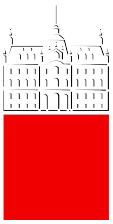
Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*



Naloga 3:
Lastne vrednosti in lastni vektorji

Matematično-fizikalni praktikum

Dimitrije Pešić

Vpisna številka: 28201072

Predavatelj: prof. dr. Borut Paul Kerševan

Ljubljana, 1.11.2023

1 Uvod

Naloga je od nas zahtevala, da z uporabo numerične diagonalizacije matrik poiščemo nekaj najnižjih lastnih vrednosti in lastnih valovnih funkcij za moteno Hamiltonko $H = H_0 + \lambda q^4$ ob vrednostih parametra $0 \leq \lambda \leq 1$. Reševali smo torej matrični problem lastnih vrednosti

$$H|n\rangle = E_n|n\rangle \, .$$

Nove (popravljene) valovne funkcije $|n\rangle$ so seveda linearna kombinacija starih (nemotenih) valovnih funkcij $|n^0\rangle$.

2 Ozadje

Enodimenzionalni linearni harmonski oscilator (delec mase m s kinetično energijo $T(p) = p^2/2m$ v kvadratičnem potencialu $V(q) = m\omega^2 q^2/2$) opišemo z brezdimenzijsko Hamiltonovo funkcijo

$$H_0 = \frac{1}{2}\left(p^2 + q^2\right) \, ,$$

tako da energijo merimo v enotah $\hbar\omega$, gibalne količine v enotah $(\hbar m\omega)^{1/2}$ in dolžine v enotah $(\hbar/m\omega)^{1/2}$. Lastna stanja $|n\rangle$ nemotenega Hamiltonovega operatorja H_0 poznamo iz osnovnega tečaja kvantne mehanike [Strnad III]: v koordinatni reprezentaciji so lastne valovne funkcije

$$|n\rangle = (2^n n!\sqrt{\pi})^{-1/2} \mathrm{e}^{-q^2/2} \, \mathcal{H}_n(q) \, ,$$

kjer so $\mathcal{H}_n$ Hermitovi polinomi. Lastne funkcije zadoščajo stacionarni Schrödingerjevi enačbi

$$H_0|n^0\rangle = E_n^0|n^0\rangle$$

z nedegeneriranimi lastnimi energijami $E_n^0 = n + 1/2$ za $n = 0, 1, 2, \ldots$. Matrika $\langle i|H_0|j\rangle$ z $i, j = 0, 1, 2, \ldots, N-1$ je očitno diagonalna, z vrednostmi $\delta_{ij}(i + 1/2)$ po diagonali. Nemoteni Hamiltonki dodamo anharmonski člen

$$H = H_0 + \lambda q^4 \, .$$

Kako se zaradi te motnje spremenijo lastne energije? Iščemo torej matrične elemente $\langle i|H|j\rangle$ *motenega* Hamiltonovega operatorja v bazi *nemotenih* valovnih funkcij $|n^0\rangle$, kar vemo iz perturbacijske teorije v najnižjem redu. Pri računu si pomagamo s pričakovano vrednostjo prehodnega matričnega elementa za posplošeno koordinato

$$q_{ij} = \langle i|q|j\rangle = \frac{1}{2}\sqrt{i + j + 1} \; \delta_{|i-j|,1} \, ,$$

ki, mimogrede, uteleša izbirno pravilo za električni dipolni prehod med nivoji harmonskega oscilatorja. V praktičnem računu moramo seveda matriki q_{ij} in $\langle i|H|j\rangle$ omejiti na neko končno razsežnost N.

3 Implementacija

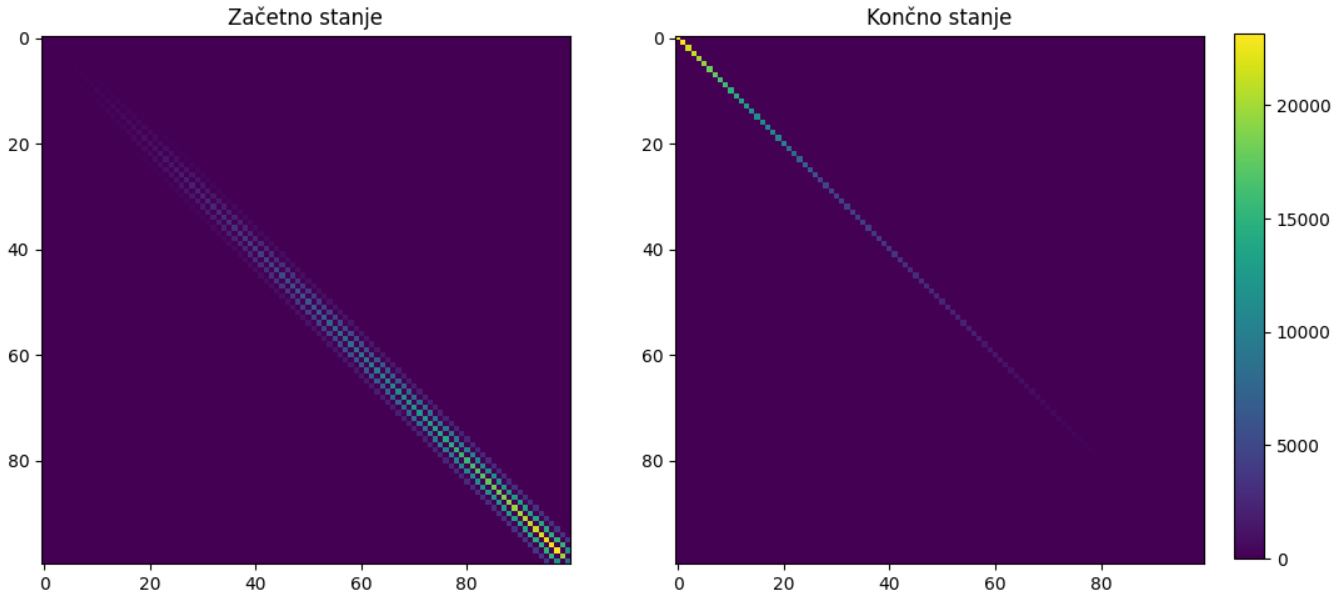
Program sem napisal v PYTHON jeziku in uporabil pakete *Numpy* in *Tensorflow* za metode za diagonalizacijo, *Matplotlib* za vizualizacijo ter *Scipy* za izračun Hermitskih polinomov.

Tokrat sem tudi bil pametnejši in implementiral knjižnico *datetime.datetime.now* s katero sem poimenoval datoteke, ki sem jih shranjeval, saj sem pri prejšnji nalogi si večkrat naredil 'overwrite' - prepis že shranjene simulacij, katerih zagon je bistven čas.

