Računarski fakultet Univerziteta UNION

Marina Radulaški

Numeričke simulacije rotirajućih Boze-Ajnštajn kondenzata

diplomski rad

Beograd, 2009. godine

Motivisana srednjoškolskim istraživanjima u Istraživačkoj stanici Petnica, koja sam sprovela u saradnji sa Institutom za fiziku u Beogradu, odlučila sam se za studije na Računarskom fakultetu Univerziteta Union, kako bih stekla dalje znanje u oblasti naučnih simulacija. Danas, nekoliko godina kasnije, zahvalna sam ovim trima institucijama na pruženom obrazovanju i praktičnim veštinama koje su omogućile da izvedem istraživanje predstavljeno u ovom radu, urađeno u Laboratoriji za primenu računara u nauci Instituta za fiziku.

Posebno sam zahvalna mentorima, doc. dr Antunu Balažu, koji me je uveo u tajne istraživanja, zainteresovao za oblast Boze-Ajnštajn kondenzacije i kroz mnogobrojne konstruktivne savete usmeravao moj rad, kao i doc. dr Radomiru Jankoviću od koga sam naučila tehnike modelovanja i simulacije.

Želim da se zahvalim i svojoj porodici i prijateljima, koji su bili uz mene i podržavali me celim tokom studija.

u Beogradu, decembra 2009. godine *Marina Radulaški*

Sadržaj

1

Uvod

U tradicionalnim pristupima istraživanju, u potpunoj paraleli sa analitičkim i sintetičkim pristupom nauci, izdvajaju se dve osnovne saznajne paradigme: eksperiment i teorija. Eksplozivni razvoj računara omogućio je pojavu i treće saznajne paradigme, numeričkih simulacija[1].

Razlozi za numerički pristup opisivanja prirode i modelovanje mogu biti višestruki:
- sistem koji želimo da opišemo je suviše komplikovan za potpuni teorijski ili numerički tretman (previše veličina koje treba uzeti u obzir, previše strukturnih elemenata),
- mikroskopsko ponašanje sistema nije u potpunosti poznato (efektivne interakcije strukturnih elemenata),
- želimo da posmatramo pojednostavljen model (u manjem broju dimenzija)
- opis originalnog sistema je nerealističan u matematičkom formalizmu koji se koristi (prelazak sa realnog na imaginarno vreme).

Ovaj pristup omogućava numeričke eksperimente na sistemu čiji opis možemo na (relativno) kompletan način da implemetiramo u numeričkim algoritmima, ali takođe i ispitivanje teorijskih modela za složene pojave u prirodi.

Fokus ovog rada je na numeričkim simulacijama specifičnog fizičkog sistema: gasa ultrahladnih atoma u rotirajućim magnetno-optičkim zamkama. Ovakav sistem pri dovoljno niskoj temperaturi doživljava fazni prelaz – Boze-Ajnštajn kondenzaciju. U ovom radu, pokazaćemo kako se pomoću numeričkih simulacija mogu u principu proučavati osobine Boze-Ajnštajn kondenzata. Naglasak rada je na ispitivanju strukture i kompleksnosti algoritama koji se koriste u ovim simulacijama.

1.1 Boze-Ajnštajn kondenzacija

U drugoj polovini XIX veka naučnici Maksvel[2] i Bolcman[3] su teorijski izveli statistički zakon po kome čestice materije zauzimaju energetske nivoe kada se nalaze u stanju termodinamičke ravnoteže. Boze[4] je, 1920. godine, napisao članak u kome je pokazao da čestice svetlosti, fotoni, ne prate Maksvel-Bolcmanovu statistiku. Ispravnost Bozeovog rada u početku nije

[1] Na engleskom jeziku je uobičajeni naziv Computational Science; za razliku od Computer Science koji predstavlja računarstvo (informatiku).

[2] James Clerk Maxwell (1831-1879), škotski teorijski fizičar i matermatičar.

