Pojedini aplikativni programi poseduju sopstvene konvencije za smeštanje više tabela unutar jedne CSV datoteke, ali te konvencije ne podležu standardima.

8. CSV format ne pruža mogućnost deduplikacije tabela unutar jednog skladišta.

Na osnovu izloženog smatra se da CSV ne zadovoljava postavljene kriterijume, te stoga nije podoban izbor za format skladišta podataka za skladištenje tabela integrala.

3.1.2 JSON

JSON (eng. *JavaScript Object Notation*) [36] je tekstualni format za jednostavnu razmenu strukturiranih podataka. Douglas Crockford je 2002.-e godine osmislio JSON format, koji je danas definisan kroz IETF RFC 4627 [37] i ECMA-404 [38] standard.

Često se koristi za programsku komunikaciju između *web* aplikacija i servera, kao jednostavnija alternativa XML formatu. Značajan broj programskih jezika poseduje podršku za rad sa JSON formatom, usled svoje opšte popularnosti na *web*-u.

Mada je idejno baziran na podskupu JavaScript programskog jezika [39,40] JSON format je portabilan i u potpunosti nezavisan od konkretnog programskog jezika.

JSON format definiše kratak skup pravila za portabilno formatiranje reprezentacija podržanih tipova podataka. JSON format poseduje 4 primitivna tipa podataka:

1. broj, koji može biti celobrojan ili u pokretnom zarezu

2. string, koji je sekvenca 0 ili više *Unicode* karaktera

3. logički tip

4. *null*, koji je specijalni tip za označavanje nedostajućih ili neispravnih vrednosti

uz 2 strukturirana tipa podataka:

1. niz, koji je uređena sekvenca 0 ili više vrednosti smeštenih u okviru `[ ]` zagrada, pri čemu svi članovi niza ne moraju biti istog tipa

2. objekat, koji predstavlja neuređeni skup 0 ili više parova *ključ:vrednost* smeštenih u okviru `{ }` zagrada, pri čemu svaki *ključ* mora biti string

Pojmovi *objekat* i *niz* su u skladu sa konvencijama JavaScript programskog jezika.

Na listingu 3.2 dat je kratak primer jednog zapisa tabele integrala u JSON formatu, dobijen korišćenjem `beam_integrals` projekta u sledećoj konfiguraciji:

- za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)
- za integral I_3
- određeni integral se izračunava do $a = 1.0$
- tabela integrala se kreira za sve kombinacije m i n do 3.-eg moda
- svi proračuni su vršeni u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat


```
[
    {
        "m": 1,
        "n": 1,
        "integral": 1.5925953511467401,
        "error": 9.999999999999999e-197
    },
```

```
    {
        "m": 1,
        "n": 2,
        "integral": 2.506316581931779,
        "error": 9.999999999999999e-190
    },
    {
        "m": 1,
        "n": 3,
        "integral": 2.1328568885828862,
        "error": 9.999999999999999e-170
    },
    {
        "m": 2,
        "n": 1,
        "integral": -15.706817907681641,
        "error": 1e-188
    },
    {
        "m": 2,
        "n": 2,
        "integral": -12.816540145876225,
        "error": 9.999999999999999e-169
    },
    {
        "m": 2,
        "n": 3,
        "integral": 3.173235940357649,
        "error": 9.999999999999999e-173
    },
    {
        "m": 3,
        "n": 1,
        "integral": 37.42626772620566,
        "error": 9.999999999999999e-167
    },
    {
        "m": 3,
        "n": 2,
        "integral": -8.885152517472523,
        "error": 1e-171
    },
    {
        "m": 3,
        "n": 3,
        "integral": -45.975507298454154,
        "error": 1e-320
    }
]
```


Listing 3.2: Kratak primer tabele integrala, snimljene u JSON formatu

Analiza podobnosti JSON formata skladišta podataka, na osnovu prethodno postavljenih kriterijuma sa strane 49:

1. JSON format je u potpunosti nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika.

2. Sâm JSON format ne podržava transparentnu kompresiju podataka, ali se rezultujuća datoteka skladišta može zasebno kompresovati željenim metodom kompresije.

3. JSON je tekstualni format koji je istovremeno i mašinski i ljudski lako čitljiv.

4. Mnogi programski jezici poseduju odgovarajuću programsku podršku za rad sa JSON formatom, bilo u okviru samog programskog jezika ili uz korišćenje dodatne programske biblioteke.

5. Usled prirode JSON formata random pristup pojedinačnim podacima jedino je ostvariv kroz parsiranje celokupnog JSON dokumenta i njegovo učitavanje u radnu memoriju, što se ne može smatrati efikasnom operacijom.

6. Sâm JSON format ne pruža mogućnost kontekstualne ugradnje metapodataka uz same podatke. Taj se mehanizam može donekle simulirati kroz proširivanje logičke strukture tabele integrala i samog formata skladišta, ali na način koji nije transparentan za korisnika.

7. JSON format ne pruža mogućnost razdvajanja različitih tabela integrala u zasebne fizičke grupe unutar jednog skladišta, dok se razdvajanje u logičke grupe može simulirati grupisanjem tabela integrala u jedan JSON niz. Međutim, time se problemi opisani u analizi 5.-og kriterijuma samo dodatno inteziviraju, naročito ako naredno traženi podaci ne pripadaju istoj tabeli integrala.

8. Sâm JSON format ne pruža mogućnost deduplikacije tabela integrala unutar jednog skladišta. Taj se mehanizam može donekle simulirati kroz proširivanje logičke strukture formata skladišta zarad uvođenja „prečica" ka datoj tabeli integrala, ali na način koji nije transparentan za korisnika.

Na osnovu izloženog smatra se da JSON ne zadovoljava postavljene kriterijume, te stoga nije podoban izbor za format skladišta podataka za skladištenje tabela integrala.

3.1.3 YAML

YAML (eng. *YAML Ain't Markup Language*) [41] je tekstualni format za jednostavnu razmenu strukturiranih podataka. Autori YAML standarda na naslovnoj stranici svoje zvanične prezentacije [41] opisuju format na sledeći način:

> „YAML is a human friendly data serialization standard for all programming languages."

Autori navode [42] da je YAML izgrađen na konceptima iz programskih jezika kao što su C [43], Java [44], Perl [45], Python [32] i Ruby [46]. Clark Evans je 2001. godine osmislio inicijalnu specifikaciju YAML formata [47], dok su mu se u razvoju kasnijih specifikacija pridružili Ingy döt Net i Oren Ben-Kiki [42].

YAML je opšti format za razmenu strukturiranih podataka, ali je pri njegovom dizajnu posebna pažnja posvećena jednostavnoj upotrebi u često potrebnim slučajevima korišćenja, kao što su [42]:

- konfiguracione datoteke
- dnevnici događaja
- razmena poruka između više različitih procesa
- razmena podataka između više različitih programskih jezika
- serijalizacija složenih struktura podataka

Štaviše, autori su se prilikom razvoja standarda vodili sledećom idejom [42]:

> „When data is easy to view and understand, programming becomes a simpler task."

Kao jedan od primera jednostavnosti YAML standarda je i to što se struktura može zadavati kroz uvlačenje koda, slično programskom jeziku Python.

Prilikom dizajna YAML standarda postavljeni su sledeći ciljevi (*sic*) [42]:

1. YAML je ljudski lako čitljiv
2. YAML odgovara nativnim strukturama podataka agilnih programskih jezika
3. YAML podaci su portabilni između više programskih jezika
4. YAML poseduje konzistenti model koji podržava generičke alate
5. YAML podržava jedno-prolazno procesiranje
6. YAML je izražajan i proširiv
7. YAML je jednostavan za implementaciju i korišćenje

YAML format poseduje 3 osnovne vrste podataka, tj. *čvorova* ako se koristi YAML terminologija [42]:

1. skalar, koji može biti jedan od atomičnih tipova [48]:
   (a) broj, koji može biti celobrojan ili u pokretnom zarezu
   (b) string, koji je sekvenca 0 ili više *Unicode* karaktera
   (c) logički tip
   (d) base64 [49] enkodirana niska binarnih podataka
   (e) tačka u vremenu, koja se sastoji od datuma i vremena zapisanih u podskupu ISO 8601 [50] formata, uz izmene [51] predložene od strane organizacije *World Wide Web Consortium* (skr. *W3C*) sa ciljem smanjenja programske složenosti i pojave mogućih grešaka
   (f) *null*, koji je specijalni tip za označavanje nedostajućih vrednosti

2. sekvenca, koja je uređen niz 0 ili više čvorova

3. mapiranje, koje predstavlja neuređeni skup 0 ili više parova *ključ:vrednost*, pri čemu svaki *ključ* mora biti jedinstven

Tokom 2005. godine autori YAML standarda načinili su interesantno zapažanje [41]:

> „08-APR-2005 – As it turns out, YAML is a superset of the JSON serialization language"

Štaviše, glavni uzrok razvoja verzije 1.2 YAML standarda [52] je ozvaničenje i formalna specifikacija YAML formata kao formalnog nadskupa JSON formata. Usled toga je, u praksi, svaki ispravan zapis u JSON formatu istovremeno i ispravan zapis u YAML formatu (dok obrnuto ne mora da važi).

Iako ranije verzije YAML standarda nisu bile striktno kompatibilne sa JSON formatom, razlike su bile retko uočljive. Stoga su YAML 1.1 parseri u praksi generalno mogli da obrade većinu JSON zapisa.

YAML poseduje nekoliko prednosti u odnosu na JSON format, kao što su: komentari unutar podataka, proširivi tipovi podataka, reference na postojeće čvorove, navođenje stringova bez njihovog obaveznog uokviravanja u navodnike, posebne vrste podataka koje mogu da održe redosled ključeva u čvoru za mapiranje [42, 53]. Navedene prednosti imaju praktičan značaj, npr. komentari u okviru konfiguracionih datoteka značajno poboljšavaju čitljivost i mogu konfiguracionu datoteku učiniti praktično samo-dokumentujućom [54].

Usled svojih prednosti u odnosu na JSON, YAML u poslednje vreme preuzima primat i postaje jedan od najpopularnijih formata za razmenu strukturiranih podataka u Open Source svetu. Veći Open Source projekti kao što su SaltStack [55], Django [56], Ruby on Rails [57], Docker [54, 58] već ga koriste kao svoj osnovni format za razmenu podataka, konfiguracione datoteke, serijalizaciju složenih struktura podataka, formalne opise stanja složenih sistema, itd.

U daljoj diskusiji težište će biti na verziji 1.1 YAML standarda [42], pošto nju najčešće i implementiraju programske biblioteke.

Za programski jezik Python biblioteka PyYAML [59, 60] smatra se referentnom implementacijom YAML 1.1 standarda.

Na listingu 3.3 dat je kratak primer jednog zapisa tabele integrala u YAML formatu, dobijen korišćenjem `beam_integrals` projekta u sledećoj konfiguraciji:

- za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)
- za integral I_3
- određeni integral se izračunava do $a = 1.0$
- tabela integrala se kreira za sve kombinacije m i n do 3.-eg moda
- svi proračuni su vršeni u podrazumevanoj radnoj preciznosti za `beam_integrals` projekat

```
- m: 1
  n: 1
  integral: 1.5925953511467401
  error: 9.999999999999999e-197
- m: 1
  n: 2
  integral: 2.506316581931779
  error: 9.999999999999999e-190
- m: 1
  n: 3
  integral: 2.1328568885828862
  error: 9.999999999999999e-170
- m: 2
  n: 1
  integral: -15.706817907681641
  error: 1.0e-188
- m: 2
  n: 2
  integral: -12.816540145876225
  error: 9.999999999999999e-169
- m: 2
  n: 3
  integral: 3.173235940357649
  error: 9.999999999999999e-173
- m: 3
  n: 1
  integral: 37.42626772620566
  error: 9.999999999999999e-167
- m: 3
  n: 2
```

```
  integral: -8.885152517472523
  error: 1.0e-171
- m: 3
  n: 3
  integral: -45.975507298454154
  error: 1.0e-320
```

Listing 3.3: Kratak primer tabele integrala, snimljene u YAML formatu

Analiza podobnosti YAML formata skladišta podataka, na osnovu prethodno postavljenih kriterijuma sa strane 49:

1. YAML format je u potpunosti nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika.

2. Sâm YAML format ne podržava transparentnu kompresiju podataka, ali se rezultujuća datoteka skladišta može zasebno kompresovati željenim metodom kompresije.

3. YAML je tekstualni format koji je istovremeno i mašinski i ljudski lako čitljiv.

4. Mnogi programski jezici poseduju odgovarajuću programsku podršku za rad sa YAML formatom, bilo u okviru samog programskog jezika (npr. Ruby) ili uz korišćenje dodatne programske biblioteke.

5. Usled prirode YAML formata random pristup pojedinačnim podacima jedino je ostvariv kroz parsiranje celokupnog YAML zapisa do tražene tabele i njenog učitavanja u radnu memoriju, što se ne može smatrati efikasnom operacijom.

6. Sâm YAML format ne pruža mogućnost kontekstualne ugradnje metapodataka uz same podatke. Taj se mehanizam može donekle simulirati kroz proširivanje logičke strukture tabele integrala i samog formata skladišta, ali na način koji nije transparentan za korisnika.

7. YAML format pruža mogućnost razdvajanja različitih tabela integrala u zasebne logičke grupe unutar jednog skladišta. To se ostvaruje grupisanjem podataka u međusobno

nezavisne entitete (YAML koristi termin „dokumenti”) u okviru istog YAML zapisa [42, 53]. YAML dokumenti su međusobno razdvojeni separatorom `---`, dok terminator `...` eksplicitno prekida trenutni dokument. Međutim, problematična je činjenica da su YAML dokumenti suštinski „bezimeni” [61] i da se različitim dokumentima pristupa u strogo sekvencijalnom redosledu. Usled toga mora se na neki način unapred proračunati u kom dokumentu se nalazi tražena tabela i potom parsirati celokupan sadržaj YAML zapisa dok se dođe do traženog dokumenta. Time se problemi opisani u analizi 5.-og kriterijuma samo dodatno inteziviraju, naročito ako naredno traženi podaci ne pripadaju istoj tabeli integrala.

8. YAML format pruža mogućnost deduplikacije tabela integrala unutar jednog skladišta, na način koji je transparentan za korisnika i nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika. To se ostvaruje kroz mehanizam YAML promenljivih (čvorovi sa prefiksom `&`) i referenci (čvorovi sa prefiksom `*`). Pošto se neki čvor definiše kao promenljiva on se kasnije može, po potrebi, referencirati više puta u YAML zapisu [42, 53].

Na osnovu izloženog smatra se da YAML ne zadovoljava postavljene kriterijume, te stoga nije podoban izbor za format skladišta podataka za skladištenje tabela integrala.

3.1.4 HDF5

HDF5 (eng. *Hierarchical Data Format 5*) [62] je format za skladištenje i organizaciju velikih količina numeričkih podataka. Njegove prvobitne verzije su proistekle na NCSA (eng. *National Center for Supercomputing Applications*) [63], ali je kasniji razvoj i brigu preuzela neprofitna organizacija *HDF Group* [64].

HDF5 je proširiv otvoreni standard, sadržan od platformski nezavisnih tehnologija koje su dostupne pod Open Source licencama. Sâm HDF5 format je dizajniran sa ciljem kreiranja skladišta koja su samo-opisna, fleksibilna, izrazito brza i efikasna u

pristupu skladištenim podacima.

HDF5 datoteka poseduje 2 osnovna tipa objekata [65]:

1. HDF5 grupa, koja predstavlja skup 0 ili više HDF5 objekata

2. HDF5 tabela (*dataset* u HDF5 terminologiji), koja predstavlja korisničke podatke organizovane u višedimenzionalni niz elemenata

Rad sa HDF5 grupama i tabelama je donekle sličan radu sa direktorijumima i datotekama pod Unix familijom operativnih sistema. Svaki objekat u HDF5 datoteci poseduje apsolutnu putanju, analogno Unix direktorijumima i datotekama, koja se koristi za adresiranje:

- `/` predstavlja osnovnu grupu, tzv. koren

- `/foo` predstavlja grupu sa nazivom `foo`, koja pripada korenu

- `/foo/bar` predstavlja tabelu sa nazivom `bar`, koja pripada grupi `foo`, koja potom pripada korenu

Jedna od interesantnih osobina HDF5 formata je što, sam po sebi, nema praktičnih ograničenja na veličinu datoteke, kao ni na veličinu i broj objekata u okviru datoteke [66, 67].

Usled svojih brojnih prednosti [67] HDF5 je postao veoma popularan format skladišta podataka, naročito u akademskim ustanovama, velikim bankarskim sistemima i aero-industriji [68, 69]. Svoju vrednost HDF5 format je dokazao u većem broju projekata [70], pri čemu se Boeing [69] i NASA [71, 72] posebno izdvajaju kao njegovi najveći i najzahtevniji korisnici.

Za programski jezik Python postoje 2 biblioteke koje se mogu smatrati referentnim implementacijama programske podrške za rukovanje podacima u HDF5 formatu:

1. PyTables [73, 74]

2. h5py [75–78]

Ova 2 projekta su dizajnirana sa različitim ciljevima.

h5py projekat za cilj ima ostvarivanje što je bliže mogućeg mapiranja HDF5 standarda i njegovih mogućnosti na Python i njegovu popularnu numeričku biblioteku NumPy [79, 80]. h5py poseduje 2 različita nivoa programskog interfejsa (eng. *Application programming interface*, skr. *API*):

- niži nivo pruža Python kodu direktan pristup gotovo celokupnom zvaničnom HDF5 API-u za programski jezik C
- viši nivo omogućava lako korišćenje postojećih Python i NumPy objekata, konvencija i metoda za rukovanje podacima u HDF5 formatu

Za razliku od njega, PyTables projekat izgrađuje svoj poseban, novi, sloj apstrakcije nad HDF5 i NumPy API-em, sa ciljem da maksimalno olakša rukovanje podacima u HDF5 formatu. PyTables nema za cilj da po svaku cenu pruži pristup celokupnom zvaničnom HDF5 API-u, već da pruži fleksibilan HDF5 bazirani alat za jednostavan rad sa velikim količinama podataka (koje mogu biti i značajno veće od kapaciteta radne memorije). Značajna prednost PyTables biblioteke je što se struktura HDF5 tabela može definisati deklarativno, kroz nasleđivanje odgovarajuće Python klase i navođenje željenih atributa koji će na bazi toga (zahvaljujući introspekciji i metaprogramiranju) formirati formalni opis i strukturu HDF5 tabele. Takođe, PyTables proširuje HDF5 standard moćnim jezikom i sistemom za pretragu [81, 82], uz dodatu mogućnost indeksiranja HDF5 tabela zarad značajno bržih pretraga [83].

Detaljnije poređenje PyTables i h5py biblioteka razmatrano je u samoj dokumentaciji za PyTables i h5py, u okviru njihovih često postavljenih pitanja [84, 85].

Za potrebe ovog istraživanja koristiće se PyTables biblioteka, usled svoje praktičnosti.

Analiza podobnosti HDF5 formata skladišta podataka, na osnovu prethodno postavljenih kriterijuma sa strane 49:

1. HDF5 format je u potpunosti nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika.

2. HDF5 format podržava transparentnu kompresiju podataka [86]. Dostupan je veći broj algoritama za kompresiju [86–88] različitih karakteristika, svrhe i stepena efikasnosti: ZLIB [89], SZIP [90], LZO [91], BZIP2 [92], LZF [93], Blosc [94], MAFISC [95], Snappy [96], LZ4 [97], JPEG-XR [98], ...

   HDF5 standard podržava uvođenje novih algoritama kompresije, ali ih je pre upotrebe potrebno zvanično registrovati [86]. Sâm mehanizam kompresije je nezavisan od hardverske platforme, operativnog sistema, konkretnog programskog jezika i algoritma kompresije. Takođe, registrovani algoritmi za kompresiju su standardni i značajnom većinom poseduju odgovarajuću Open Source implementaciju.

3. HDF5 je binarni format koji je istovremeno i mašinski i ljudski lako čitljiv. Ljudska čitljivost se postiže korišćenjem besplatnih ili komercijalnih vizuelnih alata za pregled i izmenu HDF5 datoteka, kao što su: HDFView [99], HDF Compass [100], ViTables [101], HDF Explorer [102], ...

4. Mnogi programski jezici poseduju odgovarajuću programsku podršku za rad sa HDF5 formatom, bilo u okviru samog programskog jezika (Mathematica [103], MATLAB [104]) ili uz korišćenje dodatne programske biblioteke.

5. HDF5 tabele se na disku mogu serijalizovati ili u kontinualnom ili razlomljenom (eng. *chunked*) rasporedu [76,105]. Sa kontinualnim rasporedom višedimenzionalni niz elemenata, koji čini HDF5 tabelu, se prvo projektuje na jednodimenzionalni niz (slično algoritmu koji koristi programski jezik C) koji se potom skupa serijalizuje na disk kao jedan blok podataka. Razlomljeni raspored seče višedimenzionalni niz elemenata u zasebne blokove podataka, koji se potom serijalizuju na disk, nezavisno od svog redosleda. Korisnik pri tome može da bira način sečenja višedimenzionalnog niza elemenata i veličinu odredišnog bloka podataka.

   Pošto se tako isečeni blokovi podataka mogu zasebno učitavati, random pristup pojedinačnim podacima je prilično

efikasan. Štaviše, zahvaljujući moći PyTables sistema za indeksiranje [83] i pretragu HDF5 tabela [81, 82] random pristup podacima, zasnovan na dobro definisanim kriterijumima pretrage, može se izvršavati i u gotovo realnom vremenu [106].

6. HDF5 format omogućava kontekstualnu ugradnju metapodataka uz same podatke, koristeći HDF5 atribute [107]. Metapodaci se mogu vezati za bilo koju HDF5 grupu ili tabelu.

7. HDF5 format omogućava, na efikasan način, razdvajanje različitih tabela integrala u zasebne logičke grupe unutar jednog skladišta, smeštanjem različitih tabela integrala u zasebne HDF5 tabele [65,108]. Pri tome, HDF5 format održava efikasnost sekvencijalnog i random pristupa pojedinačnim podacima.

8. HDF5 format pruža mogućnost deduplikacije tabela integrala unutar jednog skladišta, na način koji je transparentan za korisnika i nezavisan od hardverske platforme, operativnog sistema i konkretnog programskog jezika. To se ostvaruje kroz mehanizam HDF5 veza [109–111], koje mogu biti tvrde (eng. *hard link*) ili simboličke (eng. *symbolic link*). Simboličke veze mogu biti meke (eng. *soft link*) kada ukazuju na objekte unutar iste HDF5 datoteke, ili eksterne kada ukazuju na objekte van izvorne HDF5 datoteke. Na jedan postojeći HDF5 objekat može ukazivati nula ili više veza.

   Za potrebe ovog istraživanja koristiće se tvrde veze, usled svoje praktičnosti. Naime, tvrde veze prilikom korišćenja ne zahtevaju postupak dereferenciranja [109, 110], za razliku od simboličkih veza.

Na osnovu izloženog smatra se da HDF5 zadovoljava postavljene kriterijume, te je stoga podoban izbor za format skladišta podataka za skladištenje tabela integrala.

3.2 Opis logičke strukture tabela integrala i formata skladišta

HDF5 datoteka sadrži samo jednu grupu (koren), kojoj direktno pripadaju sve HDF5 tabele koje istovremeno i čine tabele integrala.
Svaka HDF5 tabela integrala može posedovati sledeće kolone:

1. `m`, integraciona promenljiva definisana jednačinom 1.34
2. `n`, integraciona promenljiva definisana jednačinom 1.34
3. `t`, integraciona promenljiva definisana jednačinom 1.34
4. `v`, integraciona promenljiva definisana jednačinom 1.34
5. `integral.str` predstavlja vrednost određenog integrala, zapisanog u originalnoj preciznosti kao string [1]
6. `error.str` predstavlja vrednost apsolutne greške pri numeričkoj integraciji, zapisane u originalnoj preciznosti kao string
7. `integral.float64` predstavlja vrednost određenog integrala, zapisanog u IEEE 754 binary64 preciznosti (kreiran na bazi kolone `integral.str`)
8. `error.float64` predstavlja vrednost apsolutne greške pri numeričkoj integraciji, zapisane u IEEE 754 binary64 preciznosti (kreirana na bazi kolone `error.str`)

Pri tome se za kolone `m`, `n`, `t` i `v` može izvršiti optimizacija koja će biti predložena u sekciji 3.3.3.

[1] String tip se koristi pošto HDF5 format ne poseduje predefinisani tip za proizvoljno precizne brojeve u pokretnom zarezu [112, 113]

Sâma HDF5 datoteka poseduje sledeće metapodatke, smeštene u koren grupu:

1. `created_at` predstavlja datum i vreme kreiranja skladišta
2. `generator.name` predstavlja naziv alata koji je kreirao tabele integrala
3. `generator.version` predstavlja verziju alata koji je kreirao tabele integrala
4. `decimal_precision` predstavlja radnu decimalnu preciznost koja je korišćena pri kreiranju tabela integrala
5. `max_mode` predstavlja maksimalni mod do kog se kreiraju tabele integrala
6. `beam_type.name` predstavlja naziv graničnog uslova za koji su kreirane tabele integrala
7. `beam_type.id` predstavlja id kôd graničnog uslova za koji su kreirane tabele integrala

Svaka HDF5 tabela integrala poseduje sledeće metapodatke:

1. `used_variables.list` predstavlja spisak integracionih promenljivih koje definišu analitičku formulu datog integrala
2. `used_variables.num` predstavlja broj integracionih promenljivih koje definišu analitičku formulu datog integrala

3.3 Optimizacija vremena potrebnog za kreiranje tabela integrala

3.3.1 Identifikacija integrala sa istovetnom kanoničkom formom

Analizom definicija integrala datih u jednačinama 1.35–1.47 uočava se da pojedini integrali poseduju istovetne kanoničke forme:

$$\begin{aligned} I_1 &\equiv I_{21} \\ &= \int_0^a Y_m(y) Y_n(y) dy \end{aligned} \tag{3.1}$$

$$\begin{aligned} I_2 &\equiv I_4 \equiv I_6 \equiv I_8 \equiv I_{25} \\ &= \int_0^a Y'_m(y) Y'_n(y) dy \end{aligned} \tag{3.2}$$

$$\begin{aligned} I_3 &\equiv I_{22} \\ &= \int_0^a Y''_m(y) Y_n(y) dy \end{aligned} \tag{3.3}$$

$$\begin{aligned} I_5 &\equiv I_{23} \\ &= \int_0^a Y_m(y) Y''_n(y) dy \end{aligned} \tag{3.4}$$

$$\begin{aligned} I_7 &\equiv I_{24} \\ &= \int_0^a Y''_m(y) Y''_n(y) dy \end{aligned} \tag{3.5}$$

Navedena analiza razmatra samo 13 integrala koji su potrebni za rešavanje problema stabilnosti u HCFSM. Međutim, broj podudaranja kanoničkih formi bi se značajno povećao ako bi se u analizu uključili svih 113 integrala koje definiše HCFSM.

Vreme potrebno za kreiranje tabela integrala može se značajno skratiti identifikacijom integrala sa istovetnom kanoničkom formom, uz uvođenje mehanizma deduplikacije istovetnih tabela integrala unutar jednog skladišta. Na primeru integrala I_2 optimizovani algoritam za deduplikaciju će:

1. Automatski uočiti postojanje integrala sa istovetnom kanoničkom formom. Za integral I_2 se iz jednačine 3.2 može videti da su to integrali I_4, I_6, I_8 i I_{25}.

2. Izračunati i kreirati tabelu integrala samo za integral I_2.

3. Za integrale I_4, I_6, I_8 i I_{25} kreirati referencu (tzv. *hard link* u HDF5 terminologiji [109,110]) na sadržaj tabele integrala I_2.

Predstavljena optimizacija dovodi do smanjenja utroška računarskih resursa prilikom kreiranja tabela integrala, uz istovremeno znatno smanjenje utroška skladišnog prostora za kreirane tabele integrala. Primera radi, za integral I_2 korišćenjem predstavljene optimizacije vremenska kompleksnost algoritma za kreiranje tabela integrala dodatno se smanjuje *5 puta*.

3.3.2 Simetrična priroda pojedinih integrala

HCFSM definiše integral I_{26} na sledeći način:

$$I_{26} = \int_0^a Y_m(y)Y_n(y)Y_t(y)Y_v(y)dy \tag{3.6}$$

Ako se iz jednačine 3.6 izdvoji podintegralna funkcija F za integral I_{26}:

$$F(y, m, n, t, v) = Y_m(y)Y_n(y)Y_t(y)Y_v(y) \tag{3.7}$$

tada se može utvrditi da su integracione promenljive m, n, t i v međusobno simetrične, tj. da važi sledeća zakonitost:

$$\begin{aligned}
&F(y, m = A, n = B, t = C, v = D) \\
\equiv\ &F(y, m = A, n = B, t = D, v = C) \\
\equiv\ &F(y, m = A, n = C, t = B, v = D) \\
\equiv\ &F(y, m = A, n = C, t = D, v = B) \\
\equiv\ &F(y, m = A, n = D, t = B, v = C) \\
\equiv\ &F(y, m = A, n = D, t = C, v = B) \\
\equiv\ &F(y, m = B, n = A, t = C, v = D) \\
\equiv\ &F(y, m = B, n = A, t = D, v = C) \\
\equiv\ &F(y, m = B, n = C, t = A, v = D) \\
\equiv\ &F(y, m = B, n = C, t = D, v = A) \\
\equiv\ &F(y, m = B, n = D, t = A, v = C) \\
\equiv\ &F(y, m = B, n = D, t = C, v = A) \\
\equiv\ &F(y, m = C, n = A, t = B, v = D) \\
\equiv\ &F(y, m = C, n = A, t = D, v = B) \\
\equiv\ &F(y, m = C, n = B, t = A, v = D) \\
\equiv\ &F(y, m = C, n = B, t = D, v = A) \\
\equiv\ &F(y, m = C, n = D, t = A, v = B) \\
\equiv\ &F(y, m = C, n = D, t = B, v = A) \\
\equiv\ &F(y, m = D, n = A, t = B, v = C) \\
\equiv\ &F(y, m = D, n = A, t = C, v = B) \\
\equiv\ &F(y, m = D, n = B, t = A, v = C) \\
\equiv\ &F(y, m = D, n = B, t = C, v = A) \\
\equiv\ &F(y, m = D, n = C, t = A, v = B) \\
\equiv\ &F(y, m = D, n = C, t = B, v = A)
\end{aligned} \tag{3.8}$$

pri čemu za A, B, C i D važi:

$$\begin{aligned}
A &\in (1, \ldots, 100) \\
B &\in (1, \ldots, 100) \\
C &\in (1, \ldots, 100) \\
D &\in (1, \ldots, 100)
\end{aligned} \tag{3.9}$$

U daljem tekstu biće pokazano da se vreme potrebno za kreiranje tabela integrala može značajno skratiti identifikacijom integrala čije su integracione promenljive međusobno simetrične, uz uvođenje mehanizma koji će u okviru jedne tabele integrala izbeći izračunavanje međusobno simetričnih oblika datog integrala.

Postupak rada optimizovanog algoritma koji uzima u obzir simetričnu prirodu pojedinih integrala:

1. Na osnovu metapodataka vezanih za sâm integral algoritam će automatski uočiti da li je dati integral simetrične prirode. Ako nije odustaje se od dalje optimizacije.

2. Na osnovu metapodataka vezanih za sâm integral algoritam će automatski odrediti integracione promenljive i njihov broj

3. Algoritam potom formira sistem za keširanje vrednosti određenih integrala u okviru date tabele integrala, pri čemu se ključ keša određuje na bazi korišćenja brojnog sistema gde je:

   (a) osnova brojnog sistema jednaka zadatom maksimalnom modu do kog su generisani podaci

   (b) broj cifara brojnog sistema jednak broju integracionih promenljivih

   (c) vrednost cifre u brojnom sistemu određena vrednošću pripadajuće joj integracione promenljive

4. Na osnovu jednačine 3.8 može se zaključiti da vrednost određenog integrala (koji je simetrične prirode) ne zavisi od međusobnog redosleda vrednosti njegovih integracionih promenljivih. Stoga se vrednosti integracionih promenljivih uvek mogu sortirati u rastući niz, bez uticaja na vrednost određenog integrala. Iz toga sledi da se i cifre opisanog brojnog sistema takođe mogu sortirati u rastući niz, bez uticaja na vrednost određenog integrala.

   Tako definisan brojni sistem će automatski grupisati međusobno simetrične oblike datog integrala u isti broj, čime će se oni grupisati i u isti ključ keša.

5. Tokom kreiranja tabele integrala algoritam će, na osnovu vrednosti integracionih promenljivih, za svaki naredni red u tabeli prvo odrediti odgovarajući ključ keša:

   (a) ukoliko se ključ već nalazi u kešu koristiće se keširana vrednost određenog integrala, čime je algoritam uspešno izbegao izračunavanje međusobno simetričnih oblika datog integrala

   (b) u suprotnom izračunaće se vrednost određenog integrala i potom smestiti u keš za dalju upotrebu

6. Po uspešnom završetku kreiranja tabele integrala keširane vrednosti više nisu potrebne. Stoga će algoritam uništiti celokupni sadržaj keša i time osloboditi zauzetu radnu memoriju.

Definišimo matematičku funkciju za sortiranje uređene n-torke od 4 integracione promenljive u nerastući niz:

$$\text{sorted}\Big((m,n,t,v)\Big) = (a_3,a_2,a_1,a_0) \tag{3.10}$$

pri čemu važe sledeći uslovi:

$$\begin{gathered}\{a_3,a_2,a_1,a_0\} = \{m,n,t,v\} \\ a_3 \geq a_2 \geq a_1 \geq a_0\end{gathered} \tag{3.11}$$

Goreopisani postupak formiranja ključa za keširanje vrednosti određenih integrala demonstriraće se na primeru integrala I_{26} (definisanog u jednačini 3.6) pošto je odabrani integral simetrične prirode i zavisi od 4 integracione promenljive (m, n, t i v). Ako se tabele integrala kreiraju do 100.-og moda, tada se matematička funkcija za formiranje ključa za keširanje definiše na sledeći način:

$$\begin{aligned}\text{key}(m,n,t,v) &= \text{sorted}\Big((m,n,t,v)\Big)_{100} \\ &= (a_3,a_2,a_1,a_0)_{100} \\ &= a_3\times 100^3 + a_2\times 100^2 + a_1\times 100^1 + a_0\times 100^0\end{aligned} \tag{3.12}$$

Za potrebe demonstracije poredićе se 3 slučaja:

$$\begin{aligned} C_1 &= (m_1, n_1, t_1, v_1) = (1, 2, 3, 4) \\ C_2 &= (m_2, n_2, t_2, v_2) = (4, 1, 2, 3) \\ C_3 &= (m_3, n_3, t_3, v_3) = (1, 1, 2, 3) \end{aligned} \tag{3.13}$$

za koje (iz jednačine 3.13) važi:

$$\begin{aligned} \{m_3, n_3, t_3, v_3\} &\neq \{m_1, n_1, t_1, v_1\} \\ \{m_3, n_3, t_3, v_3\} &\neq \{m_2, n_2, t_2, v_2\} \\ \{m_1, n_1, t_1, v_1\} &= \{m_2, n_2, t_2, v_2\} \end{aligned} \tag{3.14}$$

iz čega se može potvrditi da su oblici integrala I_{26} međusobno simetrični za slučajeve C_1 i C_2. Takođe se može videti da oblik integrala za slučaj C_3 nije simetričan ni sa oblikom za slučaj C_1 ni za slučaj C_2.