3.1 Diagonalizacija

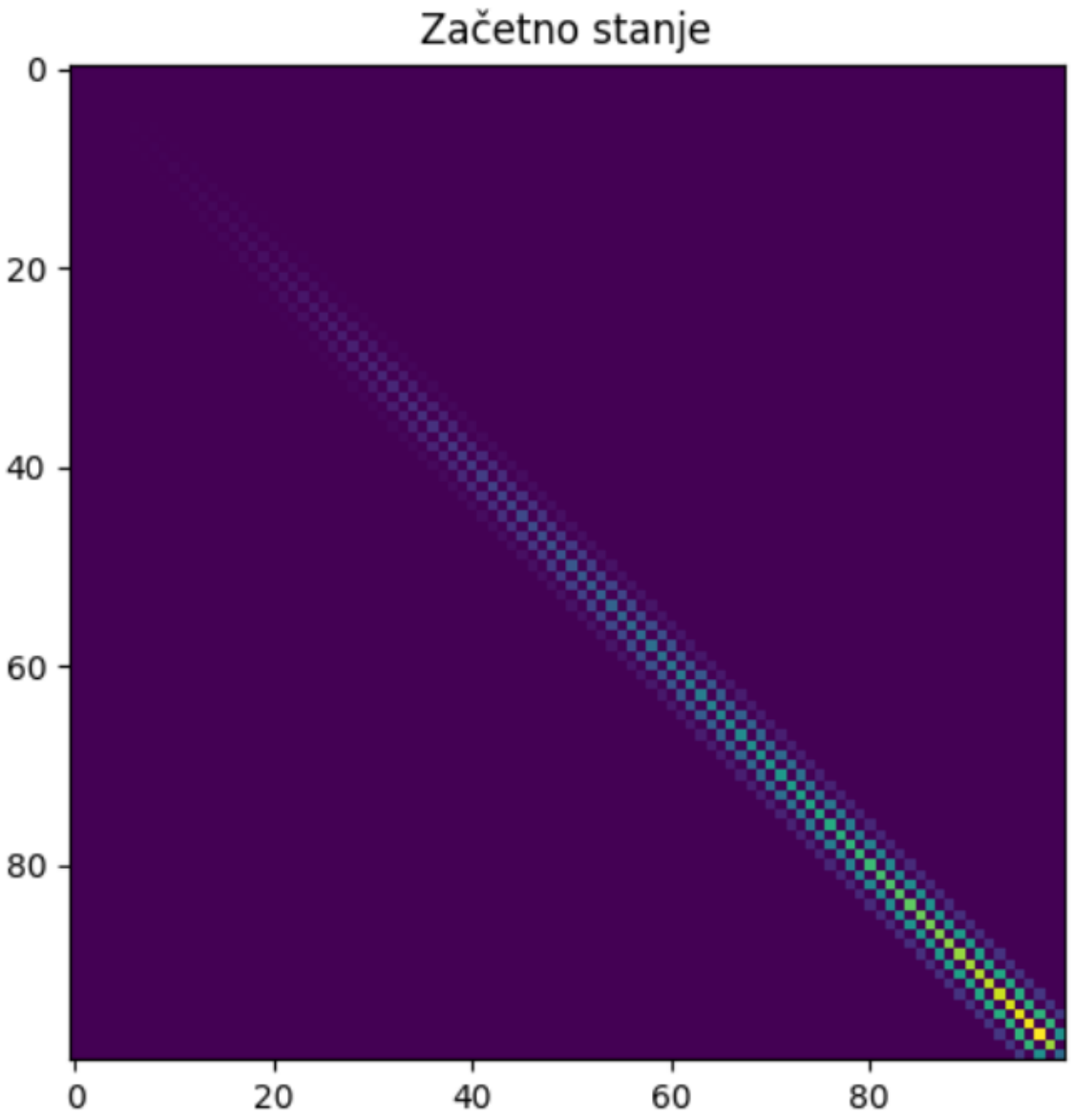
Spisal sem funkcijo, ki uporabi Householderjev algoritem za transofmracijo matrike v tridiagonalno obliko. Le to sem nato vstavil v QR metodo in dobil lastne vrednosti in vektorje. Householderjeve transformacije so linearne trasformacije, ki predstavljajo zrcaljenje preko ravnine (oz. hiperravnine), ki je podana z enotsko normalo **v** pravokotno na ravnino [3].
Funkciji za diagonalizacijo sem dal pogoj za ustavitev, ko je dosežena konvergenca - vsota absolutnih vrednosti elementov matrike za nek majhen epsilon različna, torej skoraj enaka, kot vsota absolutnih vrednosti diagonalnih elementov. Zaradi težav z natančnostjo 'floatov' je potrebno gledati enakost v neki epsilonski okolici. Za dodatno pospešitev diagonalizacije sem elemente, ki so padli pod tolerance (recimo 10^{-15} postavil kar na 0. Dodal sem še pogoj maksimalnih iteracij, kar me kasneje muči.

Na grafu 1 je prikazano začetno stanje pred diagonalizacijo z Householder metodo in končno stanje po njej.



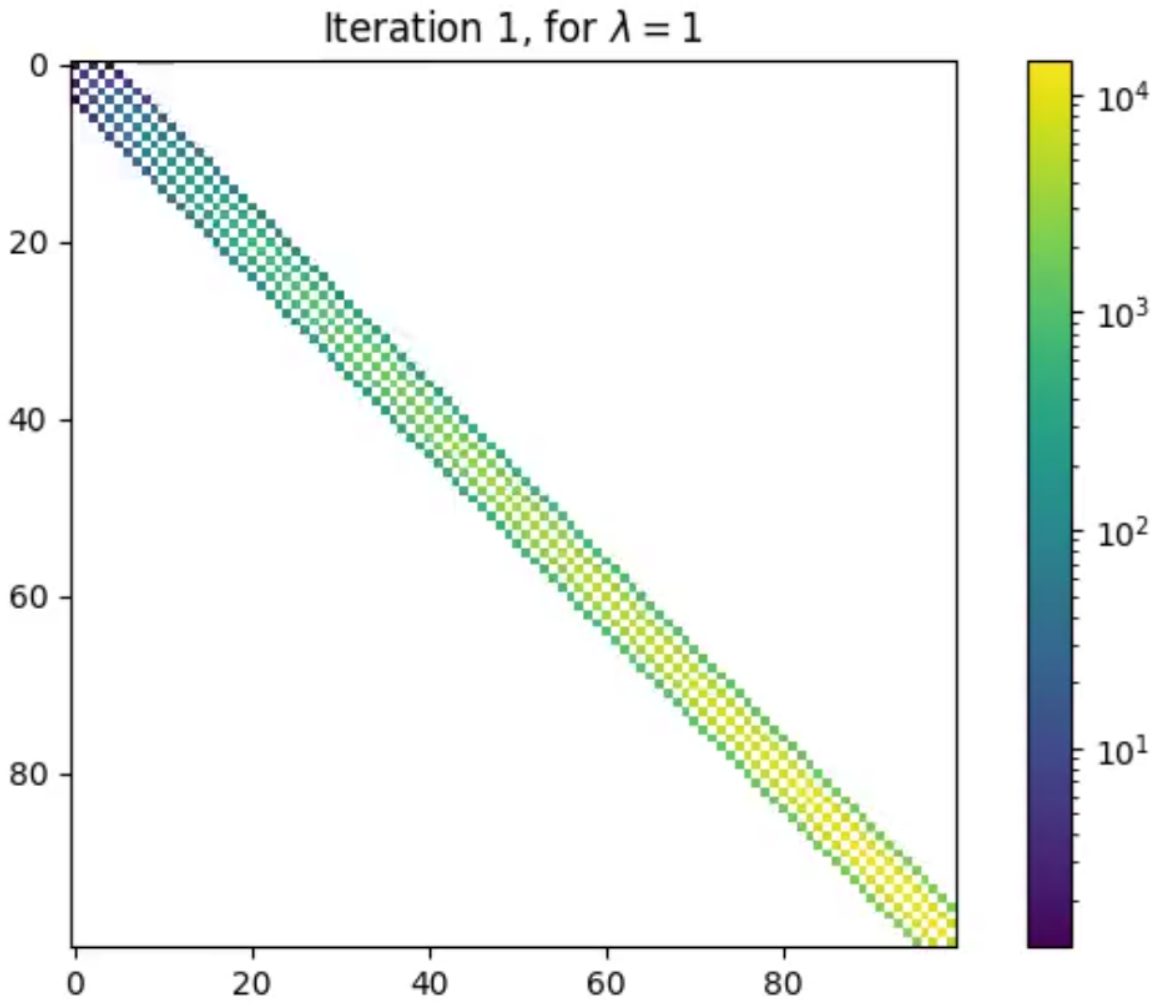
Slika 1: Začetno in končno stanje diagonalizacije.