[3] Ludwig Eduard Boltzmann (1844-1906), austrijski fizičar.

[4] Satyendra Nath Bose (1895-1974), indijski fizičar.

prepoznata u Evropi, što ga je nagnalo da se 1924. godine obrati Ajnštajnu[5], koji je preveo rad na nemački jezik i objavio ga u Bozeovo ime. Ajnštajn je ne samo uvideo značaj Bozeovog rada, već i proširio njegov zaključak na masene čestice celobrojnog spina[6]. Teorijski rad dva fizičara je predvideo novu fazu u koju prelaze bozonske čestice[7] na niskim temperaturama (bliskim apsolutnoj nuli). Ta faza je danas poznata kao Boze-Ajnštajn kondenzat (BAK), a odgovarajući fazni prelaz se naziva Boze-Ajnštajn kondenzacija.

Radi boljeg razumevanja ovog fenomena, osvrnimo se prvo na podelu elementarnih čestica na bozone i fermione. Bozoni (npr. foton) imaju celobrojne vrednosti spina, a fermioni (npr. elektron) polu-celobrojne. Više bozona se može naći u istom kvantnom stanju, karakterisanom istim vrednostima svih kvantnih brojeva, dok je fermionima to zabranjeno na osnovu Paulijevog[8] principa isključenja. Ovo se lepo može ilustrovati na primerima: u laserskoj svetlosti svi fotoni se nalaze u istom kvantnom stanju, dok se prilikom popunjavanja atomskih orbitala samo dva elektrona mogu naći na istoj orbitali, i to sa suprotnim vrednostima spina, kako bi bio zadovoljen Paulijev princip.

Boze-Ajnštajn kondenzacija [1, 2, 3] predstavlja makroskopsko popunjavanje osnovnog (najnižeg) energetskog stanja sistema bozona. Na niskim temperaturama se ovo prirodno događa usled težnje svakog fizičkog sistema da minimizuje svoju energiju. Čestice u kondenzatu se nalaze u makroskopskom koherentnom stanju. Zato ideja BAK leži u osnovi objašnjenja fenomena superfluidnosti i niskotemperaturne superprovodnosti u kojima se čestice kreću bez unutrašnjeg otpora (viskoznosti, odnosno električnog otpora).

Fermi[9] je smatrao da je u praksi nemoguće ostvariti BAK. Teorijski, ovaj fazni prelaz se postiže za slučaj kada je interakcija između čestica mala. Fermi je smatrao da su interakcije između čestica u prirodi dovoljno jake da onemoguće prelaz u BAK fazu. Danas znamo da Fermi nije bio u pravu, a ovo otkriće je pokrenulo čitavu novu oblast u fizici (hladni kvantni gasovi).

Zahvaljujući razvoju tehnika hlađenja materije na temperature reda veličine 100nK, koje su bile neophodne za eksperimentalno ostvarenje BAK, posle decenija rada BAK je 1995. godine postignut i u laboratorijskim uslovima [4, 5], za šta je 2001. godine dodeljena Nobelova nagrada iz fizike. BAK se najčešće realizuje hlađenjem neutralnih atoma alkalnih metala (izotopa sa parnim brojem neutrona u jezgru, tako da se atomi ponašaju kao bozoni), kao što su ^{87}Rb, ^{23}Na, ^{7}Li, ^{133}Cs, na temperature bliske apsolutnoj nuli. Kada se temperatura bozonskog sistema spusti ispod kritične temperature Boze-Ajnštajn kondenzacije, čestice počinju da makroskopski naseljuju najniže energetsko stanje.

Ovaj rad razmatra slučaj idealnog bozonskog gasa u velikom kanonskom ansamblu [6], odnosno sistema neinteragujućih bozona u kome je razmena energije i čestica sa okolinom ipak moguća.

