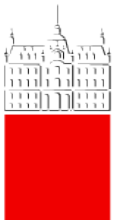
Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*



Enačbe hoda

6. naloga pri Matematično-fizikalnem praktikumu

Avtor: Marko Urbanč (28191096)
Predavatelj: prof. dr. Borut Paul Kerševan

20.11.2021

Kazalo

1 Uvod

Pri opisu fizikalnih pojavov pogosto srečamo navadne diferencialne enačbe, ki povezujejo vrednosti spremenljivk sistema z njihovimi časovnimi spremembami. Zelo preprost primer tega je enačba za časovno odvisnost temperature v stanovanju, ki je obdano z stenami z neko toplotno prevodnostji in ima zunaj določeno temperaturo. Enačba se glasi:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k\left(T - T_{\mathrm{zun}}\right) \tag{1}$$

To enačbo lahko analitično rešimo z separacijo spremenljvk. Analitična rešitev je:

$$T(t) = T_{\mathrm{zun}} + \mathrm{e}^{-kt}\left(T(0) - T_{\mathrm{zun}}\right) \ .$$

Enačbam, ki opisujejo razvoj spremenljivk sistema y po času oz. drugi neodvisni spremenljivki pravimo *enačbe hoda*. Za njihovo rešitev potrebujemo seveda še ustrezni začetni pogoj. Tako rešujemo *problem začetne vrednosti* (angl. Initial value problem). Najbolj groba numerična metoda za reševanje tega problema je **Eulerjeva** metoda, ki je pravzaprav le prepisana aproksimacija za prvi odvod:

$$y(x+h) = y(x) + h \left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_x \ . \tag{2}$$

S tem smo diferencialno enačbo prepisali v diferenčno, kjer sistem spremljamo v ekvidistančnih korakih dolžine h. Za natančnost metode moramo ustrezno zmanjšati korak. Za red natančnosti boljša metoda je **simetrizirana Eulerjeva**, ki sledi iz simetriziranega približka za prvi odvod $y' \approx (y(x+h) - y(x-h))/2h$. Računamo po shemi

$$y(x+h) = y(x-h) + 2h \left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_x \ , \tag{3}$$

ki pa je praviloma nestabilna. Želeli bi si pravzaprav nekaj takega

$$y(x+h) = y(x) + \frac{h}{2}\left[\left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_x + \left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_{x+h}\right] \ , \tag{4}$$

le da to pot ne poznamo odvoda v končni točki intervala (shema je implicitna). Pomagamo si lahko z iteracijo. Odvod lahko zapišemo kot:

$$\left.\frac{dy}{dx}\right|_x = f(x,y); \quad x_{n+1} = x_n + h, \quad y_n = y(x_n)$$

Heunova metoda je približek idealne formule z:

$$\hat{y}_{n+1} = y_n + h \cdot f(x_n, y_n) \ ,$$
$$y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, \hat{y}_{n+1})\right] \ .$$

Izvedenka tega je nato **Midpoint** metoda (tudi $\mathcal{O}(h^3)$ lokalno):

$$\begin{aligned} k_1 &= f(x_n, y_n)\,, \\ k_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}\,h\,k_1)\,, \\ y_{n+1} &= y_n + h\,k_2\,. \end{aligned}$$

Nadaljno pa seveda obstaja praktično poljubno mnogo možnih modifikacij. Dandanes so v praktični rabi druge metode zaradi željene natančnosti in numerične učinkovitosti. Uporabljajo se metode zasnovane na algoritmih prediktor-korektor, metode višjih redov iz družine Runge-Kutta ali ekstrapolacijske metode. Zelo priljubljena metoda je **Runge-Kutta 4**, katere postopek se glasi:

$$\begin{aligned} k_1 &= f\left(x,\, y(x)\right)\,, \\ k_2 &= f\left(x + \tfrac{1}{2}h,\, y(x) + \tfrac{h}{2}k_1\right)\,, \\ k_3 &= f\left(x + \tfrac{1}{2}h,\, y(x) + \tfrac{h}{2}k_2\right)\,, \\ k_4 &= f\left(x + h,\, y(x) + hk_3\right)\,, \\ y(x+h) &= y(x) + \tfrac{h}{6}\,\left(k_1 + 2k_2 + 2k_3 + k_4\right) + \mathcal{O}(h^5)\,. \end{aligned}$$