Za zabavo sem postopek diagonalizacije tudi animiral 3.1. Da bi animacija delovala, morate dokument odpreti v *Adobe PDF Reader* ali pa *Foxit PDF reader*, morda deluje tudi v katerem drugem. Nato kliknete na sliko in se animacija zažene.



Animacija diagonalizacije.

Čeprav je meni barvno lepša zgornja animacija 3.1, se mi zdi spodnja primernejša 3.1, kjer so barve v logaritemski skali, saj je bolj razvidno, da je potrebno mnogo več iteracij za dokončno diagonalizacijo.



Animacija diagonalizacije v logaritemski skali.

3.2 Primerjava metod

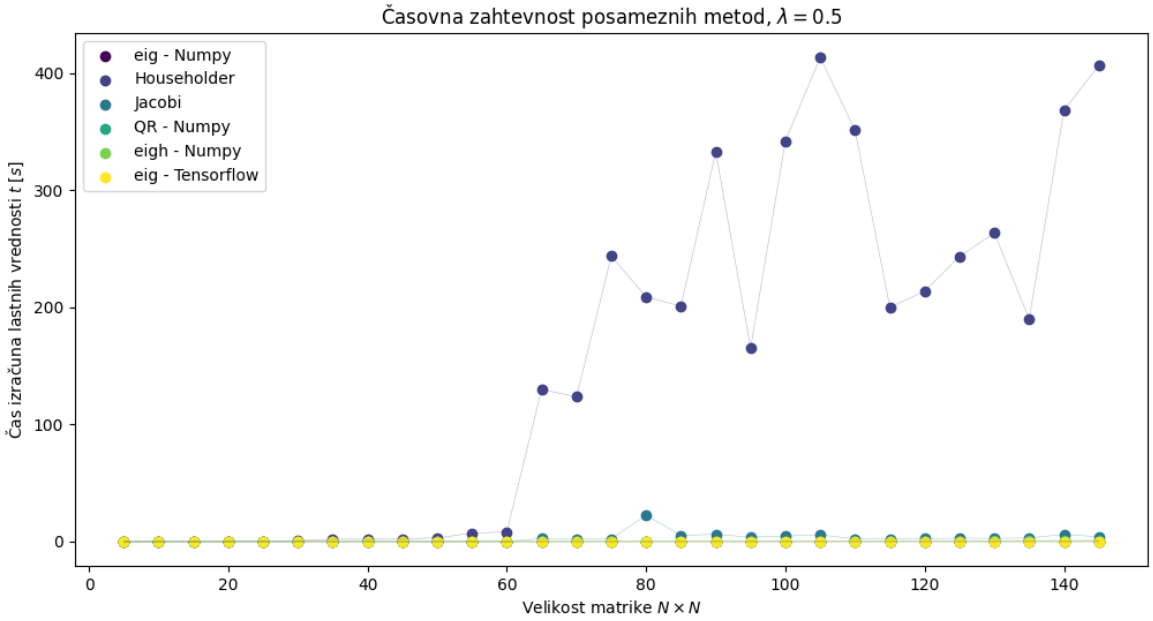
Kot je do zdaj že navada, sem svoje metode primerjal z že obstoječimi algoritmi v raznih knjižnicah.

Poleg Householder metode za tridiagonalizacija in QR metode za lastne vrednosti, sem implementiral še Jacobi metodo za tridiagonalizacijo. Ta algoritem uporablja ravninske rotacije za sistematično zmanjšanje velikosti nediagonalnih elementov, hkrati pa povečuje velikost diagonalnih elementov [4].

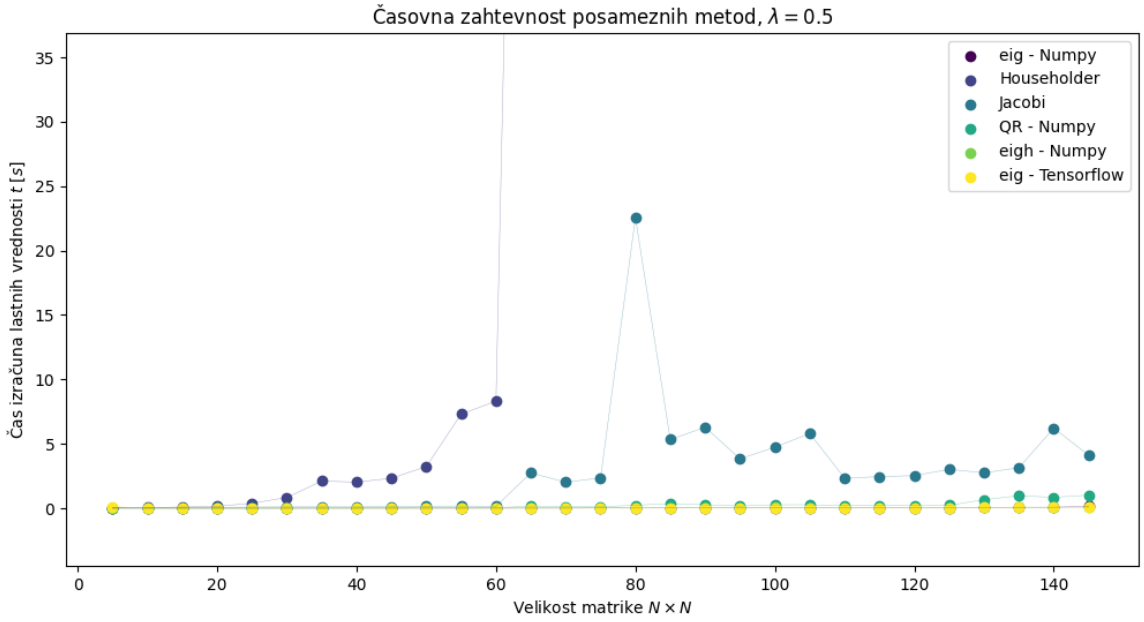
Za primerjavo sem iz *numpy.linalg* uporabil funkciji *eig* in *eigh*. Iz *tensorflow.linalg* knjižnice pa le *eig*. *eigh* funkcija se razlikuje od *eig* po tem, da predpostavi simetričnost sprejete matrike, in zato deluje kanček hitreje, vendar je manj robustna saj ne preveri pogoja simetričnosti.