Kada se funkcija key, definisana jednačinom 3.12, primeni na slučajeve opisane u jednačini 3.13 tada važe sledeće jednakosti:

$$\begin{aligned} \text{key}(m_1, n_1, t_1, v_1) &= \text{sorted}\Big((1, 2, 3, 4)\Big)_{100} \\ &= (4, 3, 2, 1)_{100} \\ &= 4 \times 100^3 + 3 \times 100^2 + 2 \times 100^1 + 1 \times 100^0 \\ &= 4,000,000 + 30,000 + 200 + 1 \\ &= 4,030,201 \end{aligned} \tag{3.15}$$

$$\begin{aligned} \text{key}(m_2, n_2, t_2, v_2) &= \text{sorted}\Big((4, 1, 2, 3)\Big)_{100} \\ &= (4, 3, 2, 1)_{100} \\ &= 4 \times 100^3 + 3 \times 100^2 + 2 \times 100^1 + 1 \times 100^0 \\ &= 4,000,000 + 30,000 + 200 + 1 \\ &= 4,030,201 \end{aligned} \tag{3.16}$$

$$
\begin{aligned}
\text{key}(m_3, n_3, t_3, v_3) &= \text{sorted}\Big((1,1,2,3)\Big)_{100} \\
&= (3,2,1,1)_{100} \\
&= 3 \times 100^3 + 2 \times 100^2 + 1 \times 100^1 + 1 \times 100^0 \\
&= 3,000,000 + 20,000 + 100 + 1 \\
&= 3,020,101
\end{aligned}
\tag{3.17}
$$

Na osnovu jednačina 3.15–3.17 važi:

$$
\begin{aligned}
&\text{key}(m_3, n_3, t_3, v_3) \neq \text{key}(m_1, n_1, t_1, v_1) \\
&\text{key}(m_3, n_3, t_3, v_3) \neq \text{key}(m_2, n_2, t_2, v_2) \\
&\text{key}(m_1, n_1, t_1, v_1) = \text{key}(m_2, n_2, t_2, v_2)
\end{aligned}
\tag{3.18}
$$

iz čega se može potvrditi da su se za integracione promenljive (m_1, n_1, t_1, v_1) i (m_2, n_2, t_2, v_2) oblici integrala I_{26} automatski grupisali u isti ključ keša, dok to ne važi za (m_3, n_3, t_3, v_3) čiji se oblik smestio u zaseban ključ keša. Stoga se, i na osnovu jednačina 3.15–3.18, takođe može potvrditi da su oblici integrala I_{26} međusobno simetrični za slučajeve C_1 i C_2. Takođe se može videti da oblik integrala za slučaj C_3 nije simetričan ni sa oblikom za slučaj C_1 ni za slučaj C_2.

Na osnovu iskazanog može se tvrditi da, u opštem slučaju integrala koji su simetrične prirode, vrednost određenog integrala ne zavisi od međusobnog redosleda vrednosti njegovih integracionih promenljivih. Tada će goreopisani postupak formiranja ključa za keširanje vrednosti određenih integrala automatski grupisati u isti ključ keša sve integracione promenljive koje su međusobno simetrične, dok će od njih automatski razdvojiti one integracione promenljive koje im nisu srodne.

Značajna prednost datog rešenja je što se celokupni opseg

vrednosti ključa keša:

$$\begin{aligned}
\text{key}_{\min} &= \text{key}(m_{\min}, n_{\min}, t_{\min}, v_{\min}) \\
&= \text{key}(1,1,1,1) \\
&= \text{sorted}\Big((1,1,1,1)\Big)_{100} \\
&= (1,1,1,1)_{100} \\
&= 1 \times 100^3 + 1 \times 100^2 + 1 \times 100^1 + 1 \times 100^0 \\
&= 1,000,000 + 10,000 + 100 + 1 \\
&= 1,010,101
\end{aligned} \tag{3.19}$$

$$\begin{aligned}
\text{key}_{\max} &= \text{key}(m_{\max}, n_{\max}, t_{\max}, v_{\max}) \\
&= \text{key}(100,100,100,100) \\
&= \text{sorted}\Big((100,100,100,100)\Big)_{100} \\
&= (100,100,100,100)_{100} \\
&= 100 \times 100^3 + 100 \times 100^2 + 100 \times 100^1 + 100 \times 100^0 \\
&= 100,000,000 + 1,000,000 + 10,000 + 100 \\
&= 101,010,100
\end{aligned} \tag{3.20}$$

može bez problema smestiti u 32-bitnu celobrojnu promenljivu, tj. $(\text{key}_{\min}, \ldots, \text{key}_{\max}) \in (\text{int32}_{\min}, \ldots, \text{int32}_{\max})$ [114–118]:

$$\begin{aligned}
\text{int32}_{\min} &= -2^{31} \\
&= -2,147,483,648
\end{aligned} \tag{3.21}$$

$$\begin{aligned}
\text{int32}_{\max} &= +2^{31} - 1 \\
&= +2,147,483,647
\end{aligned} \tag{3.22}$$

čime se značajno smanjuje opterećenje radne memorije i ubrzava formiranje keš ključa [119–122], što pojednostavljuje i značajno ubrzava međusobno poređenje keš ključeva – a samim tim i pojednostavljuje i značajno skraćuje vreme potrebno za kreiranje tabela integrala.

Na osnovu jednačina 3.8 i 3.9 ukupan broj oblika integrala (bez obzira da li je integral simetrične prirode) može se izvesti i

iz broja varijacija sa ponavljanjem [123–125]:

$$V_p = n^k \tag{3.23}$$

gde je:

- n zadati maksimalni mod do kog se generišu podaci
- k broj integracionih promenljivih od kojih zavisi zadati integral

Međutim, kao što je već pokazano, na osnovu jednačine 3.8 može se zaključiti da vrednost određenog integrala (koji je simetrične prirode) ne zavisi od međusobnog redosleda vrednosti njegovih integracionih promenljivih. Takođe, uvidom u oblik integrala za slučaj C_3 iz jednačine 3.13 može se zaključiti da je moguće da se istovetna vrednost pojavi u više integracionih promenljivih jednog istog oblika integrala. Nezavisnost redosleda članova i mogućnost ponavljanja istovetne vrednosti u više članova grupe su ključne osobine koje opisuju kombinacije sa ponavljanjem [123–125]. Stoga se, u opštem slučaju integrala koji su simetrične prirode, teoretski minimalan potreban broj oblika integrala (koji mogu opisati sve moguće oblike integrala) može izvesti iz broja kombinacija sa ponavljanjem [123–125]:

$$K_p = \left(\!\binom{n}{k}\!\right) = \binom{n+k-1}{k} = \binom{p}{k} = \frac{p!}{k!(p-k)!} \tag{3.24}$$

gde je:

- n zadati maksimalni mod do kog se generišu podaci
- k broj integracionih promenljivih od kojih zavisi zadati integral

Pretpostavka je da će goreopisani postupak formiranja ključa za keširanje takođe dovesti do optimalnog broja oblika integrala, koji će biti istovetan teoretski minimalnom broju oblika integrala izvedenom iz broja kombinacija sa ponavljanjem i datom u jednačini 3.24.

Na listingu 3.4 se, kroz interaktivnu Python sesiju, proverava korektnost algoritma formiranja ključa za keširanje, uz upoređivanje njegovog očekivanog stepena ubrzanja sa teoretski dobijenim rezultatima (izvedenih na osnovu jednačine 3.24):

```
>>> import itertools

# Compute integrals up to the 100th mode
>>> max_mode = 100
>>> modes = range(1, max_mode+1)

# Cache key function, based on the ‘BaseIntegralWithSymetricVariables.cache_key‘
# method from the ‘beam_integrals.integrals‘ module
>>> def cache_key(values):
...     return sum(
...         val * max_mode**idx
...         for idx, val in enumerate(sorted(values))
...     )

# A helper function, used to minimize code duplication in ‘theoretical_speedup‘
# and ‘expected_speedup‘ functions
>>> def _speedup_helper(num_integral_vars, num_optimized):
...     num_total = max_mode**num_integral_vars
...     speedup = 1.0*num_total/num_optimized
...
...     print "\t%d\ttotal num. of computed integrals" % num_total
...     print "\t%d\toptimized num. of computed integrals" % num_optimized
...     print "\t%.4f\tspeedup of using the optimized algo." % speedup
...
...     return num_optimized

# Iterate over the Cartesian product of integral variables up to the ‘max_mode‘.
# Eliminate any unnecessary integral computations, where 2 or more computations
# are sharing the same cache key (as determined by the ‘cache_key‘ function).
>>> def expected_speedup(num_integral_vars):
...     return _speedup_helper(
...         num_integral_vars,
...         num_optimized = len(set(
...             cache_key(v)
...             for v in itertools.product(modes, repeat=num_integral_vars)
...         ))
...     )

# Iterate over all ‘k‘-length combinations with repetions for ‘modes‘ up to the
# ‘max_mode‘, where ‘k‘ is the number of integral variables.
# Given the math, this should result in the same optimized number of computed
# integrals as the ‘expected_speedup‘ function.
>>> def theoretical_speedup(num_integral_vars):
...     return _speedup_helper(
...         num_integral_vars,
...         num_optimized = len(list(
...             itertools.combinations_with_replacement(modes, num_integral_vars)
...         ))
...     )
```

```
>>> def test_speedup_assumption(num_integral_vars):
...     print 'Theoretical speedup:'
...     t_num_optimized = theoretical_speedup(num_integral_vars)
...
...     print 'Expected speedup:'
...     e_num_optimized = expected_speedup(num_integral_vars)
...
...     # Fail the test if the methods differ in their optimized number
...     # of computed integrals
...     assert t_num_optimized == e_num_optimized

# Verify the speedup assumption for integrals with 2 integral variables
>>> test_speedup_assumption(num_integral_vars=2)
Theoretical speedup:
    10000   total num. of computed integrals
    5050    optimized num. of computed integrals
    1.9802  speedup of using the optimized algo.
Expected speedup:
    10000   total num. of computed integrals
    5050    optimized num. of computed integrals
    1.9802  speedup of using the optimized algo.

# Verify the speedup assumption for integrals with 3 integral variables
>>> test_speedup_assumption(num_integral_vars=3)
Theoretical speedup:
    1000000 total num. of computed integrals
    171700  optimized num. of computed integrals
    5.8241  speedup of using the optimized algo.
Expected speedup:
    1000000 total num. of computed integrals
    171700  optimized num. of computed integrals
    5.8241  speedup of using the optimized algo.

# Verify the speedup assumption for integrals with 4 integral variables
>>> test_speedup_assumption(num_integral_vars=4)
Theoretical speedup:
    100000000       total num. of computed integrals
    4421275 optimized num. of computed integrals
    22.6179 speedup of using the optimized algo.
Expected speedup:
    100000000       total num. of computed integrals
    4421275 optimized num. of computed integrals
    22.6179 speedup of using the optimized algo.
```

Listing 3.4: Provera korektnosti optimizovanog algoritma koji se oslanja na simetričnu prirodu integrala, uz upoređivanje njegovog očekivanog stepena ubrzanja sa teoretski dobijenim rezultatima

Broj promenljiva	Broj oblika integrala		
	Ukupan	Optimizovan	Ubrzanje
2	10 000	5 050	1.9802
3	1 000 000	171 700	5.8241
4	100 000 000	4 421 275	22.6179

Tabela 3.1: Pregled teoretski dobijenih rezultata sa listinga 3.4

Na osnovu ispisa sa listinga 3.4 formirana je tabela 3.1 koja sadrži pregled rezultata dobijenih goreopisanim teoretskim postupkom, na osnovu kombinacija sa ponavljanjem. Kolona „optimizovan broj oblika integrala" odgovara vrednostima izvedenim iz jednačine 3.24:

$$\begin{aligned} K_p|_{k=2}^{n=100} &= \left(\!\!\binom{100}{2}\!\!\right) = \binom{100+2-1}{2} \\ &= \binom{101}{2} = \frac{101!}{2!(101-2)!} \\ &= \frac{101 \times 100 \times 99!}{2! \times 99!} \\ &= \frac{101 \times 100}{2!} \\ &= 5,050 \end{aligned} \tag{3.25}$$

$$\begin{aligned} K_p|_{k=3}^{n=100} &= \left(\!\!\binom{100}{3}\!\!\right) = \binom{100+3-1}{3} \\ &= \binom{102}{3} = \frac{102!}{3!(102-3)!} \\ &= \frac{102 \times 101 \times 100 \times 99!}{3! \times 99!} \\ &= \frac{102 \times 101 \times 100}{3!} \\ &= 171,700 \end{aligned} \tag{3.26}$$

$$
\begin{aligned}
K_p|_{k=4}^{n=100} &= \left(\!\!\binom{100}{4}\!\!\right) = \binom{100+4-1}{4} \\
&= \binom{103}{4} = \frac{103!}{4!(103-4)!} \\
&= \frac{103 \times 102 \times 101 \times 100 \times 99!}{4! \times 99!} \\
&= \frac{103 \times 102 \times 101 \times 100}{4!} \\
&= 4,421,275
\end{aligned}
\tag{3.27}
$$

dok kolona „ukupan broj oblika integrala" odgovara vrednostima izvedenim iz jednačine 3.23:

$$V_p|_{k=2}^{n=100} = 100^2 = 10,000 \tag{3.28}$$

$$V_p|_{k=3}^{n=100} = 100^3 = 1,000,000 \tag{3.29}$$

$$V_p|_{k=4}^{n=100} = 100^4 = 100,000,000 \tag{3.30}$$

Na osnovu ispisa sa listinga 3.4 formirana je tabela 3.2 koja sadrži pregled rezultata dobijenih iz goreopisanog optimizovanog algoritma koji se oslanja na simetričnu prirodu integrala.

Broj promenljiva	Broj oblika integrala		
	Ukupan	Optimizovan	Ubrzanje
2	10 000	5 050	1.9802
3	1 000 000	171 700	5.8241
4	100 000 000	4 421 275	22.6179

Tabela 3.2: Pregled algoritamski dobijenih rezultata sa listinga 3.4

Analizom izvornog koda i dobijenog ispisa sa listinga 3.4 može se potvrditi korektnost opisanog algoritma formiranja ključa za keširanje i njegova usklađenost sa teoretskim postupkom. Takođe, uporednom analizom tabela 3.1 i 3.2 može se uvideti da su ostvareni rezultati istovetni u oba navedena postupka.

Predstavljena optimizacija dovodi do značajnog smanjenja utroška računarskih resursa prilikom kreiranja tabela integrala, pošto podaci sa tabele 3.2 pokazuju da se vremenska kompleksnost algoritma za kreiranje tabela integrala dodatno smanjuje:

- do $\approx$ *22.62 puta*, ako integral zavisi od 4 integracione promenljive

- do $\approx$ *5.82 puta*, ako integral zavisi od 3 integracione promenljive

- do $\approx$ *1.98 puta*, ako integral zavisi od 2 integracione promenljive

3.3.3 Racionalizacija broja potrebnih integracionih promenljiva

Osnovna verzija algoritma za kreiranje tabela integrala bi mogla da ima sledeći oblik:

```
Input: integral, max_mode
Output: integral_table

for m ← 1 to max_mode do
    for n ← 1 to max_mode do
        for t ← 1 to max_mode do
            for v ← 1 to max_mode do
                x ← integrate(integral, m, n, t, v)
                integral_table[m, n, t, v] ← x
            end
        end
    end
end
```

Međutim ovakva naivna implementacija poseduje ozbiljan nedostatak – za svaku tabelu integrala sračunava se vrednost integrala po svim promenljivama (tj. dekartov proizvod svih mogućih vrednosti promenljiva m, t, v, n) iako pojedini integrali zavise od svega nekoliko promenljivih.

Vreme potrebno za kreiranje tabela integrala može se značajno skratiti uvođenjem mehanizma za racionalizaciju broja potrebnih integracionih promenljiva. Na primeru integrala I_1 optimizovani algoritam će:

1. Na osnovu metapodataka vezanih za sâm integral automatski odrediti zavisne integracione promenljive i njihov broj. Za integral I_1 se iz jednačine 1.35 može videti da taj integral zavisi od 2 integracione promenljive (m i n).

2. Formirati Python iterator [32, 126] koji prolazi samo kroz dekartov proizvod zavisnih integracionih promenljivih, do zadatog maksimalnog moda. Za integral I_1 to je dekartov proizvod svih mogućih vrednosti promenljiva m i n.

3. Izračunati i kreirati tabelu integrala samo za vrednosti integracionih promenljivih dobijenih prolaskom kroz formirani Python iterator. Za integral I_1 vremenska kompleksnost takvog prolaska je $\approx O(\text{max\_mode}^2)$.

Predstavljena optimizacija dovodi do izrazitog smanjenja utroška računarskih resursa prilikom kreiranja tabela integrala, uz istovremeno znatno smanjenje utroška skladišnog prostora za kreirane tabele integrala. Primera radi, za integral I_1 korišćenjem predstavljene optimizacije vremenska kompleksnost algoritma za kreiranje tabela integrala dodatno se smanjuje sa $O(\text{max\_mode}^4)$ na $O(\text{max\_mode}^2)$ što dovodi do *ubrzanja od nekoliko redova veličine*:

$$\begin{aligned} \text{speedup} &\approx \frac{O(\text{max\_mode}^4)}{O(\text{max\_mode}^2)} = \text{max\_mode}^2 = 100^2 \\ &= \mathbf{10^4} \end{aligned} \tag{3.31}$$

3.3.4 Okolnosti koje su dovele do potrebe za distribuiranim sistemom

Projekat `beam_integrals` (opisan u sekciji 4.3) pruža referentnu Open Source implementaciju hibridne metode. Projekat `beam_integrals` poseduje i napredan podsistem za automatsku verifikaciju (opisan u sekciji 4.3.7), koji je baziran na standardnim Python alatima za testiranje [127] i poznatom Open Source projektu `nose` [128].

Jenkins je poznat kao vodeći Open Source *Continuous Integration* server [129] i koristi se za automatsko prevođenje, testiranje i praćenje toka razvoja projekta `beam_integrals` [130].

Naša istraživačka grupa sve svoje Jenkins projekte smešta na naš centralni Jenkins integracioni server [131], čije su osnovne tehničke karakteristike:

- HP Compaq Elite 8300 CMT Desktop Computer [132]
- Intel® Core™ i5-3470 Quad-Core CPU @ 3.20GHz [133]
- Samsung 8 GB RAM DDR3 1600 MHz
- Western Digital WD10EZEX-60Z 1 TB SATA 3.0 HDD [134]
- AMD Radeon HD 7450 DP 1 GB DDR3 [135]
- Intel® 82579LM Gigabit Network Connection [136]

Podsistem za automatsku verifikaciju `beam_integrals` projekta sadrži ≈ 4400 međusobno nezavisnih testova koje naš Jenkins integracioni server automatski izvršava svaki put kada se registruje promena u javno dostupnom repozitorijumu izvornog koda za `beam_integrals` projekat [137].

Jedan od artefakata koji se generišu prilikom testiranja je datoteka `nosetests.xml` [138], koja sadrži rezultate testiranja u standardnom XUnit XML formatu [139, 140].

U datoteci `nosetests.xml` se uz svaki test slučaj beleži i vreme njegovog izvršavanja (realni broj, u sekundama). Na listingu 3.5 dat je kratak primer iz `nosetests.xml` za projekat `beam_integrals`, dobijen sa našeg Jenkins integracionog servera:

```
<testcase classname="tests.beam_types.test_y_m_derivatives"
    name="test_caching(5, 22, 2)" time="0.002"/>
<testcase classname="tests.characteristic_equation_solvers.test_solvers"
    name="test_find_best_root(3, 98)" time="0.148"/>
<testcase classname="tests.integrals.test_integration"
    name="test_integrate(5, 6, 99, None, None, 99)" time="23.927"/>
<testcase classname="tests.integrals.test_integration"
    name="test_integrate(5, 6, 99, None, None, 100)" time="59.615"/>
<testcase classname="tests.integrals.test_integration"
    name="test_integrate(5, 6, 100, None, None, 99)" time="59.598"/>
<testcase classname="tests.integrals.test_integration"
```

```
    name="test_integrate(5, 6, 100, None, None, 100)" time="24.047"/>
<testcase classname="tests.integrals.test_cache_key"
    name="test_integral_with_symetric_variables(('m', 'n'),)" time="0.001"/>
```

Listing 3.5: Izvod iz `nosetests.xml` za projekat `beam_integrals`, puna verzija dostupna na [141]

Pretpostavka je da se instrumentacijom podsistema za automatsku verifikaciju projekta `beam_integrals`, a na osnovu podataka u javno dostupnom `nosetests.xml` [141] sa našeg Jenkins integracionog servera, može napraviti procena vremena potrebnog za sekvencijalno kreiranje tabela integrala. Korišćenjem projekta `beam_integrals` i na osnovu prethodne pretpostavke na listingu 3.6 se, kroz interaktivnu Python sesiju, pravi procena vremena:

```
>>> from lxml import etree
>>> from beam_integrals import DEFAULT_MAX_MODE
>>> from beam_integrals.beam_types import BaseBeamType
>>> from beam_integrals.integrals import BaseIntegral

>>> REPORT_SOURCE = 'http://ci.petarmaric.com/job/beam_integrals/ws/nosetests.xml'
>>> TEST_INTEGRATE_XPATH = etree.XPath(
...     '/testsuite/testcase['
...     '   @classname="tests.integrals.test_integration" '
...     '   and starts-with(@name, "test_integrate(")'
...     ']'
... )


>>> def get_average_integration_time(report_source):
...     """Returned time is a floating point measured in seconds"""
...     document = etree.parse(report_source)
...     results = TEST_INTEGRATE_XPATH(document)
...     total_time = sum(float(testcase.get('time')) for testcase in results)
...     return total_time/len(results)

>>> def estimate_time_required_for_exporting_integrals(report_source):
...     """Returned time is a floating point measured in days"""
...     num_beam_types = len(BaseBeamType.plugins.valid_ids)
...
...     canonical_integrals = (
...         cls
...         for cls in BaseIntegral.plugins.classes if not cls.has_parent()
...     )
...     num_integral_table_items = sum(
...         DEFAULT_MAX_MODE**len(cls.used_variables)
...         for cls in canonical_integrals
...     )
...
...     return num_beam_types * num_integral_table_items * \
...            get_average_integration_time(report_source) / 60 / 60 / 24
```

```
>>> print "%.2f\tAverage integration time (seconds)\n" \
...       "%.2f\tEstimated time required for exporting integrals (days)" % (
...           get_average_integration_time(REPORT_SOURCE),
...           estimate_time_required_for_exporting_integrals(REPORT_SOURCE)
...       )
10.98     Average integration time (seconds)
38.13     Estimated time required for exporting integrals (days)
```

Listing 3.6: Procena vremena potrebnog za sekvencijalno kreiranje tabela integrala, zasnovano na instrumentaciji podsistema za automatsku verifikaciju projekta `beam_integrals` (opisanog u sekciji 4.3.7)

Na osnovu ispisa sa listinga 3.6 može se videti da je procenjeno vreme potrebno za sekvencijalno kreiranje tabela integrala $\approx$ *38.13 dana*.

Međutim, navedena analiza razmatra samo 13 integrala koji su potrebni za rešavanje problema stabilnosti u HCFSM (datih u jednačinama 1.35–1.47). Vreme potrebno za sekvencijalno kreiranje tabela integrala bi se značajno povećalo ako bi se u analizu uključilo svih 113 integrala koje definiše HCFSM.

Projekat `beam_integrals` trenutno ne obuhvata svih 113 integrala koje definiše HCFSM, već samo 13 spomenutih integrala potrebnih za rešavanje problema stabilnosti. Stoga se procena vremena potrebnog za sekvencijalno kreiranje tabela *svih* integrala ne može izvršiti samo analizom datom u listingu 3.6, već je tom vremenu potrebno dodati uticaj sledećih faktora:

1. za oko red veličine jer projekat `beam_integrals` trenutno ne obuhvata svih 113 integrala, već samo 13 integrala potrebnih za rešavanje problema stabilnosti

2. za bar red veličine jer su svih 13 razmatranih integrala „jednostavniji" integrali koji zavise od svega 2 integracione promenljive. Veći broj integrala koje definiše HCFSM zavisi od sve 4 integracione promenljive te je za njih vremenska kompleksnost veća za $\approx$ *4 reda veličine* (na osnovu razmatranja u sekciji 3.3.3)

Procena naše istraživačka grupe je da bi kombinovani doprinos navedenih faktora doveo do uvećanja od $\approx$ *50 puta*, tj. da bi

procenjeno vreme potrebno za sekvencijalno kreiranje tabela *svih* integrala bilo:

$$\begin{aligned} T_{\text{sequential}} &\approx 38.13 \text{ dana} \times 50 = 1907 \text{ dana} \\ &\approx \mathbf{5 \text{ godina}} \end{aligned} \tag{3.32}$$

Naravno, ovako dugo vreme potrebno za kreiranje tabela integrala smo smatrali neprihvatljivim, pa je odlučeno da treba osmisliti načine da se vreme proračuna značajno smanji. Razmatrani su pristupi koji mogu dovesti do dodatnog smanjenja vremenske kompleksnosti proračuna, međutim većina optimizacija je već bila iscrpljena (sekcije 3.3.1–3.3.3). Takođe je uzeta u obzir i mogućnost smanjenja radne preciznosti proračuna što bi značajno smanjilo potrebu za resursima, ali je podsistem za automatsku verifikaciju projekta `beam_integrals` pokazao da se daljim smanjenjem radne preciznosti značajno ugrožava tačnost rezultata (što je već razmatrano u poglavlju 2, a može se i vizuelno potvrditi na slikama 2.6–2.10, 2.32, 2.34 i 2.40).

Stoga je odlučeno da se u rešavanju ovog problema mora pristupiti kroz paralelizaciju proračuna, tj. da čisto sekvencijalni kôd ne može kreirati tabele integrala u razumnom vremenu, pod zadatim kriterijumima. Srećom postupak kreiranja tabela integrala se može veoma dobro horizontalno skalirati [142–145], pošto se proračuni integrala mogu izvesti tako da budu međusobno nezavisni (tzv. *embarrassingly parallel problem* [146–148]).

Amdalov zakon [149,150] nalaže da je teoretsko ubrzanje koje se može dobiti izvršavanjem algoritma na sistemu sa n niti izvršavanja:

$$S_A(n) = \frac{T_A(1)}{T_A(n)} = \frac{1}{(1-f) + \frac{f}{n}} \tag{3.33}$$

gde je:

- n broj niti izvršavanja
- $0 \leq f \leq 1$ deo algoritma koji se može idealno paralelizovati

Na osnovu Amdalovog zakona se, iz jednačine 3.33, može aproksimirati vreme potrebno za izvršavanje nekog algoritma na n niti:

$$T_A(n) = T_A(1)\left((1-f) + \frac{f}{n}\right) \tag{3.34}$$

U jednačini 3.32 data je procena vremena potrebnog za sekvencijalno kreiranje tabela svih integrala. Ona nije uključivala procenu vremena T_{overhead} potrebnog za skladištenje tabela integrala, pripremu za izračunavanje određenih integrala i ostale prateće proračune. Pretpostavljeno je da T_{overhead} ima zanemarljiv uticaj na ukupno vreme potrebno za kreiranje tabela integrala pošto je:

$$T_{\text{overhead}} \ll T_{\text{sequential}} \tag{3.35}$$

Naša istraživačka grupa je odlučila da preceni ovo vreme na:

$$T_{\text{overhead}} \approx 2 \text{ dana} \tag{3.36}$$

Međutim T_{overhead} se, zbog formulacije Amdalovog zakona, ne sme zanemarivati ako se proces kreiranja tabela integrala paralelizuje, jer suštinski predstavlja serijski deo algoritma. Sâm uticaj T_{overhead} na $T_A(1)$ je relativno blag:

$$\begin{aligned} T_A(1) &= T_{\text{sequential}} + T_{\text{overhead}} = 1907 \text{ dana} + 2 \text{ dana} \\ &= 1909 \text{ dana} \end{aligned} \tag{3.37}$$

dok ima značajan uticaj na faktor f:

$$\begin{aligned} f &= 1 - \frac{T_{\text{overhead}}}{T_A(1)} = 1 - \frac{2 \text{ dana}}{1909 \text{ dana}} \\ &= \frac{1907}{1909} \\ &\approx 0.99895 \\ &\approx 99.895\% \end{aligned} \tag{3.38}$$

koji ograničava stepen teoretski maksimalnog ubrzanja [150]:

$$\begin{aligned} \lim_{n\to\infty} S_A(n) &= \lim_{n\to\infty} \frac{T_A(1)}{T_A(n)} = \lim_{n\to\infty} \frac{1}{(1-f) + \frac{f}{n}} \\ &= \frac{1}{1-f} \\ &= \frac{1}{1 - \frac{1907}{1909}} = \frac{1}{\frac{2}{1909}} = \frac{1909}{2} \\ &= 954.5 \end{aligned} \tag{3.39}$$

Uvrštavanjem vrednosti $T_A(1)$ iz jednačine 3.37 i vrednosti f iz jednačine 3.38 u jednačinu 3.34 dobija se matematički model za procenu vremena potrebnog za kreiranje tabela svih integrala, koji zavisi samo od broja niti izvršavanja n:

$$\begin{aligned} T_A(n) &= T_A(1)\left((1-\frac{1907}{1909})+\frac{\frac{1907}{1909}}{n}\right) \\ &= 1909 \text{ dana} \times \left(\frac{2}{1909}+\frac{1907}{1909\times n}\right) \\ &= 1909 \text{ dana} \times \frac{2\times n+1907}{1909\times n} \\ &= \frac{2\times n+1907}{n} \text{ dana} \\ &= 2+\frac{1907}{n} \text{ dana} \end{aligned} \tag{3.40}$$

Naš integracioni server poseduje 4 procesorska jezgra [133]. Ako se matematički model opisan jednačinom 3.40 primeni na naš integracioni server kroz lokalne paralelizme, tada će procenjeno vreme potrebno za kreiranje tabela svih integrala biti:

$$\begin{aligned} T_{\text{parallel}}|^{\text{cores}=4} &\approx T_A(4) \\ &= 2+\frac{1907}{4} \text{ dana} \\ &\approx \mathbf{479 \text{ dana}} \end{aligned} \tag{3.41}$$

Amdalov zakon pretpostavlja da se deo algoritma f može idealno paralelizovati, što nažalost u praksi nije slučaj [151]. Ako se uzme procena da će praktični nedostaci Amdalovog zakona [151] doprineti sa 15% tada će korigovano vreme biti:

$$\begin{aligned} \tilde{T}_{\text{parallel}}|^{\text{cores}=4} &= 1.15 \times T_{\text{parallel}}|^{\text{cores}=4} \\ &= 1.15 \times 479 \text{ dana} \\ &\approx \mathbf{551 \text{ dana}} \end{aligned} \tag{3.42}$$

Međutim, ovako dugo vreme smo takođe smatrali neprihvatljivim. Pošto ni sekvencijalni kôd niti lokalni paralelizam ne rešavaju problem moralo se pribeći distribuiranim proračunima.

Putem projekta Ministarstva prosvete, nauke i tehnološkog razvoja naša istraživačka grupa je došla u posed omanjeg računarskog klastera za naučne proračune. Klaster za proračune čine 1 *master* server zadužen za upravljanje proračunima uz 32 *worker* računara koji vrše proračune. Svi računari u klasteru imaju istovetne osnovne tehničke karakteristike:

- HP Z220 CMT Workstation [152, 153]
- Intel® Core™ i5-3470 Quad-Core CPU @ 3.20GHz [133]
- Micron 32 GB RAM DDR3 1600 MHz
- Western Digital WD10EZEX-60Z 1 TB SATA 3.0 HDD [134]
- NVIDIA® Quadro® 600 1 GB DDR3 [154]
- Intel® 82579LM Gigabit Network Connection [136]
- Intel® 82574L Gigabit Network Connection [155]

Proračuni potrebni za kreiranje tabela integrala prvenstveno zavise od performansi centralnog procesora. Stoga se, usled istovetnog procesora u oba slučaja [133], može smatrati da su performanse našeg integracionog servera slične računarima u klasteru, sa stanovišta problema kreiranja tabela integrala. Bez obzira na to, uporedno je ispitano vreme potrebno da se izvrši podsistem za automatsku verifikaciju projekta `beam_integrals` i nakon višestrukih testova je potvrđeno da se vremena izvršavanja zanemarljivo razlikuju ($< 2\%$).

Svi računari u klasteru poseduju 4 procesorska jezgra [133]. Ako se matematički model opisan jednačinom 3.40 primeni na distribuirani sistem za izračunavanje određenih integrala, koji se izvršava na našem klasteru za proračune, tada će procenjeno vreme potrebno za kreiranje tabela svih integrala biti:

$$\begin{aligned} T_{\text{distributed}}\big|_{\text{nodes}=32}^{\text{cores}=4} &\approx T_A(4 \times 32) \\ &= 2 + \frac{1907}{4 \times 32} \text{ dana} \\ &\approx \mathbf{17 \text{ dana}} \end{aligned} \tag{3.43}$$

Međutim, kao i u slučaju lokalnog paralelizma, Amdalov zakon pretpostavlja da se deo algoritma f može idealno paralelizovati, što nažalost nije slučaj. Kod distribuiranih sistema treba uzeti u obzir i dodatne faktore koji mogu dovesti do usporavanja rada: sistem za instrumentaciju i nadgledanje utrošenih resursa, sistem za praćenje događaja, vreme izgubljeno u mrežnoj komunikaciji, vreme potrebno klasteru da se pripremi za rad, podsistem za bezbednosno skladištenje međurezultata u slučaju otkaza ili prekida, ... Ako se uzme procena da će kombinovani uticaj navedenih faktora doprineti sa 20% tada će korigovano vreme biti:

$$\begin{aligned} \tilde{T}_{\text{distributed}}\big|_{\text{nodes=32}}^{\text{cores=4}} &= 1.20 \times T_{\text{distributed}}\big|_{\text{nodes=32}}^{\text{cores=4}} \\ &= 1.20 \times 17 \text{ dana} \\ &\approx \mathbf{20 \text{ dana}} \end{aligned} \tag{3.44}$$

što predstavlja znatno poboljšanje u odnosu na originalnu procenu datu u jednačini 3.32.

3.4 Problem normalizacije tabela integrala

Na osnovu zadatih ulaznih parametara program FSMNE i njegovi derivati [4, 9, 15, 17] za potrebe svojih proračuna koriste jednu od postojećih tabela integrala. Na osnovu zadatog graničnog uslova i dužine grede (a) FSMNE bira koju će od tabela integrala koristiti.

Takođe, u zavisnosti od tipa zadatog problema i pristupa u njegovom rešavanju [4] (npr. von Karman ili Green–Lagrange) FSMNE bira koje će integrale iz tabela koristiti. Prilikom kreiranja tabela integrala u opštem slučaju nije unapred poznato koji će integrali biti potrebni, te se stoga tabele moraju kreirati za sve podržane integrale (da bi se izbegla kasnija duplikacija posla).

Na osnovu opšteg oblika integrala iz jednačine 1.34 može se videti da, kod svih 113 integrala koje definiše HCFSM, vrednost određenog integrala zavisi i od dužine grede (a). FSMNE mora da tokom svog rada koristi odgovarajuću tabelu integrala, kod koje

važi:

$$a_{\text{FSMNE}} = a_{\text{tabela\_integrala}} \tag{3.45}$$

pošto će se u suprotnom dobiti netačni rezultati, usled pogrešne vrednosti određenog integrala.