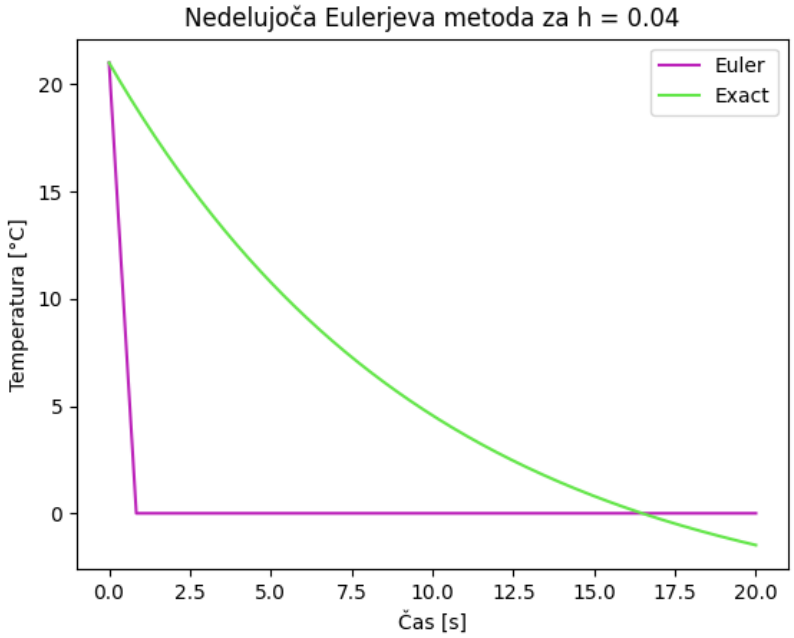
2 Naloga

Naloga od nas zahteva da enačbo (1) rešimo z Eulerjevo metodo ter še s čim več naprednejšimi metodami na primeru z začetnima temperaturama $T(0) = 21$ ali $T(0) = -15$, z zunanjo temperaturo $T_{zun} = -5$ in parametrom $k = 0.1$. Želi, da raziščemo kako je z velikostjo koraka h in da naj na koncu izračunamo družino rešitev za različne vrednosti parametra k.

3 Opis reševanja

Naloge sem se lotil v Pythonu z pomočjo paketov `matplotlib` za risanje in `Numpy` za vse numerične postopke. Pogledal sem si tudi vgrajeno `scipy.integrate.odeint()`. Dodatno je profesor dodal že nekaj spisanih rutin za numerično reševanje navadnih diferencialnih enačb v datoteki `diffeq.py`.

Z že spisanimi metodami sem imel nekoliko težave, ker z izjemo Runge-Kutta-Fehlberg niso delovale pravilno. Nekaj je bilo nenavadno, kar se tiče velikosti koraka. Manjši kot je bil korak, slabša je bila natančnost, kjer je bila najbolša dobljena natačnost, če je bilo točk na prste prešteto, tudi pravzaprav slaba.



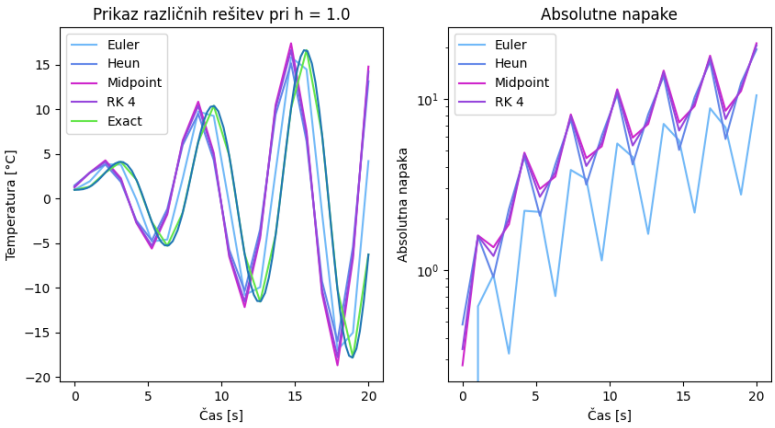
Slika 1: Čudno obnašanje Eulerjeve metode iz `diffeq.py`

Zato sem za Eulerjevo, Heunovo, Midpoint in RK4 metodo sprogramiral svoje rutine po podani psevdokodi in z pomočjo `diffeq.py`. Poskusno si zastavimo izziv rešiti enačbo:

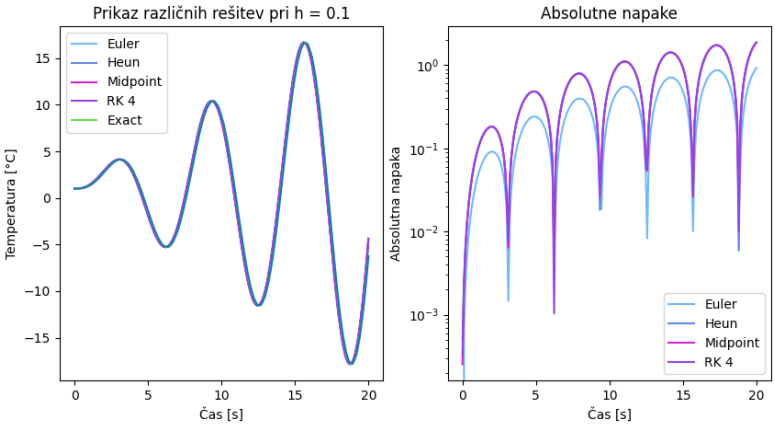
$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= t \sin t \\ x(0) &= 1 \end{aligned} \tag{5}$$