Želel sem tudi posikusiti knjižnico *tensorflow-gpu*, ki bi mi omogočila uporabo grafične kartice za računanje, saj le te specializirajo za hitro računanje z matrikami - 'ray tracing' [2]. Žal mi tega ni uspelo usposobiti.

Na grafu 2 je prikazano kako se spreminja čas izvajanja algoritma v odvisnosti od velikosti matrike. Na grafu 3 in 4 sta pa še približani sliki za boljšo razvidnost med metodami.



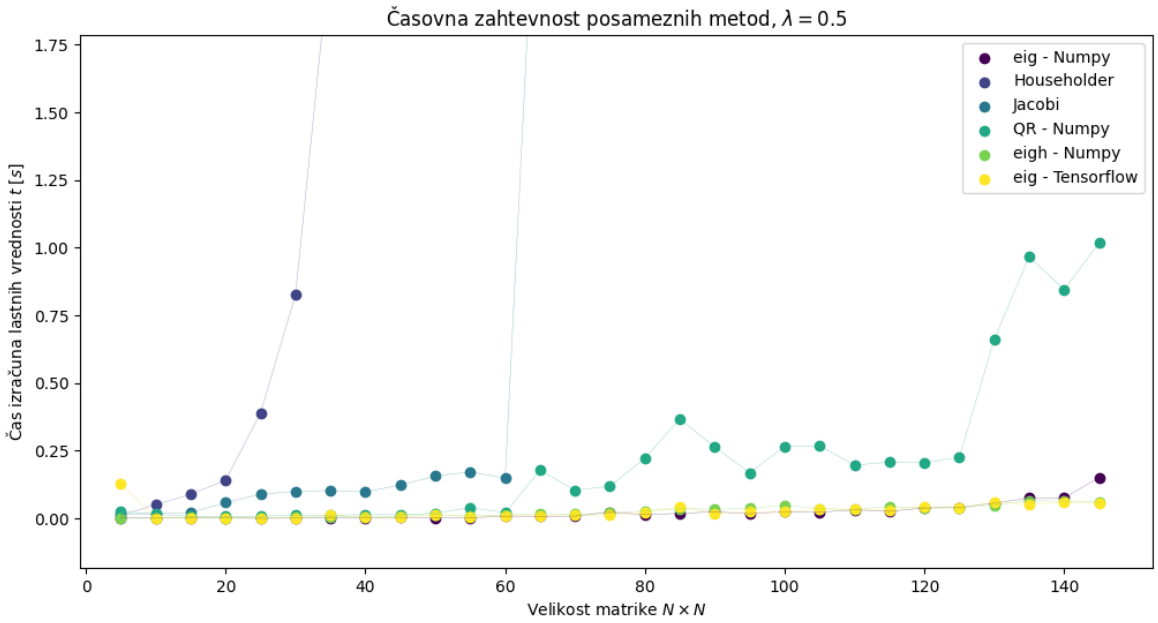
Slika 2: Čas izvajanja za različne metode.



Slika 3: Čas izvajanja za različne metode.

Iz grafa je razvidno, da imata moji implementaciji kar hitro problem ko začne rasti velikost matrike. Pri Householderjevi transformacije (levsaj tako kot sem jo jaz implementiral) nastane problem že pri matriki velikost $N \approx 60$, saj začne moj algoritem presegati 1000 iteracij kar pa žal moj računalnik ne zmore in se zruši.

Izračunal sem tudi absolutno napako izračunanih lastnih vrednosti v primerjavi z *numpy.linalg.eig* metodo. Prikazano na grafih 5, 6, 7.



Slika 4: Čas izvajanja za različne metode.

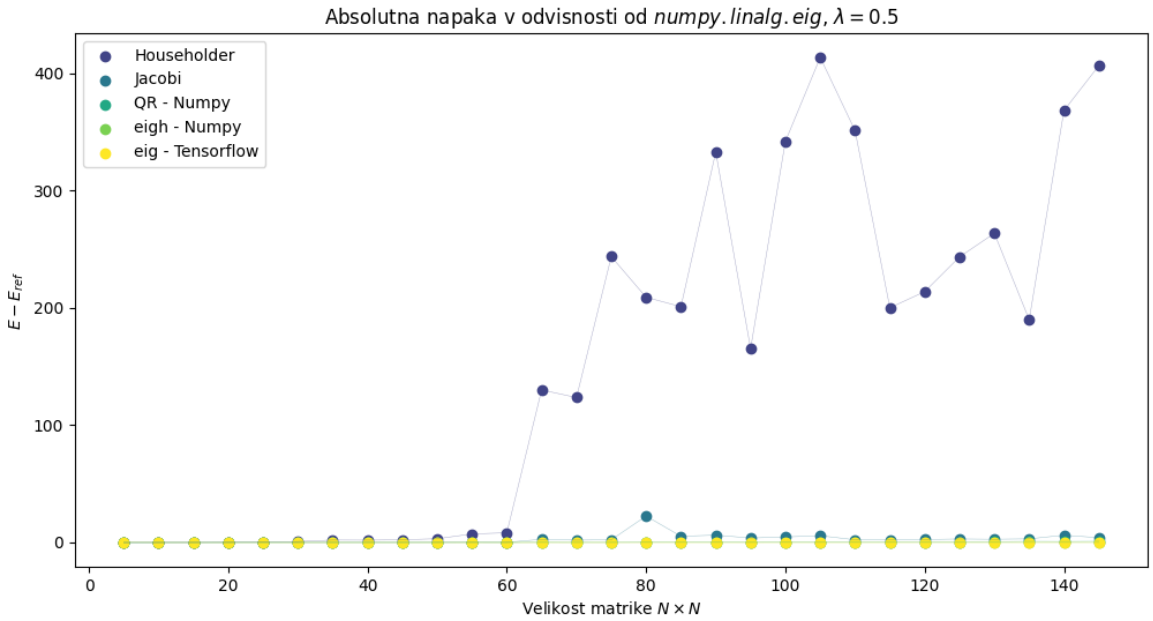
V prid časa in prihranka glavobolov, sem za nadaljne dele naloge kar uporabil *numpy.linalg.eig* kot najrobustnejšo mero.

