2. projekt pri predmetu MATEMATIČNO MODELIRANJE

Presek dveh implicitno danih ploskev

Aljaž Verlič, Lina Lumborovska, Blažka Blatnik, Luka Tavčer
Mentor: Damir Franetič

5. junij, 2017

Vsebina

1 Predstavitev problema

V $\mathbb{R}^3$ imamo podani dve poljubni implicitno dani ploskvi, opisani z enačbama $f_1(x) = C_1$ in $f_2(x) = C_2$. Presek teh dveh ploskev je množica rešitev nelinearnega sistema enačb:

$$f_1(x) = C_1,$$
$$f_2(x) = C_2.$$

Naša naloga je poiskati krivuljo K (oziroma točke na njej), ki predstavlja presek teh dveh ploskev.

2 Opis modela in uporabljenih metod

Sistem lahko gledamo tudi, kot enačbe nivojnic funkcij f_1 in f_2, krivulja K pa je presek teh nivojnic. Gradienta funkcij sta tako v vsaki točki krivulje preseka pravokotna nanjo. To opišemo:

$$F(x) = \frac{(gradf_1(x)) \times (gradf_2(x))}{\|gradf_1(x)) \times (gradf_2(x))\|}$$

in označimo z $x = x(t)$ naravno parametrizacijo krivulje K. Ta x je rešitev avtonomnega sistema diferencialnih enačb:

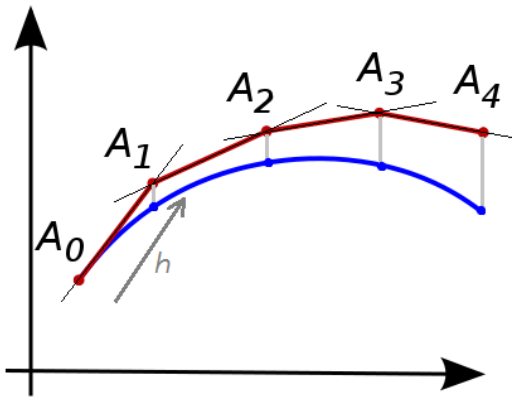
$$\dot{x} = F(x)$$

2.1 Reševanje sistema diferencialnih enačb

Za reševanje sistema diferencialnih enačb lahko uporabimo katero izmed dveh znanih metod za numerično reševanje diferencialnih enačb.

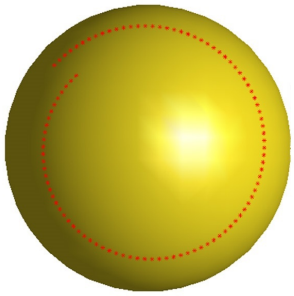
2.1.1 Eulerjeva metoda

Eulerjeva metoda je numerična metoda za reševanje diferencialnih enačb, z podanim začetnim približkom. Prednost metode je, da je preprosta in najbolj logična. Na vsakem koraku naslednjo točko (x_{i+1}, y_{i+1}) dobimo tako, da se za korak h premaknemo vzdolž tangente na rešitev (x_i, y_i). Geometrijsko lahko delovanje metode predstavimo s spodnjo sliko.

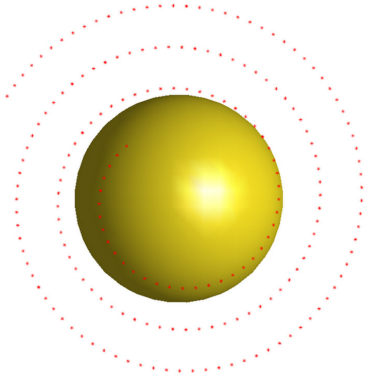


Slika 1: Geometrijski prikaz delovanja Eulerjeve metode.