Ta ima analitično rešitev:

$$x(t) = \sin t - t \cos t + x(0)$$



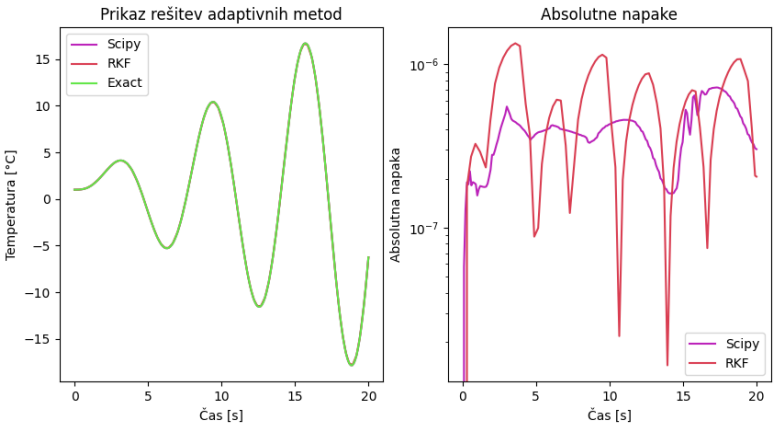
Slika 2: Rešitev enačbe (5) pri velikosti koraka $h = 1$



Slika 3: Rešitev enačbe (5) pri velikosti koraka $h = 0.1$

Vidimo, da ima velikost koraka znaten vpliv na natančnost rešitev. Torej ga lahko prilagodimo dokler ne dobimo željene natančnosti. Zanimivo mi je, da ima v tem primeru Eulerjeva metoda pravzaprav najmanjšo napako (dodatni dvomi okoli napake izraženi v 4.2). Za osnovno reševanje je to vse kar je pravzaprav potrebno. Zato so ostale rešitve prikazane v 4.1.

Pogledal sem si še kako je z naprednejšimi metodami, ki imajo adaptivni korak. To pomeni, da imajo sposobnost oceniti za vsako naslednjo točko, kolikšna bo najboljša velikost koraka. Uporabil sem že sprogramirano Runge-Kutta-Fehlberg metodo iz `diffeq.py` in vgrajeno funkcijo `scipy.integrate.odeint()`, ki rešuje po **LSODA** metodi, ki ima več različnih načinov in režimov delovanja [1].



Slika 4: Rešitev enačbe (5) z adaptivnimi metodami

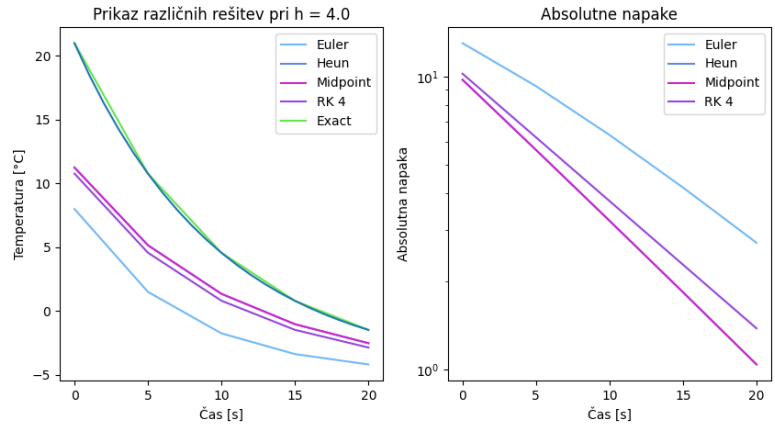
4 Rezultati

4.1 Rešitve različnih metod

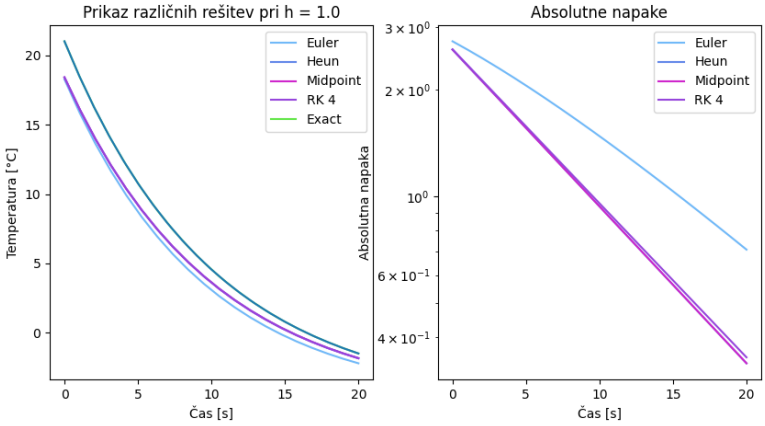
V prvem primeru rešujemo enačbo:

$$\begin{aligned} \frac{\mathrm{d}T}{\mathrm{d}t} &= -0.1(T+5) \\ T(0) &= 21 \end{aligned} \tag{6}$$