4 Rezultati

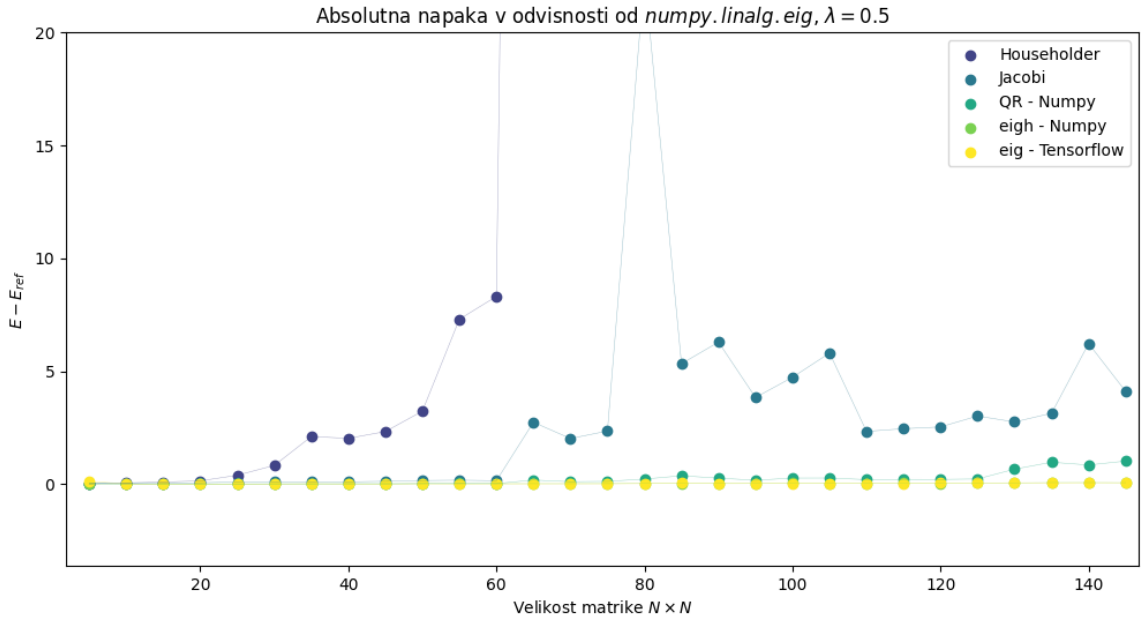
4.1 Nemoten harmonski oscilator

Sedaj ko imamo metodo za računanje diagonalizacije lahko predstavimo rešitvne motene Hamiltonke. Diagonalizacijo sem računal za matriko velikosti $N = 50$ saj nam to omogoči večjo natančnost prvih nekaj lastnih vrednosti. Na grafu 8 sem prikazal prvih 10 lastnih stanj za harmonski oscilator.

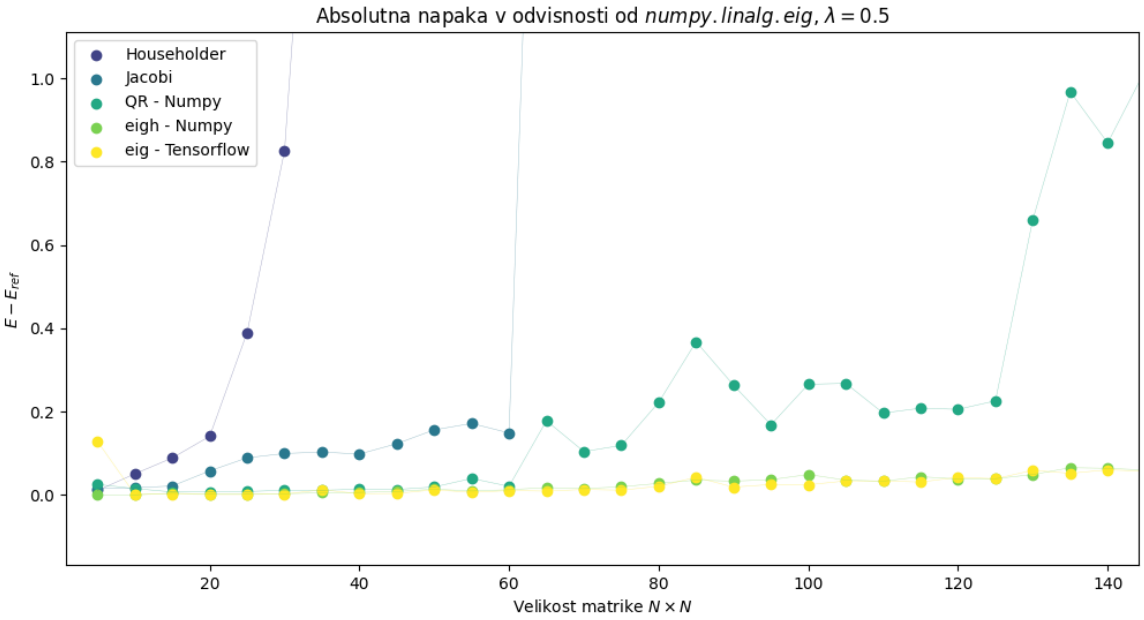
Lastne funkcije so premaknjene po skali navzgor za lepšo vizualizacijo, ne predstavljajo dejanske vrednosti lastnih funkcij, študijo le teh sem naredil v naslednjem poglavju, kjer so tudi zapisane v tabeli.



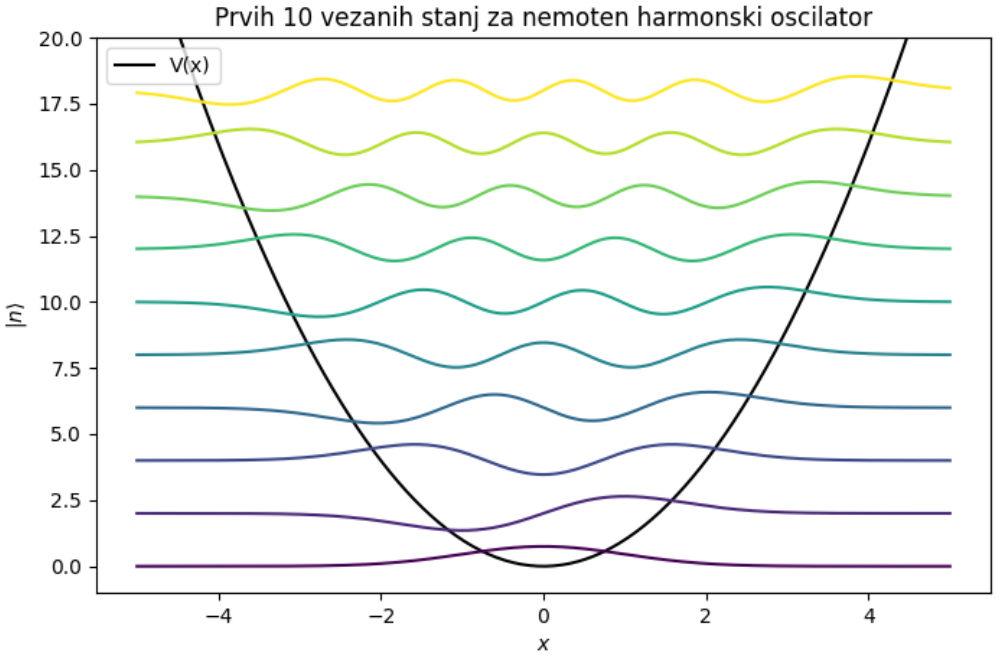
Slika 5: Absolutna napaka za različne metode.