Slabost opisanega postopka je napaka, ki se skozi iteracije izvajanja metode povečuje (napaka na vsakem koraku metode je reda $O(h^2)$, kumulativna napaka pa z vsako iteracijo narašča). Tako je napaka večja, tem večji je korak. V iskanju rešitve našega problema to predstavlja težavo, zato moramo vsak izračunan približek vedno popraviti tako, da spet leži na krivulji preseka. Na preprostem primeru iskanja presečišč valja in sfere lahko opazimo delovanje Eulerjeve metode in problem kumulativne napake (valja na sliki zaradi večje preglednosti ni)



Slika 2: Eulerjeva metoda z manjšim korakom



Slika 3: Eulerjeva metoda z večjem korakom

2.1.2 Metoda Runge-Kutta 4

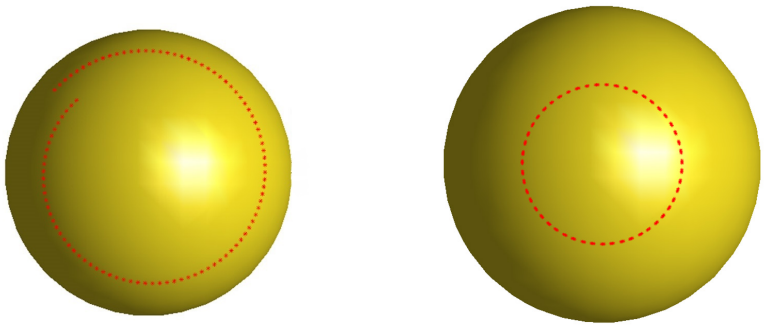
Precej bol natančna, a manj intuitivna metoda za reševanje diferencialnih enačb, je metoda Runge-Kutta 4. Tudi tu potrebujemo nek začetni približek, metoda pa potem z večjo natančnostjo računa nadaljne premike. Kumulativna napaka je konstantna in se za razliko od Eulerjeve z iteracijami ne povečuje.

2.1.3 Newtonova metoda za popravljanje približkov

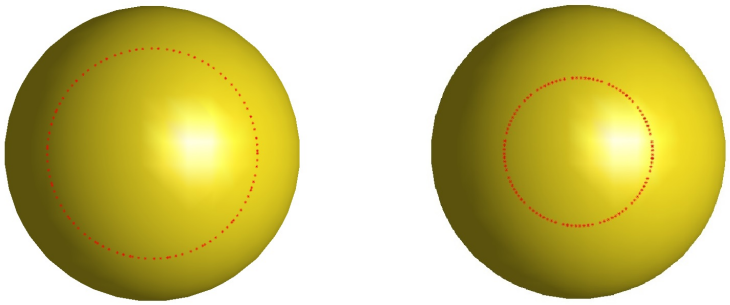
Pri uporabi Eulerjeve metode je potrebno sprotno popravljanje približkov, saj se drugače napake seštevajo do te mere, da ne dobimo željene rešitve. Dobljeni približek y, želimo popraviti na nek x, ki bo ležal na preseku. Če zapišemo $F(y)\cdot x = F(y)\cdot y$, nam to predstavlja enačbo ravnine, ki je zelo blizu normalni ravnini na krivuljo K. Z Newtonovo metodo z začetnim približkom y rešimo sistem enačb:

$$\begin{aligned} f_1(x) &= C_1 \\ f_2(x) &= C_2 \\ F(y)\cdot x &= F(y)\cdot y \end{aligned}$$

Rešitev sistema je točka, ki leži na presečišču obeh ploskev. V primeru uporabe RK4 je Newtonova metoda bolj potrebna predvsem za postavitev začetnega približka na presečišče, saj je metoda RK4 že sama po sebi precej natančna. Učinkovitost metode je razvidna iz primerov:



Slika 4: Osnovna Eulerjeva metoda (levo) in popravljena Eulerjeva metoda (desno).



Slika 5: Osnovna metoda RK4 (levo) in popravljena metoda RK4 (desno).