Kreiranje tabela integrala koje obuhvataju sve moguće dužine grede je veoma zahtevan proces po pitanju utroška računarskih resursa i vremena proračuna, jer bi celokupni postupak kreiranja tabela integrala morao nezavisno da se ponovi za svaku traženu dužinu grede. Uz to bi distribucija, odabir i upotreba odgovarajuće tabele integrala tada predstavljala ozbiljan inženjerski i organizacioni poduhvat.

Stoga je neophodno osmisliti i realizovati metod koji će omogućiti `FSMNE` programu (a i svim drugim programima koji zavise od HCFSM integrala) da koristi unapred kreiranu tabelu integrala i u slučajevima kada iskaz iz jednačine 3.45 ne važi.

Ako se za sve kombinacije integrala i graničnih uslova kreiraju jedinične (normalizovane) tabele integrala, gde je:

$$a_{\text{tabela\_integrala}} = 1.0 \tag{3.46}$$

pretpostavimo da je moguće da se za svaku pojedinačnu kombinaciju integrala i graničnog uslova izvede zasebna matematička funkcija skaliranja $z(a)$:

$$z(a) = a^{\text{scale\_factor}} \tag{3.47}$$

putem koje se može dobiti tačna vrednost datog određenog integrala za svaku traženu vrednost parametra a, a na osnovu njegove normalizovane vrednosti iz jedinične tabele integrala:

$$\begin{aligned} I_\alpha(a) &= z(a) \times I_\alpha(1.0) \\ &= z(a) \times \int_0^1 F\Big(Y_m(y),\ Y_n(y),\ Y_t(y),\ Y_v(y)\Big)dy \end{aligned} \tag{3.48}$$

Projekat `beam_integrals`, zahvaljujući SymPy biblioteci, za potrebe svojih proračuna čuva i koristi analitičku formulaciju svih integrala i graničnih uslova u orginalnom, simboličkom obliku. Stoga je moguće simbolički odrediti funkciju skaliranja $z(a)$, koristeći pristup predočen u narednom listingu:

```
>>> from sympy import Float, factor
>>> from beam_integrals import a, y, mu_m
>>> from beam_integrals.beam_types import BaseBeamType
>>> from beam_integrals.integrals import BaseIntegral

>>> def estimate_scale_function(integral, beam_type, m, t, v, n):
...     Y_m = beam_type.Y_m
...     dY_m = lambda mode: beam_type.Y_m_derivative_from_cache(mode, order=1)
...     ddY_m = lambda mode: beam_type.Y_m_derivative_from_cache(mode, order=2)
...
...     integrand = integral._integrand(Y_m, dY_m, ddY_m, m, t, v, n)
...     factorized = factor(integrand)
...
...     scale_by = Float(1) # If nothing is found
...     for arg in factorized.args:
...         atoms = arg.atoms()
...         if a in atoms:
...             # Skip atoms with anything besides 'a'
...             if y in atoms or mu_m in atoms:
...                 continue
...             else:
...                 scale_by = arg
...
...     return scale_by*a

>>> def estimate_scale_factor(integral, beam_type, m, t, v, n):
...     return estimate_scale_function(
...         integral, beam_type, m, t, v, n
...     ).as_powers_dict()[a]

>>> for beam_type in BaseBeamType.plugins.instances_sorted_by_id:
...     print "%s:" % beam_type
...     for integral in BaseIntegral.plugins.instances_sorted_by_id:
...         print "\t%s:\t%2d\t%s" % (
...             integral,
...             estimate_scale_factor(integral, beam_type, 1, 1, 1, 1),
...             estimate_scale_function(integral, beam_type, 1, 1, 1, 1),
...         )
Simply Supported Beam (id=1):
    I1:      1      a
    I2:     -1      1/a
    I3:     -1      1/a
    I4:     -1      1/a
    I5:     -1      1/a
    I6:     -1      1/a
    I7:     -3      a**(-3)
    I8:     -1      1/a
    I21:     1      a
    I22:    -1      1/a
    I23:    -1      1/a
    I24:    -3      a**(-3)
    I25:    -1      1/a
Clamped Clamped Beam (id=2):
    I1:      1      a
    I2:     -1      1/a
```

```
    I3:     -1      1/a
    I4:     -1      1/a
    I5:     -1      1/a
    I6:     -1      1/a
    I7:     -3      a**(-3)
    I8:     -1      1/a
    I21:     1      a
    I22:    -1      1/a
    I23:    -1      1/a
    I24:    -3      a**(-3)
    I25:    -1      1/a
Clamped Free Beam (id=3):
    I1:      1      a
    I2:     -1      1/a
    I3:     -1      1/a
    I4:     -1      1/a
    I5:     -1      1/a
    I6:     -1      1/a
    I7:     -3      a**(-3)
    I8:     -1      1/a
    I21:     1      a
    I22:    -1      1/a
    I23:    -1      1/a
    I24:    -3      a**(-3)
    I25:    -1      1/a
Clamped Simply Supported Beam (id=4):
    I1:      1      a
    I2:     -1      1/a
    I3:     -1      1/a
    I4:     -1      1/a
    I5:     -1      1/a
    I6:     -1      1/a
    I7:     -3      a**(-3)
    I8:     -1      1/a
    I21:     1      a
    I22:    -1      1/a
    I23:    -1      1/a
    I24:    -3      a**(-3)
    I25:    -1      1/a
Simply Supported Free Beam (id=5):
    I1:     -1      1/a
    I2:      2      a**2
    I3:      1      a
    I4:      2      a**2
    I5:      1      a
    I6:      2      a**2
    I7:      1      a
    I8:      2      a**2
    I21:    -1      1/a
    I22:     1      a
    I23:     1      a
    I24:     1      a
    I25:     2      a**2
Free Free Beam (id=6):
    I1:      1      a
    I2:      1      a
```

```
I3:     1     a
I4:     1     a
I5:     1     a
I6:     1     a
I7:     1     a
I8:     1     a
I21:    1     a
I22:    1     a
I23:    1     a
I24:    1     a
I25:    1     a
```

Listing 3.7: Demonstracija algoritma za simboličko određivanje funkcija i faktora skaliranja, kada su modovi $m = n = t = v = 1$

Na osnovu ispisa sa listinga 3.7 formirane su tabele 3.3–3.8 koje sadrže pregled simbolički određenih funkcija i faktora skaliranja.

Pretpostavka je da se dobijene funkcije i faktori skaliranja mogu primeniti i u slučajevima kada ne važi navedena zakonitost iz listinga 3.7:

$$m = n = t = v = 1 \tag{3.49}$$

usled periodičnosti i oblika funkcija $Y_m(y)$, koje se takođe mogu skalirati jer njihova vrednost zavisi od odnosa y/a – pošto se prilikom određivanja integrala taj odnos kreće u opsegu $0 \leq y/a \leq 1$, bez obzira na vrednost promenljive a.

Na osnovu funkcija skaliranja $z(a)$ datih u tabelama 3.3–3.8 formirane su tabele 3.11–3.34 (str. 105–116) koje verifikuju gorenavedenu pretpostavku za:

- sve granične uslove date u sekciji 1.2.1
- svih 13 integrala koji su potrebni za rešavanje problema stabilnosti u HCFSM (datih u jednačinama 1.35–1.47)
- različite dužine grede, $a \in \{1.7, 2.0, 4.2, 10.0\}$
- $m = 99$
- $n = 97$

Analizom sadržaja tabela 3.11–3.34 može se potvrditi generalna korektnost funkcija skaliranja, ali i utvrditi da postoje slučajevi kada pretpostavka data u jednačini 3.48 ne važi, tj:

$$z(a) \times I_\alpha(1.0) \neq I_\alpha(a) \tag{3.50}$$

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	-1	a^{-1}
I_3	-1	a^{-1}
I_4	-1	a^{-1}
I_5	-1	a^{-1}
I_6	-1	a^{-1}
I_7	-3	a^{-3}
I_8	-1	a^{-1}
I_{21}	1	a
I_{22}	-1	a^{-1}
I_{23}	-1	a^{-1}
I_{24}	-3	a^{-3}
I_{25}	-1	a^{-1}

Tabela 3.3: Pregled dobijenih rezultata sa listinga 3.7, za granični uslov gde su oba kraja slobodno oslonjena (LJ=1)

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	-1	a^{-1}
I_3	-1	a^{-1}
I_4	-1	a^{-1}
I_5	-1	a^{-1}
I_6	-1	a^{-1}
I_7	-3	a^{-3}
I_8	-1	a^{-1}
I_{21}	1	a
I_{22}	-1	a^{-1}
I_{23}	-1	a^{-1}
I_{24}	-3	a^{-3}
I_{25}	-1	a^{-1}

Tabela 3.4: Pregled dobijenih rezultata sa listinga 3.7, za granični uslov gde su oba kraja uklještena (LJ=2)

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	-1	a^{-1}
I_3	-1	a^{-1}
I_4	-1	a^{-1}
I_5	-1	a^{-1}
I_6	-1	a^{-1}
I_7	-3	a^{-3}
I_8	-1	a^{-1}
I_{21}	1	a
I_{22}	-1	a^{-1}
I_{23}	-1	a^{-1}
I_{24}	-3	a^{-3}
I_{25}	-1	a^{-1}

Tabela 3.5: Pregled dobijenih rezultata sa listinga 3.7, za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3)

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	-1	a^{-1}
I_3	-1	a^{-1}
I_4	-1	a^{-1}
I_5	-1	a^{-1}
I_6	-1	a^{-1}
I_7	-3	a^{-3}
I_8	-1	a^{-1}
I_{21}	1	a
I_{22}	-1	a^{-1}
I_{23}	-1	a^{-1}
I_{24}	-3	a^{-3}
I_{25}	-1	a^{-1}

Tabela 3.6: Pregled dobijenih rezultata sa listinga 3.7, za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4)

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	-1	a^{-1}
I_2	2	a^2
I_3	1	a
I_4	2	a^2
I_5	1	a
I_6	2	a^2
I_7	1	a
I_8	2	a^2
I_{21}	-1	a^{-1}
I_{22}	1	a
I_{23}	1	a
I_{24}	1	a
I_{25}	2	a^2

Tabela 3.7: Pregled dobijenih rezultata sa listinga 3.7, za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5)

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	1	a
I_3	1	a
I_4	1	a
I_5	1	a
I_6	1	a
I_7	1	a
I_8	1	a
I_{21}	1	a
I_{22}	1	a
I_{23}	1	a
I_{24}	1	a
I_{25}	1	a

Tabela 3.8: Pregled dobijenih rezultata sa listinga 3.7, za granični uslov gde su oba kraja slobodna (LJ=6)

Bližim uvidom u slučajeve gde se iskazane vrednosti ne poklapaju može se primetiti da su tada vrednosti određenih integrala uglavnom bliske nuli:

$$\begin{aligned} I_\alpha(1.0) &\approx 0 \\ I_\alpha(a) &\approx 0 \end{aligned} \tag{3.51}$$

Stoga se, sa praktične strane stanovišta, u slučajevima kada su vrednosti integrala izrazito male:

$$|I_\alpha(x)| \leq 10^{-9} \tag{3.52}$$

može reći da takva vrednost i jeste nula:

$$I_\alpha(x) = 0, \quad \text{ako je } |I_\alpha(1.0)| \leq 10^{-9} \tag{3.53}$$

Upotrebom opisanog postupka anuliranja malih vrednosti integrala (datog u jednačini 3.53) formirane su tabele 3.35–3.58 (str. 117–128), koristeći iste postavke kao i za tabele 3.11–3.34.

Uporednom analizom sadržaja tabela 3.35–3.58 sa sadržajem tabela 3.11–3.34 mogu se primetiti značajna poboljšanja u stepenu poklapanja vrednosti, ostvarena upotrebom postupka anuliranja.

Štaviše, za granične uslove LJ $\in \{1, 2, 3, 4\}$ nema vrednosti integrala koje se ne poklapaju, što se može videti uporednom analizom tabela 3.35–3.50. Međutim kada se analiziraju vrednosti vezane za granične uslove LJ $\in \{5, 6\}$, tj. tabele 3.51–3.58 očevidno je da se dobijene vrednosti generalno ne slažu.

Dobijena razlika potiče od, uslovno rečeno, pogrešno odabranih funkcija skaliranja. Naime, za granične uslove LJ $\in \{5, 6\}$ nije moguće odabrati univerzalne funkcije skaliranja, kao što se može učiniti za druge granične uslove. Taj problem nastaje usled složenosti njihovih jednačina svojstvenih oblika, čiji oblici nisu konstantni već zavise od traženog moda – videti jednačine 1.24, 1.29 i uporediti ih sa jednačinama 1.6, 1.9, 1.14, 1.19. Samim tim i funkcije skaliranja za granične uslove LJ $\in \{5, 6\}$ zavise od traženog moda, tj. postaju *specifične* funkcije skaliranja koje se moraju zasebno određivati za svaku kombinaciju integracionih promenljivih (modovi m, n, t i v).

Pretpostavka je da se uvažavanjem navedene zavisnosti ipak mogu odrediti odgovarajuće, *specifične*, funkcije skaliranja i za granične uslove $\mathrm{LJ} \in \{5, 6\}$.

Na listingu 3.8 se, na bazi Python kôda datog u listingu 3.7, simbolički određuju *specifične* funkcije skaliranja $z(a)$ za granične uslove $\mathrm{LJ} \in \{5, 6\}$, kada su modovi $m = 99$ i $n = 97$:

```
>>> from sympy import Float, factor
>>> from beam_integrals import a, y, mu_m
>>> from beam_integrals.beam_types import BaseBeamType
>>> from beam_integrals.integrals import BaseIntegral

>>> def estimate_scale_function(integral, beam_type, m, t, v, n):
...     Y_m = beam_type.Y_m
...     dY_m = lambda mode: beam_type.Y_m_derivative_from_cache(mode, order=1)
...     ddY_m = lambda mode: beam_type.Y_m_derivative_from_cache(mode, order=2)
...
...     integrand = integral._integrand(Y_m, dY_m, ddY_m, m, t, v, n)
...     factorized = factor(integrand)
...
...     scale_by = Float(1) # If nothing is found
...     for arg in factorized.args:
...         atoms = arg.atoms()
...         if a in atoms:
...             # Skip atoms with anything besides 'a'
...             if y in atoms or mu_m in atoms:
...                 continue
...             else:
...                 scale_by = arg
...
...     return scale_by*a

>>> def estimate_scale_factor(integral, beam_type, m, t, v, n):
...     return estimate_scale_function(
...         integral, beam_type, m, t, v, n
...     ).as_powers_dict()[a]

>>> for beam_type_id in (5, 6):
...     beam_type = BaseBeamType.coerce(beam_type_id)
...     print "%s:" % beam_type
...     for integral in BaseIntegral.plugins.instances_sorted_by_id:
...         print "\t%s:\t%2d\t%s" % (
...             integral,
...             estimate_scale_factor(integral, beam_type, 99, None, None, 97),
...             estimate_scale_function(integral, beam_type, 99, None, None, 97),
...         )
Simply Supported Free Beam (id=5):
    I1:      1      a
    I2:     -1      1/a
    I3:     -1      1/a
    I4:     -1      1/a
    I5:     -1      1/a
    I6:     -1      1/a
    I7:     -3      a**(-3)
```

```
    I8:     -1      1/a
    I21:     1      a
    I22:    -1      1/a
    I23:    -1      1/a
    I24:    -3      a**(-3)
    I25:    -1      1/a
Free Free Beam (id=6):
    I1:      1      a
    I2:     -1      1/a
    I3:     -1      1/a
    I4:     -1      1/a
    I5:     -1      1/a
    I6:     -1      1/a
    I7:     -3      a**(-3)
    I8:     -1      1/a
    I21:     1      a
    I22:    -1      1/a
    I23:    -1      1/a
    I24:    -3      a**(-3)
    I25:    -1      1/a
```

Listing 3.8: Demonstracija algoritma za simboličko određivanje *specifičnih* funkcija i faktora skaliranja za granične uslove LJ $\in \{5, 6\}$, kada su modovi $m = 99$ i $n = 97$

Na osnovu ispisa sa listinga 3.8 formirane su tabele 3.9–3.10 koje sadrže pregled simbolički određenih *specifičnih* funkcija i faktora skaliranja.

Na osnovu *specifičnih* funkcija skaliranja $z(a)$ datih u tabelama 3.9–3.10 formirane su tabele 3.59–3.66 (str. 129–132) koje verifikuju gorenavedenu pretpostavku za:

- granične uslove LJ $\in \{5, 6\}$
- svih 13 integrala koji su potrebni za rešavanje problema stabilnosti u HCFSM (datih u jednačinama 1.35–1.47)
- različite dužine grede, $a \in \{1.7, 2.0, 4.2, 10.0\}$
- $m = 99$
- $n = 97$

Uporednom analizom sadržaja tabela 3.59–3.66 sa sadržajem tabela 3.51–3.58 mogu se primetiti značajna poboljšanja u stepenu poklapanja vrednosti, ostvarena upotrebom *specifičnih* funkcija skaliranja.

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	-1	a^{-1}
I_3	-1	a^{-1}
I_4	-1	a^{-1}
I_5	-1	a^{-1}
I_6	-1	a^{-1}
I_7	-3	a^{-3}
I_8	-1	a^{-1}
I_{21}	1	a
I_{22}	-1	a^{-1}
I_{23}	-1	a^{-1}
I_{24}	-3	a^{-3}
I_{25}	-1	a^{-1}

Tabela 3.9: Pregled dobijenih rezultata sa listinga 3.8, za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5), kada su modovi $m = 99$ i $n = 97$

Integral	Faktor skaliranja	Funkcija skaliranja $z(a)$
I_1	1	a
I_2	-1	a^{-1}
I_3	-1	a^{-1}
I_4	-1	a^{-1}
I_5	-1	a^{-1}
I_6	-1	a^{-1}
I_7	-3	a^{-3}
I_8	-1	a^{-1}
I_{21}	1	a
I_{22}	-1	a^{-1}
I_{23}	-1	a^{-1}
I_{24}	-3	a^{-3}
I_{25}	-1	a^{-1}

Tabela 3.10: Pregled dobijenih rezultata sa listinga 3.8, za granični uslov gde su oba kraja slobodna (LJ=6), kada su modovi $m = 99$ i $n = 97$

Štaviše, detaljnim uvidom u tabele 3.59–3.66 može se primetiti da više uopšte i nema vrednosti integrala koje se ne poklapaju.

Na kraju, da bi se korisnicima omogućilo jednostavno korišćenje faktora skaliranja, treba izmeniti predloženu strukturu HDF5 tabela integrala (sekcija 3.2) dodavanjem kolone `scale_factor`.

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	4.845×10^{-156}	8.237×10^{-156}	1.991×10^{-47}
I_2	a^{-1}	5.067×10^{-151}	2.981×10^{-151}	4.005×10^{-13}
I_3	a^{-1}	-4.687×10^{-151}	-2.757×10^{-151}	-6.665×10^{-43}
I_4	a^{-1}	5.067×10^{-151}	2.981×10^{-151}	4.005×10^{-13}
I_5	a^{-1}	-4.499×10^{-151}	-2.647×10^{-151}	-6.398×10^{-43}
I_6	a^{-1}	5.067×10^{-151}	2.981×10^{-151}	4.005×10^{-13}
I_7	a^{-3}	4.352×10^{-146}	8.859×10^{-147}	2.141×10^{-38}
I_8	a^{-1}	5.067×10^{-151}	2.981×10^{-151}	4.005×10^{-13}
I_{21}	a	4.845×10^{-156}	8.237×10^{-156}	1.991×10^{-47}
I_{22}	a^{-1}	-4.687×10^{-151}	-2.757×10^{-151}	-6.665×10^{-43}
I_{23}	a^{-1}	-4.499×10^{-151}	-2.647×10^{-151}	-6.398×10^{-43}
I_{24}	a^{-3}	4.352×10^{-146}	8.859×10^{-147}	2.141×10^{-38}
I_{25}	a^{-1}	5.067×10^{-151}	2.981×10^{-151}	4.005×10^{-13}

Tabela 3.11: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 1.7$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	4.845×10^{-156}	9.690×10^{-156}	9.690×10^{-156}
I_2	a^{-1}	5.067×10^{-151}	2.534×10^{-151}	2.534×10^{-151}
I_3	a^{-1}	-4.687×10^{-151}	-2.343×10^{-151}	-2.343×10^{-151}
I_4	a^{-1}	5.067×10^{-151}	2.534×10^{-151}	2.534×10^{-151}
I_5	a^{-1}	-4.499×10^{-151}	-2.250×10^{-151}	-2.250×10^{-151}
I_6	a^{-1}	5.067×10^{-151}	2.534×10^{-151}	2.534×10^{-151}
I_7	a^{-3}	4.352×10^{-146}	5.440×10^{-147}	5.440×10^{-147}
I_8	a^{-1}	5.067×10^{-151}	2.534×10^{-151}	2.534×10^{-151}
I_{21}	a	4.845×10^{-156}	9.690×10^{-156}	9.690×10^{-156}
I_{22}	a^{-1}	-4.687×10^{-151}	-2.343×10^{-151}	-2.343×10^{-151}
I_{23}	a^{-1}	-4.499×10^{-151}	-2.250×10^{-151}	-2.250×10^{-151}
I_{24}	a^{-3}	4.352×10^{-146}	5.440×10^{-147}	5.440×10^{-147}
I_{25}	a^{-1}	5.067×10^{-151}	2.534×10^{-151}	2.534×10^{-151}

Tabela 3.12: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 2.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	4.845×10^{-156}	2.035×10^{-155}	-3.064×10^{-46}
I_2	a^{-1}	5.067×10^{-151}	1.206×10^{-151}	-2.983×10^{-13}
I_3	a^{-1}	-4.687×10^{-151}	-1.116×10^{-151}	1.680×10^{-42}
I_4	a^{-1}	5.067×10^{-151}	1.206×10^{-151}	-2.983×10^{-13}
I_5	a^{-1}	-4.499×10^{-151}	-1.071×10^{-151}	1.613×10^{-42}
I_6	a^{-1}	5.067×10^{-151}	1.206×10^{-151}	-2.983×10^{-13}
I_7	a^{-3}	4.352×10^{-146}	5.874×10^{-148}	-8.844×10^{-39}
I_8	a^{-1}	5.067×10^{-151}	1.206×10^{-151}	-2.983×10^{-13}
I_{21}	a	4.845×10^{-156}	2.035×10^{-155}	-3.064×10^{-46}
I_{22}	a^{-1}	-4.687×10^{-151}	-1.116×10^{-151}	1.680×10^{-42}
I_{23}	a^{-1}	-4.499×10^{-151}	-1.071×10^{-151}	1.613×10^{-42}
I_{24}	a^{-3}	4.352×10^{-146}	5.874×10^{-148}	-8.844×10^{-39}
I_{25}	a^{-1}	5.067×10^{-151}	1.206×10^{-151}	-2.983×10^{-13}

Tabela 3.13: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 4.2$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	4.845×10^{-156}	4.845×10^{-155}	5.404×10^{-44}
I_2	a^{-1}	5.067×10^{-151}	5.067×10^{-152}	5.261×10^{-13}
I_3	a^{-1}	-4.687×10^{-151}	-4.687×10^{-152}	-5.228×10^{-41}
I_4	a^{-1}	5.067×10^{-151}	5.067×10^{-152}	5.261×10^{-13}
I_5	a^{-1}	-4.499×10^{-151}	-4.499×10^{-152}	-5.018×10^{-41}
I_6	a^{-1}	5.067×10^{-151}	5.067×10^{-152}	5.261×10^{-13}
I_7	a^{-3}	4.352×10^{-146}	4.352×10^{-149}	4.854×10^{-38}
I_8	a^{-1}	5.067×10^{-151}	5.067×10^{-152}	5.261×10^{-13}
I_{21}	a	4.845×10^{-156}	4.845×10^{-155}	5.404×10^{-44}
I_{22}	a^{-1}	-4.687×10^{-151}	-4.687×10^{-152}	-5.228×10^{-41}
I_{23}	a^{-1}	-4.499×10^{-151}	-4.499×10^{-152}	-5.018×10^{-41}
I_{24}	a^{-3}	4.352×10^{-146}	4.352×10^{-149}	4.854×10^{-38}
I_{25}	a^{-1}	5.067×10^{-151}	5.067×10^{-152}	5.261×10^{-13}

Tabela 3.14: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 10.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	5.890×10^{-24}	1.001×10^{-23}	1.001×10^{-23}
I_2	a^{-1}	-6.187×10^{2}	-3.639×10^{2}	-3.639×10^{2}
I_3	a^{-1}	6.187×10^{2}	3.639×10^{2}	3.639×10^{2}
I_4	a^{-1}	-6.187×10^{2}	-3.639×10^{2}	-3.639×10^{2}
I_5	a^{-1}	6.187×10^{2}	3.639×10^{2}	3.639×10^{2}
I_6	a^{-1}	-6.187×10^{2}	-3.639×10^{2}	-3.639×10^{2}
I_7	a^{-3}	-4.088×10^{-14}	-8.321×10^{-15}	5.362×10^{-8}
I_8	a^{-1}	-6.187×10^{2}	-3.639×10^{2}	-3.639×10^{2}
I_{21}	a	5.890×10^{-24}	1.001×10^{-23}	1.001×10^{-23}
I_{22}	a^{-1}	6.187×10^{2}	3.639×10^{2}	3.639×10^{2}
I_{23}	a^{-1}	6.187×10^{2}	3.639×10^{2}	3.639×10^{2}
I_{24}	a^{-3}	-4.088×10^{-14}	-8.321×10^{-15}	5.362×10^{-8}
I_{25}	a^{-1}	-6.187×10^{2}	-3.639×10^{2}	-3.639×10^{2}

Tabela 3.15: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 1.7$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	5.890×10^{-24}	1.178×10^{-23}	1.178×10^{-23}
I_2	a^{-1}	-6.187×10^{2}	-3.094×10^{2}	-3.094×10^{2}
I_3	a^{-1}	6.187×10^{2}	3.094×10^{2}	3.094×10^{2}
I_4	a^{-1}	-6.187×10^{2}	-3.094×10^{2}	-3.094×10^{2}
I_5	a^{-1}	6.187×10^{2}	3.094×10^{2}	3.094×10^{2}
I_6	a^{-1}	-6.187×10^{2}	-3.094×10^{2}	-3.094×10^{2}
I_7	a^{-3}	-4.088×10^{-14}	-5.110×10^{-15}	-5.110×10^{-15}
I_8	a^{-1}	-6.187×10^{2}	-3.094×10^{2}	-3.094×10^{2}
I_{21}	a	5.890×10^{-24}	1.178×10^{-23}	1.178×10^{-23}
I_{22}	a^{-1}	6.187×10^{2}	3.094×10^{2}	3.094×10^{2}
I_{23}	a^{-1}	6.187×10^{2}	3.094×10^{2}	3.094×10^{2}
I_{24}	a^{-3}	-4.088×10^{-14}	-5.110×10^{-15}	-5.110×10^{-15}
I_{25}	a^{-1}	-6.187×10^{2}	-3.094×10^{2}	-3.094×10^{2}

Tabela 3.16: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 2.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	5.890×10^{-24}	2.474×10^{-23}	2.474×10^{-23}
I_2	a^{-1}	-6.187×10^{2}	-1.473×10^{2}	-1.473×10^{2}
I_3	a^{-1}	6.187×10^{2}	1.473×10^{2}	1.473×10^{2}
I_4	a^{-1}	-6.187×10^{2}	-1.473×10^{2}	-1.473×10^{2}
I_5	a^{-1}	6.187×10^{2}	1.473×10^{2}	1.473×10^{2}
I_6	a^{-1}	-6.187×10^{2}	-1.473×10^{2}	-1.473×10^{2}
I_7	a^{-3}	-4.088×10^{-14}	-5.518×10^{-16}	-6.542×10^{-9}
I_8	a^{-1}	-6.187×10^{2}	-1.473×10^{2}	-1.473×10^{2}
I_{21}	a	5.890×10^{-24}	2.474×10^{-23}	2.474×10^{-23}
I_{22}	a^{-1}	6.187×10^{2}	1.473×10^{2}	1.473×10^{2}
I_{23}	a^{-1}	6.187×10^{2}	1.473×10^{2}	1.473×10^{2}
I_{24}	a^{-3}	-4.088×10^{-14}	-5.518×10^{-16}	-6.542×10^{-9}
I_{25}	a^{-1}	-6.187×10^{2}	-1.473×10^{2}	-1.473×10^{2}

Tabela 3.17: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 4.2$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	5.890×10^{-24}	5.890×10^{-23}	5.890×10^{-23}
I_2	a^{-1}	-6.187×10^{2}	-6.187×10^{1}	-6.187×10^{1}
I_3	a^{-1}	6.187×10^{2}	6.187×10^{1}	6.187×10^{1}
I_4	a^{-1}	-6.187×10^{2}	-6.187×10^{1}	-6.187×10^{1}
I_5	a^{-1}	6.187×10^{2}	6.187×10^{1}	6.187×10^{1}
I_6	a^{-1}	-6.187×10^{2}	-6.187×10^{1}	-6.187×10^{1}
I_7	a^{-3}	-4.088×10^{-14}	-4.088×10^{-17}	2.036×10^{-9}
I_8	a^{-1}	-6.187×10^{2}	-6.187×10^{1}	-6.187×10^{1}
I_{21}	a	5.890×10^{-24}	5.890×10^{-23}	5.890×10^{-23}
I_{22}	a^{-1}	6.187×10^{2}	6.187×10^{1}	6.187×10^{1}
I_{23}	a^{-1}	6.187×10^{2}	6.187×10^{1}	6.187×10^{1}
I_{24}	a^{-3}	-4.088×10^{-14}	-4.088×10^{-17}	2.036×10^{-9}
I_{25}	a^{-1}	-6.187×10^{2}	-6.187×10^{1}	-6.187×10^{1}

Tabela 3.18: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 10.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	1.371×10^{-26}	*2.331×10^{-26}*	*4.885×10^{-17}*
I_2	a^{-1}	6.125×10^{2}	3.603×10^{2}	3.603×10^{2}
I_3	a^{-1}	6.252×10^{2}	3.678×10^{2}	3.678×10^{2}
I_4	a^{-1}	6.125×10^{2}	3.603×10^{2}	3.603×10^{2}
I_5	a^{-1}	6.001×10^{2}	3.530×10^{2}	3.530×10^{2}
I_6	a^{-1}	6.125×10^{2}	3.603×10^{2}	3.603×10^{2}
I_7	a^{-3}	4.011×10^{-16}	*8.165×10^{-17}*	*-8.451×10^{-17}*
I_8	a^{-1}	6.125×10^{2}	3.603×10^{2}	3.603×10^{2}
I_{21}	a	1.371×10^{-26}	*2.331×10^{-26}*	*4.885×10^{-17}*
I_{22}	a^{-1}	6.252×10^{2}	3.678×10^{2}	3.678×10^{2}
I_{23}	a^{-1}	6.001×10^{2}	3.530×10^{2}	3.530×10^{2}
I_{24}	a^{-3}	4.011×10^{-16}	*8.165×10^{-17}*	*-8.451×10^{-17}*
I_{25}	a^{-1}	6.125×10^{2}	3.603×10^{2}	3.603×10^{2}

Tabela 3.19: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 1.7$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	1.371×10^{-26}	2.742×10^{-26}	2.742×10^{-26}
I_2	a^{-1}	6.125×10^{2}	3.063×10^{2}	3.063×10^{2}
I_3	a^{-1}	6.252×10^{2}	3.126×10^{2}	3.126×10^{2}
I_4	a^{-1}	6.125×10^{2}	3.063×10^{2}	3.063×10^{2}
I_5	a^{-1}	6.001×10^{2}	3.001×10^{2}	3.001×10^{2}
I_6	a^{-1}	6.125×10^{2}	3.063×10^{2}	3.063×10^{2}
I_7	a^{-3}	4.011×10^{-16}	5.014×10^{-17}	5.014×10^{-17}
I_8	a^{-1}	6.125×10^{2}	3.063×10^{2}	3.063×10^{2}
I_{21}	a	1.371×10^{-26}	2.742×10^{-26}	2.742×10^{-26}
I_{22}	a^{-1}	6.252×10^{2}	3.126×10^{2}	3.126×10^{2}
I_{23}	a^{-1}	6.001×10^{2}	3.001×10^{2}	3.001×10^{2}
I_{24}	a^{-3}	4.011×10^{-16}	5.014×10^{-17}	5.014×10^{-17}
I_{25}	a^{-1}	6.125×10^{2}	3.063×10^{2}	3.063×10^{2}

Tabela 3.20: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 2.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	1.371×10^{-26}	*5.758×10^{-26}*	*-2.220×10^{-16}*
I_2	a^{-1}	6.125×10^{2}	1.458×10^{2}	1.458×10^{2}
I_3	a^{-1}	6.252×10^{2}	1.489×10^{2}	1.489×10^{2}
I_4	a^{-1}	6.125×10^{2}	1.458×10^{2}	1.458×10^{2}
I_5	a^{-1}	6.001×10^{2}	1.429×10^{2}	1.429×10^{2}
I_6	a^{-1}	6.125×10^{2}	1.458×10^{2}	1.458×10^{2}
I_7	a^{-3}	4.011×10^{-16}	*5.414×10^{-18}*	*8.128×10^{-18}*
I_8	a^{-1}	6.125×10^{2}	1.458×10^{2}	1.458×10^{2}
I_{21}	a	1.371×10^{-26}	*5.758×10^{-26}*	*-2.220×10^{-16}*
I_{22}	a^{-1}	6.252×10^{2}	1.489×10^{2}	1.489×10^{2}
I_{23}	a^{-1}	6.001×10^{2}	1.429×10^{2}	1.429×10^{2}
I_{24}	a^{-3}	4.011×10^{-16}	*5.414×10^{-18}*	*8.128×10^{-18}*
I_{25}	a^{-1}	6.125×10^{2}	1.458×10^{2}	1.458×10^{2}

Tabela 3.21: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 4.2$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	1.371×10^{-26}	*1.371×10^{-25}*	*2.220×10^{-15}*
I_2	a^{-1}	6.125×10^{2}	6.125×10^{1}	6.125×10^{1}
I_3	a^{-1}	6.252×10^{2}	6.252×10^{1}	6.252×10^{1}
I_4	a^{-1}	6.125×10^{2}	6.125×10^{1}	6.125×10^{1}
I_5	a^{-1}	6.001×10^{2}	6.001×10^{1}	6.001×10^{1}
I_6	a^{-1}	6.125×10^{2}	6.125×10^{1}	6.125×10^{1}
I_7	a^{-3}	4.011×10^{-16}	*4.011×10^{-19}*	*8.431×10^{-19}*
I_8	a^{-1}	6.125×10^{2}	6.125×10^{1}	6.125×10^{1}
I_{21}	a	1.371×10^{-26}	*1.371×10^{-25}*	*2.220×10^{-15}*
I_{22}	a^{-1}	6.252×10^{2}	6.252×10^{1}	6.252×10^{1}
I_{23}	a^{-1}	6.001×10^{2}	6.001×10^{1}	6.001×10^{1}
I_{24}	a^{-3}	4.011×10^{-16}	*4.011×10^{-19}*	*8.431×10^{-19}*
I_{25}	a^{-1}	6.125×10^{2}	6.125×10^{1}	6.125×10^{1}