Slika 6: Absolutna napaka za različne metode.



Slika 7: Absolutna napaka za različne metode.



Slika 8: Prvih 10 lastnih stanj za harmonski oscilator.

4.2 Različno računanje q

Namesto da računamo matrične elemente $q_{ij} = \langle i|q|j\rangle$ in perturbacijsko matriko razumemo kot $[q_{ij}]^4$, bi lahko računali tudi matrične elemente kvadrata koordinate

$$q^{(2)}_{ij} = \langle i|q^2|j\rangle$$

in motnjo razumeli kot kvadrat ustrezne matrike,

$$\lambda q^4 \to \lambda \left[q_{ij}^{(2)} \right]^2 ,$$

ali pa bi računali matrične elemente četrte potence koordinate

$$q_{ij}^{(4)} = \langle i|q^4|j\rangle$$

in kar to matriko razumeli kot motnjo,

$$\lambda q^4 \to \lambda \left[q_{ij}^{(4)} \right] .$$

Kakšne so razlike med naštetimi tremi načini izračuna lastnih vrednosti in funkcij? Pri računu poleg enačbe za $\langle i|q|j\rangle$ uporabi še enačbi

$$\langle i|q^2|j\rangle = \frac{1}{2}\left[\sqrt{j(j-1)}\,\delta_{i,j-2} + (2j+1)\,\delta_{i,j} + \sqrt{(j+1)(j+2)}\,\delta_{i,j+2}\right]$$

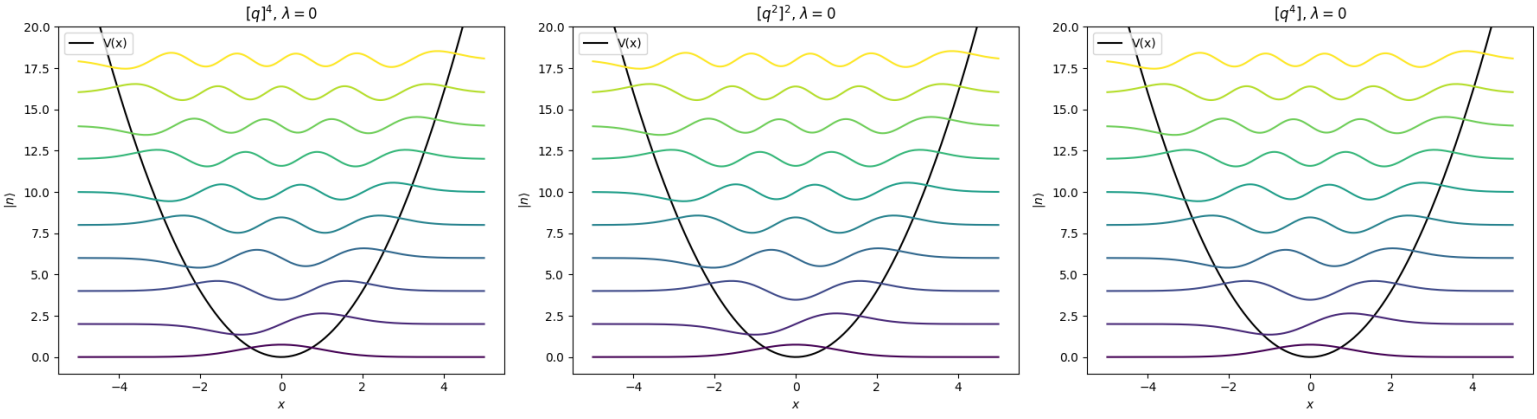
ter

$$\langle i|q^4|j\rangle = \frac{1}{2^4}\sqrt{\frac{2^i\, i!}{2^j\, j!}}\Big[\quad \delta_{i,j+4} + 4\,(2j+3)\,\delta_{i,j+2} + 12\left(2j^2+2j+1\right)\,\delta_{i,j}$$
$$+ \quad 16j\left(2j^2-3j+1\right)\,\delta_{i,j-2} + 16j\left(j^3-6j^2+11j-6\right)\,\delta_{i,j-4}\Big] ,$$

le te se izpelje iz rekurzijskih zvez za Hermitove polinome [1]. Vsi načini so med seboj podobni, ampak vrnejo nekoliko drugačne matrike.

4.3 Rezultati

Na grafih 9 in 10, 11 so predstavljene lastne funkcije za prvih 10 vezanih stanj za anharmonski oscilator. Pri $\lambda = 0$ se prepričamo, da zares dobimo harmonski oscilator. Vrednosti lastnih energij za posamezno metodo so prikazana v tabelah 3, 2, 2.

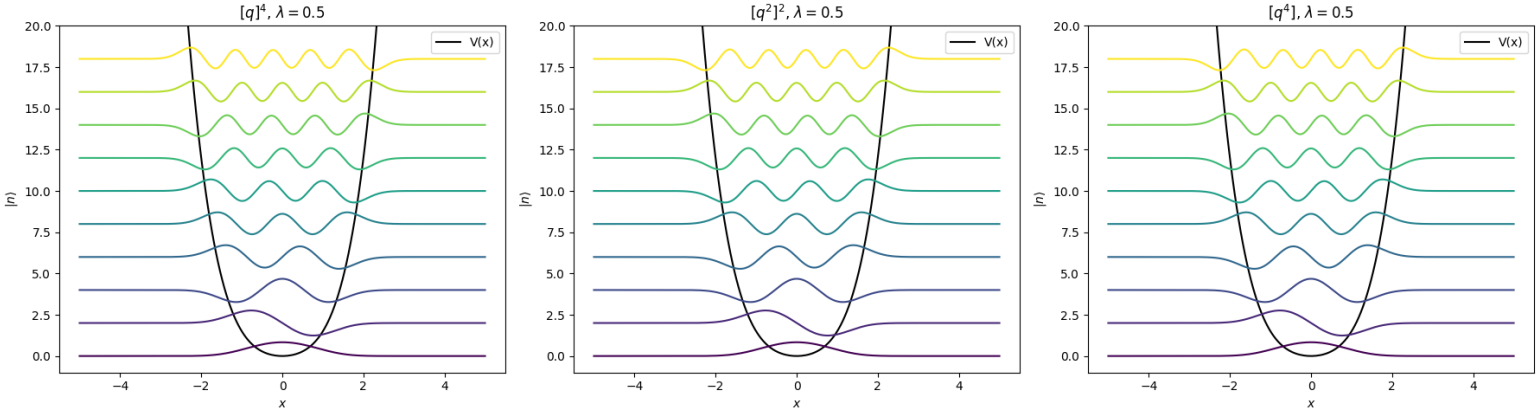


Slika 9: Prvih 10 lastnih stanj za anharmonski oscilator z $\lambda = 0$.

Kot je razvidno iz tabele 3, za $\lambda = 0$ zares dobimo le harmonski oscilator kar nam nakazuje na pravilnoste implementacije.