3 Potreben pogoj in Jacobijeva matrika

Potreben pogoj za delovanje metod je, da sta funkciji f_1 in f_2 parcialno odvedljivi in da ima Jacobijeva matrika parcialnih odvodov poln rang 2. Za uspešno delovanje Newtonove metode moramo poiskati Jacobijevo matriko leve strani sistema nelinearnih enačb:

$$\text{JG} = \begin{bmatrix} grad(f_1) \\ grad(f_2) \\ grad(\vec{v} \cdot \vec{x}) \end{bmatrix} \text{ oziroma JG} = \begin{bmatrix} grad(f_1) \\ grad(f_2) \\ grad(\vec{v}^\intercal) \end{bmatrix}$$

Newtnova metoda rešuje sistem:

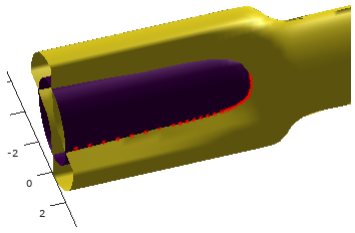
$$\vec{F}(\vec{x}) = \vec{0},$$

zato je potrebno poiskati tudi matriko $\vec{F}(\vec{x})$, ki je enaka:

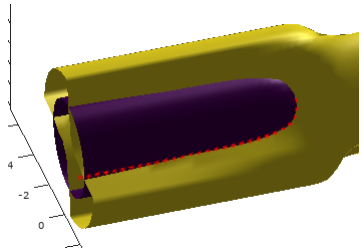
$$\vec{F}(\vec{x}) = \begin{bmatrix} f_1(\vec{v}) - C_1 \\ f_2(\vec{v}) - C_2 \\ \vec{v} \cdot \vec{x} - \vec{v} \cdot \vec{y}) \end{bmatrix}$$

4 Adaptivni korak

Adaptivni korak omogoča natančnejšo rešitev problema, saj dinamično prilagaja dolžino koraka. Tako lahko na bolj preprostih predelih preseka uporabljamo večji korak in je metoda hitrejša, na kompleksnejših delih preseka, kjer je potrebna večja natančnost, pa se premikamo z manjšim korakom. Program poveča oziroma zmanjša korak glede na število korakov, ki jih izvede Newtonova metoda za eno točko. Empirično smo določili, da je velikost koraka h na intervalu $h \in [10^{-7}, 100]$ in da se korak povečuje oziroma zmanjšuje s faktorjem 2. Korak se ne spremeni, če je število korakov enako 2 ali 3.



Slika 6: Adaptivni korak



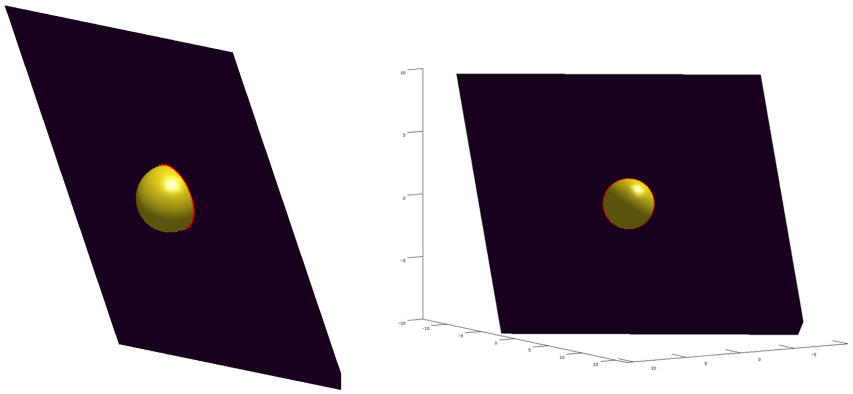
Slika 7: Brez adaptivnega koraka

5 Implementacija, testiranje in primeri

Delovanje našega programa lahko preverimo s programom, ki smo ga napisali v Octave-u. Kot vhodne parametre mu podamo obe implicitno podani funkciji f_1, f_2, $C1$, $C2$, $grad(f_1)$, $grad(f_2)$. Določimo tudi začetni približek x_0, začetno dolžino koraka in pa parameter, ki določa metodo delovanja (Euler/Runge-Kutta).
Program poženemo na različnih primerih in štejemo povprečno dolžino koraka ter število porabljenih korakov:

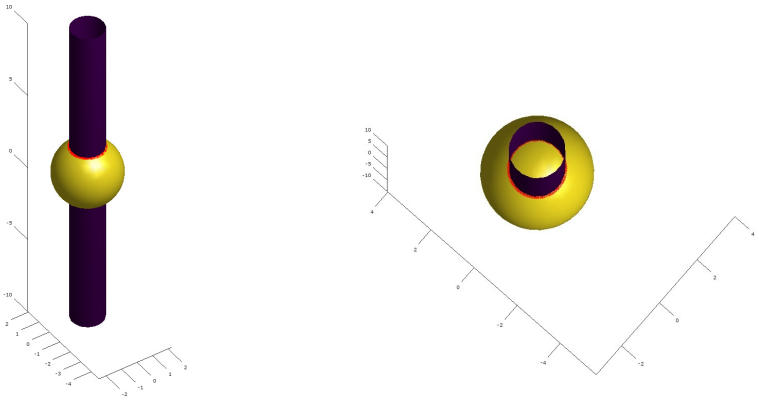
Primer 1:

- $f_1(x, y, z) = x^2 + y^2 + z^2 = 4$
- $f_2(x, y, z) = 3x + 2y + z = 1$



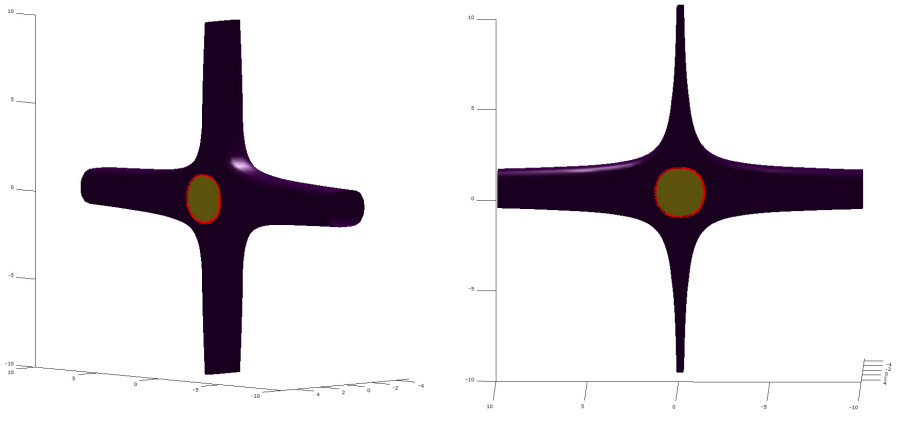
Primer 2:

- $f_1(x, y, z) = x^2 + y^2 + z^2 = 4$
- $f_2(x, y, z) = x^2 + y^2 = 1$



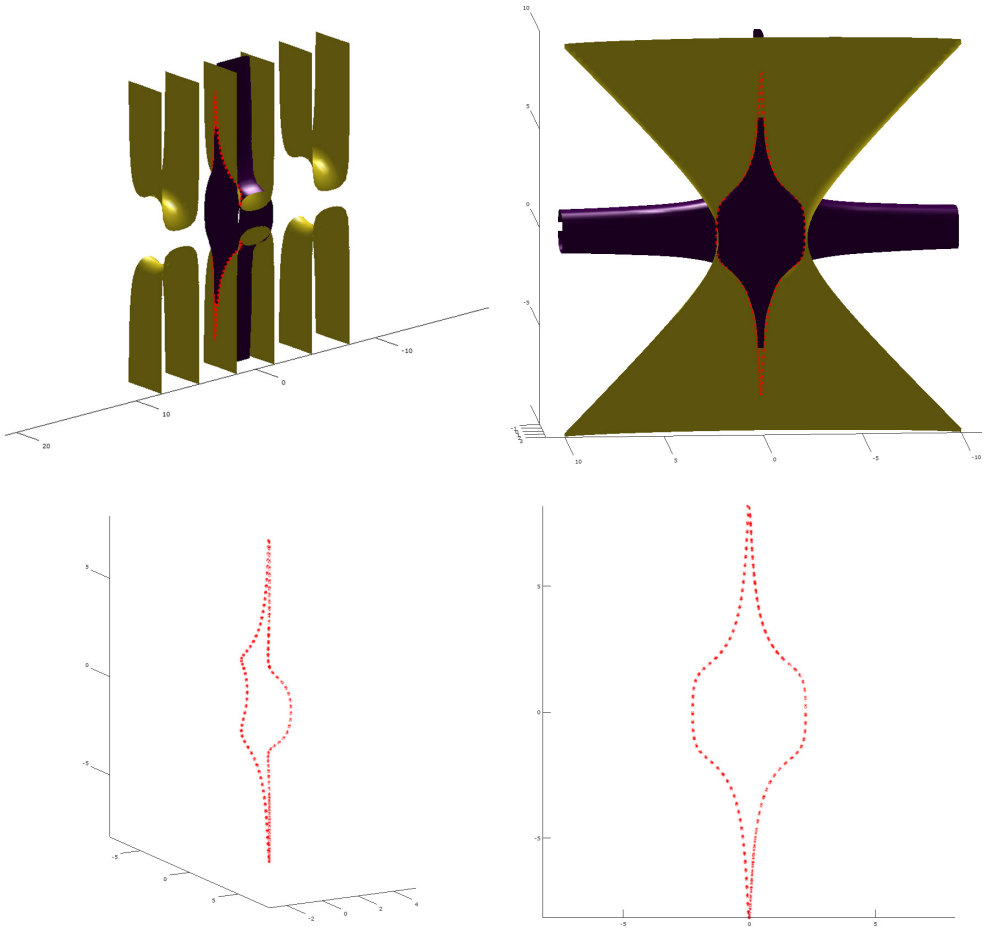
Primer 3:

- $f_1(x, y, z) = x^2 + y^2 + z^2 = 4$
- $f_2(x, y, z) = y^4 + log(x^2 + 1)z^2 - 4 = 1$



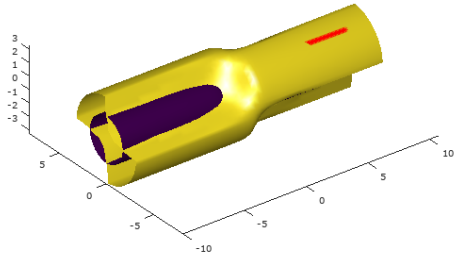
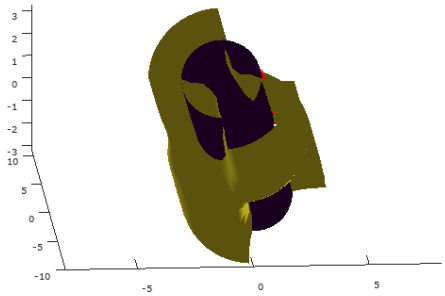
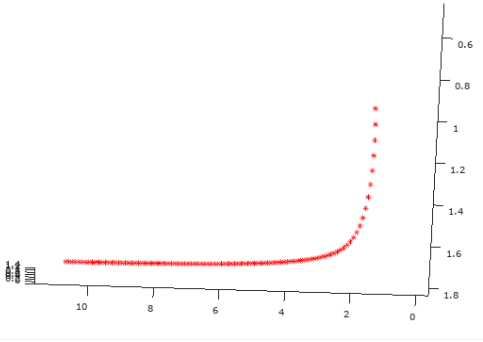
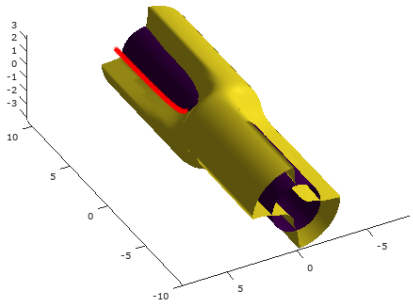
Primer 4:

- $f_1(x, y, z) = x^2 + cos(y)z^2 - 12 = 4$
- $f_2(x, y, z) = y^4 + log(x^2 + 1)z^2 - 4 = 1$