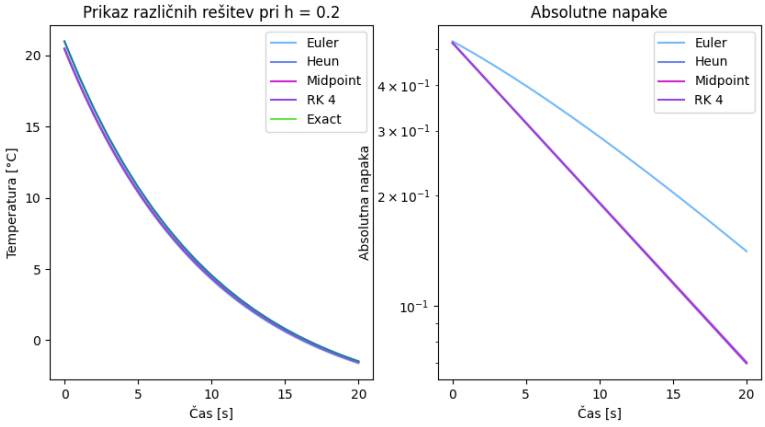
Rešitve sem izračunal pri različnih velikostih koraka h. Korak lahko pravzaprav manjšamo poljubno, da dosežemo željeno natančnost ob ceni numerične hitrosti računanja.



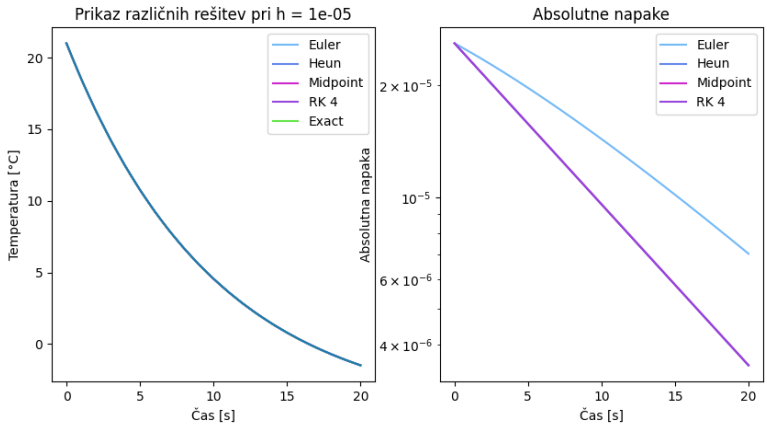
Slika 5: Rešitev enačbe (6) pri velikosti koraka $h = 4$



Slika 6: Rešitev enačbe (6) pri velikosti koraka $h = 1$



Slika 7: Rešitev enačbe (6) pri velikosti koraka $h = 0.2$

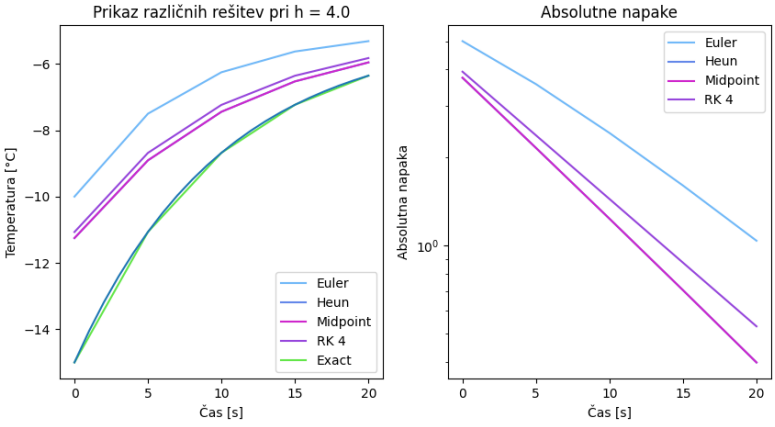


Slika 8: Rešitev enačbe (6) pri velikosti koraka $h = 10^{-5}$

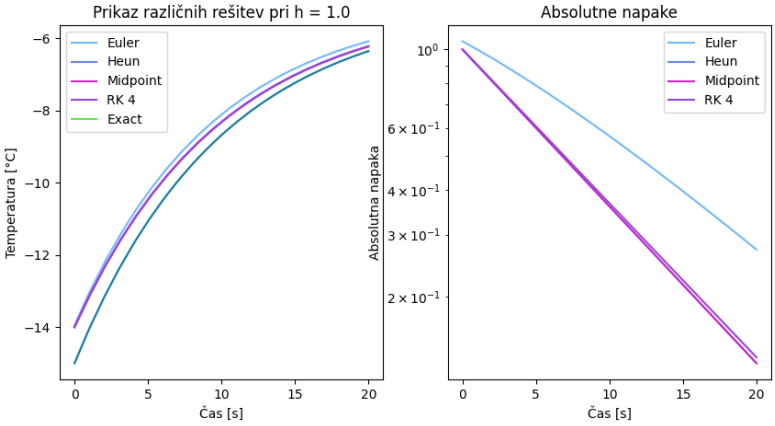
Vidimo, da temperatura res eksponentno pada s časom in se asimptotsko bliža T_{zun}. Pričakovano. Hiša se hladi. Po popolnoma enakem postopku lahko rešimo tudi primer z drugo začetno vrednostjo:

$$\begin{aligned} \frac{\mathrm{d}T}{\mathrm{d}t} &= -0.1(T+5) \\ T(0) &= -15 \end{aligned} \tag{7}$$