Tabela 3.22: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 10.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-4.240×10^{-155}	*-7.208×10^{-155}*	*2.952×10^{-77}*
I_2	a^{-1}	-1.543×10^{2}	-9.075×10^{1}	-9.075×10^{1}
I_3	a^{-1}	1.543×10^{2}	9.075×10^{1}	9.075×10^{1}
I_4	a^{-1}	-1.543×10^{2}	-9.075×10^{1}	-9.075×10^{1}
I_5	a^{-1}	1.543×10^{2}	9.075×10^{1}	9.075×10^{1}
I_6	a^{-1}	-1.543×10^{2}	-9.075×10^{1}	-9.075×10^{1}
I_7	a^{-3}	-3.841×10^{-145}	*-7.818×10^{-146}*	*2.654×10^{-8}*
I_8	a^{-1}	-1.543×10^{2}	-9.075×10^{1}	-9.075×10^{1}
I_{21}	a	-4.240×10^{-155}	*-7.208×10^{-155}*	*2.952×10^{-77}*
I_{22}	a^{-1}	1.543×10^{2}	9.075×10^{1}	9.075×10^{1}
I_{23}	a^{-1}	1.543×10^{2}	9.075×10^{1}	9.075×10^{1}
I_{24}	a^{-3}	-3.841×10^{-145}	*-7.818×10^{-146}*	*2.654×10^{-8}*
I_{25}	a^{-1}	-1.543×10^{2}	-9.075×10^{1}	-9.075×10^{1}

Tabela 3.23: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 1.7$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-4.240×10^{-155}	-8.480×10^{-155}	-8.480×10^{-155}
I_2	a^{-1}	-1.543×10^{2}	-7.714×10^{1}	-7.714×10^{1}
I_3	a^{-1}	1.543×10^{2}	7.714×10^{1}	7.714×10^{1}
I_4	a^{-1}	-1.543×10^{2}	-7.714×10^{1}	-7.714×10^{1}
I_5	a^{-1}	1.543×10^{2}	7.714×10^{1}	7.714×10^{1}
I_6	a^{-1}	-1.543×10^{2}	-7.714×10^{1}	-7.714×10^{1}
I_7	a^{-3}	-3.841×10^{-145}	-4.801×10^{-146}	-4.801×10^{-146}
I_8	a^{-1}	-1.543×10^{2}	-7.714×10^{1}	-7.714×10^{1}
I_{21}	a	-4.240×10^{-155}	-8.480×10^{-155}	-8.480×10^{-155}
I_{22}	a^{-1}	1.543×10^{2}	7.714×10^{1}	7.714×10^{1}
I_{23}	a^{-1}	1.543×10^{2}	7.714×10^{1}	7.714×10^{1}
I_{24}	a^{-3}	-3.841×10^{-145}	-4.801×10^{-146}	-4.801×10^{-146}
I_{25}	a^{-1}	-1.543×10^{2}	-7.714×10^{1}	-7.714×10^{1}

Tabela 3.24: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 2.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-4.240×10^{-155}	*-1.781×10^{-154}*	*-1.537×10^{-75}*
I_2	a^{-1}	-1.543×10^{2}	-3.673×10^{1}	-3.673×10^{1}
I_3	a^{-1}	1.543×10^{2}	3.673×10^{1}	3.673×10^{1}
I_4	a^{-1}	-1.543×10^{2}	-3.673×10^{1}	-3.673×10^{1}
I_5	a^{-1}	1.543×10^{2}	3.673×10^{1}	3.673×10^{1}
I_6	a^{-1}	-1.543×10^{2}	-3.673×10^{1}	-3.673×10^{1}
I_7	a^{-3}	-3.841×10^{-145}	*-5.185×10^{-147}*	*-3.238×10^{-9}*
I_8	a^{-1}	-1.543×10^{2}	-3.673×10^{1}	-3.673×10^{1}
I_{21}	a	-4.240×10^{-155}	*-1.781×10^{-154}*	*-1.537×10^{-75}*
I_{22}	a^{-1}	1.543×10^{2}	3.673×10^{1}	3.673×10^{1}
I_{23}	a^{-1}	1.543×10^{2}	3.673×10^{1}	3.673×10^{1}
I_{24}	a^{-3}	-3.841×10^{-145}	*-5.185×10^{-147}*	*-3.238×10^{-9}*
I_{25}	a^{-1}	-1.543×10^{2}	-3.673×10^{1}	-3.673×10^{1}

Tabela 3.25: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 4.2$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-4.240×10^{-155}	*-4.240×10^{-154}*	*4.783×10^{-72}*
I_2	a^{-1}	-1.543×10^{2}	-1.543×10^{1}	-1.543×10^{1}
I_3	a^{-1}	1.543×10^{2}	1.543×10^{1}	1.543×10^{1}
I_4	a^{-1}	-1.543×10^{2}	-1.543×10^{1}	-1.543×10^{1}
I_5	a^{-1}	1.543×10^{2}	1.543×10^{1}	1.543×10^{1}
I_6	a^{-1}	-1.543×10^{2}	-1.543×10^{1}	-1.543×10^{1}
I_7	a^{-3}	-3.841×10^{-145}	*-3.841×10^{-148}*	*1.008×10^{-9}*
I_8	a^{-1}	-1.543×10^{2}	-1.543×10^{1}	-1.543×10^{1}
I_{21}	a	-4.240×10^{-155}	*-4.240×10^{-154}*	*4.783×10^{-72}*
I_{22}	a^{-1}	1.543×10^{2}	1.543×10^{1}	1.543×10^{1}
I_{23}	a^{-1}	1.543×10^{2}	1.543×10^{1}	1.543×10^{1}
I_{24}	a^{-3}	-3.841×10^{-145}	*-3.841×10^{-148}*	*1.008×10^{-9}*
I_{25}	a^{-1}	-1.543×10^{2}	-1.543×10^{1}	-1.543×10^{1}

Tabela 3.26: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 10.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	-4.249×10^{-155}	-2.499×10^{-155}	2.442×10^{-17}
I_2	a^2	4.582×10^{2}	1.324×10^{3}	2.695×10^{2}
I_3	a	1.591×10^{2}	2.705×10^{2}	9.360×10^{1}
I_4	a^2	4.582×10^{2}	1.324×10^{3}	2.695×10^{2}
I_5	a	1.466×10^{2}	2.491×10^{2}	8.621×10^{1}
I_6	a^2	4.582×10^{2}	1.324×10^{3}	2.695×10^{2}
I_7	a	-3.689×10^{-145}	-6.271×10^{-145}	2.955×10^{-68}
I_8	a^2	4.582×10^{2}	1.324×10^{3}	2.695×10^{2}
I_{21}	a^{-1}	-4.249×10^{-155}	-2.499×10^{-155}	2.442×10^{-17}
I_{22}	a	1.591×10^{2}	2.705×10^{2}	9.360×10^{1}
I_{23}	a	1.466×10^{2}	2.491×10^{2}	8.621×10^{1}
I_{24}	a	-3.689×10^{-145}	-6.271×10^{-145}	2.955×10^{-68}
I_{25}	a^2	4.582×10^{2}	1.324×10^{3}	2.695×10^{2}

Tabela 3.27: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 1.7$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	-4.249×10^{-155}	-2.124×10^{-155}	-8.497×10^{-155}
I_2	a^2	4.582×10^{2}	1.833×10^{3}	2.291×10^{2}
I_3	a	1.591×10^{2}	3.182×10^{2}	7.956×10^{1}
I_4	a^2	4.582×10^{2}	1.833×10^{3}	2.291×10^{2}
I_5	a	1.466×10^{2}	2.931×10^{2}	7.328×10^{1}
I_6	a^2	4.582×10^{2}	1.833×10^{3}	2.291×10^{2}
I_7	a	-3.689×10^{-145}	-7.378×10^{-145}	-4.611×10^{-146}
I_8	a^2	4.582×10^{2}	1.833×10^{3}	2.291×10^{2}
I_{21}	a^{-1}	-4.249×10^{-155}	-2.124×10^{-155}	-8.497×10^{-155}
I_{22}	a	1.591×10^{2}	3.182×10^{2}	7.956×10^{1}
I_{23}	a	1.466×10^{2}	2.931×10^{2}	7.328×10^{1}
I_{24}	a	-3.689×10^{-145}	-7.378×10^{-145}	-4.611×10^{-146}
I_{25}	a^2	4.582×10^{2}	1.833×10^{3}	2.291×10^{2}

Tabela 3.28: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 2.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	-4.249×10^{-155}	-1.012×10^{-155}	-1.110×10^{-16}
I_2	a^2	4.582×10^{2}	8.083×10^{3}	1.091×10^{2}
I_3	a	1.591×10^{2}	6.683×10^{2}	3.789×10^{1}
I_4	a^2	4.582×10^{2}	8.083×10^{3}	1.091×10^{2}
I_5	a	1.466×10^{2}	6.155×10^{2}	3.489×10^{1}
I_6	a^2	4.582×10^{2}	8.083×10^{3}	1.091×10^{2}
I_7	a	-3.689×10^{-145}	-1.549×10^{-144}	-4.131×10^{-68}
I_8	a^2	4.582×10^{2}	8.083×10^{3}	1.091×10^{2}
I_{21}	a^{-1}	-4.249×10^{-155}	-1.012×10^{-155}	-1.110×10^{-16}
I_{22}	a	1.591×10^{2}	6.683×10^{2}	3.789×10^{1}
I_{23}	a	1.466×10^{2}	6.155×10^{2}	3.489×10^{1}
I_{24}	a	-3.689×10^{-145}	-1.549×10^{-144}	-4.131×10^{-68}
I_{25}	a^2	4.582×10^{2}	8.083×10^{3}	1.091×10^{2}

Tabela 3.29: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 4.2$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	-4.249×10^{-155}	-4.249×10^{-156}	1.110×10^{-15}
I_2	a^2	4.582×10^{2}	4.582×10^{4}	4.582×10^{1}
I_3	a	1.591×10^{2}	1.591×10^{3}	1.591×10^{1}
I_4	a^2	4.582×10^{2}	4.582×10^{4}	4.582×10^{1}
I_5	a	1.466×10^{2}	1.466×10^{3}	1.466×10^{1}
I_6	a^2	4.582×10^{2}	4.582×10^{4}	4.582×10^{1}
I_7	a	-3.689×10^{-145}	-3.689×10^{-144}	4.000×10^{-66}
I_8	a^2	4.582×10^{2}	4.582×10^{4}	4.582×10^{1}
I_{21}	a^{-1}	-4.249×10^{-155}	-4.249×10^{-156}	1.110×10^{-15}
I_{22}	a	1.591×10^{2}	1.591×10^{3}	1.591×10^{1}
I_{23}	a	1.466×10^{2}	1.466×10^{3}	1.466×10^{1}
I_{24}	a	-3.689×10^{-145}	-3.689×10^{-144}	4.000×10^{-66}
I_{25}	a^2	4.582×10^{2}	4.582×10^{4}	4.582×10^{1}

Tabela 3.30: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 10.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-6.910×10^{-27}	-1.175×10^{-26}	4.885×10^{-17}
I_2	a	6.063×10^{2}	1.031×10^{3}	3.566×10^{2}
I_3	a	6.190×10^{2}	1.052×10^{3}	3.641×10^{2}
I_4	a	6.063×10^{2}	1.031×10^{3}	3.566×10^{2}
I_5	a	5.938×10^{2}	1.010×10^{3}	3.493×10^{2}
I_6	a	6.063×10^{2}	1.031×10^{3}	3.566×10^{2}
I_7	a	4.828×10^{-17}	8.208×10^{-17}	1.256×10^{-17}
I_8	a	6.063×10^{2}	1.031×10^{3}	3.566×10^{2}
I_{21}	a	-6.910×10^{-27}	-1.175×10^{-26}	4.885×10^{-17}
I_{22}	a	6.190×10^{2}	1.052×10^{3}	3.641×10^{2}
I_{23}	a	5.938×10^{2}	1.010×10^{3}	3.493×10^{2}
I_{24}	a	4.828×10^{-17}	8.208×10^{-17}	1.256×10^{-17}
I_{25}	a	6.063×10^{2}	1.031×10^{3}	3.566×10^{2}

Tabela 3.31: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 1.7$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-6.910×10^{-27}	-1.382×10^{-26}	-1.382×10^{-26}
I_2	a	6.063×10^{2}	1.213×10^{3}	3.031×10^{2}
I_3	a	6.190×10^{2}	1.238×10^{3}	3.095×10^{2}
I_4	a	6.063×10^{2}	1.213×10^{3}	3.031×10^{2}
I_5	a	5.938×10^{2}	1.188×10^{3}	2.969×10^{2}
I_6	a	6.063×10^{2}	1.213×10^{3}	3.031×10^{2}
I_7	a	4.828×10^{-17}	9.656×10^{-17}	6.035×10^{-18}
I_8	a	6.063×10^{2}	1.213×10^{3}	3.031×10^{2}
I_{21}	a	-6.910×10^{-27}	-1.382×10^{-26}	-1.382×10^{-26}
I_{22}	a	6.190×10^{2}	1.238×10^{3}	3.095×10^{2}
I_{23}	a	5.938×10^{2}	1.188×10^{3}	2.969×10^{2}
I_{24}	a	4.828×10^{-17}	9.656×10^{-17}	6.035×10^{-18}
I_{25}	a	6.063×10^{2}	1.213×10^{3}	3.031×10^{2}

Tabela 3.32: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 2.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-6.910×10^{-27}	-2.902×10^{-26}	-2.220×10^{-16}
I_2	a	6.063×10^{2}	2.546×10^{3}	1.443×10^{2}
I_3	a	6.190×10^{2}	2.600×10^{3}	1.474×10^{2}
I_4	a	6.063×10^{2}	2.546×10^{3}	1.443×10^{2}
I_5	a	5.938×10^{2}	2.494×10^{3}	1.414×10^{2}
I_6	a	6.063×10^{2}	2.546×10^{3}	1.443×10^{2}
I_7	a	4.828×10^{-17}	2.028×10^{-16}	1.347×10^{-18}
I_8	a	6.063×10^{2}	2.546×10^{3}	1.443×10^{2}
I_{21}	a	-6.910×10^{-27}	-2.902×10^{-26}	-2.220×10^{-16}
I_{22}	a	6.190×10^{2}	2.600×10^{3}	1.474×10^{2}
I_{23}	a	5.938×10^{2}	2.494×10^{3}	1.414×10^{2}
I_{24}	a	4.828×10^{-17}	2.028×10^{-16}	1.347×10^{-18}
I_{25}	a	6.063×10^{2}	2.546×10^{3}	1.443×10^{2}

Tabela 3.33: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 4.2$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	-6.910×10^{-27}	-6.910×10^{-26}	2.220×10^{-15}
I_2	a	6.063×10^{2}	6.063×10^{3}	6.063×10^{1}
I_3	a	6.190×10^{2}	6.190×10^{3}	6.190×10^{1}
I_4	a	6.063×10^{2}	6.063×10^{3}	6.063×10^{1}
I_5	a	5.938×10^{2}	5.938×10^{3}	5.938×10^{1}
I_6	a	6.063×10^{2}	6.063×10^{3}	6.063×10^{1}
I_7	a	4.828×10^{-17}	4.828×10^{-16}	2.442×10^{-20}
I_8	a	6.063×10^{2}	6.063×10^{3}	6.063×10^{1}
I_{21}	a	-6.910×10^{-27}	-6.910×10^{-26}	2.220×10^{-15}
I_{22}	a	6.190×10^{2}	6.190×10^{3}	6.190×10^{1}
I_{23}	a	5.938×10^{2}	5.938×10^{3}	5.938×10^{1}
I_{24}	a	4.828×10^{-17}	4.828×10^{-16}	2.442×10^{-20}
I_{25}	a	6.063×10^{2}	6.063×10^{3}	6.063×10^{1}

Tabela 3.34: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 10.0$

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_3	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_4	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_5	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_6	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{23}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0

Tabela 3.35: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_3	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_4	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_5	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_6	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{23}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0

Tabela 3.36: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_3	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_4	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_5	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_6	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{23}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0

Tabela 3.37: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_3	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_4	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_5	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_6	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{23}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0

Tabela 3.38: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodno oslonjena (LJ=1) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-6.187×10^2	-3.639×10^2	-3.639×10^2
I_3	a^{-1}	6.187×10^2	3.639×10^2	3.639×10^2
I_4	a^{-1}	-6.187×10^2	-3.639×10^2	-3.639×10^2
I_5	a^{-1}	6.187×10^2	3.639×10^2	3.639×10^2
I_6	a^{-1}	-6.187×10^2	-3.639×10^2	-3.639×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-6.187×10^2	-3.639×10^2	-3.639×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.187×10^2	3.639×10^2	3.639×10^2
I_{23}	a^{-1}	6.187×10^2	3.639×10^2	3.639×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-6.187×10^2	-3.639×10^2	-3.639×10^2

Tabela 3.39: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-6.187×10^2	-3.094×10^2	-3.094×10^2
I_3	a^{-1}	6.187×10^2	3.094×10^2	3.094×10^2
I_4	a^{-1}	-6.187×10^2	-3.094×10^2	-3.094×10^2
I_5	a^{-1}	6.187×10^2	3.094×10^2	3.094×10^2
I_6	a^{-1}	-6.187×10^2	-3.094×10^2	-3.094×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-6.187×10^2	-3.094×10^2	-3.094×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.187×10^2	3.094×10^2	3.094×10^2
I_{23}	a^{-1}	6.187×10^2	3.094×10^2	3.094×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-6.187×10^2	-3.094×10^2	-3.094×10^2

Tabela 3.40: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-6.187×10^2	-1.473×10^2	-1.473×10^2
I_3	a^{-1}	6.187×10^2	1.473×10^2	1.473×10^2
I_4	a^{-1}	-6.187×10^2	-1.473×10^2	-1.473×10^2
I_5	a^{-1}	6.187×10^2	1.473×10^2	1.473×10^2
I_6	a^{-1}	-6.187×10^2	-1.473×10^2	-1.473×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-6.187×10^2	-1.473×10^2	-1.473×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.187×10^2	1.473×10^2	1.473×10^2
I_{23}	a^{-1}	6.187×10^2	1.473×10^2	1.473×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-6.187×10^2	-1.473×10^2	-1.473×10^2

Tabela 3.41: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-6.187×10^2	-6.187×10^1	-6.187×10^1
I_3	a^{-1}	6.187×10^2	6.187×10^1	6.187×10^1
I_4	a^{-1}	-6.187×10^2	-6.187×10^1	-6.187×10^1
I_5	a^{-1}	6.187×10^2	6.187×10^1	6.187×10^1
I_6	a^{-1}	-6.187×10^2	-6.187×10^1	-6.187×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-6.187×10^2	-6.187×10^1	-6.187×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.187×10^2	6.187×10^1	6.187×10^1
I_{23}	a^{-1}	6.187×10^2	6.187×10^1	6.187×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-6.187×10^2	-6.187×10^1	-6.187×10^1

Tabela 3.42: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja uklještena (LJ=2) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.125×10^2	3.603×10^2	3.603×10^2
I_3	a^{-1}	6.252×10^2	3.678×10^2	3.678×10^2
I_4	a^{-1}	6.125×10^2	3.603×10^2	3.603×10^2
I_5	a^{-1}	6.001×10^2	3.530×10^2	3.530×10^2
I_6	a^{-1}	6.125×10^2	3.603×10^2	3.603×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.125×10^2	3.603×10^2	3.603×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.252×10^2	3.678×10^2	3.678×10^2
I_{23}	a^{-1}	6.001×10^2	3.530×10^2	3.530×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.125×10^2	3.603×10^2	3.603×10^2

Tabela 3.43: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.125×10^2	3.063×10^2	3.063×10^2
I_3	a^{-1}	6.252×10^2	3.126×10^2	3.126×10^2
I_4	a^{-1}	6.125×10^2	3.063×10^2	3.063×10^2
I_5	a^{-1}	6.001×10^2	3.001×10^2	3.001×10^2
I_6	a^{-1}	6.125×10^2	3.063×10^2	3.063×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.125×10^2	3.063×10^2	3.063×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.252×10^2	3.126×10^2	3.126×10^2
I_{23}	a^{-1}	6.001×10^2	3.001×10^2	3.001×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.125×10^2	3.063×10^2	3.063×10^2

Tabela 3.44: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.125×10^2	1.458×10^2	1.458×10^2
I_3	a^{-1}	6.252×10^2	1.489×10^2	1.489×10^2
I_4	a^{-1}	6.125×10^2	1.458×10^2	1.458×10^2
I_5	a^{-1}	6.001×10^2	1.429×10^2	1.429×10^2
I_6	a^{-1}	6.125×10^2	1.458×10^2	1.458×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.125×10^2	1.458×10^2	1.458×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.252×10^2	1.489×10^2	1.489×10^2
I_{23}	a^{-1}	6.001×10^2	1.429×10^2	1.429×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.125×10^2	1.458×10^2	1.458×10^2

Tabela 3.45: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.125×10^2	6.125×10^1	6.125×10^1
I_3	a^{-1}	6.252×10^2	6.252×10^1	6.252×10^1
I_4	a^{-1}	6.125×10^2	6.125×10^1	6.125×10^1
I_5	a^{-1}	6.001×10^2	6.001×10^1	6.001×10^1
I_6	a^{-1}	6.125×10^2	6.125×10^1	6.125×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.125×10^2	6.125×10^1	6.125×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.252×10^2	6.252×10^1	6.252×10^1
I_{23}	a^{-1}	6.001×10^2	6.001×10^1	6.001×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.125×10^2	6.125×10^1	6.125×10^1

Tabela 3.46: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-1.543×10^2	-9.075×10^1	-9.075×10^1
I_3	a^{-1}	1.543×10^2	9.075×10^1	9.075×10^1
I_4	a^{-1}	-1.543×10^2	-9.075×10^1	-9.075×10^1
I_5	a^{-1}	1.543×10^2	9.075×10^1	9.075×10^1
I_6	a^{-1}	-1.543×10^2	-9.075×10^1	-9.075×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-1.543×10^2	-9.075×10^1	-9.075×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.543×10^2	9.075×10^1	9.075×10^1
I_{23}	a^{-1}	1.543×10^2	9.075×10^1	9.075×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-1.543×10^2	-9.075×10^1	-9.075×10^1

Tabela 3.47: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-1.543×10^2	-7.714×10^1	-7.714×10^1
I_3	a^{-1}	1.543×10^2	7.714×10^1	7.714×10^1
I_4	a^{-1}	-1.543×10^2	-7.714×10^1	-7.714×10^1
I_5	a^{-1}	1.543×10^2	7.714×10^1	7.714×10^1
I_6	a^{-1}	-1.543×10^2	-7.714×10^1	-7.714×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-1.543×10^2	-7.714×10^1	-7.714×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.543×10^2	7.714×10^1	7.714×10^1
I_{23}	a^{-1}	1.543×10^2	7.714×10^1	7.714×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-1.543×10^2	-7.714×10^1	-7.714×10^1

Tabela 3.48: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-1.543×10^2	-3.673×10^1	-3.673×10^1
I_3	a^{-1}	1.543×10^2	3.673×10^1	3.673×10^1
I_4	a^{-1}	-1.543×10^2	-3.673×10^1	-3.673×10^1
I_5	a^{-1}	1.543×10^2	3.673×10^1	3.673×10^1
I_6	a^{-1}	-1.543×10^2	-3.673×10^1	-3.673×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-1.543×10^2	-3.673×10^1	-3.673×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.543×10^2	3.673×10^1	3.673×10^1
I_{23}	a^{-1}	1.543×10^2	3.673×10^1	3.673×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-1.543×10^2	-3.673×10^1	-3.673×10^1

Tabela 3.49: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	-1.543×10^2	-1.543×10^1	-1.543×10^1
I_3	a^{-1}	1.543×10^2	1.543×10^1	1.543×10^1
I_4	a^{-1}	-1.543×10^2	-1.543×10^1	-1.543×10^1
I_5	a^{-1}	1.543×10^2	1.543×10^1	1.543×10^1
I_6	a^{-1}	-1.543×10^2	-1.543×10^1	-1.543×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	-1.543×10^2	-1.543×10^1	-1.543×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.543×10^2	1.543×10^1	1.543×10^1
I_{23}	a^{-1}	1.543×10^2	1.543×10^1	1.543×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	-1.543×10^2	-1.543×10^1	-1.543×10^1

Tabela 3.50: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^2	4.582×10^2	1.324×10^3	2.695×10^2
I_3	a	1.591×10^2	2.705×10^2	9.360×10^1
I_4	a^2	4.582×10^2	1.324×10^3	2.695×10^2
I_5	a	1.466×10^2	2.491×10^2	8.621×10^1
I_6	a^2	4.582×10^2	1.324×10^3	2.695×10^2
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^2	4.582×10^2	1.324×10^3	2.695×10^2
I_{21}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	1.591×10^2	2.705×10^2	9.360×10^1
I_{23}	a	1.466×10^2	2.491×10^2	8.621×10^1
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^2	4.582×10^2	1.324×10^3	2.695×10^2

Tabela 3.51: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^2	4.582×10^2	1.833×10^3	2.291×10^2
I_3	a	1.591×10^2	3.182×10^2	7.956×10^1
I_4	a^2	4.582×10^2	1.833×10^3	2.291×10^2
I_5	a	1.466×10^2	2.931×10^2	7.328×10^1
I_6	a^2	4.582×10^2	1.833×10^3	2.291×10^2
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^2	4.582×10^2	1.833×10^3	2.291×10^2
I_{21}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	1.591×10^2	3.182×10^2	7.956×10^1
I_{23}	a	1.466×10^2	2.931×10^2	7.328×10^1
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^2	4.582×10^2	1.833×10^3	2.291×10^2

Tabela 3.52: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^2	4.582×10^2	8.083×10^3	1.091×10^2
I_3	a	1.591×10^2	6.683×10^2	3.789×10^1
I_4	a^2	4.582×10^2	8.083×10^3	1.091×10^2
I_5	a	1.466×10^2	6.155×10^2	3.489×10^1
I_6	a^2	4.582×10^2	8.083×10^3	1.091×10^2
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^2	4.582×10^2	8.083×10^3	1.091×10^2
I_{21}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	1.591×10^2	6.683×10^2	3.789×10^1
I_{23}	a	1.466×10^2	6.155×10^2	3.489×10^1
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^2	4.582×10^2	8.083×10^3	1.091×10^2

Tabela 3.53: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^2	4.582×10^2	4.582×10^4	4.582×10^1
I_3	a	1.591×10^2	1.591×10^3	1.591×10^1
I_4	a^2	4.582×10^2	4.582×10^4	4.582×10^1
I_5	a	1.466×10^2	1.466×10^3	1.466×10^1
I_6	a^2	4.582×10^2	4.582×10^4	4.582×10^1
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^2	4.582×10^2	4.582×10^4	4.582×10^1
I_{21}	a^{-1}	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	1.591×10^2	1.591×10^3	1.591×10^1
I_{23}	a	1.466×10^2	1.466×10^3	1.466×10^1
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^2	4.582×10^2	4.582×10^4	4.582×10^1

Tabela 3.54: Verifikacija funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a	6.063×10^2	1.031×10^3	3.566×10^2
I_3	a	6.190×10^2	1.052×10^3	3.641×10^2
I_4	a	6.063×10^2	1.031×10^3	3.566×10^2
I_5	a	5.938×10^2	1.010×10^3	3.493×10^2
I_6	a	6.063×10^2	1.031×10^3	3.566×10^2
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a	6.063×10^2	1.031×10^3	3.566×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	6.190×10^2	1.052×10^3	3.641×10^2
I_{23}	a	5.938×10^2	1.010×10^3	3.493×10^2
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a	6.063×10^2	1.031×10^3	3.566×10^2

Tabela 3.55: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a	6.063×10^2	1.213×10^3	3.031×10^2
I_3	a	6.190×10^2	1.238×10^3	3.095×10^2
I_4	a	6.063×10^2	1.213×10^3	3.031×10^2
I_5	a	5.938×10^2	1.188×10^3	2.969×10^2
I_6	a	6.063×10^2	1.213×10^3	3.031×10^2
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a	6.063×10^2	1.213×10^3	3.031×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	6.190×10^2	1.238×10^3	3.095×10^2
I_{23}	a	5.938×10^2	1.188×10^3	2.969×10^2
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a	6.063×10^2	1.213×10^3	3.031×10^2

Tabela 3.56: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a	6.063×10^2	2.546×10^3	1.443×10^2
I_3	a	6.190×10^2	2.600×10^3	1.474×10^2
I_4	a	6.063×10^2	2.546×10^3	1.443×10^2
I_5	a	5.938×10^2	2.494×10^3	1.414×10^2
I_6	a	6.063×10^2	2.546×10^3	1.443×10^2
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a	6.063×10^2	2.546×10^3	1.443×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	6.190×10^2	2.600×10^3	1.474×10^2
I_{23}	a	5.938×10^2	2.494×10^3	1.414×10^2
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a	6.063×10^2	2.546×10^3	1.443×10^2

Tabela 3.57: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a	6.063×10^2	6.063×10^3	6.063×10^1
I_3	a	6.190×10^2	6.190×10^3	6.190×10^1
I_4	a	6.063×10^2	6.063×10^3	6.063×10^1
I_5	a	5.938×10^2	5.938×10^3	5.938×10^1
I_6	a	6.063×10^2	6.063×10^3	6.063×10^1
I_7	a	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a	6.063×10^2	6.063×10^3	6.063×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a	6.190×10^2	6.190×10^3	6.190×10^1
I_{23}	a	5.938×10^2	5.938×10^3	5.938×10^1
I_{24}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a	6.063×10^2	6.063×10^3	6.063×10^1

Tabela 3.58: Verifikacija funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	4.582×10^2	2.695×10^2	2.695×10^2
I_3	a^{-1}	1.591×10^2	9.360×10^1	9.360×10^1
I_4	a^{-1}	4.582×10^2	2.695×10^2	2.695×10^2
I_5	a^{-1}	1.466×10^2	8.621×10^1	8.621×10^1
I_6	a^{-1}	4.582×10^2	2.695×10^2	2.695×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	4.582×10^2	2.695×10^2	2.695×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.591×10^2	9.360×10^1	9.360×10^1
I_{23}	a^{-1}	1.466×10^2	8.621×10^1	8.621×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	4.582×10^2	2.695×10^2	2.695×10^2

Tabela 3.59: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	4.582×10^2	2.291×10^2	2.291×10^2
I_3	a^{-1}	1.591×10^2	7.956×10^1	7.956×10^1
I_4	a^{-1}	4.582×10^2	2.291×10^2	2.291×10^2
I_5	a^{-1}	1.466×10^2	7.328×10^1	7.328×10^1
I_6	a^{-1}	4.582×10^2	2.291×10^2	2.291×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	4.582×10^2	2.291×10^2	2.291×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.591×10^2	7.956×10^1	7.956×10^1
I_{23}	a^{-1}	1.466×10^2	7.328×10^1	7.328×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	4.582×10^2	2.291×10^2	2.291×10^2

Tabela 3.60: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	4.582×10^2	1.091×10^2	1.091×10^2
I_3	a^{-1}	1.591×10^2	3.789×10^1	3.789×10^1
I_4	a^{-1}	4.582×10^2	1.091×10^2	1.091×10^2
I_5	a^{-1}	1.466×10^2	3.489×10^1	3.489×10^1
I_6	a^{-1}	4.582×10^2	1.091×10^2	1.091×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	4.582×10^2	1.091×10^2	1.091×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.591×10^2	3.789×10^1	3.789×10^1
I_{23}	a^{-1}	1.466×10^2	3.489×10^1	3.489×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	4.582×10^2	1.091×10^2	1.091×10^2

Tabela 3.61: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	4.582×10^2	4.582×10^1	4.582×10^1
I_3	a^{-1}	1.591×10^2	1.591×10^1	1.591×10^1
I_4	a^{-1}	4.582×10^2	4.582×10^1	4.582×10^1
I_5	a^{-1}	1.466×10^2	1.466×10^1	1.466×10^1
I_6	a^{-1}	4.582×10^2	4.582×10^1	4.582×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	4.582×10^2	4.582×10^1	4.582×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	1.591×10^2	1.591×10^1	1.591×10^1
I_{23}	a^{-1}	1.466×10^2	1.466×10^1	1.466×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	4.582×10^2	4.582×10^1	4.582×10^1

Tabela 3.62: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.063×10^2	3.566×10^2	3.566×10^2
I_3	a^{-1}	6.190×10^2	3.641×10^2	3.641×10^2
I_4	a^{-1}	6.063×10^2	3.566×10^2	3.566×10^2
I_5	a^{-1}	5.938×10^2	3.493×10^2	3.493×10^2
I_6	a^{-1}	6.063×10^2	3.566×10^2	3.566×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.063×10^2	3.566×10^2	3.566×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.190×10^2	3.641×10^2	3.641×10^2
I_{23}	a^{-1}	5.938×10^2	3.493×10^2	3.493×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.063×10^2	3.566×10^2	3.566×10^2

Tabela 3.63: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 1.7$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.063×10^2	3.031×10^2	3.031×10^2
I_3	a^{-1}	6.190×10^2	3.095×10^2	3.095×10^2
I_4	a^{-1}	6.063×10^2	3.031×10^2	3.031×10^2
I_5	a^{-1}	5.938×10^2	2.969×10^2	2.969×10^2
I_6	a^{-1}	6.063×10^2	3.031×10^2	3.031×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.063×10^2	3.031×10^2	3.031×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.190×10^2	3.095×10^2	3.095×10^2
I_{23}	a^{-1}	5.938×10^2	2.969×10^2	2.969×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.063×10^2	3.031×10^2	3.031×10^2

Tabela 3.64: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 2.0$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.063×10^2	1.443×10^2	1.443×10^2
I_3	a^{-1}	6.190×10^2	1.474×10^2	1.474×10^2
I_4	a^{-1}	6.063×10^2	1.443×10^2	1.443×10^2
I_5	a^{-1}	5.938×10^2	1.414×10^2	1.414×10^2
I_6	a^{-1}	6.063×10^2	1.443×10^2	1.443×10^2
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.063×10^2	1.443×10^2	1.443×10^2
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.190×10^2	1.474×10^2	1.474×10^2
I_{23}	a^{-1}	5.938×10^2	1.414×10^2	1.414×10^2
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.063×10^2	1.443×10^2	1.443×10^2

Tabela 3.65: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 4.2$. Male vrednosti integrala su anulirane

I_α	$z(a)$	$I_\alpha(1)$	$z(a) \times I_\alpha(1)$	$I_\alpha(a)$
I_1	a	0.000×10^0	0.000×10^0	0.000×10^0
I_2	a^{-1}	6.063×10^2	6.063×10^1	6.063×10^1
I_3	a^{-1}	6.190×10^2	6.190×10^1	6.190×10^1
I_4	a^{-1}	6.063×10^2	6.063×10^1	6.063×10^1
I_5	a^{-1}	5.938×10^2	5.938×10^1	5.938×10^1
I_6	a^{-1}	6.063×10^2	6.063×10^1	6.063×10^1
I_7	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_8	a^{-1}	6.063×10^2	6.063×10^1	6.063×10^1
I_{21}	a	0.000×10^0	0.000×10^0	0.000×10^0
I_{22}	a^{-1}	6.190×10^2	6.190×10^1	6.190×10^1
I_{23}	a^{-1}	5.938×10^2	5.938×10^1	5.938×10^1
I_{24}	a^{-3}	0.000×10^0	0.000×10^0	0.000×10^0
I_{25}	a^{-1}	6.063×10^2	6.063×10^1	6.063×10^1

Tabela 3.66: Verifikacija *specifičnih* funkcija skaliranja za granični uslov gde su oba kraja slobodna (LJ=6) uz $m = 99$, $n = 97$ i $a = 10.0$. Male vrednosti integrala su anulirane

Glava 4

Uvid u tehničku implementaciju

4.1 Pomoćni projekat `friendly_name_mixin`

Projekat `friendly_name_mixin` [156, 157] nastao je usled potrebe projekta `beam_integrals` za automatskim dodeljivanjem ljudski čitljivih imena postojećim Python [32] klasama.