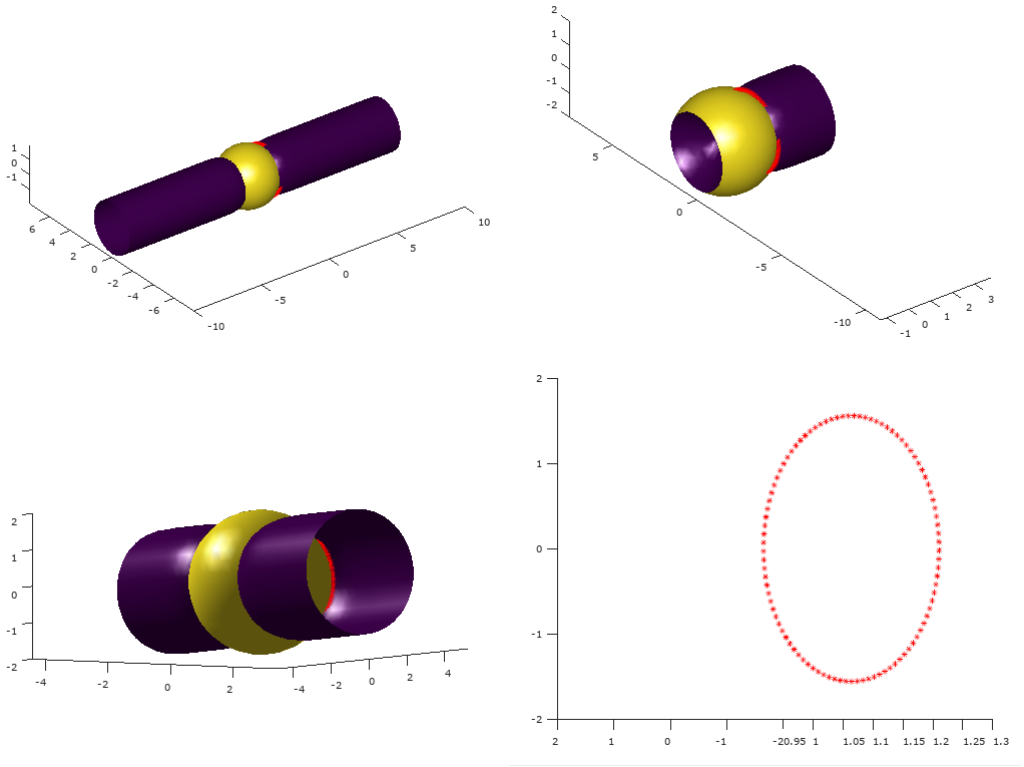
Primer 5:

- $f_1(x, y, z) = e^{(-x^2+1)} + y^2 + z^2 = 3$
- $f_2(x, y, z) = e^{(xyz)} + y^2 + z^2 = 10$

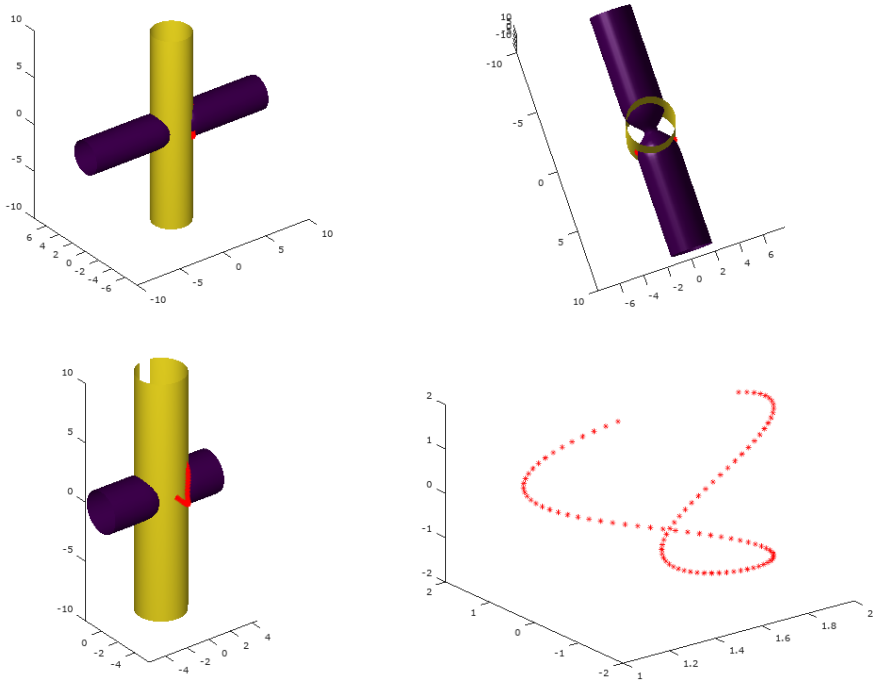
Primer 6:

- $f_1(x, y, z) = e^{(-x^2+1)} + y^2 + z^2 = 3$
- $f_2(x, y, z) = x^2 + y^2 + z^2 = 4$



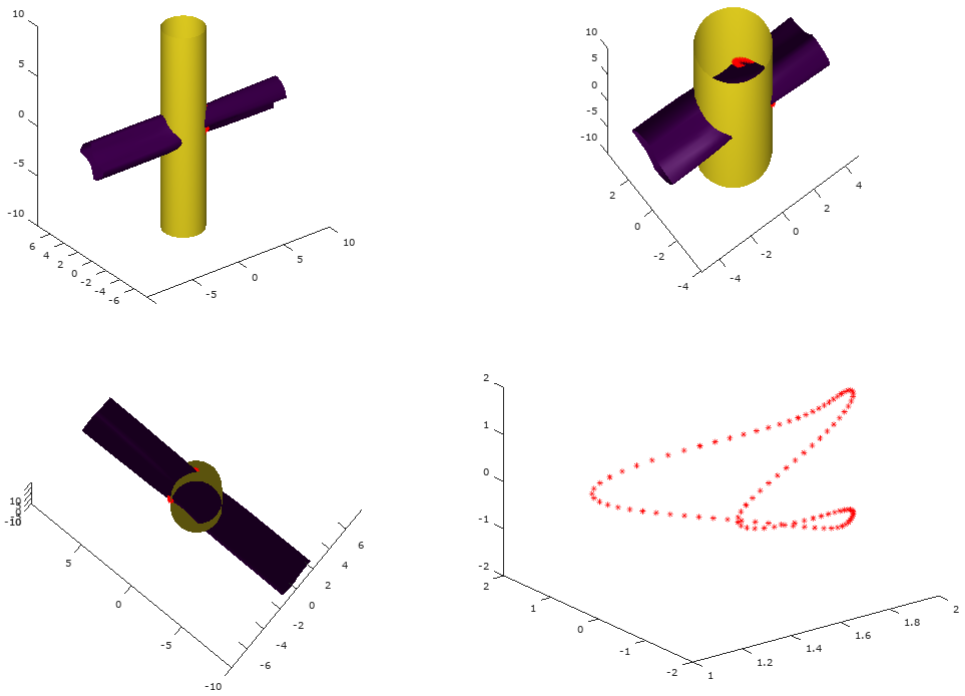
Primer 7:

- $f_1(x, y, z) = e^{(-x^2+1)} + y^2 + z^2 = 3$
- $f_2(x, y, z) = x^2 + y^2 = 1$



Primer 8:

- $f_2(x, y, z) = e^{(xyz)} + y^2 + z^2 = 10$
- $f_2(x, y, z) = x^2 + y^2 = 1$



6 Analiza povprečnega števila korakov Newtonove metode

Na koncu smo za vseh 8 primerov naredili analizo, tako da smo izmerili povprečno število korakov Newtonove metode za RK4 in Eulerjevo matodo na dva načina: z adaptivnim in fiksnim korakom. Pri adaptivnem koraku smo izračunali tudi minimalno in maksimalno število korakov.