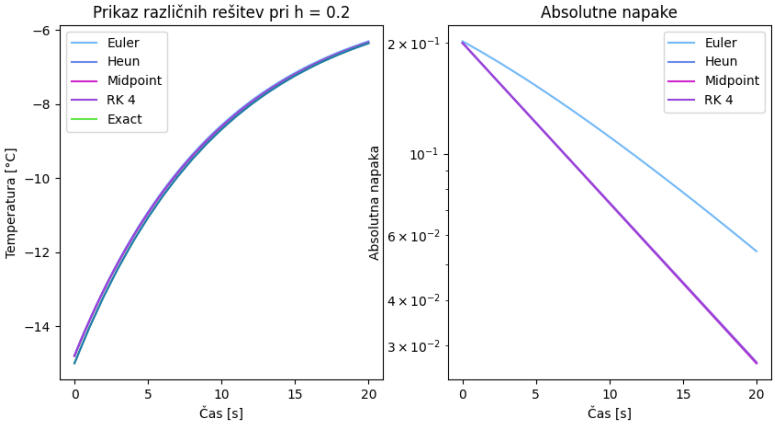
Tu pa pričakujemo logaritmično naraščanje temperature, ki se asimptotsko bliža T_{zun}. Hiša se segreva iz okolice.



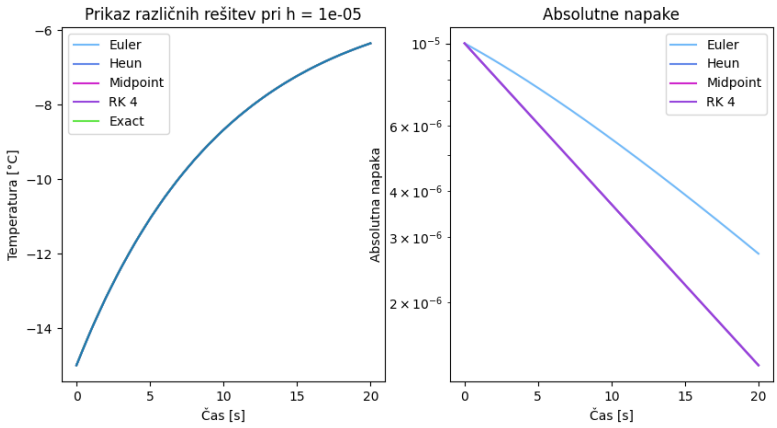
Slika 9: Rešitev enačbe (7) pri velikosti koraka $h = 4$