Kada posmatramo BAK fenomen, glavna fizička veličina koju koristimo za karakterizaciju je

[5] Albert Einstein (1879-1955), nemački teorijski fizičar.
[6] Spin predstavlja jedan od osnovnih parametara opisa kvantnog stanja čestice.
[7] Bozoni su čestice celobrojnog spina.
[8] Wolfgang Ernst Pauli (1900-1958), austrijski teorijski fizičar.
[9] Enrico Fermi (1901-1954), italijanski fizičar.

kondenzaciona temperatura T_c. Ona predstavlja temperaturu ispod koje je broj čestica N_0 u osnovnom stanju makroskopski, a iznad koje se zanemarljivi broj atoma nalazi u osnovnom stanju. Za sistem idealnih bozona [1, 2, 3], broj atoma u osnovnom stanju se određuje pomoću jednačine:

$$N_0 = N - \sum_{m=1}^{\infty} [e^{m\beta E_0} Z_1(m\beta) - 1],$$

gde je N ukupan broj čestica u sistemu (broj atoma u magnetno-optičkoj zamci), $\beta = \frac{1}{k_B T}$ se naziva inverzna temperatura, k_B je Bolcmanova konstanta, a T je termodinamička temperatura. U gornjoj jednačini E_0 predstavlja energiju osnovnog stanja jednočestičkog potencijala, a $Z_1(\beta)$ je odgovarajuća jednočestična particiona funkcija definisana sa:

$$Z_1(\beta) = \sum_{k=0}^{\infty} e^{-\beta E_k}.$$

Da bismo mogli da izračunamo kondenzacionu temperaturu, potrebno je da znamo energetski spektar teorije, odnosno vrednosti energetskih nivoa E_k jednočestičnog sistema. Ovo su svojstvene vrednosti operatora energije (hamiltonijana) $\hat{H}$ sistema. Da bismo našli spektar teorije, iskoristićemo metod opisan u radovima [7, 8]. Razmotrićemo vremensku evoluciju sistema u kratkom (imaginarnom) vremenskom periodu ϵ i amplitudu verovatnoće prelaza čestice iz položaja $\mathbf{r}$ u položaj $\mathbf{r}'$ za dato vreme, $A(\mathbf{r}, \mathbf{r}'; \epsilon) = \langle \mathbf{r}' | e^{-\epsilon \hat{H}} | \mathbf{r} \rangle$. Svojstvene vrednosti ovako definisane matrice evolucionog operatora A su povezane sa svojstvenim vrednostima energije pomoću izraza $e^{-\epsilon E_k}$. Stoga, ako znamo matricu A i u mogućnosti smo da izračunamo njene svojstvene vrednosti, pomoću njih lako možemo odrediti i energetske nivoe E_k sistema idealnih bozona, kao i kondenzacionu temperaturu BAK, odnosno druge globalne termodinamičke osobine kondenzata.

Matrični elementi evolucionog operatora se ne mogu izračunati egzaktno u opštem slučaju i ovo predstavlja ključan problem kod proučavanja sistema sa netrivijalnim potencijalima. Međutim, numerički se amplitude prelaza $A(\mathbf{r}, \mathbf{r}'; \epsilon)$ mogu efikasno izračunati pomoću metoda efektivnih dejstava, razvijenog u radovima [7, 8, 9, 10, 11]. Greška određivanja energetskih nivoa ovim metodom se skalira sa ϵ^p, gde prirodan broj p predstavlja odabrani nivo efektivnog dejstva (pogledati sliku 3.1 u poglavlju Rezultati). Stoga, za kratka vremena evolucije ($\epsilon < 1$), greška računanja amplituda verovatnoće metodom efektivnih dejstava za dovoljno veliku vrednost parametra p se praktično može zanemariti. Uz pogodno odabranu prostornu diskretizaciju sistema [7, 8, 9, 10, 11], ovako možemo izračunati članove matrice A. Korišćenjem jednog od algoritama dijagonalizacije[10] u stanju smo da svojstvene vrednosti i svojstvene vektore numerički egzaktno odredimo i rešimo posmatrani problem.