Projekat `friendly_name_mixin` razvijan je u Python [32] programskom jeziku, uz oslanjanje na Python standardnu biblioteku [158]. Usled korišćenja Python programskog jezika projekat `friendly_name_mixin` je, sa praktične strane, platformski neutralan [158–160].

Izvorni kôd projekta `friendly_name_mixin`, uz prateći podsistem za automatsku verifikaciju, javno je dostupan [156] pod BSD Open Source licencom [21]. Štaviše, celokupni proces i istorija razvoja projekta su, od samog početka, javno dostupni [156].

Celokupni izvorni kôd projekta sadržan je u jednom Python modulu pod nazivom `friendly_name_mixin`. Na listingu 4.1 dat je izvod iz izvornog kôda glavnog modula:

```
import re


class FriendlyNameFromClassMixin(object):
    _name_re = re.compile('((?<=[a-z])[A-Z])|([A-Z](?![A-Z0-9]|$))')

    @property
    def name(self):
        if not hasattr(self, '_name'):
            class_name = self.__class__.__name__
            self._name = self._name_re.sub(r' \1', class_name).strip()

        return self._name
```

Listing 4.1: Izvod iz Python modula `friendly_name_mixin`, puna verzija dostupna na [156]

pri čemu je implementacija inspirisana funkcijom `django.db.models.options.get_verbose_name` iz popularnog Django [56] Open Source projekta.

Python programski jezik podržava *mixin* klase [161, 162] i višestruko nasleđivanje [163, 164]. Stoga je za pravilnu upotrebu dovoljno da ciljna klasa nasledi i `FriendlyNameFromClassMixin` *mixin* klasu (prikazanu na listingu 4.1) i potom prosto pristupa svom novom svojstvu (eng. *property*) [165, 166] `name`. U pozadini će svako pristupanje svojstvu `name` u ciljnoj Python klasi dovesti do sledećeg postupka:

1. Proverava se da li ciljna Python klasa poseduje keširani atribut `_name`. Ako da, skače se na korak 5

2. U suprotnom, mehanizmom introspekcije [167] dobavlja se naziv ciljne klase i smešta u lokalnu promenljivu `class_name`

3. Upotrebom regularnog izraza [168, 169], smeštenog u atribut `_name_re`, transformiše se sadržaj lokalne promenljive `class_name` u ljudski čitljivo ime ciljne klase

4. U ciljnu klasu dinamički se dodaje keširani atribut `_name`, sa vrednošću prethodno određenog ljudski čitljivog imena

5. Vraća se vrednost keširanog atributa `_name`

Jedan primer upotrebe projekta `friendly_name_mixin` dat je na listingu 4.2, kroz interaktivnu Python sesiju:

```
>>> from friendly_name_mixin import FriendlyNameFromClassMixin

>>> class IsHTML5BetterThanFlash11OrIsItMe(FriendlyNameFromClassMixin):
...     answer = 'yes'

>>> print IsHTML5BetterThanFlash11OrIsItMe().name
Is HTML5 Better Than Flash11 Or Is It Me
```

Listing 4.2: Primer upotrebe projekta `friendly_name_mixin`

gde se na osnovu naziva Python klase:

```
IsHTML5BetterThanFlash11OrIsItMe
```

automatski određuje njeno ljudski čitljivo ime:

```
Is HTML5 Better Than Flash11 Or Is It Me
```

Jenkins je poznat kao vodeći Open Source *Continuous Integration* server [129] i koristi se za automatsko prevođenje, testiranje i praćenje toka razvoja projekta `friendly_name_mixin`, čiji je javan Jenkins projekat dostupan na [170]. Naša istraživačka grupa sve svoje Jenkins projekte smešta na naš centralni Jenkins integracioni server [131].

Naš Jenkins integracioni server automatski izvršava podsistem za automatsku verifikaciju `friendly_name_mixin` projekta svaki put kada se registruje promena u javno dostupnom repozitorijumu izvornog koda za `friendly_name_mixin` projekat [156].

4.2 Pomoćni projekat `simple_plugins`

Projekat `simple_plugins` [171, 172] nastao je usled potrebe projekta `beam_integrals` za programskom bibliotekom za brzi razvoj jednostavnih, pouzdanih, moćnih i proširivih podsistema za automatsko registrovanje programskih dodataka (eng. *plugin system*). Naime, u projektu `beam_integrals` granični uslovi, integrali i algoritmi za rešavanje karakterističnih jednačina su implementirani kao *zasebni* i *međusobno nezavisni* podsistemi programskih dodataka (detaljnije u sekcijama 4.3.2–4.3.5).

Projekat `simple_plugins` razvijan je u Python programskom jeziku, uz oslanjanje na Python standardnu biblioteku. Usled korišćenja Python programskog jezika projekat `simple_plugins` je, sa praktične strane, platformski neutralan.

Izvorni kôd projekta `simple_plugins`, uz prateći podsistem za automatsku verifikaciju, javno je dostupan [171] pod BSD Open Source licencom. Štaviše, celokupni proces i istorija razvoja projekta su, od samog početka, javno dostupni [171].

Celokupni izvorni kôd projekta sadržan je u jednom Python modulu pod nazivom `simple_plugins`. Na listingu 4.3 dat je izvod iz izvornog kôda glavnog modula:

```
from warnings import warn


class PerformanceWarning(UserWarning):
    """Warning issued when something causes a performance degradation"""


class CoercionError(Exception):
    """The given value can't be coerced to a valid instance"""


class AttrDict(dict):
    __getattr__= dict.__getitem__
    __setattr__= dict.__setitem__
    __delattr__= dict.__delitem__


class PluginMount(type):
    def __init__(cls, name, bases, attrs): #@UnusedVariable
        if not hasattr(cls, '_plugin_registry'):
            # This branch only executes when processing the mount point itself.
            # So, since this is a new plugin type, not an implementation, this
            # class shouldn't be registered as a plugin. Instead, it sets up a
            # list where plugins can be registered later.
            cls._plugin_registry = []

            meta = getattr(cls, 'Meta', object())
            def override_options(options):
                return dict(
                    (name, getattr(meta, name, default_value))
                    for name, default_value in options.items()
                )

            cls._meta = AttrDict(_base_class=cls)
            cls._meta.update(override_options({
                'id_field': 'id',
                'id_field_coerce': int,
            }))
        else:
```

```
            # Don't register 'Base*' classes as plugin implementations
            if not name.startswith('Base'):
                # This must be a plugin implementation, which should be registered.
                # Simply appending it to the list is all that's needed to keep
                # track of it later.
                cls._plugin_registry.append(cls)

        base_cls = cls._meta._base_class
        base_cls._plugins = None # Clear plugin cache

    def _unregister_plugin(self):
        base_cls = self._meta._base_class
        base_cls._plugin_registry.remove(self)
        base_cls._plugins = None # Clear plugin cache

    @property
    def plugins(self):
        base_cls = self._meta._base_class
        if base_cls._plugins is None:
            x = AttrDict()
            x.classes = set(self._plugin_registry)
            x.instances = set(cls() for cls in x.classes)
            x.id_to_instance = dict(
                (getattr(obj, self._meta.id_field), obj) for obj in x.instances
            )
            x.id_to_class = dict((k, type(v)) for k, v in x.id_to_instance.items())
            x.class_to_id = dict((v, k) for k, v in x.id_to_class.items())
            x.instances_sorted_by_id = [
                v for _, v in sorted(x.id_to_instance.items())
            ]
            x.valid_ids = set(x.id_to_instance)

            if hasattr(base_cls, '_contribute_to_plugins'):
                base_cls._contribute_to_plugins(_plugins=x)

            base_cls._plugins = x

        return base_cls._plugins

    def coerce(self, value):
        """Coerce the passed value into the right instance"""
        perf_warn_msg = "Creating too many %s instances may be expensive, "\
                        "passing the objects id is generally preferred" %\
                        self._meta._base_class

        # Check if the passed value is already a `_base_class` instance
        if isinstance(value, self._meta._base_class):
            warn(perf_warn_msg, category=PerformanceWarning)
            return value # No coercion needed

        # Check if the passed value is a `_base_class` subclass
        try:
            if issubclass(value, self._meta._base_class):
                warn(perf_warn_msg, category=PerformanceWarning)
                return value()
        except TypeError:
```

```
            pass # Passed value is not a class

        # Check if the passed value is a valid object id
        try:
            object_id = self._meta.id_field_coerce(value)
            try:
                return self.plugins.id_to_instance[object_id]
            except KeyError:
                raise CoercionError("%d is not a valid object id" % object_id)
        except (TypeError, ValueError):
            pass # Passed value can't be coerced to the object id type

        # Can't coerce an unknown type
        raise CoercionError("Can't coerce %r to a valid %s instance" % (
            value, self._meta._base_class)
        )
```

Listing 4.3: Izvod iz Python modula `simple_plugins`, puna verzija dostupna na [171]

pri čemu je osnovna ideja bazirana na interesantnom rešenju koje je predstavio Marty Alchin [173] dok je implementacija `AttrDict` klase preuzeta sa [174]. Implementacija konfiguracije podsistema programskih dodataka, kroz unutrašnju `Meta` klasu, inspirisana je Python paketom `django.db.models` iz popularnog Django Open Source projekta.

Python podržava koncept *metaklasa* [175–179], štaviše u programskom jeziku Python svaka klasa je instanca odgovarajuće metaklase [175], kao što je i svaki objekat instanca odgovarajuće klase. Samim tim metaklasa može da kontroliše atribute i ponašanje klase, ili pak da utiče na sam način kreiranja klase [176]. Stoga metaklase značajno olakšavaju napredno metaprogramiranje i pružaju nove interesantne mogućnosti u razvoju složenih softverskih rešenja (npr. videti Python paket `django.db.models` iz popularnog Django Open Source projekta). Marty Alchin je svoje rešenje za automatsko registrovanje programskih dodataka [173] upravo bazirao na korišćenju metaklasa.

Osnovni proces registrovanja programskih dodataka Marty Alchin je objasnio u svom članku [173]. Projekat `simple_plugins` proširuje njegovo rešenje i uvodi dodatna poboljšanja, npr:

1. Svaka klasa koja nasledi baznu klasu podsistema programskih dodataka (eng. *plugin mount point*) [173] biće registrovana kao programski dodatak, osim u slučaju da naziv

te klase počinje sa stringom `'Base'`. Time se programeru pruža mogućnost da izbegne registrovanje apstraktnih ili parcijalno definisanih klasa.

2. Svaki kreirani podsistem programskih dodataka se može konfigurisati kroz dodavanje unutrašnje `Meta` klase u baznu klasu podsistema programskih dodataka, pri čemu su dostupna sledeća podešavanja:

   (a) `id_field`, koji predstavlja naziv atributa klase koji će se koristiti za jednoznačnu identifikaciju konkretne klase unutar podsistema programskih dodataka. Ako se ovo podešavanje ne navede, koristiće se atribut `id`.

   (b) `id_field_coerce`, koji predstavlja funkciju koja će se koristiti prilikom poziva metode `coerce` za konverziju joj prosleđene vrednosti u tip podatka koji odgovara atributu klase na koji ukazuje podešavanje `id_field`, a zarad identifikacije konkretne klase unutar podsistema programskih dodataka. Ako se ovo podešavanje ne navede, koristiće se funkcija `int`.

3. Mogućnost deregistracije postojećeg programskog dodatka korišćenjem skrivene metode `_unregister_plugin`, koja je automatski ubačena u baznu klasu podsistema programskih dodataka

4. Keširano dinamičko svojstvo `plugins` koje je automatski ubačeno u baznu klasu podsistema programskih dodataka i sadrži bogate informacije o trenutno registrovanim programskim dodacima:

   (a) `classes`, koji predstavlja spisak klasa registrovanih programskih dodataka

   (b) `instances`, koji predstavlja spisak objekata koji su instance klasa registrovanih programskih dodataka. Koristi se za optimizaciju utroška radne memorije i kreiranje isključivo jednog objekta (tzv. *singleton* [180]) za svaki registrovani programski dodatak

(c) `id_to_instance`, koji predstavlja mapiranje sa *ključa* na *vrednost* (Python tip *dict* [181]), gde je *ključ* vrednost atributa na koji ukazuje podešavanje `id_field` a *vrednost* je *singleton* objekat registrovanog programskog dodatka

(d) `id_to_class`, koji predstavlja mapiranje sa *ključa* na *vrednost*, gde je *ključ* vrednost atributa na koji ukazuje podešavanje `id_field` a *vrednost* je klasa registrovanog programskog dodatka

(e) `class_to_id`, koji predstavlja mapiranje sa *ključa* na *vrednost*, gde je *ključ* klasa registrovanog programskog dodatka a *vrednost* je vrednost atributa na koji ukazuje podešavanje `id_field`

(f) `instances_sorted_by_id`, koji predstavlja spisak objekata koji su instance klasa registrovanih programskih dodataka, gde je spisak sortiran po vrednosti atributa na koji ukazuje podešavanje `id_field`

(g) `valid_ids`, koji predstavlja spisak vrednosti atributa na koji ukazuje podešavanje `id_field`, za sve klase registrovanih programskih dodataka

5. Metoda `coerce`, koja je automatski ubačena u baznu klasu podsistema programskih dodataka, pruža mogućnost konverzije prosleđene joj vrednosti u odgovarajući *singleton* objekat registrovanog programskog dodatka, primenom funkcije na koji ukazuje podešavanje `id_field_coerce` a na osnovu dinamičkog svojstva `plugins.id_to_instance`.

Jedan primer upotrebe projekta `simple_plugins` dat je na listingu 4.4, kroz interaktivnu Python sesiju:

```
>>> from pprint import pprint
>>> from simple_plugins import PluginMount

>>> class BaseHttpResponse(object):
...     """Mount point is not registered as a plugin"""
...
...     status_code = None
...
...     __metaclass__ = PluginMount
```

```
...
...     class Meta:
...         id_field = 'status_code'
...
...     def __repr__(self):
...         return "<%s: %s>" % (self.__class__.__name__, self.status_code)
...

>>> class OK(BaseHttpResponse):
...     status_code = 200
...

>>> class BaseRedirection(BaseHttpResponse):
...     """'Base*' classes are not registered as plugins"""
...     pass
...

>>> class MovedPermanently(BaseRedirection):
...     status_code = 301
...

>>> class NotModified(BaseRedirection):
...     status_code = 304
...

>>> class BadRequest(BaseHttpResponse):
...     status_code = 400
...

>>> class NotFound(BaseHttpResponse):
...     status_code = 404
...

# All plugin info
>>> BaseHttpResponse.plugins.keys()
['valid_ids', 'instances_sorted_by_id', 'id_to_class', 'instances',
 'classes', 'class_to_id', 'id_to_instance']

# Plugin info can be accessed using either dict...
>>> pprint(BaseHttpResponse.plugins['valid_ids'])
set([200, 301, 304, 400, 404])

# ... or object notation
>>> pprint(BaseHttpResponse.plugins.valid_ids)
set([200, 301, 304, 400, 404])

>>> pprint(BaseHttpResponse.plugins.classes)
set([<class '__main__.OK'>,
     <class '__main__.MovedPermanently'>,
     <class '__main__.NotModified'>,
     <class '__main__.BadRequest'>,
     <class '__main__.NotFound'>])

>>> BaseHttpResponse.plugins.id_to_class[200]
<class '__main__.OK'>
```

```
>>> BaseHttpResponse.plugins.id_to_instance[200]
<OK: 200>

>>> pprint(BaseHttpResponse.plugins.instances_sorted_by_id)
[<OK: 200>,
 <MovedPermanently: 301>,
 <NotModified: 304>,
 <BadRequest: 400>,
 <NotFound: 404>]

# Unregister the ‘NotFound‘ plugin
>>> NotFound._unregister_plugin()
>>> BaseHttpResponse.plugins.instances_sorted_by_id
[<OK: 200>, <MovedPermanently: 301>, <NotModified: 304>, <BadRequest: 400>]

# Coerce the passed value into the right instance
>>> BaseHttpResponse.coerce(200)
<OK: 200>
```

Listing 4.4: Primer mogućnosti projekta `simple_plugins`

Dok se na listingu 4.5, kroz interaktivnu Python sesiju, pokazuje da su podsistemi programskih dodataka *međusobno nezavisni*:

```
>>> from friendly_name_mixin import FriendlyNameFromClassMixin
>>> from simple_plugins import PluginMount

>>> class BaseDirection(FriendlyNameFromClassMixin):
...     dx = None
...     dy = None
...
...     __metaclass__ = PluginMount
...
...     class Meta:
...         id_field = 'name'
...         id_field_coerce = str
...
...     def __repr__(self):
...         return "<%s: dx=%d, dy=%d>" % (self.name, self.dx, self.dy)
...

>>> class Up(BaseDirection):
...     dx =  0
...     dy = +1
...

>>> class Down(BaseDirection):
...     dx =  0
...     dy = -1
...

>>> class Left(BaseDirection):
...     dx = -1
...     dy =  0
...
```

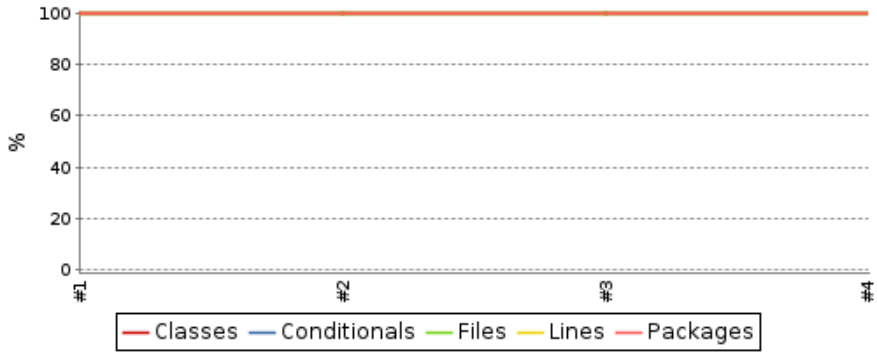
```
>>> class Right(BaseDirection):
...     dx = +1
...     dy =  0
...

>>> class BaseGZipCompressionLevel(FriendlyNameFromClassMixin):
...     level = None
...
...     __metaclass__ = PluginMount
...
...     class Meta:
...         id_field = 'level'
...
...     def __repr__(self):
...         return "<%s: %d>" % (self.name, self.level)
...

>>> class Best(BaseGZipCompressionLevel):
...     level = 9
...

>>> class Fast(BaseGZipCompressionLevel):
...     level = 1
...

>>> class Normal(BaseGZipCompressionLevel):
...     level = 6
...

>>> BaseDirection.plugins.instances_sorted_by_id
[<Down: dx=0, dy=-1>, <Left: dx=-1, dy=0>, <Right: dx=1, dy=0>, <Up: dx=0, dy=1>]

>>> BaseGZipCompressionLevel.plugins.instances_sorted_by_id
[<Fast: 1>, <Normal: 6>, <Best: 9>]

# Verify that plugins are not shared between mount points
>>> BaseDirection.plugins.classes.isdisjoint(
...     BaseGZipCompressionLevel.plugins.classes
... )
True
```

Listing 4.5: Primer međusobne nezavisnosti 2 odvojena podsistema programskih dodataka

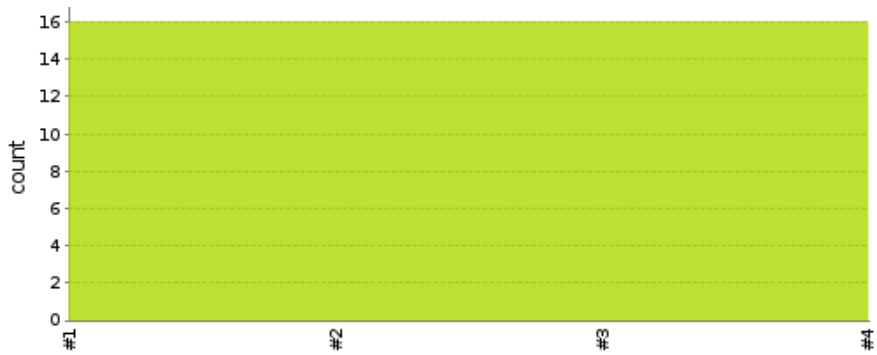
Jenkins Open Source *Continuous Integration* server se koristi za automatsko prevođenje, testiranje i praćenje toka razvoja projekta `simple_plugins`, čiji je javan Jenkins projekat dostupan na [182].

Podsistem za automatsku verifikaciju `simple_plugins` projekta razvijan je sa ciljem da pokrije svaki aspekt projekta uz 100% pokrivenosti koda testovima (slika 4.1) i sadrži 16 međusobno nezavisnih testova (slika 4.2) koje naš Jenkins integracioni

server automatski izvršava svaki put kada se registruje promena u javno dostupnom repozitorijumu izvornog koda za projekat `simple_plugins` [171].



Slika 4.1: Istorijski izveštaj pokrivenosti koda testovima, preuzet sa naslovne strane Jenkins projekta `simple_plugins` [182]. Horizontalna osa predstavlja redni broj Jenkins prevođenja (eng. *build number*)



Slika 4.2: Istorijski izveštaj broja i stepena prolaznosti testova u okviru podsistema za automatsku verifikaciju, preuzet sa naslovne strane Jenkins projekta `simple_plugins` [182]. Horizontalna osa predstavlja redni broj Jenkins prevođenja (eng. *build number*)

4.3 Projekat `beam_integrals`

Projekat `beam_integrals` [137, 183] pruža referentnu Open Source implementaciju hibridne metode (opisana u sekciji 2.1).

Projekat `beam_integrals` razvijan je u Python programskom jeziku, uz oslanjanje na Python standardnu biblioteku i sledeće eksterne programske biblioteke:

1. SymPy [25, 184] je interaktivni alat i Open Source biblioteka za simboličku matematiku, sa ciljem da postane vodeći Open Source *Computer Algebra System*

2. mpmath [185, 186] je Open Source biblioteka koja uvodi podršku za proizvoljno-preciznu aritmetiku u pokretnom zarezu (eng. *arbitrary-precision floating point arithmetic*). U verziji 0.7.1 SymPy biblioteke, koja se koristi za ovo istraživanje, mpmath je već integrisan u SymPy (Python paket `sympy.mpmath`).

3. projekat `friendly_name_mixin`, već opisan u sekciji 4.1

4. projekat `simple_plugins`, već opisan u sekciji 4.2

Takođe, projekat `beam_integrals` u svojoj dokumentaciji preporučuje instalaciju Open Source biblioteke gmpy [187, 188], koja nije obavezna. Međutim, bez nje će svi proračuni biti značajno sporiji, naročito u režimu visoke preciznosti [185, 187].

Ukoliko se koristi starija podržana verzija (2.6) Python programskog jezika potrebno je instalirati i Open Source biblioteku argparse [189, 190], koja je sastavni deo standardne biblioteke u novijim verzijama Python programskog jezika.

Usled korišćenja Python programskog jezika `beam_integrals` projekat je, sa praktične strane, platformski neutralan.

Izvorni kôd projekta `beam_integrals`, uz prateći podsistem za automatsku verifikaciju, javno je dostupan [137] pod BSD Open Source licencom. Štaviše, celokupni proces i istorija razvoja projekta su, od samog početka, javno dostupni [137].

4.3.1 Modul `beam_integrals`

Modul `beam_integrals` definiše konstante sa podrazumevanim vrednostima globalnih podešavanja `beam_integrals` projekta i zajedničke promenljive koje koriste ostali moduli.

Na listingu 4.6 dat je izvod iz izvornog kôda `beam_integrals` modula:

```
import os
from sympy import symbols


PROJECT_SETTINGS_DIR = os.path.expanduser('~/.beam_integrals')
DEFAULT_MAX_MODE = 100
DEFAULT_DECIMAL_PRECISION = 155

# Commonly used symbols
a, y, mu_m = symbols('a, y, mu_m')
```

Listing 4.6: Izvod iz Python modula `beam_integrals`, puna verzija dostupna na [137]

Dostupna su sledeća globalna podešavanja:

1. `PROJECT_SETTINGS_DIR`, podrazumevani direktorijum u koji se smeštaju svi podaci vezani za rad `beam_integrals` projekta. Koristi se prvenstveno za smeštaj keširanih vrednosti korenova karakteristične jednačine (detaljnije u sekciji 4.3.3)

2. `DEFAULT_MAX_MODE`, podrazumevani maksimalni mod do kog se generišu podaci i vrše proračuni

3. `DEFAULT_DECIMAL_PRECISION`, podrazumevana i optimalna (tj. minimalna potrebna) radna preciznost `beam_integrals` projekta i njegovog hibridnog metoda. Eksperimentalno je određena na 155 decimalnih mesta (detaljnije na strani 26)

Promenljive `a`, `y` i `mu_m` su SymPy simboličke promenljive [191] i direktno se mapiraju na matematičke promenljive a, y i μ_m iz jednačina datih u poglavlju 1.

4.3.2 Modul `beam_integrals.beam_types`

Zahvaljujući korišćenju SymPy biblioteke celokupna implementacija proračuna u projektu `beam_integrals` je hibridnog tipa. Naime, sve moguće definicije i proračuni su SymPy simbolički izrazi i pretvaraju se u numeričke podatke tek kada je to apsolutno neophodno, npr. prilikom numeričke optimizacije korenova (sekcija 4.3.3) ili numeričke integracije (sekcija 4.3.5).

Modul `beam_integrals.beam_types` simbolički definiše sve granične uslove date u sekciji 1.2.1, uz prateću programsku podršku za jednostavnu i efikasnu upotrebu graničnih uslova u drugim modulima.

Na listingu 4.7 dat je izvod iz izvornog kôda `beam_integrals.beam_types` modula:

```
from friendly_name_mixin import FriendlyNameFromClassMixin
from simple_plugins import PluginMount
from sympy import cos, cosh, diff, Float, pi, sin, sinh, tan, tanh
from . import a, y, mu_m


class BaseBeamType(FriendlyNameFromClassMixin):
    id = None
    characteristic_function = None

    dont_improve_mu_m_for_modes = ()
    mu_m_initial_search_width = pi/10
    mu_m_increase_search_width_by = Float(1.05)

    __metaclass__ = PluginMount

    def __str__(self):
        return "%s (id=%d)" % (self.name, self.id)

    @property
    def filename(self):
        return "%d-%s" % (self.id, '-'.join(self.name.lower().split()))

    @property
    def characteristic_equation_str(self):
        return "%s = 0" % self.characteristic_function

    def mu_m_initial_guess(self, mode):
        raise NotImplementedError

    def Y_m(self, mode):
        raise NotImplementedError

    _Y_M_DERIVATIVES_CACHE_MAX_ORDER = 2
    _Y_m_derivatives_cache = None
```

```
    def Y_m_derivative_from_cache(self, mode, order):
        if self._Y_m_derivatives_cache is None:
            self._Y_m_derivatives_cache = {}
            cache_keys = [None] + list(self.dont_improve_mu_m_for_modes)
            for n in range(1, self._Y_M_DERIVATIVES_CACHE_MAX_ORDER+1):
                self._Y_m_derivatives_cache[n] = dict(
                    (k, diff(self.Y_m(k), y, n))
                    for k in cache_keys
                )

        key = mode if mode in self.dont_improve_mu_m_for_modes else None
        return self._Y_m_derivatives_cache[order][key]


class SimplySupportedBeam(BaseBeamType):
    id = 1
    characteristic_function = sin(mu_m)

    def mu_m_initial_guess(self, mode):
        return mode * pi

    def Y_m(self, mode):
        return sin(mu_m*y/a)


class ClampedClampedBeam(BaseBeamType):
    id = 2
    characteristic_function = cos(mu_m)*cosh(mu_m) - 1

    def mu_m_initial_guess(self, mode):
        return (2*mode + 1) * pi/2

    def Y_m(self, mode):
        return 1/(cos(mu_m)-cosh(mu_m)) * (
              sin (mu_m*y/a)*cos (mu_m) - sin (mu_m*y/a)*cosh(mu_m)
            - sinh(mu_m*y/a)*cos (mu_m) + sinh(mu_m*y/a)*cosh(mu_m)
            - cos (mu_m*y/a)*sin (mu_m) + cosh(mu_m*y/a)*sin (mu_m)
            + cos (mu_m*y/a)*sinh(mu_m) - cosh(mu_m*y/a)*sinh(mu_m)
        )


class ClampedFreeBeam(BaseBeamType):
    id = 3
    characteristic_function = cos(mu_m)*cosh(mu_m) + 1

    def mu_m_initial_guess(self, mode):
        return (2*mode - 1) * pi/2

    def Y_m(self, mode):
        return 1/(cos(mu_m)+cosh(mu_m)) * (
              sin (mu_m*y/a)*cos (mu_m) + sin (mu_m*y/a)*cosh(mu_m)
            - sinh(mu_m*y/a)*cos (mu_m) - sinh(mu_m*y/a)*cosh(mu_m)
            - cos (mu_m*y/a)*sin (mu_m) + cosh(mu_m*y/a)*sin (mu_m)
            - cos (mu_m*y/a)*sinh(mu_m) + cosh(mu_m*y/a)*sinh(mu_m)
        )
```

```
class ClampedSimplySupportedBeam(BaseBeamType):
    id = 4
    characteristic_function = tan(mu_m) - tanh(mu_m)

    def mu_m_initial_guess(self, mode):
        return (4*mode + 1) * pi/4

    def Y_m(self, mode):
        return 1/sinh(mu_m) * (sin(mu_m*y/a)*sinh(mu_m) - sinh(mu_m*y/a)*sin(mu_m))


class SimplySupportedFreeBeam(ClampedSimplySupportedBeam):
    id = 5

    dont_improve_mu_m_for_modes = (1,)

    def mu_m_initial_guess(self, mode):
        # Special case for this mode
        if mode == 1:
            return Float(1)

        return super(SimplySupportedFreeBeam, self).mu_m_initial_guess(mode-1)

    def Y_m(self, mode):
        # Special case for this mode
        if mode == 1:
            return y/a

        return 1/sinh(mu_m) * (sin(mu_m*y/a)*sinh(mu_m) + sinh(mu_m*y/a)*sin(mu_m))


class FreeFreeBeam(ClampedClampedBeam):
    id = 6

    dont_improve_mu_m_for_modes = (1, 2)

    def mu_m_initial_guess(self, mode):
        # Special case for these modes
        if mode == 1:
            return Float(0)
        elif mode == 2:
            return Float(1)

        return super(FreeFreeBeam, self).mu_m_initial_guess(mode-2)

    def Y_m(self, mode):
        # Special case for these modes
        if mode == 1:
            return Float(1)
        elif mode == 2:
            return 1 - 2*y/a

        return 1/(cos(mu_m)+cosh(mu_m)) * (
              sin (mu_m*y/a)*cos (mu_m) + sin (mu_m*y/a)*cosh(mu_m)
            - sinh(mu_m*y/a)*cos (mu_m) - sinh(mu_m*y/a)*cosh(mu_m)
```

```
            - cos (mu_m*y/a)*sin (mu_m) + cosh(mu_m*y/a)*sin (mu_m)
            - cos (mu_m*y/a)*sinh(mu_m) + cosh(mu_m*y/a)*sinh(mu_m)
        )
```

Listing 4.7: Izvod iz Python modula `beam_integrals.beam_types`, puna verzija dostupna na [137]

Klasa `BaseBeamType` predstavlja baznu klasu podsistema programskih dodataka za granične uslove. Stoga će sve njene klase naslednice biti automatski registrovane kao odgovarajući programski dodaci, tj. biće automatski vidljivi drugim modulima kao zasebni granični uslovi. Klasa `BaseBeamType` nasleđuje predstavljenu `FriendlyNameFromClassMixin` *mixin* klasu (sekcija 4.1) i sadrži sledeće atribute, svojstva i metode:

1. `id` je atribut koji sadrži jednoznačnu numeričku identifikaciju klase unutar podsistema programskih dodataka. Njegova vrednost odgovara vrednosti oznake LJ, navedene uz svaki granični uslov (sekcija 1.2.1).

2. `characteristic_function` je atribut koji sadrži simbolički izraz odgovarajuće karakteristične funkcije f, izvedene iz pripadajuće karakteristične jednačine na način koji je opisan u sekciji 2.1.

3. `characteristic_equation_str` je pomoćno svojstvo koje sadrži string reprezentaciju kanonske forme odgovarajuće karakteristične jednačine, a na osnovu vrednosti atributa `characteristic_function`.

4. `name` je pomoćno svojstvo, automatski ubačeno prilikom nasleđivanja `FriendlyNameFromClassMixin` *mixin* klase (sekcija 4.1), koje sadrži ljudski čitljivo ime objekta klase datog graničnog uslova.

5. `__str__` je specijalna Python metoda koja vraća string reprezentaciju objekta klase datog graničnog uslova. String reprezentacija formirana je na osnovu vrednosti svojstva `name` i atributa `id`.

6. `filename` je pomoćno Python svojstvo koje sadrži naziv datoteke za objekat klase datog graničnog uslova. Predloženi

naziv datoteke formiran je na osnovu vrednosti svojstva `name` i atributa `id`.

7. `mu_m_initial_guess` je metoda koja vraća simbolički izraz aproksimativno analitičkog rešenja karakteristične jednačine. Ova metoda prima parametar `mode` pošto se za granične uslove LJ $\in \{5, 6\}$ formulacija rešenja karakteristične jednačine menja u zavisnosti od traženog moda.

8. `dont_improve_mu_m_for_modes` je svojstvo koje sadrži spisak modova za koje ne treba vršiti numeričku optimizaciju korenova karakteristične jednačine (detaljnije u sekciji 4.3.3). Ovo svojstvo je potrebno usled raznorodne formulacije rešenja karakteristične jednačine za granične uslove LJ $\in \{5, 6\}$.

9. `mu_m_initial_search_width` je atribut koji sadrži inicijalnu širinu intervala oko aproksimativno analitičkog rešenja karakteristične jednačine, u kome će se vršiti numerička optimizacija korenova karakteristične jednačine (detaljnije u sekciji 4.3.3).

10. `mu_m_increase_search_width_by` je atribut koji sadrži faktor uvećanja širine intervala opisanog navedenim atributom `mu_m_initial_search_width` (detaljnije u sekciji 4.3.3). Ovaj atribut se progresivno primenjuje ukoliko se traženi koren ne nalazi u prethodno definisanom intervalu.

11. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$. Ova metoda prima parametar `mode` pošto se za granične uslove LJ $\in \{5, 6\}$ formulacija bazne funkcije $Y_m(y)$ menja u zavisnosti od traženog moda.

12. `Y_m_derivative_from_cache` je pomoćna metoda za simboličko određivanje i keširanje prvog i drugog izvoda bazne funkcije $Y_m(y)$. Ova metoda prima parametar `mode` pošto se za granične uslove LJ $\in \{5, 6\}$ formulacija bazne funkcije $Y_m(y)$ menja u zavisnosti od traženog moda.