Slika 10: Rešitev enačbe (7) pri velikosti koraka $h = 1$



Slika 11: Rešitev enačbe (7) pri velikosti koraka $h = 0.2$

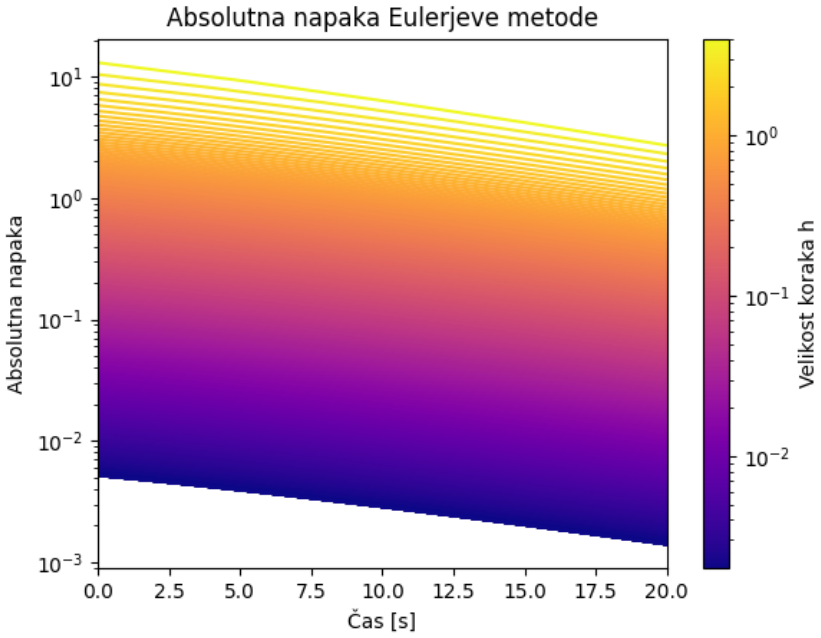


Slika 12: Rešitev enačbe (7) pri velikosti koraka $h = 10^{-5}$

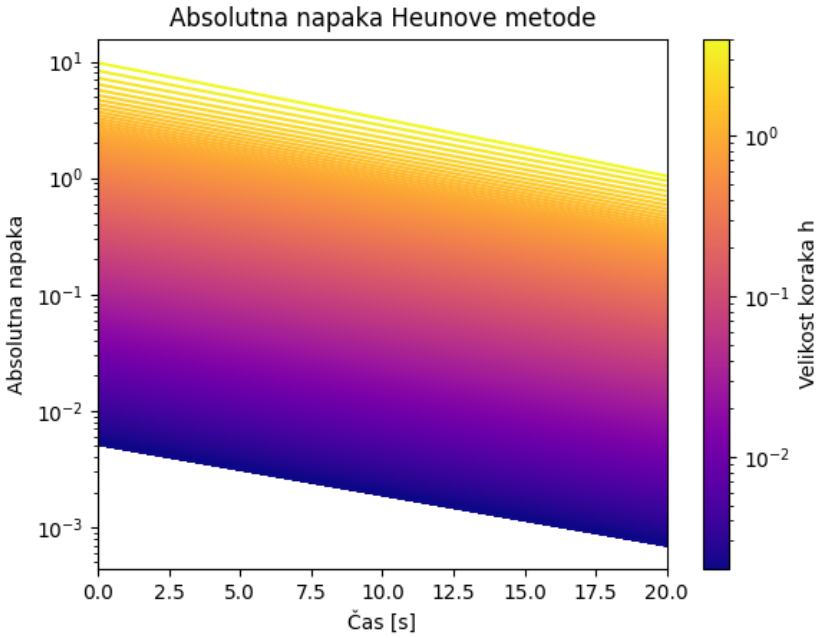
4.2 Natančnost in čas računanja pri različnih velikosti h

Podrobneje sem si želel pogledati, kako je z natančnostjo in časom računanja, ko se manjša korak. Podatke sem izračunal za nekaj metod za h na intervalu:

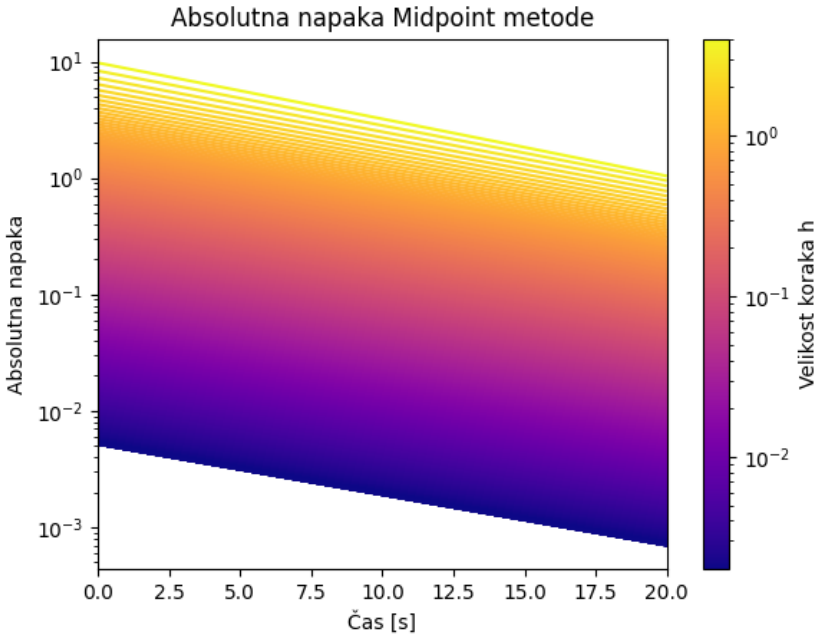
$$h \in [0.002,\ 4]$$



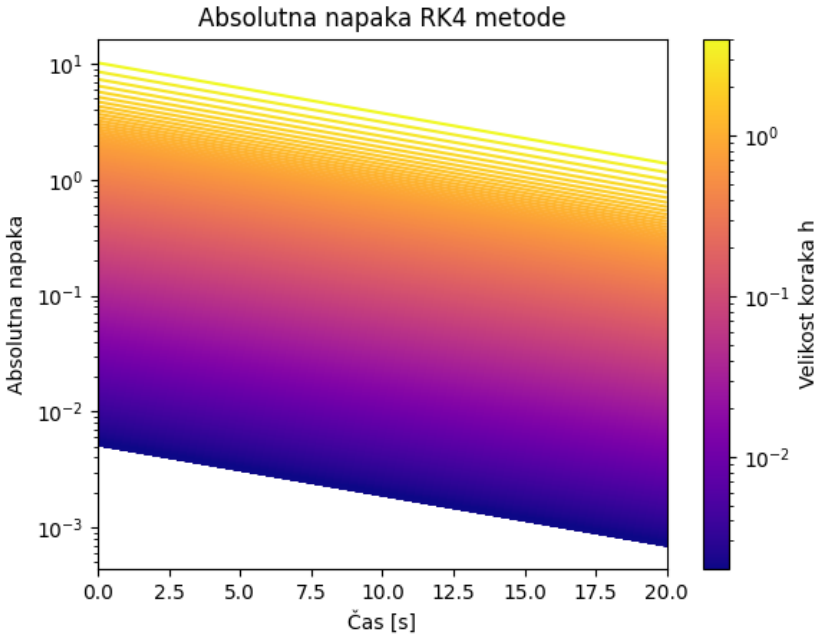
Slika 13: Natančnost reševanja (6) z Eulerjevo metodo