1.2 Rotirajući Boze-Ajnštajn kondenzat

Među aktuelnim BAK eksperimentima nalaze se i kondenzati u brzo rotirajućim magneto-optičkim zamkama. Njihov značaj je u doprinosu razumevanja formiranja i evolucije kvant-

[10] Dijagonalizacija predstavlja svođenje matrice na ekvivalentnu matricu dijagonalne forme; kada se matrica prevede u ovu formu, njeni nenulti elementi (koji se nalaze na glavnoj dijagonali) su jednaki svojstvenim vrednostima matrice.

nih vorteksa, intrigantnog fenomena fundamentalne fizike [12, 13], koji predstavlja odgovor superfluida na rotaciona pobuđenja sistema - kvantne rotacije. To se u eksperimentu vidi kao stvaranje manjih nezavisnih vrtloga, a osobina kojom ih karakterišemo (vrtložnost) ima kvantovane vrednosti.

U eksperimentu Dalibara i saradnika [14], atomi ^{87}Rb su zarobljeni u harmonijskom potencijalu koji ima i mali anharmonijski deo u $x-y$ ravni, a ceo sistem rotira oko z-ose ugaonom brzinom Ω, tako da je rezultujući potencijal koji atomi osećaju jednak

$$V(x,y,z) = \frac{M}{2}\omega_\perp^2(1-r^2)(x^2+y^2) + \frac{M}{2}\omega_z^2 + \frac{k_n}{24}(x^2+y^2)^2,$$

gde M predstavlja masu čestice, k_n anharmonicitet, $\omega_\perp = 407.15$Hz harmonijsku frekvenciju zamke u $x-y$ ravni, $\omega_z = 69.115$Hz frekvenciju zamke duž z-ose, a r parametar rotacije sistema, $r = \frac{\Omega}{\omega_\perp}$. Da bismo operisali sa bezdimenzionim jedinicama koristili smo prirodne jedinice u kojima je $M = \hbar = 1$, dok je na taj način reskalirana vrednost anharmoniciteta k_n izražena u jedinicama $\frac{\hbar}{M^2\omega_\perp^3}$ i jednaka $k_n = 0.00195$, dok se dužine izražavaju u jedinicama $\sqrt{\frac{\hbar}{M\omega_\perp}}$.

Vrednost potencijala figuriše u računanju amplituda verovatnoće prelaza, što predstavlja prvi korak procesa određivanja kondenzacione temperature u okviru posmatranog pristupa.

1.3 Cilj rada

Pri određivanju kondenzacione temperature Boze-Ajnštajnove kondenzacije na opisani način, ključni numerički izazov se ogleda u određivanju svojstvenih vrednosti i vektora evolucionog operatora. Numerički se ovaj problem rešava dijagonalizacijom matrice prostorno diskretizovanog operatora. Precizan opis ovakvih fizičkih sistema zahteva veliki broj svojstvenih vrednosti i svojstvenih vektora, što je moguće dobiti samo pomoću veoma finih diskretizacija. Naravno, ovo zahteva matrice realnih dimenzija tako da dijagonalizacija postaje i numerički i memorijski veoma zahtevna.

Kao što je poznato, postoji više razvijenih algoritama kojima se može izvršiti dijagonalizacija. Cilj ovog rada je upoređivanje tačnosti, vremenske i prostorne složenosti dijagonalizacije Lancoš algoritmom [15] u odnosu na standardnu implementaciju algoritma potpune dijagonalizacije, na primeru rešavanja svojstvenog problema evolucionog operatora opisanog rotirajućeg Boze-Ajnštajn kondenzata, posmatranog kao gas idealnih bozona ^{87}Rb.