E_n	$\|n^0\rangle$	$\|n^1\rangle$	$\|n^2\rangle$	$\|n^3\rangle$	$\|n^4\rangle$	$\|n^5\rangle$	$\|n^6\rangle$	$\|n^7\rangle$	$\|n^8\rangle$	$\|n^9\rangle$
$[q_{ij}]^4$	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5
$\left[q_{ij}^{(2)}\right]^2$	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5
$\left[q_{ij}^{(4)}\right]$	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5

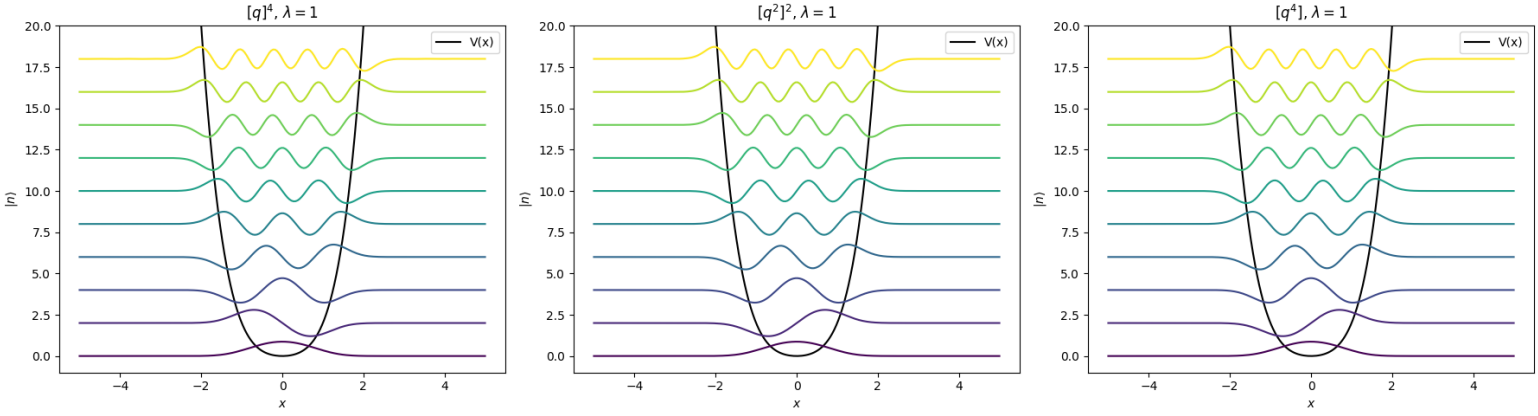
Tabela 1: Vrednosti E_n za vsako metodo pri $\lambda = 0$.



Slika 10: Prvih 10 lastnih stanj za anharmonski oscilator z $\lambda = 0.5$.

E_n	$\|n^0\rangle$	$\|n^1\rangle$	$\|n^2\rangle$	$\|n^3\rangle$	$\|n^4\rangle$	$\|n^5\rangle$	$\|n^6\rangle$	$\|n^7\rangle$	$\|n^8\rangle$	$\|n^9\rangle$
$[q_{ij}]^4$	0.8038	2.7379	5.1793	7.9424	10.9636	14.2031	17.634	21.2367	24.9955	28.8907
$\left[q_{ij}^{(2)}\right]^2$	0.8038	2.7379	5.1793	7.9424	10.9636	14.2031	17.634	21.2362	24.9955	28.9018
$\left[q_{ij}^{(4)}\right]$	0.8038	2.7379	5.1793	7.9424	10.9636	14.2031	17.6341	21.2366	24.9973	28.9025

Tabela 2: Vrednosti E_n za vsako metodo pri $\lambda = 0.5$.

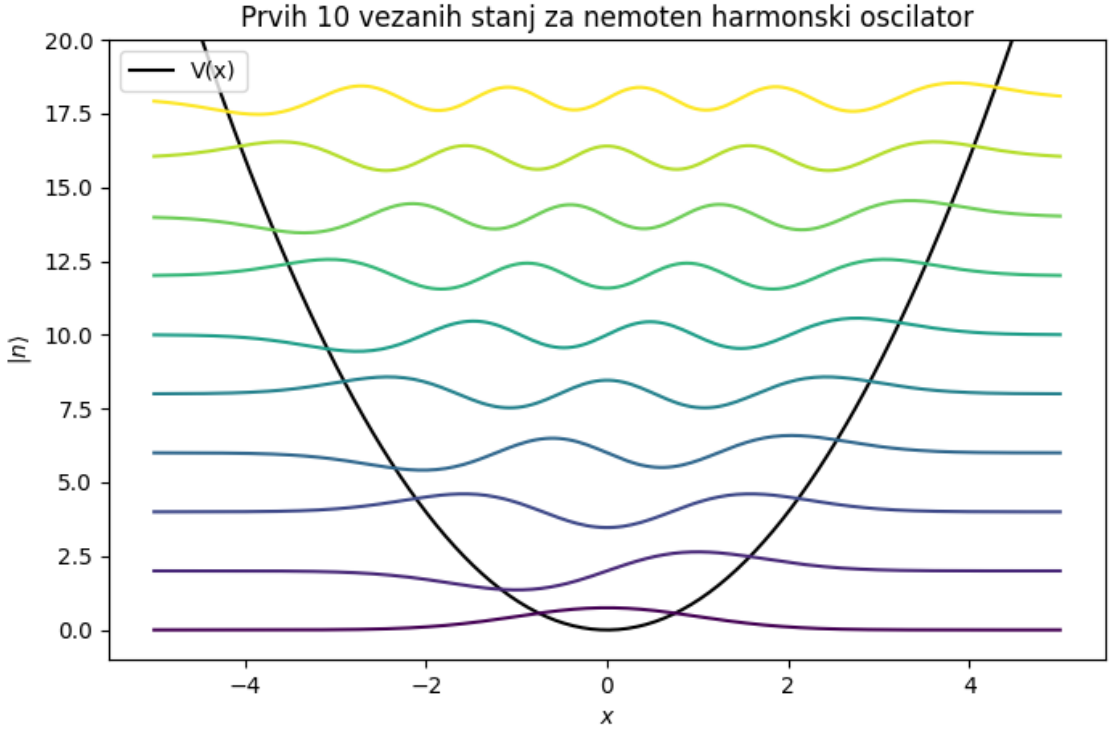


Slika 11: Prvih 10 lastnih stanj za anharmonski oscilator z $\lambda = 1$.