Klasa `SimplySupportedBeam` predstavlja granični uslov gde su oba kraja slobodno oslonjena (LJ=1, opisan u sekciji 1.2.1.1). Klasa nasleđuje baznu klasu `BaseBeamType` i redefiniše sledeće atribute i metode:

1. `id` je atribut koji sadrži numeričku vrednost oznake LJ

2. `characteristic_function` je atribut koji sadrži simbolički izraz karakteristične funkcije f, izvedene iz karakteristične jednačine 1.7

3. `mu_m_initial_guess` je metoda koja vraća simbolički izraz analitičkog rešenja karakteristične jednačine, datog u jednačini 1.8

4. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$, date u jednačini 1.6

Klasa `ClampedClampedBeam` predstavlja granični uslov gde su oba kraja ukliještena (LJ=2, opisan u sekciji 1.2.1.2). Klasa nasleđuje baznu klasu `BaseBeamType` i redefiniše sledeće atribute i metode:

1. `id` je atribut koji sadrži numeričku vrednost oznake LJ

2. `characteristic_function` je atribut koji sadrži simbolički izraz karakteristične funkcije f, izvedene iz karakteristične jednačine 1.10

3. `mu_m_initial_guess` je metoda koja vraća simbolički izraz aproksimativno analitičkog rešenja karakteristične jednačine, datog u jednačini 1.12

4. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$, date u jednačini 1.9

Klasa `ClampedFreeBeam` predstavlja granični uslov gde je jedan kraj uklješten a drugi slobodan (LJ=3, opisan u sekciji 1.2.1.3). Klasa nasleđuje baznu klasu `BaseBeamType` i redefiniše sledeće atribute i metode:

1. `id` je atribut koji sadrži numeričku vrednost oznake LJ

2. `characteristic_function` je atribut koji sadrži simbolički izraz karakteristične funkcije f, izvedene iz karakteristične jednačine 1.15

3. `mu_m_initial_guess` je metoda koja vraća simbolički izraz aproksimativno analitičkog rešenja karakteristične jednačine, datog u jednačini 1.16

4. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$, date u jednačini 1.14

Klasa `ClampedSimplySupportedBeam` predstavlja granični uslov gde je jedan kraj slobodno oslonjen a drugi uklješten (LJ=4, opisan u sekciji 1.2.1.4). Klasa nasleđuje baznu klasu `BaseBeamType` i redefiniše sledeće atribute i metode:

1. `id` je atribut koji sadrži numeričku vrednost oznake LJ

2. `characteristic_function` je atribut koji sadrži simbolički izraz karakteristične funkcije f, izvedene iz karakteristične jednačine 1.20

3. `mu_m_initial_guess` je metoda koja vraća simbolički izraz aproksimativno analitičkog rešenja karakteristične jednačine, datog u jednačini 1.21

4. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$, date u jednačini 1.19

Klasa `SimplySupportedFreeBeam` predstavlja granični uslov gde je jedan kraj slobodno oslonjen a drugi slobodan (LJ=5, opisan u sekciji 1.2.1.5). Klasa nasleđuje `ClampedSimplySupported Beam` klasu i redefiniše sledeće atribute i metode:

1. `id` je atribut koji sadrži numeričku vrednost oznake LJ

2. `mu_m_initial_guess` je metoda koja vraća simbolički izraz aproksimativno analitičkog rešenja karakteristične jednačine, datog u jednačini 1.26

3. `dont_improve_mu_m_for_modes` je svojstvo koje sadrži spisak modova za koje ne treba vršiti numeričku optimizaciju korenova karakteristične jednačine, izvedeno iz jednačine 1.26

4. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$, date u jednačini 1.24

Klasa `FreeFreeBeam` predstavlja granični uslov gde su oba kraja slobodna (LJ=6, opisan u sekciji 1.2.1.6). Klasa nasleđuje `ClampedClampedBeam` klasu i redefiniše sledeće atribute i metode:

1. `id` je atribut koji sadrži numeričku vrednost oznake LJ

2. `mu_m_initial_guess` je metoda koja vraća simbolički izraz aproksimativno analitičkog rešenja karakteristične jednačine, datog u jednačini 1.31

3. `dont_improve_mu_m_for_modes` je svojstvo koje sadrži spisak modova za koje ne treba vršiti numeričku optimizaciju korenova karakteristične jednačine, izvedeno iz jednačine 1.31

4. `Y_m` je metoda koja vraća simbolički izraz bazne funkcije $Y_m(y)$, date u jednačini 1.29

4.3.3 Modul `beam_integrals.characteristic_equation_solvers`

Modul `beam_integrals.characteristic_equation_solvers` implementira proširiv podsistem numeričkih određivača korenova karakterističnih jednačina, uz prateću programsku podršku za jednostavnu i efikasnu upotrebu u drugim modulima. Ovaj modul suštinski predstavlja srž implementacije hibridne metode, predstavljene u sekciji 2.1.

Na listingu 4.8 dat je izvod iz izvornog kôda `beam_integrals.characteristic_equation_solvers` modula:

```
import multiprocessing
from operator import itemgetter
import os
import cPickle as pickle
from friendly_name_mixin import FriendlyNameFromClassMixin
from simple_plugins import AttrDict, PluginMount
from sympy import Abs, Float, nan, mpmath, sign
from . import PROJECT_SETTINGS_DIR, DEFAULT_MAX_MODE, DEFAULT_DECIMAL_PRECISION
from . import exceptions as exc
from .beam_types import BaseBeamType


class BaseRootfinder(FriendlyNameFromClassMixin):
    max_iterations = 100

    __metaclass__ = PluginMount

    class Meta:
        id_field = 'solver_name'
        id_field_coerce = str

    @property
    def solver_name(self):
        return self.name.lower().split()[0]

    def _get_f(self, beam_type, decimal_precision):
        def f(mu_m):
            return beam_type.characteristic_function.evalf(
                n=decimal_precision,
                subs={'mu_m': mu_m}
            )

        return f

    def find_root(self, beam_type, mode,
                  decimal_precision=DEFAULT_DECIMAL_PRECISION, **kwargs):
        if mode in beam_type.dont_improve_mu_m_for_modes:
            mu_m = beam_type.mu_m_initial_guess(mode).evalf(n=decimal_precision)
            mu_m_error = nan # characteristic equation not defined for this `mode`
        else:
```

```
            mu_m = self.improve_mu_m(beam_type, mode, decimal_precision, **kwargs)
            f = self._get_f(beam_type, decimal_precision)
            mu_m_error = Abs(f(mu_m))

        return mu_m, mu_m_error

    def improve_mu_m(self, beam_type, mode,
                     decimal_precision=DEFAULT_DECIMAL_PRECISION, **kwargs):
        f = self._get_f(beam_type, decimal_precision)

        with mpmath.workdps(decimal_precision):
            # If not converted to `sympy.Float` precision will be lost after the
            # original `mpmath` context is restored
            return Float(
                mpmath.findroot(
                    f=f,
                    x0=self.x0(beam_type, mode, decimal_precision),
                    solver=self.solver_name,
                    maxsteps=self.max_iterations,
                    verify=False,
                    **kwargs
                ),
                decimal_precision
            )

    def x0(self, beam_type, mode, decimal_precision=DEFAULT_DECIMAL_PRECISION):
        raise NotImplementedError

    def __call__(self, beam_type, mode,
                 decimal_precision=DEFAULT_DECIMAL_PRECISION, **kwargs):
        return self.find_root(beam_type, mode, decimal_precision, **kwargs)


class BaseIntervalBasedRootfinder(BaseRootfinder):
    def x0(self, beam_type, mode, decimal_precision=DEFAULT_DECIMAL_PRECISION):
        """Guesses the optimal `mu_m` search interval"""
        if mode in beam_type.dont_improve_mu_m_for_modes:
            raise exc.UndefinedRootError(
                "%s: root is undefined for mode = %d" % (
                    beam_type, mode
            ))

        f = self._get_f(beam_type, decimal_precision)
        center = beam_type.mu_m_initial_guess(mode).evalf(n=decimal_precision)
        search_width = beam_type.mu_m_initial_search_width

        while True:
            a = (center - search_width/2.).evalf(n=decimal_precision)
            b = (center + search_width/2.).evalf(n=decimal_precision)
            f_a = f(a)
            f_b = f(b)

            if f_a == f_b == 0.:
                raise exc.MultipleRootsError(
                    "%s: Found multiple roots in area [%s, %s] for mode = %d" % (
                        beam_type, a, b, mode
```

```
                ))

            if sign(f_a) == -sign(f_b):
                break

            search_width *= beam_type.mu_m_increase_search_width_by

        return a, b


class BaseStartingPointBasedRootfinder(BaseRootfinder):
    def x0(self, beam_type, mode, decimal_precision=DEFAULT_DECIMAL_PRECISION):
        return beam_type.mu_m_initial_guess(mode).evalf(n=decimal_precision)


class AndersonRootfinder(BaseIntervalBasedRootfinder):
    pass


# Disabled, as it's too slow and not a top rootfinder
#class BisectRootfinder(BaseIntervalBasedRootfinder):
#    pass


class IllinoisRootfinder(BaseIntervalBasedRootfinder):
    pass


class PegasusRootfinder(BaseIntervalBasedRootfinder):
    pass


# Disabled, as it's too slow and not a top rootfinder
#class RidderRootfinder(BaseIntervalBasedRootfinder):
#    pass


class SecantRootfinder(BaseStartingPointBasedRootfinder):
    pass


def find_root_candidates(beam_type, mode,
                         decimal_precision=DEFAULT_DECIMAL_PRECISION, **kwargs):
    return dict(
        (name, rootfinder(beam_type, mode, decimal_precision, **kwargs))
        for name, rootfinder in BaseRootfinder.plugins.id_to_instance.items()
    )

def find_best_root(beam_type, mode, decimal_precision=DEFAULT_DECIMAL_PRECISION,
                   include_error=False, use_cache=True, cache_instance=None,
                   **kwargs):
    if use_cache:
        cache_instance = cache_instance or best_roots_cache
        result = cache_instance.get(beam_type, mode, decimal_precision)
    else:
        result = min(
```

```
            find_root_candidates(
                beam_type, mode, decimal_precision, **kwargs
            ).values(),
            key=itemgetter(1)
        )

    return result if include_error else result[0]

def _init_pool(*data):
    global _pool_data

    data_keys = 'beam_type, decimal_precision, include_error, kwargs'.split(', ')
    _pool_data = AttrDict(zip(data_keys, data))

def _worker(mode):
    c = _pool_data
    return find_best_root(
        c.beam_type, mode, c.decimal_precision, include_error=c.include_error,
        use_cache=False, **c.kwargs
    )

def find_best_roots(beam_type, max_mode=DEFAULT_MAX_MODE,
    decimal_precision=DEFAULT_DECIMAL_PRECISION, include_error=True, **kwargs):
    pool = multiprocessing.Pool(
        initializer=_init_pool,
        initargs=(beam_type, decimal_precision, include_error, kwargs)
    )
    results = pool.map(func=_worker, iterable=range(1, max_mode+1))

    pool.close()
    pool.join()
    return results


class BestRootsCache(object):
    disk_cache_dir = os.path.join(
        PROJECT_SETTINGS_DIR, 'cache', 'characteristic-equations'
    )
    _ram_cache = {}

    def __init__(self, disk_cache_dir=None):
        self.disk_cache_dir = disk_cache_dir or self.disk_cache_dir
        if not os.path.exists(self.disk_cache_dir):
            os.makedirs(self.disk_cache_dir)

    def disk_cache_filename(self, decimal_precision=DEFAULT_DECIMAL_PRECISION):
        return os.path.join(
            self.disk_cache_dir,
            "best-roots.decimal-precision=%d.pickle" % decimal_precision
        )

    def regenerate(self, max_mode=DEFAULT_MAX_MODE,
                   decimal_precision=DEFAULT_DECIMAL_PRECISION, **kwargs):
        results = dict(
            (
                beam_type.id,
```

```
                find_best_roots(beam_type, max_mode, decimal_precision, **kwargs)
            )
            for beam_type in BaseBeamType.plugins.instances
        )

        with open(self.disk_cache_filename(decimal_precision), 'wb') as f:
            pickle.dump(results, f, protocol=pickle.HIGHEST_PROTOCOL)

        if decimal_precision in self._ram_cache: # Clear out the old RAM cache
            del self._ram_cache[decimal_precision]

    def _load_disk_cache(self, decimal_precision=DEFAULT_DECIMAL_PRECISION):
        filename = self.disk_cache_filename(decimal_precision)
        try:
            with open(filename, 'rb') as f:
                self._ram_cache[decimal_precision] = pickle.load(f)
        except (IOError, pickle.UnpicklingError), e:
            raise exc.UnableToLoadBestRootsCacheError(
                "Unable to load cache from '%s': %s. You'll need to "\
                "regenerate the cache by calling the console app "\
                "'beam_integrals' or by using 'beam_integrals."\
                "characteristic_equation_solvers.best_roots_cache.regenerate' "\
                "Python API call." %
                (filename, e)
            )

    def get(self, beam_type, mode, decimal_precision=DEFAULT_DECIMAL_PRECISION):
        if decimal_precision not in self._ram_cache:
            self._load_disk_cache(decimal_precision)

        try:
            if mode <= 0:
                raise exc.InvalidModeError(
                    "Mode has to be a positive number (%s given)" % mode
                )

            # Mode list is 0 indexed
            return self._ram_cache[decimal_precision][beam_type.id][mode-1]
        except KeyError:
            # Old version cache, doesn't support this beam type
            raise exc.BeamTypeNotFoundInCacheError(
                "No best root found in cache for %s. You'll need to"\
                "regenerate the cache by calling the console app "\
                "'beam_integrals' or by using 'beam_integrals."\
                "characteristic_equation_solvers.best_roots_cache.regenerate' "\
                "Python API call." % beam_type
            )
        except IndexError:
            raise exc.ModeNotFoundInCacheError(
                "No best root found in cache for %s when mode = %d. You'll "\
                "need to regenerate the cache by calling the console app "\
                "'beam_integrals' or by using 'beam_integrals."\
                "characteristic_equation_solvers.best_roots_cache.regenerate' "\
                "Python API call. Please remember to increase max_mode to the "\
                "desired limit." %
                (beam_type, mode)
```

```
        )


best_roots_cache = BestRootsCache()
```

Listing 4.8: Izvod iz `beam_integrals.characteristic_equation_solvers` Python modula, puna verzija dostupna na [137]

U projektu `beam_integrals` svaka funkcija ili metoda vezana za numeričke proračune prima dodatni, opcioni, parametar `decimal_precision` kojim se na jednostavan način može dinamički menjati radna preciznost između API poziva, bez kompromitovanja originalne preciznosti prethodnih rezultata. Ovo se postiže korišćenjem `sympy.Float` klase [192] koja omogućava ugradnju informacije o radnoj preciznosti izvršenih proračuna uz svaki pojedinačni rezultat. Alternativa bi bila korišćenje mehanizma mpmath konteksta [193–195], ali bi tada originalna preciznost proračuna i rezultata bila izgubljena odmah po napuštanju trenutno aktuelnog mpmath konteksta ili čak po izvršenom API pozivu koji menja radnu preciznost.

Klasa `BaseRootfinder` predstavlja baznu klasu podsistema programskih dodataka za numeričke određivače korenova karakterističnih jednačina. Stoga će sve njene klase naslednice biti automatski registrovane kao odgovarajući programski dodaci, tj. biće automatski vidljivi ovom i drugim modulima kao zasebni numerički određivači. Klasa `BaseRootfinder` nasleđuje predstavljenu `FriendlyNameFromClassMixin` *mixin* klasu (sekcija 4.1) i sadrži sledeće atribute, svojstva i metode:

1. `max_iterations` je atribut koji sadrži maksimalno dozvoljen broj iteracija koje dati numerički određivač korenova može izvršiti pre odustajanja.

2. `name` je pomoćno svojstvo, automatski ubačeno prilikom nasleđivanja `FriendlyNameFromClassMixin` *mixin* klase (sekcija 4.1), koje sadrži ljudski čitljivo ime objekta klase datog numeričkog određivača.

3. `solver_name` je pomoćno Python svojstvo koje sadrži formalni naziv datog numeričkog određivača, u skladu sa terminologijom mpmath programske biblioteke [185,186]. Ovo

svojstvo, formirano na osnovu vrednosti svojstva `name`, koristi se za jednoznačnu identifikaciju klase unutar podsistema programskih dodataka i određuje koji će se od mpmath numeričkih određivača korenova [196] koristiti.

4. `_get_f` je pomoćna metoda koja na osnovu karakteristične funkcije datog graničnog uslova i tražene radne preciznosti formira i vraća unutrašnju funkciju `f`, koja zavisi samo od vrednosti parametra `mu_m`. Formiranje unutrašnje funkcije `f` potrebno je zbog specifičnosti načina rada mpmath numeričkih određivača korenova [196].

5. `x0` je pomoćna metoda koja vraća ili numeričku vrednost aproksimativno analitičkog rešenja karakteristične jednačine ili interval oko aproksimativno analitičkog rešenja u kome će se vršiti numerička optimizacija korenova karakteristične jednačine. Tip povratne vrednosti zavisi od vrste zadatog mpmath numeričkog određivača korena (detaljnije na strani 162).

6. `improve_mu_m` je metoda koja vraća vrednost numeričko optimizovanog korena, dobijenog korišćenjem zadatog mpmath numeričkog određivača korena (na osnovu svojstva `solver_name`). Za početnu pretpostavku rešenja koristi se povratna vrednost metode `x0`.

7. `find_root` je metoda koja odlučuje da li se uopšte sme vršiti numerička optimizacija korenova karakteristične jednačine, a na osnovu vrednosti `dont_improve_mu_m_for_modes` svojstva klase `BaseBeamType` i njenih naslednica (detaljnije na strani 150).

   Ova metoda vraća uređeni par vrednosti *(koren, greška)*. Ukoliko je numerička optimizacija dozvoljena uređeni par *(koren, greška)* će činiti vrednosti numeričko optimizovanog korena i njegove greške. U suprotnom *koren* će biti numerička vrednost aproksimativno analitičkog rešenja karakteristične jednačine, dok će *greška* tada biti `sympy.nan`, tj. nedefinisana (eng. *Not A Number* [26, 27]).

8. `__call__` je specijalna Python metoda koja datom objektu pruža mogućnost da bude pozivan kao funkcija, tzv. *callable object* [197, 198] u terminologiji programskog jezika Python. U ovom slučaju prosto se prosleđuje poziv metodi `find_root` i vraća njena povratna vrednost. Primera radi, ova metoda pruža mogućnost da se poziv:
   `rootfinder.find_root(beam_type, mode, precision)`
   može izvršiti i kroz sam objekat:
   `rootfinder(beam_type, mode, precision)`

Klasa `BaseStartingPointBasedRootfinder` predstavlja baznu klasu za mpmath numeričke određivače korenova koji koriste jednu tačku [196] kao početnu pretpostavku približne lokacije traženog korena. Klasa nasleđuje baznu klasu `BaseRootfinder` i redefiniše pomoćnu metodu `x0`, koja sada vraća numeričku vrednost aproksimativno analitičkog rešenja karakteristične jednačine.

Klasa `BaseIntervalBasedRootfinder` predstavlja baznu klasu za mpmath numeričke određivače korenova koji traže koren u okviru zadatog intervala [196]. Klasa nasleđuje baznu klasu `BaseRootfinder` i redefiniše pomoćnu metodu `x0`, koja sada vraća interval oko aproksimativno analitičkog rešenja u kome će se vršiti numerička optimizacija traženog korena karakteristične jednačine. Ovi numerički određivači korenova očekuju da traženi koren μ_m pripada zadatom intervalu:

$$\mu_m \in [a, b] \tag{4.1}$$

pri čemu važi:

$$\begin{aligned} a &= c - \frac{s}{2} \\ b &= c + \frac{s}{2} \end{aligned} \tag{4.2}$$

gde je:

- c centar intervala, postavljen na numeričku vrednost aproksimativno analitičkog rešenja karakteristične jednačine
- s automatski određena optimalna širina intervala

Klasa `BaseIntervalBasedRootfinder` i njene naslednice koriste svoju pomoćnu metodu `_get_f` i njenu unutrašnju funkciju `f` da bi odredile optimalnu širinu intervala s, koja zadovoljava sledeći kriterijum:

$$\begin{aligned} \operatorname{sgn}(f(a_x)) &= -\operatorname{sgn}(f(b_x)) \neq 0 \\ a_x &= c - \frac{s_x}{2} \\ b_x &= c + \frac{s_x}{2} \end{aligned} \tag{4.3}$$

pri čemu se s_x određuje iterativnim postupkom:

$$s_x = \begin{cases} s_0 & , x = 0 \\ \beta s_{x-1} & , x > 0 \end{cases} \tag{4.4}$$

gde je:

- s_0 inicijalna širina intervala, postavljena na vrednost svojstva `mu_m_initial_search_width` klase `BaseBeamType`
- β faktor uvećanja širine intervala, postavljena na vrednost svojstva `mu_m_increase_search_width_by` klase `BaseBeamType`

Klasa `AndersonRootfinder` nasleđuje `BaseIntervalBasedRootfinder` klasu i potom se putem metaprogramiranja uvezuje na mpmath programsku biblioteku [185,186] i njen `anderson` numerički određivač korenova [196], koji je baziran na Anderson–Björck metodu [199,200].

Klasa `BisectRootfinder` nasleđuje klasu `BaseIntervalBasedRootfinder` i potom se putem metaprogramiranja uvezuje na mpmath programsku biblioteku [185,186] i njen `bisect` numerički određivač korenova [196], koji je baziran na metodu bisekcije [201]. Projekat `beam_integrals` trenutno ne koristi ovaj metod (zakomentarisan je izvorni kôd) pošto se tokom istraživanja metod pokazao veoma sporim i neefikasnim u rešavanju karakterističnih jednačina. Dodatno, na listingu 4.9 je za svaku zasebnu kombinaciju graničnog uslova i moda analiziran učinak ovog metoda u turnirski baziranom takmičenju između numeričkih određivača korenova:

```
>>> from collections import defaultdict
>>> from beam_integrals import DEFAULT_DECIMAL_PRECISION, DEFAULT_MAX_MODE
>>> from beam_integrals import characteristic_equation_solvers as ces
>>> from beam_integrals.beam_types import BaseBeamType


# Reactivate the bisect rootfinder, to showcase it's never the best rootfinder
>>> class BisectRootfinder(ces.BaseIntervalBasedRootfinder):
...     pass


# Reactivate the ridder rootfinder, to showcase it's never the best rootfinder
>>> class RidderRootfinder(ces.BaseIntervalBasedRootfinder):
...     pass


>>> def get_best_rootfinder(beam_type, mode,
...                         decimal_precision=DEFAULT_DECIMAL_PRECISION,
...                         find_best=True):
...     r = sorted(
...         (
...             (error, name)
...             for name, (root, error) in ces.find_root_candidates(
...                 beam_type, mode, decimal_precision=decimal_precision
...             ).items()
...         ),
...         reverse=not find_best
...     )
...
...     return 'multiple rootfinders' if r[0][0] == r[1][0] else r[0][1]

>>> def get_rootfinder_stats(beam_type, max_mode=DEFAULT_MAX_MODE,
...                          decimal_precision=DEFAULT_DECIMAL_PRECISION,
...                          find_best=True):
...     stats = defaultdict(int)
...     for mode in range(1, max_mode+1):
...         key = get_best_rootfinder(
...             beam_type, mode,
...             decimal_precision=decimal_precision, find_best=find_best
...         )
...         stats[key] += 1
...
...     return stats

>>> def analyze_rootfinder_stats(beam_type, max_mode=DEFAULT_MAX_MODE,
...                              decimal_precision=DEFAULT_DECIMAL_PRECISION,
...                              find_best=True):
...     stats = get_rootfinder_stats(
...         beam_type, max_mode=max_mode,
...         decimal_precision=decimal_precision, find_best=find_best
...     )
...     sorted_stats = sorted(((v, k) for k, v in stats.items()), reverse=True)
...     for count, name in sorted_stats:
...         print "\t\t%3d%% %s" % (100.0 * count/max_mode, name)
```

```
>>> for beam_type in BaseBeamType.plugins.instances_sorted_by_id:
...     print "%s:" % beam_type
...     for find_best in (True, False):
...         print "\t%s rootfinders:" % ('Best' if find_best else 'Worst')
...         analyze_rootfinder_stats(beam_type, find_best=find_best)
Simply Supported Beam (id=1):
        Best rootfinders:
                100% multiple rootfinders
        Worst rootfinders:
                 91% bisect
                  9% ridder
Clamped Clamped Beam (id=2):
        Best rootfinders:
                100% multiple rootfinders
        Worst rootfinders:
                 95% bisect
                  5% ridder
Clamped Free Beam (id=3):
        Best rootfinders:
                100% multiple rootfinders
        Worst rootfinders:
                 94% bisect
                  6% ridder
Clamped Simply Supported Beam (id=4):
        Best rootfinders:
                100% multiple rootfinders
        Worst rootfinders:
                 99% bisect
                  1% ridder
Simply Supported Free Beam (id=5):
        Best rootfinders:
                100% multiple rootfinders
        Worst rootfinders:
                 98% bisect
                  1% ridder
                  1% multiple rootfinders
Free Free Beam (id=6):
        Best rootfinders:
                100% multiple rootfinders
        Worst rootfinders:
                 93% bisect
                  5% ridder
                  2% multiple rootfinders
```

Listing 4.9: Analiza učinka pojedinačnih metoda u turnirski baziranom takmičenju između numeričkih određivača korenova

na osnovu čega je utvrđeno da njegova deaktivacija neće uticati na opštu tačnost korenova jer do sada nije pobedio ni na jednom turniru.

Klasa `IllinoisRootfinder` nasleđuje `BaseIntervalBased Rootfinder` klasu i potom se putem metaprogramiranja uvezuje na mpmath programsku biblioteku [185,186] i njen `illinois` nu-

merički određivač korenova [196], koji je baziran na *Illinois* metodu [200, 202].

Klasa `PegasusRootfinder` nasleđuje `BaseIntervalBased Rootfinder` klasu i potom se putem metaprogramiranja uvezuje na mpmath programsku biblioteku [185, 186] i njen `pegasus` numerički određivač korenova [196], koji je baziran na *Pegasus* metodu [200, 203].

Klasa `RidderRootfinder` nasleđuje `BaseIntervalBased Rootfinder` klasu i potom se putem metaprogramiranja uvezuje na mpmath programsku biblioteku [185, 186] i njen `ridder` numerički određivač korenova [196], koji je baziran na *Ridders* metodu [204]. Projekat `beam_integrals` trenutno ne koristi ovaj metod (zakomentarisan je izvorni kôd) pošto se tokom istraživanja metod pokazao veoma sporim i neefikasnim u rešavanju karakterističnih jednačina. Dodatno, na listingu 4.9 je za svaku zasebnu kombinaciju graničnog uslova i moda analiziran učinak ovog metoda u turnirski baziranom takmičenju između numeričkih određivača korenova, na osnovu čega je utvrđeno da njegova deaktivacija neće uticati na opštu tačnost korenova jer do sada nije pobedio ni na jednom turniru.

Klasa `SecantRootfinder` nasleđuje `BaseStartingPointBased Rootfinder` klasu i potom se putem metaprogramiranja uvezuje na mpmath programsku biblioteku [185, 186] i njen `secant` numerički određivač korenova [196], koji je baziran na metodu sečice [201].

`find_root_candidates` je funkcija koja organizuje turnirski bazirano takmičenju između svih numeričkih određivača korenova karakterističnih jednačina, koji su registrovani unutar podsistema programskih dodataka. Ova funkcija vraća sve potencijalne kandidate za traženi koren u obliku mapiranja sa *ključa* na *vrednost*, gde je:

- *ključ* vrednost svojstva `solver_name` odgovarajuće klase numeričkog određivača korenova
- *vrednost* uređeni par vrednosti *(koren, greška)*, dobijen pozivom metode `find_root` odgovarajuće klase numeričkog određivača korenova

`find_best_root` je funkcija koja ima dva režima rada, u zavisnosti od vrednosti parametra `use_cache`:

1. direktno sračunavanje optimalne vrednosti traženog korena, korišćenjem funkcije `find_root_candidates`

2. upotreba unapred sračunatih, keširanih, vrednosti korenova karakteristične jednačine (kroz postupak opisan u daljem tekstu)

Ova funkcija, u zavisnosti od vrednosti parametra `include_error`, vraća:

- ili uređeni par vrednosti *(koren, greška)*

- ili samo vrednost traženog korena

`find_best_roots` je pomoćna funkcija koja sprovodi turnirski bazirano takmičenje između svih numeričkih određivača korenova karakterističnih jednačina za zadati granični uslov u traženoj decimalnoj preciznosti, a za sve modove od prvog do zadatog maksimalnog moda. Samo sračunavanje vrednosti korenova je značajno ubrzano kroz lokalni paralelizam upotrebom `multiprocessing` modula [205] Python standardne biblioteke, uz pomoć funkcija `_init_pool` i `_worker`. Ova funkcija, u zavisnosti od vrednosti parametra `include_error`, vraća spisak sačinjen od:

- ili uređenih parova vrednosti *(koren, greška)*

- ili samo vrednosti korenova

Klasa `BestRootsCache` predstavlja osnovu mehanizma za transparentno keširanje i upotrebu unapred sračunatih vrednosti korenova karakterističnih jednačina. Postupak sračunavanja vrednosti korenova karakterističnih jednačina je veoma zahtevan po pitanju utroška računarskih resursa i vremena proračuna. Keširanjem unapred sračunatih vrednosti korenova, za sve kombinacije modova i graničnih uslova, ubuduće se izbegavaju ponovni i veoma zahtevni proračuni. Ostvarene uštede dodatno dobijaju na značaju tokom izračunavanja određenih integrala, pošto se sračunate vrednosti korena koriste u podintegralnoj funkciji opšteg

oblika integrala, koja se potom često poziva tokom procesa numeričke integracije. Sâm keš je hibridnog tipa i sastoji se iz dva skrivena sloja:

1. disk, na kome se trajno skladište sračunati korenovi
2. radna memorija, u koju se korenovi, po zahtevu, privremeno učitavaju sa diska

Ova klasa sadrži sledeće atribute i metode:

1. `disk_cache_dir` je atribut koji sadrži putanju direktorijuma na disku, u kome će biti trajno uskladišteni svi sračunati korenovi
2. `disk_cache_filename` je metoda koja vraća naziv datoteke na disku, u koju će biti trajno uskladišteni sračunati korenovi zadate decimalne preciznosti
3. `_ram_cache` je pomoćni atribut koji sadrži keš uskladišten u radnoj memoriji, u obliku mapiranja sa *ključ1* na *vrednost1*, gde je:
   - *ključ1* decimalna preciznost korišćena pri sračunavanju vrednosti korenova
   - *vrednost1* mapiranje sa *ključ2* na *vrednost2*, gde je:
     - *ključ2* vrednost atributa `id` odgovarajuće klase graničnog uslova
     - *vrednost2* spisak indeksiran po modu, sačinjen od uređenih parova vrednosti *(koren, greška)*
4. `regenerate` je metoda koja regeneriše keš na disku i u radnoj memoriji upotrebom `find_best_roots` pomoćne funkcije, za zadatu decimalnu preciznost i sve modove od prvog do zadatog maksimalnog moda
5. `_load_disk_cache` je pomoćna metoda koja pokušava da učita keš sa diska u radnu memoriju, za zadatu decimalnu preciznost. Ukoliko traženi keš na disku ne postoji doći će do greške, uz preporuku da se regeneriše keš.

6. `get` je metoda koja vraća keširanu vrednost korena za zadati granični uslov, mod i decimalnu preciznost.

   Ova metoda vrednost traženog korena prvo pokušava da dobavi iz keša radne memorije. Ukoliko tamo nije pronađena pokušaće da je dobavi iz keša na disku. Ukoliko ni tamo nije pronađena doći će do greške, uz preporuku da se regeneriše keš.

`best_roots_cache` je globalno dostupni objekat i instanca klase `BestRootsCache`, koji olakšava upotrebu `find_best_root` funkcije i razvoj podsistema za automatsku verifikaciju projekta `beam_integrals`.

4.3.4 Modul `beam_integrals.exceptions`

Modul `beam_integrals.exceptions` definiše sve izuzetke do kojih može doći prilikom upotrebe projekta `beam_integrals`.

Na listingu 4.10 dat je izvod iz izvornog kôda `beam_integrals.exceptions` modula:

```
class BeamTypesException(Exception):
    """A generic exception for all others to extend"""


class InvalidModeError(BeamTypesException):
    """The given value isn't a valid mode"""


class RootfinderError(BeamTypesException):
    """Exception raised when something causes a rootfinder error"""


class UndefinedRootError(RootfinderError):
    """Root is undefined for the given mode"""


class MultipleRootsError(RootfinderError):
    """Multiple roots found while guessing the optimal `mu_m` search interval"""


class BestRootsCacheError(RootfinderError):
    """Exception raised when something causes a best roots cache error"""


class UnableToLoadBestRootsCacheError(BestRootsCacheError):
    """Unable to load the best roots cache"""
```

```
class BeamTypeNotFoundInCacheError(BestRootsCacheError):
    """Given beam type not found in the best roots cache"""


class ModeNotFoundInCacheError(BestRootsCacheError):
    """Given mode not found in the best roots cache"""


class ShellCommandError(Exception):
    """Exception raised when a shell command causes an error"""
```

Listing 4.10: Izvod iz Python modula `beam_integrals.exceptions`, puna verzija dostupna na [137]

Klasa izuzetka `BeamTypesException` predstavlja baznu klasu izuzetaka do kojih može doći prilikom upotrebe `beam_integrals` projekta. Svi dolenavedeni izuzeci (osim `ShellCommandError`) su organizovani u hijerarhiju, kroz nasleđivanje zajedničkog pretka klase `BeamTypesException`.

Klasa izuzetka `InvalidModeError` nasleđuje klasu `BeamTypesException` i predstavlja izuzetak do koga dolazi ako zadati mod nije validan (npr. negativna vrednost).

Klasa izuzetka `RootfinderError` nasleđuje klasu `BeamTypesException` i predstavlja izuzetak do koga dolazi ako, iz nekog razloga, nastane problem prilikom numeričkog određivanja korenova karakteristične jednačine. Klase naslednice bliže opisuju uzrok nastalog problema.

Klasa izuzetka `UndefinedRootError` nasleđuje `RootfinderError` klasu i predstavlja izuzetak do koga dolazi ako numerički određivač ustanovi da je numerička optimizacija korena nemoguća za traženu kombinaciju moda i graničnog uslova, npr. mode $\in \{1, 2\}$ za granični uslov gde su oba kraja slobodna (LJ=6, opisan u sekciji 1.2.1.6).

Klasa izuzetka `MultipleRootsError` nasleđuje `RootfinderError` klasu i predstavlja izuzetak do koga dolazi ako numerički određivač prilikom proširivanja intervala oko aproksimativno analitičkog rešenja karakteristične jednačine ustanovi postojanje više korenova u toj oblasti.

Klasa izuzetka `BestRootsCacheError` nasleđuje `RootfinderError` klasu i predstavlja izuzetak do koga dolazi ako, iz nekog razloga, nastane problem u korišćenju keša korenova karakteri-

stične jednačine. Klase naslednice bliže opisuju uzrok nastalog problema.

Klasa izuzetka `UnableToLoadBestRootsCacheError` nasleđuje klasu `BestRootsCacheError` i predstavlja izuzetak do koga dolazi ako, iz nekog razloga, nije moguće učitati traženu datoteku keš korenova karakteristične jednačine.

Klasa izuzetka `BeamTypeNotFoundInCacheError` nasleđuje klasu `BestRootsCacheError` i predstavlja izuzetak do koga dolazi ako traženi granični uslov, iz nekog razloga, nije pronađen u kešu korenova karakteristične jednačine.

Klasa izuzetka `ModeNotFoundInCacheError` nasleđuje klasu `BestRootsCacheError` i predstavlja izuzetak do koga dolazi ako traženi mod, iz nekog razloga, nije pronađen u kešu korenova karakteristične jednačine.