2

Numeričke simulacije Boze-Ajnštajn kondenzata

U posmatranom pristupu numeričkom opisivanju Boze-Ajnštajn kondenzata neophodno je uvesti odgovarajuću diskretizaciju prostora. Ona se mora uvoditi na kontrolisan način, tako da je greška koja se na taj način uključuje u rezultate može proceniti i na siguran način ukloniti iz rezultata. U radu [7] je izvedena procena analitičke greške koja se čini uvođenjem diskretizacije, a u radu [8] je pokazano kako se ova greška može praktično eliminisati pogodnim izborom parametara diskretizacije. U ovom radu ćemo koristiti pomenuti pristup. Koordinate položaja atoma se biraju tako da uzimaju vrednosti $n\Delta$, gde ceo broj n uzima $L = 2N + 1$ vrednosti iz skupa $[-N, N]$. Zbog binarne osnove reprezentovanja brojeva na računaru, mnogi numerički algoritmi se brže izvršavaju za parne vrednosti veličine matrice. Stoga se obično za skup vrednosti n uzima $n \in [-N, N)$, čime postižemo parnost broja tačaka kojima reprezentujemo jednu prostornu dimenziju, $L = 2N$. Pri ovakvoj diskretizaciji, matrica evolucionog operatora se može predstaviti kao matrica dimenzije $D \times D$, gde je $D = L^d$, a d broj prostornih dimenzija sistema. Najznačajniji problem pri numeričkom simuliranju Boze-Ajnštajn kondenzata jeste nalaženje svojstvenih vrednosti λ ove matrice, odnosno

$$Av = \lambda v,$$

gde v predstavlja odgovarajući svojstveni vektor.

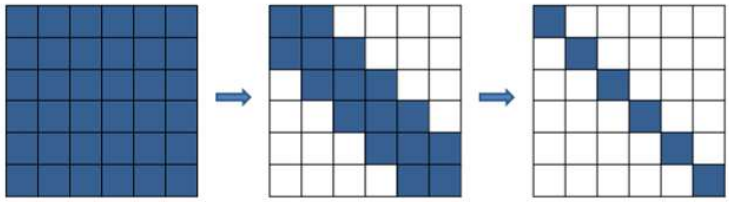
S obzirom da je posmatrana matrica A, realna $A \in \mathbb{R}^{D^2}$ i simetrična $A^T = A$, postoji ortogonalna[11] matrica P koja operacijom sličnosti $P^T AP$ pretvara matricu A u matricu koja ima dijagonalnu formu [16]. Vrednosti na dijagonali matrice $P^T AP$ predstavljaju svojstvene vrednosti matrice A, a vektori od kojih se sastoji matrica P (njene kolone) su svojstveni vektori matrice A. Broj ovako dobijenih svojstvenih vrednosti i svojstvenih vektora je D i jednak je linearnoj dimenziji matrice.

U nastavku su predstavljena dva algoritma dijagonalizacije koja proučavamo u ovom radu: algoritam potpune dijagonalizacije i Lancoš algoritam.

[11] Za ortogonalne matrice važi jednakost $PP^T = I$.

2.1 Algoritam potpune dijagonalizacije

Algoritam potpune dijagonalizacije egzaktno određuje sve svojstvene vrednosti i svojstvene vektore. Njihov broj je jednak dimenziji matrice D. Sastoji se iz dva koraka koji su ilustrovani na slici 2.1.



Slika 2.1: Ilustracija algoritma dijagonalizacije matrice: matrica se prvo svodi na tridijagonalnu formu, a zatim na dijagonalnu.

U prvom koraku matrica se transformiše u realnu simetričnu tridijagonalnu[12] matricu

$$A = QA_QQ^T$$

gde je Q pogodno odabrana ortogonalna matrica. Da bi se došlo do ove forme koristi se QR dekompozicija koju su 1961. godine nezavisno razvili Frensis[13] i Kublanovskaja[14].