Slika 14: Natančnost reševanja (6) z Heunevo metodo



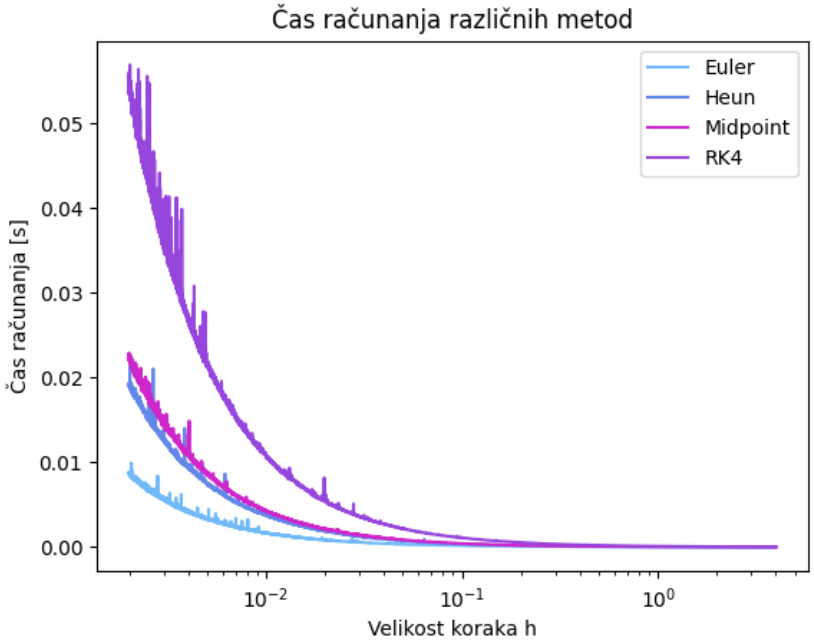
Slika 15: Natančnost reševanja (6) z Midpoint metodo



Slika 16: Natančnost reševanja (6) z RK4 metodo

Zanimivo mi je, da so si napake med metodami tako podobne. Večkrat sem preveril svojo kodo in funkcije, ker sem začel dvomiti v pravilnost svojih metod. Napak nisem našel. Torej je očitno tako? Metode so si po napakah in manjšanju napake z manjšanjem koraka podobne. Kar pa se spremeni je njihov čas računanja.

Za "štopanje" časa računanja funkcije sem uporabil Pythonovo knjižnico `time`. Prvotno sem uporabljal ukaz `time.time()`, ki ima natančnost le 1 ms. Tako so se mi pojavili diskretizirani nivoji. Natančnost ure sem poskusil izboljšati z uporabo naprednejšega ukaza `time.time_ns()`, ki ima resolucijo 1 ns. Vseeno je prišlo do nekakšne čudne diskretizacije časov, ki si je ne znam razložiti. Na koncu sem uporabil `time.clock()`, ki naj bi vrnila trenuten procesorski čas in je najbolj primerna za merjenje časov računanja funkcij. Prejšnje težave z diskretizacijo so se nekoliko popravile.