Klasa izuzetka `ShellCommandError` predstavlja izuzetak do koga dolazi ako, iz nekog razloga, nastane problem u korišćenju konzolne aplikacije pod nazivom `beam_integrals`, definisane u modulu `beam_integrals.shell` (sekcija 4.3.6).

4.3.5 Modul `beam_integrals.integrals`

Modul `beam_integrals.integrals` simbolički definiše sve integrale date u sekciji 1.2.2, uz prateću programsku podršku za jednostavnu i efikasnu upotrebu integrala i njihovu numeričku integraciju.

Na listingu 4.11 dat je izvod iz izvornog kôda `beam_integrals.integrals` modula:

```
import itertools
import re
from friendly_name_mixin import FriendlyNameFromClassMixin
from simple_plugins import PluginMount
from sympy import Float, mpmath
from . import DEFAULT_DECIMAL_PRECISION
from .characteristic_equation_solvers import find_best_root


class BaseIntegral(FriendlyNameFromClassMixin):
    _id_re = re.compile('I(\d+)')

    used_variables = ('m', 't', 'v', 'n')

    __metaclass__ = PluginMount
```

```
    def __str__(self):
        return self.name

    @property
    def id(self):
        if not hasattr(self, '_id'):
            self._id = int(self._id_re.match(self.name).group(1))

        return self._id

    @staticmethod
    def _contribute_to_plugins(_plugins):
        _plugins.child_id_to_parent_id = dict(
            (id, _plugins.class_to_id[cls.__base__])
            for cls, id in _plugins.class_to_id.items()
            if cls.__base__ in _plugins.classes
        )

    @classmethod
    def parent_id(cls):
        return cls.plugins.child_id_to_parent_id.get(cls.plugins.class_to_id[cls])

    @classmethod
    def has_parent(cls):
        return cls.parent_id() is not None

    def iterate_over_used_variables(self, max_mode, start_mode=1):
        modes = range(start_mode, max_mode+1)
        get_modes = lambda var: modes if var in self.used_variables else (None,)

        var_modes = (get_modes(var) for var in BaseIntegral.used_variables)
        return itertools.product(*var_modes)

    def cache_key(self, m, t, v, n, max_mode):
        d = locals()
        return sum(
            d[var] * max_mode**idx
            for idx, var in enumerate(self.used_variables)
        )

    def integrand(self, beam_type, m, t, v, n,
                  decimal_precision=DEFAULT_DECIMAL_PRECISION):
        def resolve_mu_m(func, *args, **kwargs):
            def wrapper(mode):
                return func(mode, *args, **kwargs).subs('mu_m',
                    find_best_root(beam_type, mode, decimal_precision)
                )

            return wrapper

        Y_m = resolve_mu_m(beam_type.Y_m)
        dY_m = resolve_mu_m(beam_type.Y_m_derivative_from_cache, order=1)
        ddY_m = resolve_mu_m(beam_type.Y_m_derivative_from_cache, order=2)

        return self._integrand(Y_m, dY_m, ddY_m, m, t, v, n)
```

```
    def _integrand(self, Y_m, dY_m, ddY_m, m, t, v, n):
        raise NotImplementedError

    def __call__(self, beam_type, m, t, v, n,
                 decimal_precision=DEFAULT_DECIMAL_PRECISION):
        return self.integrand(beam_type, m, t, v, n, decimal_precision)


class BaseIntegralWithSymetricVariables(BaseIntegral):
    def cache_key(self, m, t, v, n, max_mode):
        d = locals()
        values = sorted(d[var] for var in self.used_variables)
        return sum(
            val * max_mode**idx
            for idx, val in enumerate(values)
        )


class I1(BaseIntegralWithSymetricVariables):
    used_variables = ('m', 'n')

    def _integrand(self, Y_m, dY_m, ddY_m, m, t, v, n):
        return Y_m(m) * Y_m(n)

class I21(I1): pass


class I2(BaseIntegralWithSymetricVariables):
    used_variables = ('m', 'n')

    def _integrand(self, Y_m, dY_m, ddY_m, m, t, v, n):
        return dY_m(m) * dY_m(n)

class I4(I2): pass
class I6(I2): pass
class I8(I2): pass
class I25(I2): pass


class I3(BaseIntegral):
    used_variables = ('m', 'n')

    def _integrand(self, Y_m, dY_m, ddY_m, m, t, v, n):
        return ddY_m(m) * Y_m(n)

class I22(I3): pass


class I5(BaseIntegral):
    used_variables = ('m', 'n')

    def _integrand(self, Y_m, dY_m, ddY_m, m, t, v, n):
        return Y_m(m) * ddY_m(n)

class I23(I5): pass
```

```
class I7(BaseIntegralWithSymetricVariables):
    used_variables = ('m', 'n')

    def _integrand(self, Y_m, dY_m, ddY_m, m, t, v, n):
        return ddY_m(m) * ddY_m(n)

class I24(I7): pass


def integrate(integral, beam_type, a, m=None, t=None, v=None, n=None,
              decimal_precision=DEFAULT_DECIMAL_PRECISION, **kwargs):
    cached_subs = integral(beam_type, m, t, v, n, decimal_precision).subs('a', a)
    f = lambda y: cached_subs.evalf(n=decimal_precision, subs={'y': y})

    with mpmath.workdps(decimal_precision):
        result = mpmath.quad(f, (0., a), **kwargs)

        # If not converted to `sympy.Float` precision will be lost after the
        # original `mpmath` context is restored
        if isinstance(result, tuple): # Integration error included
            return tuple(Float(x, decimal_precision) for x in result)
        else:
            return Float(result, decimal_precision)
```

Listing 4.11: Izvod iz Python modula `beam_integrals.integrals`, puna verzija dostupna na [137]

Klasa `BaseIntegral` predstavlja baznu klasu podsistema programskih dodataka za integrale date u sekciji 1.2.2. Stoga će sve njene klase naslednice biti automatski registrovane kao odgovarajući programski dodaci, tj. biće automatski vidljivi drugim modulima kao zasebni integrali. Klasa `BaseIntegral` nasleđuje predstavljenu `FriendlyNameFromClassMixin` *mixin* klasu (sekcija 4.1) i sadrži sledeće atribute, svojstva i metode:

1. `name` je pomoćno svojstvo, automatski ubačeno prilikom nasleđivanja `FriendlyNameFromClassMixin` *mixin* klase (sekcija 4.1), koje sadrži ljudski čitljivo ime objekta klase datog integrala.

2. `__str__` je specijalna Python metoda koja vraća string reprezentaciju objekta klase datog integrala. String reprezentacija formirana je na osnovu vrednosti svojstva `name`.

3. `id` je pomoćno Python svojstvo koje sadrži sadrži jednoznačnu numeričku identifikaciju klase unutar podsistema

programskih dodataka, dobijen na osnovu naziva integrala. U pozadini će svako pristupanje svojstvu `id` u ciljnoj Python klasi dovesti do sledećeg postupka:

   (a) Proverava se da li ciljna Python klasa poseduje keširani atribut `_id`. Ako da, skače se na korak 3d

   (b) U suprotnom, upotrebom regularnog izraza [168,169], smeštenog u atribut `_id_re`, transformiše se sadržaj svojstva `name` u numeričku oznaku datog integrala

   (c) U ciljnu klasu potom se dinamički dodaje keširani atribut `_id`, sa vrednošću prethodno određene numeričke oznake

   (d) Vraća se vrednost keširanog atributa `_id`

4. `used_variables` je atribut koji sadrži spisak integracionih promenljivih koje definišu analitičku formulu datog integrala. Ovaj atribut koristi se tokom implementacije optimizacija predstavljenih u sekcijama 3.3.2 i 3.3.3.

5. `_contribute_to_plugins` je specijalna metoda u okviru podsistema programskih dodataka koja može da dodaje nove i menja postojeće informacije o trenutno registrovanim programskim dodacima, koje su uskladištene u `plugins` svojstvu. U ovom slučaju dodaje se nova informacija o integralima sa istovetnom kanoničkom formom u podatak `plugins.child_id_to_parent_id`, koji predstavlja mapiranje sa *ključa* na *vrednost*, gde je *ključ* vrednost svojstva `id` klase koja poseduje kanoničku formu a *vrednost* je vrednost svojstva `id` klase koja je direktni predak klasi sadržanoj u *ključu* i predstavlja njenu kanoničku formu. Ova metoda koristi se tokom implementacije optimizacije predstavljene u sekciji 3.3.1.

6. `has_parent` je metoda koja za dati integral vraća informaciju da li postoji roditeljska klasa koja predstavlja njegovu kanoničku formu. Ova metoda koristi se tokom implementacije optimizacije predstavljene u sekciji 3.3.1.

7. `parent_id` je metoda koja za dati integral vraća vrednost svojstva `id` roditeljske klase koja predstavlja njegovu kanoničku formu. Ukoliko dati integral ne poseduje kanoničku formu vraća se specijalna Python konstanta `None` [206]. Ova metoda koristi se tokom implementacije optimizacije predstavljene u sekciji 3.3.1.

8. `iterate_over_used_variables` je metoda koja formira i vraća Python iterator [126] koji prolazi samo kroz dekartov proizvod zavisnih integracionih promenljivih, do zadatog maksimalnog moda. Ova metoda koristi se tokom implementacije optimizacije predstavljene u sekciji 3.3.3.

9. `cache_key` je metoda koja formira sistem za keširanje vrednosti određenih integrala u okviru date tabele integrala i vraća numeričku vrednost ključa keša za zadate zavisne integracione promenljive. Ova metoda koristi se tokom implementacije optimizacije predstavljene u sekciji 3.3.2.

10. `_integrand` je metoda koja definiše i vraća analitičku formulu podintegralne funkcije (tzv. integrand) datog integrala.

11. `integrand` je pomoćna metoda koja značajno olakšava korišćenje `_integrand` metode prilikom numeričke integracije. Ova metoda formira i vraća simbolički izraz integranda za zadati granični uslov i zavisne integracione promenljive.

12. `__call__` je specijalna Python metoda koja datom objektu pruža mogućnost da bude pozivan kao funkcija, tzv. *callable object* [197, 198] u terminologiji programskog jezika Python. U ovom slučaju prosto se prosleđuje poziv metodi `integrand` i vraća njena povratna vrednost. Primera radi, ova metoda pruža mogućnost da se poziv:
    ```
    i1.integrand(beam_type, m, t, v, n, precision)
    ```
    može izvršiti i kroz sam objekat:
    ```
    i1(beam_type, m, t, v, n, precision)
    ```

`BaseIntegralWithSymetricVariables` predstavlja baznu klasu za sve integrale koji su simetrične prirode. Klasa nasleđuje baznu klasu `BaseIntegral` i redefiniše metodu `cache_key`, koja sada vraća identični ključ keša za sve međusobno simetrične oblike datog integrala, u skladu sa algoritmom predstavljenim u sekciji 3.3.2.

Klasa `I1` predstavlja integral I_1, dat u jednačini 1.35. Klasa nasleđuje baznu klasu `BaseIntegralWithSymetricVariables`, automatski označavajući da je integral simetrične prirode, i redefiniše sledeće atribute i metode:

1. `used_variables` je atribut koji sadrži spisak integracionih promenljivih koje definišu analitičku formulu ovog integrala, izveden iz jednačine 1.35

2. `_integrand` je metoda koja vraća analitičku formulu podintegralne funkcije ovog integrala, izvedenu iz jednačine 1.35

Klasa `I2` predstavlja integral I_2, dat u jednačini 1.36. Klasa nasleđuje baznu klasu `BaseIntegralWithSymetricVariables`, automatski označavajući da je integral simetrične prirode, i redefiniše sledeće atribute i metode:

1. `used_variables` je atribut koji sadrži spisak integracionih promenljivih koje definišu analitičku formulu ovog integrala, izveden iz jednačine 1.36

2. `_integrand` je metoda koja vraća analitičku formulu podintegralne funkcije ovog integrala, izvedenu iz jednačine 1.36

Klasa `I3` predstavlja integral I_3, dat u jednačini 1.37. Klasa nasleđuje baznu klasu `BaseIntegral` i redefiniše sledeće atribute i metode:

1. `used_variables` je atribut koji sadrži spisak integracionih promenljivih koje definišu analitičku formulu ovog integrala, izveden iz jednačine 1.37

2. `_integrand` je metoda koja vraća analitičku formulu podintegralne funkcije ovog integrala, izvedenu iz jednačine 1.37

Klasa `I4` predstavlja integral I_4, dat u jednačini 1.38. Klasa nasleđuje klasu `I2`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_2. Ovaj tip optimizacije predstavljen je u sekciji 3.3.1.

Klasa `I5` predstavlja integral I_5, dat u jednačini 1.39. Klasa nasleđuje baznu klasu `BaseIntegral` i redefiniše sledeće atribute i metode:

1. `used_variables` je atribut koji sadrži spisak integracionih promenljivih koje definišu analitičku formulu ovog integrala, izveden iz jednačine 1.39

2. `_integrand` je metoda koja vraća analitičku formulu podintegralne funkcije ovog integrala, izvedenu iz jednačine 1.39

Klasa `I6` predstavlja integral I_6, dat u jednačini 1.40. Klasa nasleđuje klasu `I2`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_2.

Klasa `I7` predstavlja integral I_7, dat u jednačini 1.41. Klasa nasleđuje baznu klasu `BaseIntegralWithSymetricVariables`, automatski označavajući da je integral simetrične prirode, i redefiniše sledeće atribute i metode:

1. `used_variables` je atribut koji sadrži spisak integracionih promenljivih koje definišu analitičku formulu ovog integrala, izveden iz jednačine 1.41

2. `_integrand` je metoda koja vraća analitičku formulu podintegralne funkcije ovog integrala, izvedenu iz jednačine 1.41

Klasa `I8` predstavlja integral I_8, dat u jednačini 1.42. Klasa nasleđuje klasu `I2`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_2.

Klasa `I21` predstavlja integral I_{21}, dat u jednačini 1.43. Klasa nasleđuje klasu `I1`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_1.

Klasa `I22` predstavlja integral I_{22}, dat u jednačini 1.44. Klasa nasleđuje klasu `I3`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_3.

Klasa `I23` predstavlja integral I_{23}, dat u jednačini 1.45. Klasa nasleđuje klasu `I5`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_5.

Klasa `I24` predstavlja integral I_{24}, dat u jednačini 1.46. Klasa nasleđuje klasu `I7`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_7.

Klasa `I25` predstavlja integral I_{25}, dat u jednačini 1.47. Klasa nasleđuje klasu `I2`, automatski označavajući da ovaj integral poseduje istovetnu kanoničku formu kao i integral I_2.

`integrate` je funkcija koja sprovodi numeričku integraciju datog integrala za zadati granični uslov, gornju granicu integracije i zavisne integracione promenljive, a u traženoj decimalnoj preciznosti. Za sam proces numeričke integracije koristi se mpmath programska biblioteka [185, 186] i njena `quad` funkcija [207]. Ova funkcija, u zavisnosti od vrednosti parametra `error`, vraća:

- ili uređeni par vrednosti (v, e), gde:
  - v predstavlja vrednost određenog integrala
  - e predstavlja procenu apsolutne greške pri numeričkoj integraciji
- ili samo vrednost određenog integrala

4.3.6 Modul `beam_integrals.shell`

Modul `beam_integrals.shell` implementira proširivu konzolnu aplikaciju pod nazivom `beam_integrals` koja pruža konzolni pristup API-u projekta `beam_integrals`, putem interfejsa komandne linije (eng. *Command Line Interface*, skr. *CLI*).

Na listingu 4.12 dat je izvod iz izvornog kôda `beam_integrals.shell` modula:

```
import argparse
import sys
import beam_integrals as b
from beam_integrals import characteristic_equation_solvers as ces
from beam_integrals.exceptions import ShellCommandError


# Decorator for argparse args
def arg(*args, **kwargs):
    def _decorator(func):
        # Because of the sematics of decorator composition if we just append
        # to the options list positional options will appear to be backwards
        func.__dict__.setdefault('arguments', []).insert(0, (args, kwargs))
        return func
    return _decorator


class Shell(object):
    def __init__(self):
        self.parser = argparse.ArgumentParser(
            prog='beam_integrals',
            description='Determines beam integrals of all 6 supported beam '\
                        'types, as described in D.D. Milasinovic, "The Finite '\
                        'Strip Method in Computational Mechanics"',
            epilog='See "beam_integrals help COMMAND" for help on a command',
            add_help=False,
            formatter_class=argparse.ArgumentDefaultsHelpFormatter
        )

        # Global arguments
        self.parser.add_argument('-h', '--help',
            action='help',
            help= argparse.SUPPRESS,
        )
        self.parser.add_argument('--version',
            action='version',
            version="%(prog)s " + b.__version__
        )

        # Subcommands
        subparsers = self.parser.add_subparsers(metavar='<subcommand>')
        self.subcommands = {}

        # Everything that's do_* is a subcommand
        for attr in (a for a in dir(self) if a.startswith('do_')):
            command = attr[3:].replace('_', '-')
            callback = getattr(self, attr)
            desc = callback.__doc__ or ''
            help = desc.strip()
            arguments = getattr(callback, 'arguments', [])

            subparser = subparsers.add_parser(command,
                help=help,
```

```
            description=desc,
            add_help=False,
            formatter_class=argparse.ArgumentDefaultsHelpFormatter
        )
        subparser.add_argument('-h', '--help',
            action='help',
            help=argparse.SUPPRESS,
        )
        self.subcommands[command] = subparser
        for (args, kwargs) in arguments:
            subparser.add_argument(*args, **kwargs)
        subparser.set_defaults(func=callback)

def main(self, argv=None):
    argv = argv if argv is not None else sys.argv[1:]

    # Show help if called without any arguments
    if not argv:
        self.parser.print_help()
        return 0

    # Parse args and call whatever callback was selected
    args = self.parser.parse_args(argv)

    # Short-circuit and deal with help right away
    if args.func == self.do_help:
        self.do_help(args)
        return 0

    args.func(args)

@arg(
    'command',
    metavar='<subcommand>',
    nargs='?',
    help='Display help for <subcommand>'
)
def do_help(self, args):
    """Display help about this program or one of its subcommands"""
    if args.command:
        if args.command in self.subcommands:
            self.subcommands[args.command].print_help()
        else:
            raise ShellCommandError(
                "'%s' is not a valid subcommand" % args.command
            )
    else:
        self.parser.print_help()

@arg(
    '--max-mode',
    metavar='<mode>',
    type=int,
    default=b.DEFAULT_MAX_MODE,
    help='Maximum mode'
)
```

```
    @arg(
        '--decimal-precision',
        metavar='<precision>',
        type=int,
        default=b.DEFAULT_DECIMAL_PRECISION,
        help='Decimal precision'
    )
    def do_best_roots_of_characteristic_equations_regenerate_cache(self, args):
        """
        Regenerate the best roots of characteristic equations cache, for all
        supported beam types
        """
        ces.best_roots_cache.regenerate(
            max_mode=args.max_mode,
            decimal_precision=args.decimal_precision
        )


def main():
    try:
        Shell().main()
    except ShellCommandError, e:
        print >> sys.stderr, e
        sys.exit(1)

if __name__ == '__main__':
    main()
```

Listing 4.12: Izvod iz Python modula `beam_integrals.shell`, puna verzija dostupna na [137]

pri čemu je implementacija bazirana na interesantnom rešenju koje je predstavio i pojasnio Jacob Kaplan-Moss [208].

Trenutno je kroz konzolnu aplikaciju izložen samo mali deo celokupnog API-a projekta `beam_integrals`. Konkretno, izložen je deo zadužen za regenerisanje keša korenova karakteristične jednačine.

Na listingu 4.13 dat je jedan primer poziva konzolne aplikacije `beam_integrals`, bez zadatih argumenata komandne linije:

```
$ beam_integrals
usage: beam_integrals [--version] <subcommand> ...

Determines beam integrals of all 6 supported beam types, as described in D.D.
Milasinovic, "The Finite Strip Method in Computational Mechanics"

positional arguments:
  <subcommand>
    best-roots-of-characteristic-equations-regenerate-cache
                        Regenerate the best roots of characteristic equations
                        cache, for all supported beam types
    help                Display help about this program or one of its
```

```
                        subcommands

optional arguments:
  --version             show program's version number and exit

See "beam_integrals help COMMAND" for help on a specific command
```

Listing 4.13: Primer poziva konzolne aplikacije `beam_integrals`, bez zadatih argumenata komandne linije

Zatim, na listingu 4.14 dat je drugi primer poziva konzolne aplikacije `beam_integrals`, u režimu pružanja pomoći u korišćenju komande za regenerisanje keša korenova karakteristične jednačine:

```
$ beam_integrals help best-roots-of-characteristic-equations-regenerate-cache
usage: beam_integrals best-roots-of-characteristic-equations-regenerate-cache
       [--max-mode <mode>] [--decimal-precision <precision>]

Regenerate the best roots of characteristic equations cache, for all supported
beam types

optional arguments:
  --max-mode <mode>     Maximum mode (default: 100)
  --decimal-precision <precision>
                        Decimal precision (default: 155)
```

Listing 4.14: Primer poziva konzolne aplikacije `beam_integrals`, u režimu pružanja pomoći u korišćenju zadate komande

Dok je na listingu 4.15 dat primer poziva konzolne aplikacije `beam_integrals` sa zadavanjem dve različite komande za regenerisanje keša korenova karakteristične jednačine:

```
$ beam_integrals best-roots-of-characteristic-equations-regenerate-cache
$ beam_integrals best-roots-of-characteristic-equations-regenerate-cache \
> --decimal-precision=15

$ ls ~/.beam_integrals/cache/characteristic-equations
best-roots.decimal-precision=15.pickle  best-roots.decimal-precision=155.pickle
```

Listing 4.15: Primer poziva konzolne aplikacije `beam_integrals`, sa zadavanjem komandi za regenerisanje keša korenova karakteristične jednačine

gde se može i videti da su kreirane dve odvojene keš datoteke:

1. `best-roots.decimal-precision=155.pickle`, koristeći podrazumevanu radnu preciznost projekta `beam_integrals`
2. `best-roots.decimal-precision=15.pickle`, koristeći zadatu radnu preciznost od 15 decimalnih mesta

4.3.7 Podsistem za automatsku verifikaciju

Projekat `beam_integrals` poseduje napredan podsistem za automatsku verifikaciju, koji rigorozno verifikuje karakteristike hibridne metode i njene referentne Open Source implementacije, za sve opisane granične uslove. Podsistem je baziran na standardnim Python alatima za testiranje [127], uz oslanjanje na sledeće eksterne programske biblioteke:

1. nose [128, 209] je Open Source biblioteka za lakši i brži razvoj testova i naprednih podsistema za automatsku verifikaciju.

2. coverage [210, 211] je interaktivni alat i Open Source biblioteka koja meri pokrivenost koda testovima.

3. mock [212, 213] je Open Source biblioteka koja uvodi podršku za tzv. *mock* [212] objekte. Njenim korišćenjem se, tokom testiranja, uživo mogu menjati ili lažirati razni delovi sistema.

4. nosexcover [214, 215] je Open Source biblioteka koja proširuje nose biblioteku i uvodi podršku za XML format izveštaja pokrivenosti koda testovima. Jenkins Open Source *Continuous Integration* server oslanja se upravo na ovaj format za svoje analize pokrivenosti koda testovima.

5. pomoćni projekat `nose_extra_tools` [216, 217] je Open Source biblioteka koja proširuje nose biblioteku i uvodi dodatne alate za lakši i brži razvoj testova.

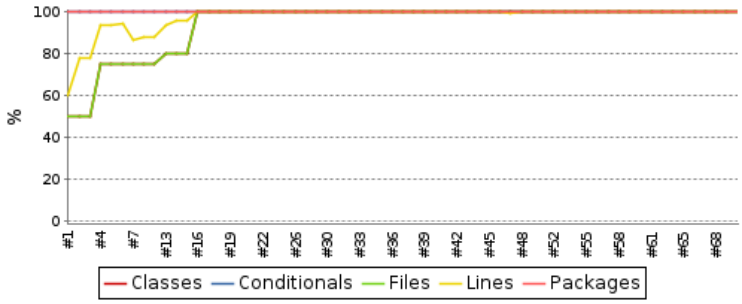
Jenkins Open Source *Continuous Integration* server se koristi za automatsko prevođenje, testiranje i praćenje toka razvoja projekta `beam_integrals`, čiji je javan Jenkins projekat dostupan na [130].

Podsistem za automatsku verifikaciju `beam_integrals` projekta razvijan je sa ciljem da pokrije svaki aspekt projekta uz 100% pokrivenosti koda testovima (slika 4.3) i sadrži $\approx$ 4400 međusobno nezavisnih testova (slika 4.4) koje naš Jenkins integracioni server automatski izvršava svaki put kada se registruje

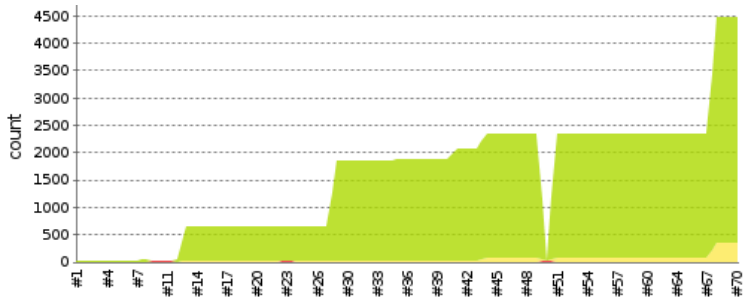
promena u javno dostupnom repozitorijumu izvornog koda za projekat `beam_integrals` [137].

Štaviše, svi bitni moduli u okviru `beam_integrals` projekta detaljno su pokriveni testovima [218]:

- `beam_integrals.beam_types` modul ispituju 1218 međusobno nezavisna testa koji pokrivaju 100% njegovog koda
- `beam_integrals.characteristic_equation_solvers` modul ispituju 619 međusobno nezavisna testa koji pokrivaju 100% njegovog koda
- `beam_integrals.integrals` modul ispituju 2631 međusobno nezavisna testa koji pokrivaju 100% njegovog koda



Slika 4.3: Istorijski izveštaj pokrivenosti koda testovima, preuzet sa naslovne strane Jenkins projekta `beam_integrals` [130]



Slika 4.4: Istorijski izveštaj broja i stepena prolaznosti testova u okviru podsistema za automatsku verifikaciju, preuzet sa naslovne strane Jenkins projekta `beam_integrals` [130]

Glava 5

Zaključak

Ovo istraživanje pokazalo je da direktno uvrštavanje aproksimativno analitičkog rešenja u karakteristične jednačine složenijih graničnih uslova rezultuje previsokim greškama, što takođe dovodi i do značajnog gubitka tačnosti prilikom rešavanja jednačina svojstvenih oblika i izračunavanja određenih integrala.

Predstavljena je hibridna metoda putem koje se na pouzdan način određuju korenovi karakterističnih jednačina, rešavaju jednačine svojstvenih oblika i izračunavaju određeni integrali. Razvijen je prototip referentne Open Source implementacije hibridne metode, uz podsistem za automatsku verifikaciju koji rigorozno verifikuje karakteristike hibridne metode i njene referentne implementacije, za sve navedene granične uslove i integrale.

Razmatrani su različiti izazovi koji proističu tokom kreiranja i normalizacije tabela integrala, za sve kombinacije modova i graničnih uslova, i predloženi su načini rešavanja uočenih problema.

5.1 Doprinos rada i mogućnosti primene

Rezultati dobijeni ovim istraživanjem omogućiće pouzdano korišćenje i viših modova tokom rešavanja problema stabilnosti u HCFSM, za sve navedene granične uslove. Samim tim ovaj rad će doprineti vernijim simulacijama fizičkih modela.

Od primena posebno dolaze u obzir problemi stabilnosti za armirano betonske konstrukcije koje po dužini menjaju karakteristike poprečnog preseka, zbog usvajanja različite količine armature. Takođe tu su i problemi kompozitnih materijala koji po slojevima u debljini trake imaju različite materijalne karakteristike.

5.2 Pravci daljeg istraživanja

Pravci daljeg istraživanja u ovoj temi mogu da obuhvate:

1. Dopunu projekta `beam_integrals` sa ciljem da obuhvati svih 113 integrala koje definiše HCFSM.

2. Razvoj *cloud computing* [219] bazirane distribuirane softverske arhitekture za kreiranje normalizovanih tabela integrala, za sve navedene granične uslove.

3. Osmišljavanje pouzdanog, efikasnog i jeftinog načina distribuiranja kreiranih tabela integrala, npr. model decentralizovane distribucije po uzoru na *BitTorrent* protokol [220].

4. Razvoj softverskog rešenja koje unakrsno poredi sadržaj tabela integrala kreiranih upotrebom različitih postupaka, npr. poređenje rezultata ostvarenih upotrebom IEEE 754 binary64 radne preciznosti i podrazumevane radne preciznosti za projekat `beam_integrals`. Takvo rešenje bi takođe moglo da grafički predstavi i analizira razlike među rezultatima.

5. Uvođenje alternativnih formulacija za bazne funkcije i pripadajuće karakteristične jednačine u `beam_integrals` projekat, kroz nasleđivanje postojećih graničnih uslova. Npr. za *Bradford bazne funkcije tipa 2* [20] mogu se kreirati novi granični uslovi LJ $\in \{11, 12, 13, 14, 15\}$, koji će nasleđivati postojeće granične uslove LJ $\in \{1, 2, 3, 4, 5\}$. Pomoću softverskog rešenja razmatranog u stavci 4 tada se može unakrsno uporediti sadržaj i analizirati potencijalne

razlike između tabela integrala kreiranih upotrebom graničnih uslova LJ $\in \{11, 12, 13, 14, 15\}$ sa LJ $\in \{1, 2, 3, 4, 5\}$.

6. Razvoj novih i prilagođavanje postojećih softverskih rešenja baziranih na HCFSM, koja će primeniti kreirane tabele integrala u rešavanju problema stabilnosti i slobodne vibracije.

5.3 Zahvalnica

Ovo istraživanje je deo projekta Ministarstva prosvete, nauke i tehnološkog razvoja „Računarska mehanika u teoriji konstrukcija" (OI 174027). Hvala im na pruženoj podršci.

Hvala i Open Source zajednici na uloženom trudu i sjajnim projektima kao što su Python [32], SymPy [25], mpmath [185] i Matplotlib [22]. Naše drage kolege John D. Hunter [22] i Malcolm Tredinnick [56] će nam svima nedostajati.

5.4 Napomene

Celokupno istraživanje bazirano je na Open Source alatima. Sve slike u ovoj tezi kreirane su programski kroz API pozive upućene projektu `beam_integrals`, uz sledeće izuzetke:

- slike 1.1 i 1.2 preuzete su iz ranijih radova naše istraživačke grupe [4, 9]
- slike 4.1 i 4.2 preuzete su sa naslovne strane našeg Jenkins projekta `simple_plugins` [182]
- slike 4.3 i 4.4 preuzete su sa naslovne strane našeg Jenkins projekta `beam_integrals` [130]

Svi primeri programskog kôda u ovoj tezi, koji su dati kroz interaktivnu Python sesiju, automatski su verifikovani od strane `doctest` modula [221] Python standardne biblioteke.

Bibliografija

[1] Y. Cheung, "The finite strip method in the analysis of elastic plates with two opposite simply supported ends," in *ICE Proceedings*, vol. 40, no. 1. Thomas Telford, 1968, pp. 1–7.

[2] ——, "Finite strip method analysis of elastic slabs," in *Journal of Engineering Mechanics*, vol. 94. American Society of Civil Engineers, 1968, pp. 1365–1378.

[3] G. Dhatt, E. Lefrançois, and G. Touzot, *Finite element method*. John Wiley & Sons, 2012.

[4] M. Hajduković, D. D. Milašinović, M. Nikolić, P. S. Rakić, Ž. Živanov, and L. Stričević, "Scope of MPI/OpenMP/CUDA Parallelization of Harmonic Coupled Finite Strip Method Applied on Large Displacement Stability Analysis of Prismatic Shell Structures," *Computer Science and Information Systems*, vol. 9, no. 2, pp. 741–761, 2012.

[5] Y. Cheung and L. G. Tham, *The Finite Strip Method*. CRC Press, 1997.

[6] D. D. Milašinović, "Geometric non-linear analysis of thin plate structures using the harmonic coupled finite strip method," *Thin-Walled Structures*, vol. 49, no. 2, pp. 280–290, 2011.

[7] Y. Cheung, "The analysis of cylindrical orthotropic curved bridge decks," in *IABSE Proceedings*, vol. 29, 1969, pp. 41–51.

[8] D. D. Milašinović, *The finite strip method in computational mechanics*. Subotica, Budapest, Belgrade: Faculties of Civil Engineering: University of Novi Sad, Technical University of Budapest and University of Belgrade, 1997.

[9] P. Rakić, D. D. Milašinović, Ž. Živanov, Z. Suvajdžin, M. Nikolić, and M. Hajduković, "MPI–CUDA parallelization of a finite-strip program for geometric nonlinear analysis: A hybrid approach," *Advances in Engineering Software*, vol. 42, no. 5, pp. 273–285, 2011.

[10] D. D. Milašinović, "Harmonic coupled finite strip method applied on buckling-mode interaction analysis of composite thin-walled wide-flange columns," *Thin-Walled Structures*, vol. 50, no. 1, pp. 95–105, 2012.

[11] D. D. Milašinović, D. Goleš, A. Borković, D. Kukaras, A. Landović, Z. Živanov, and P. Rakić, "Rheological-dynamical limit analysis of reinforced concrete folded plate structures using the harmonic coupled finite-strip method," *BHV Topping*, 2012.

[12] D. D. Milašinović and D. Goleš, "Finite strip modeling for optimal design of reinforced concrete folded plate structures," *Facta universitatis-series: Architecture and Civil Engineering*, vol. 10, no. 3, pp. 275–290, 2012.

[13] D. Goleš, D. D. Milašinović, and Ž. Živanov, "Analysis of ultimate resistance of long reinforced concrete folded plate structure," *Zbornik radova Građevinskog fakulteta, Subotica*, no. 22, pp. 67–77, 2013.

[14] D. D. Milašinović and D. Goleš, "Analiza stabilnosti armiranobetonskih složenica," *Građevinar*, vol. 65, no. 5, pp. 411–422, 2013.

[15] D. D. Milašinović, A. Borković, Ž. Živanov, P. Rakić, M. Nikolić, L. Stričević, and M. Hajduković, "Large displacement stability analysis of thin plate structures: Scope of mpi-/openmp parallelization in harmonic coupled finite strip

analysis," *Advances in Engineering Software*, vol. 66, pp. 40–51, 2013.

[16] D. D. Milasinovic and D. Goleš, "Geometric nonlinear analysis of reinforced concrete folded plate structures by the harmonic coupled finite strip method," *Civil Engineering*, 2014.

[17] M. Nikolić, M. Hajduković, D. D. Milašinović, D. Goleš, P. Marić, and Ž. Živanov, "Hybrid MPI/OpenMP cloud parallelization of harmonic coupled finite strip method applied on reinforced concrete prismatic shell structure," *Advances in Engineering Software*, 2015.

[18] V. A. Vujičić, *Teorija oscilacija.* Savremena administracija, 1969.

[19] I. E. Harik and R. Ekambaram, "Seminumerical solution for buckling of rectangular plates," *Computers & structures*, vol. 23, no. 5, pp. 649–655, 1986.

[20] M. A. Bradford and M. Azhari, "Buckling of plates with different end conditions using the finite strip method," *Computers & structures*, vol. 56, no. 1, pp. 75–83, 1995.