Primer in rezultati izvajanje enega testa:

```
>> zazeniPrimer(5,10,1,1,[0,10])
* ------------------- *
 REZULTATI:
Primer: 5, Euler , adaptivni
 Cas: 9.90702
 Dolzina koraka: min = 0.138889 / max = 0.138889
 Povprecno st. korakov Newtonove metode: 3
* ------------------- *
>> zazeniPrimer(5,10,1,0,[0,10])
* ------------------- *
 REZULTATI:
Primer: 5, Euler , fiksni
 Cas: 7.31774
 Dolzina koraka: 1.11111
 Povprecno st. korakov Newtonove metode: 4.66667
* ------------------- *
>> zazeniPrimer(5,10,0,1,[0,10])
* ------------------- *
 REZULTATI:
Primer: 5, RK4 , adaptivni
 Cas: 7.40928
 Dolzina koraka: min = 1.11111 / max = 1.11111
 Povprecno st. korakov Newtonove metode: 2.22222
* ------------------- *
>> zazeniPrimer(5,10,0,0,[0,10])
* ------------------- *
 REZULTATI:
Primer: 5, RK4 , fiksni
 Cas: 7.24917
 Dolzina koraka: 1.11111
 Povprecno st. korakov Newtonove metode: 2.66667
* ------------------- *
```

Po izvajanju tega postopka za vse primere smo dobili naslednje rezultate:

Funkcija	**Euler**		**RK4**	
	adaptivno	fiksno	adaptivno	fiksno
$f_1(x,y,z) = x^2 + y^2 + z^2 = 4$ $f_2(x,y,z) = x^2 + y^2 = 1$	3	5	3	3.222
$f_1(x,y,z) = x^2 + y^2 + z^2 = 4$ $f_2(x,y,z) = 3x + 2y + z = 1$	3	4.222	3	3.333
$f_1(x,y,z) = x^2 + y^2 + z^2 = 4$ $f_2(x,y,z) = y^4 + log(x^2+1)z^2 - 4 = 1$	3	5	3	3.222
$f_1(x,y,z) = x^2 + cos(y)z^2 - 12 = 4$ $f_2(x,y,z) = y^4 + log(x^2+1)z^2 - 4 = 1$	3	4.111	2.222	2.556
$f_1(x,y,z) = e^{(-x^2+1)} + y^2 + z^2 = 3$ $f_2(x,y,z) = e^{(xyz)} + y^2 + z^2 = 10$	3	4.667	2.222	2.667
$f_1(x,y,z) = e^{(-x^2+1)} + y^2 + z^2 = 3$ $f_2(x,y,z) = x^2 + y^2 + z^2 = 4$	3	5	3	3.222
$f_1(x,y,z) = e^{(-x^2+1)} + y^2 + z^2 = 3$ $f_2(x,y,z) = x^2 + y^2 = 1$	3	5	3	3.333
$f_1(x,y,z) = e^{(xyz)} + y^2 + z^2 = 10$ $f_2(x,y,z) = x^2 + y^2 = 1$	3	5	3	3.556

Pri bolj kompleksnih funkcijah kot so npr.:

- $f_1(x,y,z) = x^2 + cos(y)z^2 - 12 = 4$ in
- $f_2(x,y,z) = y^4 + log(x^2+1)z^2 - 4 = 1$,

smo dobili pri adaptivnem koraku manjše povprečno število korakov, ker je metoda hitro konvergirala, vendar za to ceno zmanjšala korak in delovala počasneje (približek je bil blizu pravilne rešitve, saj je bil premik majhen). Tudi minimalno in maksimalno število korakov sta bila manjša kot pri vseh ostalih. Adaptivni korak se pri teh funkcijah približa spodnji meji (2), dolžina koraka pa se manjša.

Pri fiksnem koraku smo prišli do podobnega zaključka kot pri adaptivnem, da bolj kompleksne funkcije hitreje konvergirajo, saj mora biti korak manjši (ni prikazano v tabeli).

7 Koda

8 Delitev dela v skupini

8.1 Programerski del

- Aljaž: Adaptivni korak, test skripta
- Lina: Reševanje primerov, empirično določanje parametrov
- Blažka: Ogrodje programa, združitev funkcij, test skripta
- Luka: Reševanje primerov, empirično določanje parametrov

8.2 Poročilo

- Aljaž: Adaptivni korak, Jacobijeva matrika
- Lina: Analiza, primeri, testiranje
- Blažka: Predstavitev problema, opis modela in metod
- Luka: Opis modela in metod (primeri delovanja)

9 Reference

- Zapiski s predavanj: Diferencialne enačbe
- `https://en.wikipedia.org/wiki/Euler_method`