Ker Velja, da je slika vredna 1000 besed, lahko ekstrapoliramo in rečemo, da je videoposnetek vreden 1000 slik (ali v mojem primeru le 200 slik skupaj zlepljenih). Zato sem zopet izdelal tudi animacijo kako se spreminjajo lastna stanja z večanje λ 4.3

E_n	$\|n^0\rangle$	$\|n^1\rangle$	$\|n^2\rangle$	$\|n^3\rangle$	$\|n^4\rangle$	$\|n^5\rangle$	$\|n^6\rangle$	$\|n^7\rangle$	$\|n^8\rangle$	$\|n^9\rangle$
$[q_{ij}]^4$	0.6962	2.3244	4.3275	6.5784	9.0288	11.6487	14.4177	17.3204	20.3452	23.4826
$\left[q_{ij}^{(2)}\right]^2$	0.6962	2.3244	4.3275	6.5784	9.0288	11.6487	14.4177	17.3204	20.3452	23.4824
$\left[q_{ij}^{(4)}\right]$	0.6962	2.3244	4.3275	6.5784	9.0288	11.6487	14.4177	17.3204	20.3452	23.4825

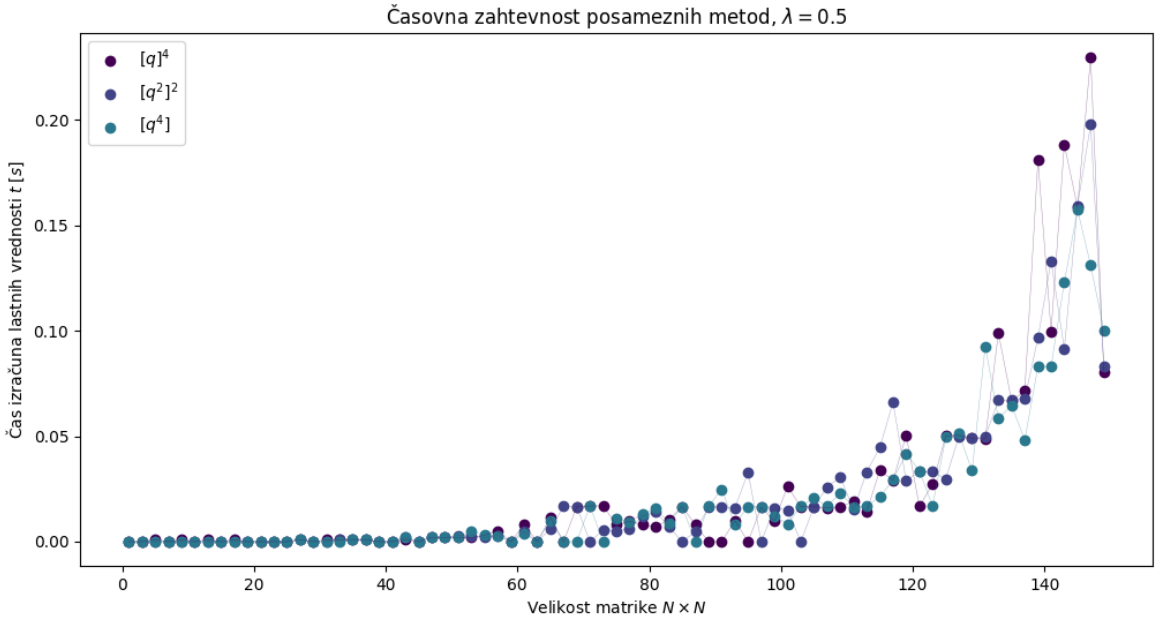
Tabela 3: Vrednosti E_n za vsako metodo pri $\lambda = 1$.



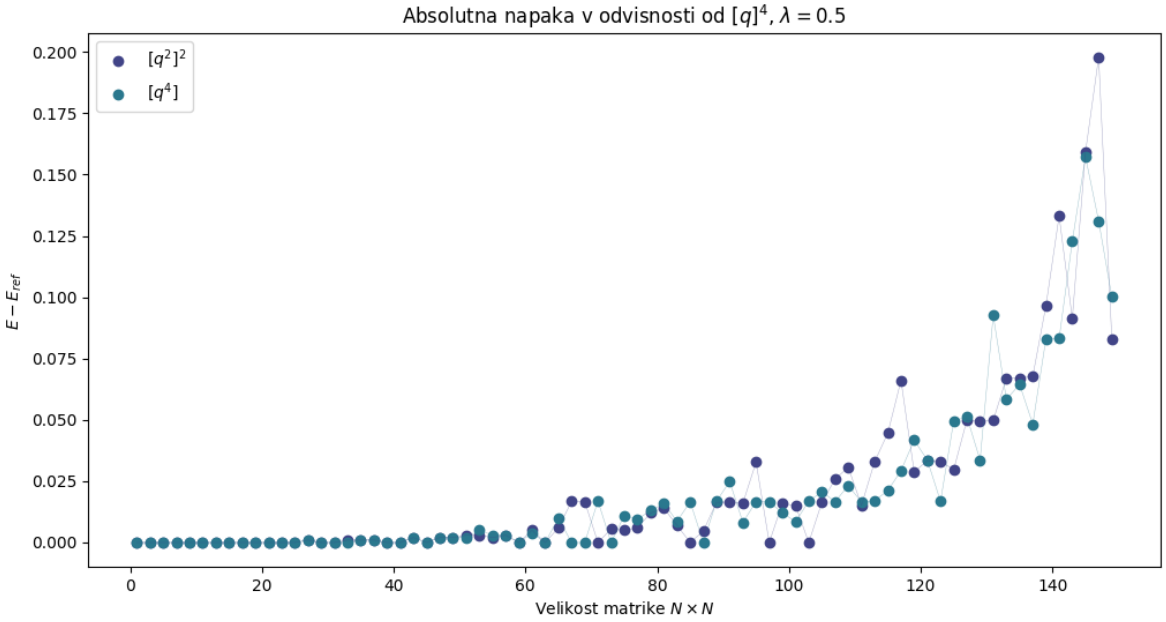
Animacija spreminjanja lastnih stanj za $0 \leq \lambda \leq 1$.

4.4 Najboljša metoda

Naloga je tudi zahtevala, da pogruntamo katera od metod računanja q je najprimernejša. Študija tega je prikazana na grafu 12 in 13.



Slika 12: Čas izvajanja za različne metode.



Slika 13: Čas izvajanja funkcij za različne metode.

Kot je razvidno iz grafov in tabel v prejšnjem poglavju, med metodami ni bistvenih razlik. Razlike se začnejo kazati pri višjih lastnih vrednostih. Verjamem, da če bi imel več časa bi lahko naredil simulacijo za mnogo večje matrike in s tem pridobil boljšo informacijo o tem kater način izračuna je

boljši. Ne dvomim, da bi se znala najti razlika pri metodah ali simulacijo izvajamo na procesorju ali grafični kartici. Prepuščam to nekomu, ki ima malo več znanja in je spal več kot 8 ure v zadnjih 3 dneh.

5 Zaključek

Cilj te naloge je bil pridobivanje znanja numerične diagonalizacije matrik, ki je uporabno na mnogo različnih področij fizike in drugod. Kljub ekstremni časovni stiski (zaradi ne samo ene selitve, vendar dveh v enem tednu...) sem 'padel' v nalogo, kar je bilo v mojo pogubo. Zapravil sem veliko časa poskušajoč izmeriti CPU in RAM porabo, žal pa moj računalnik ni želel sodelovati. Poleg tega me je še močno mučila implementacija multithreadinga in GPU threadinga nad katero sem moral obupati, kar je razlog bolj skopih velikosti matrik saj so moje metode obupale nad velikostmi $N \approx 100$.

Literatura

[1] Simon Bukovšek, *Skripta predavanj Matematika 4, Ljubljana 2023.*

[2] Ray tracing Wikipedia

[3] Householder transformation

[4] Jacobi eigenvalue algorithm