[21] Open Source Initiative, *The BSD 3-Clause License*, 2013. [Online]. Available: http://opensource.org/licenses/BSD-3-Clause

[22] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[23] StackOverflow, *Difference between log and symlog*, 2014. [Online]. Available: http://stackoverflow.com/questions/3305865/what-is-the-difference-between-log-and-symlog

[24] G. H. Golub and C. F. van Van Loan, *Matrix computations.* The Johns Hopkins University Press, 1996.

[25] SymPy Development Team, *SymPy - Python library for symbolic mathematics*, 2012. [Online]. Available: http://www.sympy.org

[26] IEEE Task P754, *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic.* New York, NY, USA: IEEE, Aug. 1985, revised 1990. A preliminary draft was published in the January 1980 issue of IEEE Computer, together with several companion articles. Also standardized as *IEC 60559 (1989-01) Binary floating-point arithmetic for microprocessor systems.* [Online]. Available: http://standards.ieee.org/reading/ieee/std/busarch/754-1985.pdf

[27] ——, *IEEE 754-2008, Standard for Floating-Point Arithmetic.* New York, NY, USA: IEEE, Aug. 2008. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=4610933

[28] J. Demmel, *Basic Issues in Floating Point Arithmetic and Error Analysis*, 2014. [Online]. Available: http://www.cs.berkeley.edu/~demmel/cs267/lecture21/lecture21.html

[29] E. Anderson, *LAPACK Users' guide.* Siam, 1999, vol. 9.

[30] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[31] K. R. Ghazi, V. Lefèvre, P. Théveny, and P. Zimmermann, "Why and how to use arbitrary precision," *Computing in Science & Engineering*, vol. 12, no. 1-3, pp. 5–5, 2010.

[32] G. Van Rossum, "Python programming language," 1994.

[33] R. G. Bartle, *A modern theory of integration.* American Mathematical Society, 2001, vol. 32.

[34] Sun Microsystems, *FORTRAN 77 Language Reference: List-Directed I/O*, 1999. [Online]. Available: http://docs.oracle.com/cd/E19957-01/805-4939/6j4m0vnc5/index.html

[35] Y. Shafranovich, *RFC 4180: Common format and MIME type for Comma-Separated Values (CSV) files*, 2005. [Online]. Available: http://tools.ietf.org/html/rfc4180

[36] JSON community, *Introducing JSON*, 2014. [Online]. Available: http://json.org/

[37] D. Crockford, *RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON)*, 2006. [Online]. Available: http://tools.ietf.org/html/rfc4627

[38] Ecma International, *Standard ECMA-404: The JSON Data Interchange Format*, October 2013. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-404.htm

[39] D. Crockford, "Douglas Crockford's Javascript," 2014. [Online]. Available: http://javascript.crockford.com/

[40] Ecma International, *Standard ECMA-262: ECMAScript Language Specification*, 5th ed., June 2011. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[41] O. Ben-Kiki, C. Evans, and I. döt Net, "YAML," 2014. [Online]. Available: http://yaml.org/

[42] ——, *YAML Ain't Markup Language (YAML™) Version 1.1*, 2005. [Online]. Available: http://yaml.org/spec/1.1/

[43] B. W. Kemighan and D. M. Ritchie, *The C programming language*, 1st ed. Prentice Hall, 1978.

[44] Oracle, *Java Platform SE 7*, 2014. [Online]. Available: http://docs.oracle.com/javase/7/docs/index.html

[45] L. Wall, “Perl programming language,” 1987. [Online]. Available: https://www.perl.org/

[46] Y. Matsumoto, “Ruby programming language,” 1993. [Online]. Available: https://www.ruby-lang.org/

[47] C. Evans, “YAML Draft 0.1,” 2001. [Online]. Available: http://tech.groups.yahoo.com/group/sml-dev/message/4710

[48] O. Ben-Kiki, C. Evans, and B. Ingerson, *Language–Independent Types for YAML™ Version 1.1*, 2005. [Online]. Available: http://yaml.org/type/

[49] N. Freed and N. S. Borenstein, *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One – Format of Internet Message Bodies*, 1996. [Online]. Available: http://tools.ietf.org/html/rfc2045

[50] International Organization for Standardization (ISO), *ISO 8601: Data elements and interchange formats – Information interchange – Representation of dates and times*, 1988. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=15903

[51] M. Wolf and C. Wicksteed, *Date and Time Formats*, 1997. [Online]. Available: http://www.w3.org/TR/NOTE-datetime

[52] O. Ben-Kiki, C. Evans, and I. döt Net, *YAML Ain't Markup Language (YAML™) Version 1.2*, 2009. [Online]. Available: http://www.yaml.org/spec/1.2/spec.html

[53] J. Noller, “YAML ain't Markup Language,” 2009. [Online]. Available: http://jessenoller.com/blog/2009/04/13/yaml-aint-markup-language-completely-different

[54] Docker Inc., *Docker Compose documentation*, 2015. [Online]. Available: https://docs.docker.com/compose/

[55] T. S. Hatch, *SaltStack Docs*, 2015. [Online]. Available: http://docs.saltstack.com/en/latest/

[56] Django Software Foundation, "Django: The Web framework for perfectionists with deadlines," 2015. [Online]. Available: https://www.djangoproject.com/

[57] D. H. Hansson, "Ruby on Rails," 2015. [Online]. Available: http://rubyonrails.org/

[58] Docker Inc., "Docker – Build, Ship, and Run Any App, Anywhere," 2015. [Online]. Available: https://www.docker.com/

[59] K. Simonov, "PyYAML – YAML parser and emitter for Python," 2014. [Online]. Available: http://pyyaml.org/wiki/PyYAML

[60] ——, "Python Package Index: PyYAML 3.11," 2014. [Online]. Available: https://pypi.python.org/pypi/PyYAML/3.11

[61] J. Angold, "Ways of loading multiple YAML Documents in a single file," 2011. [Online]. Available: https://groups.google.com/forum/#!topic/snakeyaml-core/DhJqkJCCx98

[62] The HDF Group, "HDF5," 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/

[63] University of Illinois, "National Center for Supercomputing Applications," 2015. [Online]. Available: http://www.ncsa.illinois.edu/

[64] The HDF Group, "About the HDF Group," 2013. [Online]. Available: https://www.hdfgroup.org/about/

[65] ——, *HDF5 Tutorial: Learning the Basics – HDF5 File Organization*, 2011. [Online]. Available: https://www.hdfgroup.org/HDF5/Tutor/fileorg.html

[66] ——, *Limits in HDF5*, 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/faq/limits.html

[67] ——, *HDF5 Technologies*, 2011. [Online]. Available: https://www.hdfgroup.org/about/hdf_technologies.html

[68] ——, "HDF5 Users," 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/users5.html

[69] ——, "HDF Boeing Project," 2011. [Online]. Available: https://www.hdfgroup.org/projects/boeing/

[70] ——, "HDF Group Projects," 2015. [Online]. Available: https://www.hdfgroup.org/projects/

[71] ——, "HDF ESDIS Project," 2013. [Online]. Available: https://www.hdfgroup.org/projects/esdis/

[72] ——, "HDF JPSS Project: Handling Large Datasets Efficiently," 2014. [Online]. Available: https://www.hdfgroup.org/projects/jpss/

[73] PyTables maintainers, *PyTables 3.2.0 documentation*, 2015. [Online]. Available: http://www.pytables.org/

[74] ——, "Python Package Index: tables 3.2.0," 2015. [Online]. Available: https://pypi.python.org/pypi/tables/3.2.0

[75] A. Collette, "HDF5 for Python," 2015. [Online]. Available: http://www.h5py.org/

[76] ——, *Python and HDF5*. O'Reilly Media, 2013.

[77] ——, *h5py 2.5.0 documentation*, 2015. [Online]. Available: http://docs.h5py.org/en/latest/

[78] ——, "Python Package Index: h5py 2.5.0," 2015. [Online]. Available: https://pypi.python.org/pypi/h5py/2.5.0

[79] NumPy Developers, "NumPy: array processing for numbers, strings, records, and objects," 2015. [Online]. Available: http://www.numpy.org/

[80] ——, "Python Package Index: numpy 1.9.2," 2015. [Online]. Available: https://pypi.python.org/pypi/numpy/1.9.2

[81] PyTables maintainers, *PyTables 3.2.0 documentation: Optimization tips – In-kernel searches*, 2015. [Online]. Available: http://www.pytables.org/usersguide/optimization.html#in-kernel-searches

[82] ——, *PyTables 3.2.0 documentation: Condition Syntax*, 2015. [Online]. Available: http://www.pytables.org/usersguide/condition_syntax.html

[83] ——, *PyTables 3.2.0 documentation: Optimization tips – Indexed searches*, 2015. [Online]. Available: http://www.pytables.org/usersguide/optimization.html#indexed-searches

[84] ——, *PyTables 3.2.0 documentation: FAQ – How does PyTables compare with the h5py project?*, 2015. [Online]. Available: http://www.pytables.org/FAQ.html#how-does-pytables-compare-with-the-h5py-project

[85] A. Collette, *h5py 2.5.0 documentation: FAQ – What's the difference between h5py and PyTables?*, 2015. [Online]. Available: http://docs.h5py.org/en/latest/faq.html#what-s-the-difference-between-h5py-and-pytables

[86] The HDF Group, *Using Compresion in HDF5*, 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/faq/compression.html

[87] ——, *HDF5 FAQ: What kind of compression methods does HDF5 support?*, 2015. [Online]. Available: https://www.hdfgroup.org/hdf5-quest.html#gcomp

[88] ——, *HDF5: Filters*, 2015. [Online]. Available: https://www.hdfgroup.org/services/filters.html

[89] J.-l. Gailly and M. Adler, "zlib: A massively spiffy yet delicately unobtrusive compression library," 2013. [Online]. Available: http://www.zlib.net/

[90] The HDF Group, "SZIP Compression in HDF Products," 2011. [Online]. Available: https://www.hdfgroup.org/doc_resource/SZIP/

[91] M. F. Oberhumer, "LZO real-time data compression library," 2015. [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[92] J. Seward, "bzip2," 2007. [Online]. Available: http://www.bzip.org/

[93] M. Lehmann, "LibLZF," 2008. [Online]. Available: http://oldhome.schmorp.de/marc/liblzf.html

[94] F. Alted and V. Hänel, "Blosc, an extremely fast, multi-threaded, meta-compressor library," 2015. [Online]. Available: http://blosc.org/

[95] N. Hübbe, "Compression of scientific data with MAFISC," 2013. [Online]. Available: http://wr.informatik.uni-hamburg.de/research/projects/icomex/mafisc

[96] M. Rissi, "Snappy – a fast compressor/decompressor," 2013. [Online]. Available: https://code.google.com/p/snappy/

[97] Y. Collet, "LZ4 – Extremely fast compression," 2015. [Online]. Available: http://cyan4973.github.io/lz4/

[98] M. Albert, "JPEG-XR filter for HDF5," 2013. [Online]. Available: https://hdf5jpegxr.codeplex.com/

[99] The HDF Group, "HDFView," 2014. [Online]. Available: https://www.hdfgroup.org/products/java/hdfview/index.html

[100] ——, "HDF Compass," 2015. [Online]. Available: https://www.hdfgroup.org/projects/compass/

[101] V. Mas, "ViTables," 2015. [Online]. Available: http://vitables.org/

[102] Space Research Software LLC, "HDF Explorer," 2012. [Online]. Available: http://www.space-research.org/

[103] Wolfram Research, Inc., "Mathematica version 10.1," Champaign, Illinois, 2015.

[104] The MathWorks, Inc., "MATLAB version 8.5 (R2015a)," Natick, Massachusetts, 2015.

[105] The HDF Group, *Chunking in HDF5*, 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/doc/Advanced/Chunking/

[106] ——, *Why use HDF: Access needs include random access*, 2011. [Online]. Available: https://www.hdfgroup.org/why_hdf/#random

[107] ——, *HDF5 Tutorial: Learning the Basics – Creating an Attribute*, 2014. [Online]. Available: http://www.hdfgroup.org/HDF5/Tutor/crtatt.html

[108] ——, *HDF5 Tutorial: Learning the Basics – Creating a Dataset*, 2014. [Online]. Available: https://www.hdfgroup.org/HDF5/Tutor/crtdat.html

[109] ——, *HDF5 Tutorial: Advanced Topics – References to Objects*, 2011. [Online]. Available: https://www.hdfgroup.org/HDF5/Tutor/reftoobj.html

[110] ——, *HDF5 User's Guide: Chapter 4 – HDF5 Groups*, 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/doc/UG/UG_frame09Groups.html

[111] PyTables maintainers, *PyTables 3.2.0 documentation: Tutorials – Using links for more convenient access to nodes*, 2015. [Online]. Available: http://www.pytables.org/usersguide/tutorials.html#linkstutorial

[112] The HDF Group, *HDF5 Reference Manual: HDF5 Predefined Datatypes*, 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/doc/RM/PredefDTypes.html

[113] ——, *HDF5 User's Guide: Chapter 6 – HDF5 Datatypes*, 2015. [Online]. Available: https://www.hdfgroup.org/HDF5/doc/UG/UG_frame11Datatypes.html

[114] The Scipy community, *NumPy v1.9 Manual: Data types*, 2014. [Online]. Available: http://docs.scipy.org/doc/numpy/user/basics.types.html

[115] Microsoft, *.NET Framework 4.5 Class Library: System.Int32 Fields*, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.int32_fields(v=vs.110).aspx

[116] Oracle, *Java Platform SE 7: Class Integer*, 2014. [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html

[117] E. Giguere, "The ANSI Standard: A Summary for the C Programmer," 1987. [Online]. Available: http://www.ericgiguere.com/articles/ansi-c-summary.html

[118] IEEE and The Open Group, *POSIX.1-2008 with TC1 (IEEE Std 1003.1): <limits.h>*, 2013. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/limits.h.html

[119] S. Sanfilippo, "Redis: Hash Tables Implementation – dictIntHashFunction()," 2015. [Online]. Available: https://github.com/antirez/redis/blob/c77081a/src/dict.c#L103

[120] D. Interactive, "Memcached: MurmurHash3 hash table algorithm," 2015. [Online]. Available: https://github.com/memcached/memcached/blob/05ca809/murmur3_hash.h

[121] ——, "Memcached: Jenkins hash table algorithm," 2015. [Online]. Available: https://github.com/memcached/memcached/blob/05ca809/jenkins_hash.h

[122] B. Jenkins, "A hash function for hash table lookup," *Dr. Dobbs Journal*, 1996. [Online]. Available: http://burtleburtle.net/bob/hash/doobs.html

[123] M. Stojaković, *Slučajni procesi*. Symbol, 2004.

[124] R. A. Brualdi, *Introductory combinatorics*, 5th ed. Pearson, 2009.

[125] D. Cvetković, *Diskretne matematičke strukture*, 2nd ed. Naučna knjiga, 1983.

[126] Python Software Foundation, *Python 2.7.9 documentation: itertools.product*, 2015. [Online]. Available: https://docs.python.org/2/library/itertools.html#itertools.product

[127] ——, *Python 2.7.9 documentation: unittest – unit testing framework*, 2015. [Online]. Available: https://docs.python.org/2/library/unittest.html

[128] J. Pellerin, *nose documentation*, 2015. [Online]. Available: https://nose.readthedocs.org/en/latest/

[129] Jenkins CI community, *Jenkins CI - An extendable open source continuous integration server*, 2013. [Online]. Available: http://jenkins-ci.org/

[130] P. Marić, "beam_integrals Jenkins project page." [Online]. Available: http://ci.petarmaric.com/job/beam_integrals/

[131] ——, "Jenkins Dashboard." [Online]. Available: http://ci.petarmaric.com/

[132] Hewlett-Packard, *HP Compaq Elite 8300 CMT Business PC Technical Specifications*, 2014. [Online]. Available: http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04128234

[133] Intel ARK, *Intel® Core™ i5-3470 Processor*, 2012. [Online]. Available: http://ark.intel.com/products/68316/Intel-Core-i5-3470-Processor-6M-Cache-up-to-3_60-GHz

[134] Western Digital Technologies, *WD Caviar® Blue™ Desktop Hard Drives*, 2012. [Online]. Available: http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701277.pdf

[135] Hewlett-Packard, *AMD Radeon HD 7450 DP (1GB) PCIe x16 Graphics Card*, 2012. [Online]. Available: http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04111040

[136] Intel ARK, *Intel® 82579LM Gigabit Ethernet PHY*, 2011. [Online]. Available: http://ark.intel.com/products/47620/Intel-82579LM-Gigabit-Ethernet-PHY

[137] P. Marić and D. D. Milašinović, "beam_integrals public code repository." [Online]. Available: https://bitbucket.org/petar/beam_integrals/src/

[138] J. Pellerin, *nose documentation: XUnit – output test results in XUnit format*, 2015. [Online]. Available: http://nose.readthedocs.org/en/latest/plugins/xunit.html

[139] P. Poulard, *XUnit cookbook*, 2010. [Online]. Available: http://reflex.gforge.inria.fr/xunit.html

[140] Windy Road Technology, *Apache Ant JUnit XML Schema*, 2011. [Online]. Available: http://windyroad.com.au/2011/02/07/apache-ant-junit-xml-schema/

[141] P. Marić, "beam_integrals Jenkins project workspace: view nosetests.xml." [Online]. Available: http://ci.petarmaric.com/job/beam_integrals/ws/nosetests.xml

[142] StackOverflow, "Difference between scaling horizontally and vertically for databases," 2015. [Online]. Available: http://stackoverflow.com/q/11707879/137064

[143] D. Beaumont, "How to explain vertical and horizontal scaling in the cloud," 2014. [Online]. Available: http://www.thoughtsoncloud.com/2014/04/explain-vertical-horizontal-scaling-cloud/

[144] The Linux Information Project, "Scalable definition," 2006. [Online]. Available: http://www.linfo.org/scalable.html

[145] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–8.

[146] I. Foster, *Designing and building parallel programs*. Addison Wesley Publishing Company, 1995.

[147] B. Massingill, "Embarrassingly parallel design pattern," 2001. [Online]. Available: http://www.cise.ufl.edu/research/ParallelPatterns/PatternLanguage/AlgorithmStructure/EmbParallel.htm

[148] B. Wilkinson and M. Allen, "Slides for parallel programming techniques and applications using networked workstations and parallel computers," 2002. [Online]. Available: http://www.ime.usp.br/~song/mac5705/slides3.pdf

[149] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[150] A. Karbowski, "Amdahl's and Gustafson-Barsis laws revisited," *arXiv preprint arXiv:0809.1177*, 2008.

[151] A. Suleman, "Parallel Programming: When Amdahl's law is inapplicable?" 2011. [Online]. Available: http://www.futurechips.org/thoughts-for-researchers/parallel-programming-gene-amdahl-said.html

[152] Hewlett-Packard, "HP Z220 Convertible Minitower Workstation," 2013. [Online]. Available: http://www8.hp.com/ca/en/products/workstations/product-detail.html?oid=5260575

[153] ——, *HP Z220 CMT Workstation Technical Specifications*, 2013. [Online]. Available: http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04128192

[154] NVIDIA Corporation, *NVIDIA® Quadro® 600 Product Specifications*, 2010. [Online]. Available: http://www.nvidia.com/docs/IO/40049/NV_DS_QUADRO_600_US_LR.pdf

[155] Intel ARK, *Intel® 82574L Gigabit Ethernet Controller*, 2008. [Online]. Available: http://ark.intel.com/products/32209/Intel-82574L-Gigabit-Ethernet-Controller

[156] P. Marić, "friendly_name_mixin public code repository," 2013. [Online]. Available: https://bitbucket.org/petar/friendly_name_mixin/src/

[157] ——, "Python Package Index: friendly_name_mixin 1.0.1," 2013. [Online]. Available: https://pypi.python.org/pypi/friendly_name_mixin/1.0.1

[158] Python Software Foundation, *Python 2.7.9 documentation: The Python Standard Library*, 2015. [Online]. Available: https://docs.python.org/2/library/

[159] Cunningham & Cunningham, Inc., "c2 wiki: Platform independence," 2009. [Online]. Available: http://c2.com/cgi/wiki?PlatformIndependence

[160] Python Software Foundation, *Python 2.7.9 documentation: Distributing Python Modules*, 2015. [Online]. Available: https://docs.python.org/2/distutils/index.html

[161] M. Lutz, *Programming Python*. O'Reilly Media, 2006.

[162] C. Esterbrook, "Using Mix-ins with Python," *Linux Journal*, vol. 48, 2001. [Online]. Available: http://www.linuxjournal.com/article/4540

[163] G. Van Rossum, "Method resolution order," 2010. [Online]. Available: http://python-history.blogspot.com/2010/06/method-resolution-order.html

[164] Python Software Foundation, *Python 2.7.9 documentation: The Python Tutorial – Classes*, 2015. [Online]. Available: https://docs.python.org/2/tutorial/classes.html

[165] ——, *Python 2.7.9 documentation: Property decorator*, 2015. [Online]. Available: https://docs.python.org/2/library/functions.html#property

[166] StackOverflow, "How does the @property decorator work?" 2013. [Online]. Available: http://stackoverflow.com/q/17330160

[167] P. O'Brien, "Guide to python introspection," 2002. [Online]. Available: http://www.ibm.com/developerworks/library/l-pyint/

[168] Python Software Foundation, *Python 2.7.9 documentation: Regular Expression HOWTO*, 2015. [Online]. Available: https://docs.python.org/2/howto/regex.html

[169] ——, *Python 2.7.9 documentation: re – Regular expression operations*, 2015. [Online]. Available: https://docs.python.org/2/library/re.html

[170] P. Marić, "friendly_name_mixin Jenkins project page." [Online]. Available: http://ci.petarmaric.com/job/friendly_name_mixin/

[171] ——, "simple_plugins public code repository," 2013. [Online]. Available: https://bitbucket.org/petar/simple_plugins/src/

[172] ——, "Python Package Index: simple_plugins 1.0.1," 2013. [Online]. Available: https://pypi.python.org/pypi/simple_plugins/1.0.1

[173] M. Alchin, "A simple plugin framework," 2008. [Online]. Available: http://martyalchin.com/2008/jan/10/simple-plugin-framework/

[174] StackOverflow, "Javascript style dot notation for dictionary keys unpythonic?" 2008. [Online]. Available: http://stackoverflow.com/q/224026

[175] C. Mărieș, "Understanding python metaclasses," 2015. [Online]. Available: http://blog.ionelmc.ro/2015/02/09/understanding-python-metaclasses/

[176] E. Bendersky, "Python metaclasses by example," 2011. [Online]. Available: http://eli.thegreenplace.net/2011/08/14/python-metaclasses-by-example/

[177] StackOverflow, "What is a metaclass in python?" 2014. [Online]. Available: http://stackoverflow.com/questions/100003/what-is-a-metaclass-in-python

[178] Python Software Foundation, *Python 2.7.9 documentation: Customizing class creation*, 2015. [Online]. Available: https://docs.python.org/2/reference/datamodel.html#customizing-class-creation

[179] J. Vanderpla, "A primer on python metaclasses," 2012. [Online]. Available: https://jakevdp.github.io/blog/2012/12/01/a-primer-on-python-metaclasses/

[180] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[181] Python Software Foundation, *Python 2.7.9 documentation: The Python Tutorial – Data Structures*, 2015. [Online]. Available: https://docs.python.org/2/tutorial/datastructures.html

[182] P. Marić, "simple_plugins Jenkins project page." [Online]. Available: http://ci.petarmaric.com/job/simple_plugins/

[183] ——, "Python Package Index: beam_integrals 1.1.0," 2015. [Online]. Available: https://pypi.python.org/pypi/beam_integrals/1.1.0

[184] SymPy Development Team, "Python Package Index: sympy 0.7.1," 2011. [Online]. Available: https://pypi.python.org/pypi/sympy/0.7.1

[185] F. Johansson, *mpmath - Python library for arbitrary-precision floating-point arithmetic*, 2013. [Online]. Available: http://code.google.com/p/mpmath

[186] ——, "Python Package Index: mpmath 0.19," 2014. [Online]. Available: https://pypi.python.org/pypi/mpmath/0.19

[187] C. Van Horsen, "gmpy – multiple-precision arithmetic for python," 2013. [Online]. Available: https://code.google.com/p/gmpy/

[188] ——, "Python Package Index: gmpy 1.17," 2013. [Online]. Available: https://pypi.python.org/pypi/gmpy/1.17

[189] S. Bethard, "argparse – python command line parsing," 2014. [Online]. Available: https://code.google.com/p/argparse/

[190] ——, "Python Package Index: argparse 1.3.0," 2014. [Online]. Available: https://pypi.python.org/pypi/argparse/1.3.0

[191] SymPy Development Team, *SymPy v0.7.1 documentation: Tutorial*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/tutorial.html

[192] ——, *SymPy v0.7.1 documentation: Float*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/modules/core.html#float

[193] ——, *SymPy v0.7.1 documentation: mpmath – temporarily changing the precision*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/modules/mpmath/basics.html#temporarily-changing-the-precision

[194] ——, *SymPy v0.7.1 documentation: mpmath contexts*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/modules/mpmath/contexts.html

[195] ——, *SymPy v0.7.1 documentation: mpmath workprec() context manager*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/modules/mpmath/general.html#workprec

[196] ——, *SymPy v0.7.1 documentation: mpmath root-finding and optimization*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/modules/mpmath/calculus/optimization.html

[197] Python Software Foundation, *Python 2.7.9 documentation: callable built-in function*, 2015. [Online]. Available: https://docs.python.org/2/library/functions.html#callable

[198] ——, *Python 2.7.9 documentation: Emulating callable objects*, 2015. [Online]. Available: https://docs.python.org/2/reference/datamodel.html#emulating-callable-objects

[199] N. Anderson and Å. Björck, "A new high order method of regula falsi type for computing a root of an equation," *BIT Numerical Mathematics*, vol. 13, no. 3, pp. 253–264, 1973.

[200] J. Ford, "Improved algorithms of Illinois-type for the numerical solution of nonlinear equations," University of Essex, Department of Computer Science, Tech. Rep., 1995.

[201] A. Kaw and E. Eric Kalu, *Numerical Methods with Applications: Abridged*, 2nd ed. http://www.autarkaw.com, 2011.

[202] M. Dowell and P. Jarratt, "A modified regula falsi method for computing the root of an equation," *BIT Numerical Mathematics*, vol. 11, no. 2, pp. 168–174, 1971.

[203] ——, "The 'Pegasus' method for computing the root of an equation," *BIT Numerical Mathematics*, vol. 12, no. 4, pp. 503–508, 1972.

[204] C. Ridders, "A new algorithm for computing a single root of a real continuous function," *IEEE Transactions on circuits and systems*, pp. 979–980, 1979.

[205] Python Software Foundation, *Python 2.7.9 documentation: multiprocessing – Process-based 'threading' interface*, 2015. [Online]. Available: https://docs.python.org/2/library/multiprocessing.html

[206] ——, *Python 2.7.9 documentation: Built-in constants*, 2015. [Online]. Available: https://docs.python.org/2/library/constants.html

[207] SymPy Development Team, *SymPy v0.7.1 documentation: mpmath numerical integration (quadrature)*, 2011. [Online]. Available: http://docs.sympy.org/0.7.1/modules/mpmath/calculus/integration.html

[208] J. Kaplan-Moss, "Python bindings to the OpenStack Compute API," 2011. [Online]. Available: https://github.com/jacobian-archive/openstack.compute

[209] J. Pellerin, "Python Package Index: nose 1.3.7," 2015. [Online]. Available: https://pypi.python.org/pypi/nose/1.3.7

[210] N. Batchelder, *Ned Batchelder: coverage.py*, 2013. [Online]. Available: http://nedbatchelder.com/code/coverage/

[211] ——, "Python Package Index: coverage 3.7.1," 2014. [Online]. Available: https://pypi.python.org/pypi/coverage/3.7.1

[212] Testing Cabal, "The python mock library," 2015. [Online]. Available: https://github.com/testing-cabal/mock

[213] ——, "Python Package Index: mock 1.3.0," 2015. [Online]. Available: https://pypi.python.org/pypi/mock/1.3.0

[214] C. Heisel, "nose-xcover: fork of nose.plugins.cover that enables XML output," 2014. [Online]. Available: https://github.com/cmheisel/nose-xcover/

[215] ——, "Python Package Index: nosexcover 1.0.10," 2014. [Online]. Available: https://pypi.python.org/pypi/nosexcover/1.0.10

[216] P. Marić, "nose_extra_tools: extra testing goodies for nose.tools," 2013. [Online]. Available: https://bitbucket.org/petar/nose_extra_tools

[217] ——, "Python Package Index: nose_extra_tools 1.0.0," 2013. [Online]. Available: https://pypi.python.org/pypi/nose_extra_tools/1.0.0

[218] ——, "beam_integrals Jenkins project: Build 70 Test Results." [Online]. Available: http://ci.petarmaric.com/job/beam_integrals/70/testReport/

[219] P. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.

[220] B. Cohen, *The BitTorrent protocol specification*, 2011. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[221] Python Software Foundation, *Python 2.7.9 documentation: doctest – Test interactive Python examples*, 2015. [Online]. Available: https://docs.python.org/2/library/doctest.html

Indeks slika

Indeks listinga

Indeks tabela

Biografija

Petar Marić rođen je 28.12.1984. godine u Čačku. Diplomirao je na Fakultetu tehničkih nauka u Novom Sadu, na studijskom programu Računarstvo i automatika – primenjene računarske nauke sa prosečnom ocenom 9,11. Diplomski-master rad na temu „Proširiv okvir za razvoj web baziranih rešenja namenjenih istraživanju stavova definisane populacije" odbranio je 28.05.2009. godine sa ocenom 10.

Na doktorske studije Fakulteta tehničkih nauka, studijski program Računarstvo i automatika – primenjene računarske nauke, upisao se školske 2009/2010 godine i položio sve ispite predviđene planom i programom sa prosečnom ocenom 10,00.

Od 28.09.2009. godine radi kao saradnik u nastavi na Fakultetu tehničkih nauka u Novom Sadu, a u zvanje asistenta izabran je 29.08.2011. godine. Od početka rada na fakultetu učestvuje u organizovanju i izvođenju vežbi iz sledećih predmeta: Arhitektura računara, Objektno programiranje, Programski jezici i strukture podataka, Programski prevodioci.

U toku studija i rada na fakultetu objavio je 5 naučnih radova, od čega 1 ranga M21 i 4 ranga M33. Od 2011. godine učesnik je projekta Ministarstva prosvete, nauke i tehnološkog razvoja „Računarska mehanika u teoriji konstrukcija" (OI 174027).

Oženjen je, otac jednog deteta i živi u Novom Sadu. Od stranih jezika govori engleski jezik.

Univerzitet u Novom Sadu • Fakultet tehničkih nauka
Ključna dokumentacijska informacija

Redni broj, **RBR**:	
Identifikacioni broj, **IBR**:	
Tip dokumentacije, **TD**:	Monografska dokumentacija
Tip zapisa, **TZ**:	Tekstualni štampani materijal
Vrsta rada, **VR**:	Doktorski rad
Autor, **AU**:	Petar Marić
Mentor, **MN**:	dr Žarko Živanov, docent
Naslov rada, **NR**:	Hibridna softverska arhitektura kao podrška primeni harmonijski spojenog metoda konačnih traka
Jezik publikacije, **JP**:	Srpski (latinica)
Jezik izvoda, **JI**:	Srpski (latinica), Engleski
Zemlja publikovanja, **ZP**:	Republika Srbija
Uže geografsko područje, **UGP**:	Vojvodina
Godina, **GO**:	2015
Izdavač, **IZ**:	Autorski reprint
Mesto i adresa, **MA**:	Novi Sad, Trg Dositeja Obradovića 6
Fizički opis rada, **FO**:	5 poglavlja, 231 strana, 221 citata, 66 tabela, 46 slika i 0 priloga
Naučna oblast, **NO**:	Elektrotehničko i računarsko inženjerstvo
Naučna disciplina, **ND**:	Primenjene računarske nauke
Predmetna odrednica, **PO**:	harmonijski spojen metod konačnih traka, korenovi karakterističnih jednačina, tačnost numeričkog sračunavanja, numerička analiza, metaprogramiranje
UDK	
Čuva se, **ČU**:	Biblioteka Fakulteta tehničkih nauka
Važna napomena, **VN**:	
Izvod, **IZ**:	Ova doktorska teza analizira problem rešavanja karakterističnih jednačina, koje se koriste prilikom rešavanja jednačina svojstvenih oblika, definisanih kroz harmonijski spojen metod konačnih traka. U slučaju složenijih graničnih uslova pokazano je da greške određivanja korenova karakteristične jednačine rastu gotovo eksponencijalno sa svakim narednim modom, usled prirode sâmih karakterističnih jednačina. Tako dobijeni korenovi karakteristične jednačine značajno i nepovoljno utiču na tačnost rešavanja jednačina svojstvenih oblika, izračunavanja određenih integrala, kao i celokupnih proračuna. Predstavljena je hibridna metoda putem koje se na pouzdan način određuju korenovi karakterističnih jednačina, rešavaju jednačine svojstvenih oblika i izračunavaju određeni integrali. Razvijen je prototip referentne Open Source implementacije hibridne metode, uz podsistem za automatsku verifikaciju koji rigorozno verifikuje karakteristike hibridne metode i njene referentne implementacije, za sve navedene granične uslove i integrale.
Datum prihvatanja teme, **DP**:	12.11.2015.
Datum odbrane, **DO**:	
Članovi komisije, **KO**:	
Predsednik:	dr Ivan Luković, redovni profesor
Član:	dr Miroslav Hajduković, redovni profesor
Član:	dr Dragan Milašinović, redovni profesor
Član:	dr Ilija Kovačević, redovni profesor
Član:	dr Dušan Malbaški, redovni profesor
Član, mentor:	dr Žarko Živanov, docent

University of Novi Sad • Faculty of technical sciences
Key words documentation

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	PhD Thesis
Author, **AU**:	Petar Marić
Mentor, **MN**:	Asst. Prof. Žarko Živanov, PhD
Title, **TI**:	A hybrid software architecture for supporting the harmonic coupled finite strip method
Language of text, **LT**:	Serbian (latin)
Language of abstract, **LA**:	Serbian (latin), English
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2015
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Trg Dositeja Obradovića 6
Physical description, **PD**:	5 chapters, 231 pages, 221 references, 66 tables, 46 figures and 0 appendix
Scientific field, **SF**:	Electrical and computer engineering
Scientific discipline, **SD**:	Applied computer science
Subject/Key words, **S/KW**:	harmonic coupled finite strip method, roots of characteristic equations, accuracy of numerical evaluation, numerical analysis, metaprogramming
UC	
Holding data, **HD**:	The Library of Faculty of technical sciences
Note, **N**:	
Abstract, **AB**:	This PhD thesis analyzes the problem of solving the characteristic equations of the basic functions, as defined by the harmonic coupled finite strip method. It's found that with each increasing mode the characteristic equation root-finding error grows exponentially for all but the most trivial edge boundary conditions, due to the hyperbolic functions involved. These large root-finding errors will lead to severe accuracy issues when computing basic functions and their integrals, especially for higher modes. A hybrid method for accurately solving characteristic equations and obtaining the required integrals is presented, along with its reference Open Source implementation. An extensive test suite has been developed to verify the hybrid method and its implementation for all the presented boundary conditions and integrals.
Accepted by the Scientific Board on, **ASB**:	2015-11-12
Defended on, **DE**:	
Defended Board, **DB**:	
President:	Prof. Ivan Luković, PhD
Member:	Prof. Miroslav Hajduković, PhD
Member:	Prof. Dragan Milašinović, PhD
Member:	Prof. Ilija Kovačević, PhD
Member:	Prof. Dušan Malbaški, PhD
Member, Mentor:	Asst. Prof. Žarko Živanov, PhD