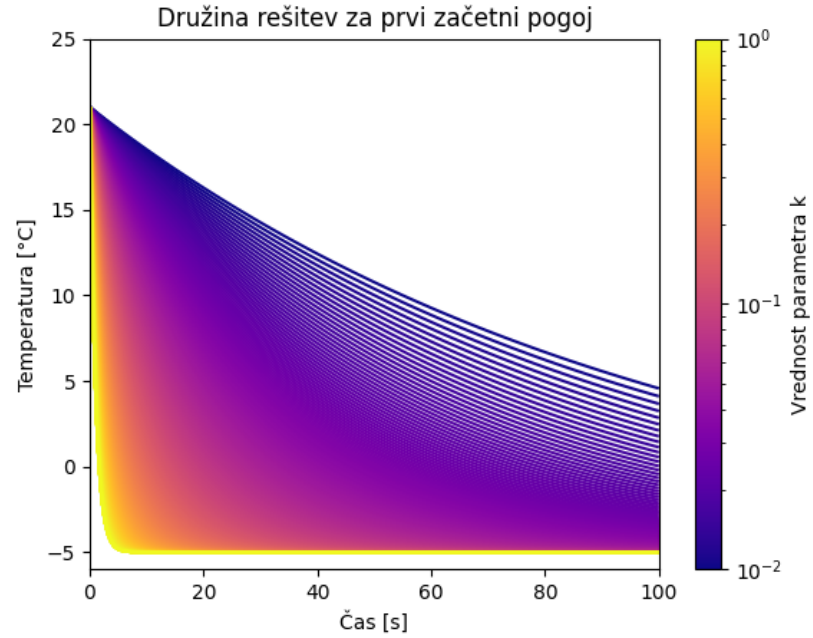


Slika 17: Odvisnost časa računanja od velikosti koraka h

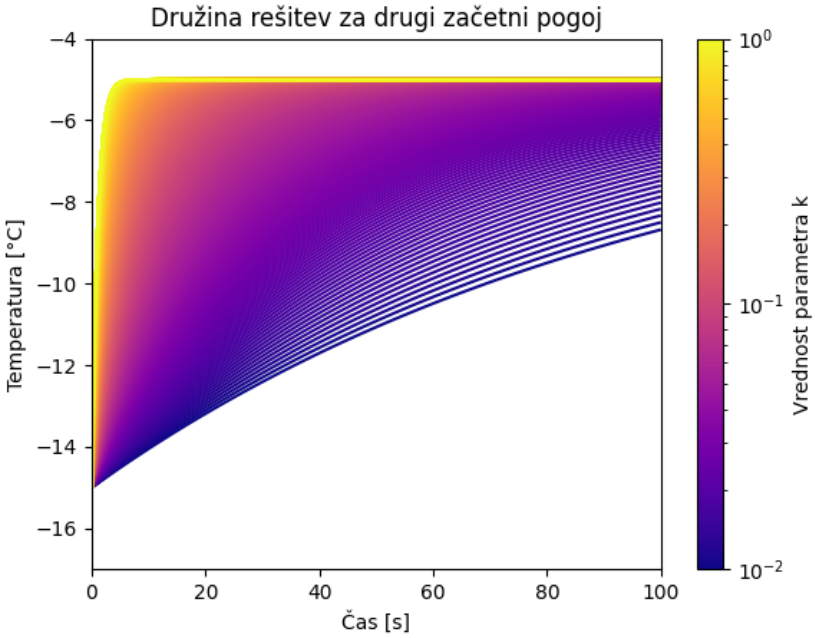
Po eni strani mi je popolnoma smiselno, da ima Runge-Kutta 4 najdaljši čas računanja in tudi najbolj strmo naraščanje, ko se manjša korak. Ima namreč od vseh pomerjenih metod najbolj kompliciran postopek izračuna. Ampak ob prejšnji ugotovitvi, da so si metode med sabo podobne po napakah tu zgleda, kot da je Eulerjeva metoda pravzaprav najboljša, kar se tiče natančnosti za dan čas računanja. To pa mi je nekoliko nenavadno, ker bi to pričakoval ravno od Runge-Kutta 4, ki se uporablja v praksi ravno zaradi dobre natančnosti in hitrega računanja. Nekoliko sem zmeden. Res sem ponovno preveril svojo kodo za morebitne napake, ki jih nisem uspel najti.

4.3 Množica rešitev za različne vrednosti k

Za končno rešitev sem izbral Runge-Kutta-Fehlberg metodo. Vgrajena Scipyjeva je sicer tudi dobra, ampak mi je njeno delovanje manj jasno. Nujuni napaki pa sta primerljivi. Parameter k povezuje različne lastnosti sten torej njihovo površino, debelino in toplotno prevodnost. Pričakujemo, da bo hlajenje/segrevanje bolj strmo večji kot bo parameter k. Za vsakega od začetnih pogojev sem narisal rešitve pri različnih vrednostih parametra.



Slika 18: Družina rešitev enačbe (6)



Slika 19: Družina rešitev enačbe (7)

5 Komentarji in izboljšave

Zdi se mi, da sem spet večino pomembnih komentarjev podal že sproti. Najbolj me pravzaprav bega učinkovitost moje rutine za Runge-Kutta 4. Pričakoval sem, da bi se jasno pokazalo zakaj je dobra metoda, ampak težko rečem, da to vidim. Tudi zakaj mi ni uspelo pravilno uporabiti metode iz `diffeq.py` mi je pravzaprav vprašanje. Vsekakor bi se lahko "štopanje" izboljšalo z povprečenjem po večih meritvah, ampak je že tako ena meritev vzela nekaj časa in tega potem nisem uspel narediti. Če bi imel več časa bi si pogledal še kakšno drugo metodo, sploh iz družine prediktor-korektor, ampak to mora očitno ostati za drugič. Nekaj težav sem imel tudi z smiselno predstavitvijo podatkov, ker imam zopet veliko plotov, ki jih na koncu nisem uporabil. Sem se pa po dosti mukah naučil uporabljati `matplotlib`-ove *Line Collection*-e za bolj učinkovito risanje colormap plotov. Dosti težav mi je povzročal tudi moj možgan, ki se je odločil, da bo za brez razloga spet žalosten in mi je naloga vzela veliko več časa, kot sem mislil (Slika 20). Zdaj je ogrožena moja priprava na kolokvije.

Žal mi ravno zaradi bližujočih (*berite:* voda mi teče v grlo) kolokvijev ni uspelo narediti "dodatne" naloge, kjer sem mislil predstaviti nekaj rešitev Lane-Emdenove enačbe za zvezdne strukture. Mogoče si vseeno to lahko pogledam pri naslednji nalogi, ali pa ko bo čas, kadarkoli bo to..

Slika 20: *Ha ha. Humor based on my pain.*

Literatura

[1] Linda Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM Journal on Scientific and Statistical Computing*, 4(1):136–148, 1983.