U drugom koraku se izračunavaju svojstvene vrednosti i svojstveni vektori tridijagonalne matrice A_Q. Ova matrica se može dekomponovati kao $T = SA_DS^T$, gde je A_D dijagonalna matrica čiji su elementi ujedno i njene svojstvene vrednosti, a S je matrica svojstvenih vektora matrice A_Q. Ako sa Z označimo ortogonalnu matricu $Z = QS$, onda važi $A = ZA_DZ^T$, pa će vrednosti na dijagonali A_D predstavljati svojstvene vektore matrice A, a Z matricu njenih svojstvenih vektora. Primetimo da su svojstvene vrednosti za matrice A, A_Q i A_D iste.

Preciznost ovog algoritma dijagonalizacije zavisi samo od numeričke preciznosti računara. Sam algoritam ne uvodi greške ocene svojstvenih vrednosti zadate matrice, ali njegova numerička implementacija ograničava broj dobijenih značajnih cifara. S obzirom da su svojstvene vrednosti koje dobijamo dijagonalizacijom jednake $e^{-\epsilon E_k}$, mala promena vrednosti energije E_k dovodi do eksponencijalne promene svojstvenih vrednosti, što ilustruje značaj preciznog računa u okviru posmatranog problema. U ovom radu koristićemo LAPACK [17] implementaciju potpunog algoritma dijagonalizacije.

2.2 Algoritam Lancoš dijagonalizacije

Lancoš algoritam [15] je egzaktan iterativni algoritam za određivanje traženog broja (maksimalno D) svojstvenih vrednosti i vektora matrica. Posebno je pogodan za velike retke[15] matrice. Razvio ga je Lancoš[16] sredinom XX veka. Kao i potpuni algoritam, sastoji se

[12] Tridijagonalna matrica je matrica čije su sve vrednosti jednake nuli, osim vrednosti na glavnoj i prvim susednim dijagonalama.

[13] John G. F. Francis (1934-), engleski informatičar.

[14] Vera Nikolaevna Kublanovskaya (1920-), ruska matematičarka.

[15] Matrice sa velikim brojem elemenata jednakih nuli (eng. sparse).

[16] Cornelius Lanczos (1893-1974), mađarski matematičar i fizičar.

od dva koraka prikazana na slici 2.1: svođenje na tridijagonalnu formu T i dijagonalizaciju tridijagonalne matrice.

Lancošev algoritam se bazira na sledećem razmatranju. Poći ćemo od slučajnog vektora q_1 i konstruisaćemo niz vektora pomoću matrice A na sledeći način: $q_{k+1} = Aq_k$. Za ovakav niz vektora može se pokazati da će vrednost $\frac{\|q_{k+1}\|}{\|q_k\|}$ težiti najvećoj svojstvenoj vrednosti, a $\frac{q_k}{\|q_k\|}$ odgovarajućem svojstvenom vektoru matrice A. Vektori q_k čine takozvani Krilovljev[17] potprostor. U Lancoš algoritmu, pri svođenju matrice A na tridijagonalnu formu, ovi vektori se koriste za određivanje elemenata α_i, β_i simetrične tridijagonalne matrice T zapisane u obliku

$$\begin{pmatrix} \alpha_1 & \beta_1 & 0 & . & . & . & 0 \\ \beta_1 & \alpha_2 & \beta_2 & . & . & . & 0 \\ 0 & \beta_2 & \alpha_3 & . & . & . & 0 \\ . & . & & . & & & . \\ . & . & & & . & & . \\ . & . & & & & . & \beta_{k-1} \\ 0 & 0 & 0 & . & . & \beta_{k-1} & \alpha_k \end{pmatrix}$$

Broj potrebnih iteracija zavisi od broja traženih svojstvenih vrednosti i konvergencije algoritma. Tipično, broj iteracija je dvostruko veći od vrednosti broja traženih svojstvenih vrednosti n_{eig}. Radi poboljšanja preciznosti algoritma, potrebno je u svakom koraku ortonormirati skup vektora q_k.

Iteracije Lancoš algoritma možemo predstaviti u obliku pseudokoda:

$q_1 = \frac{q_{rand}}{\|q_{rand}\|}$
for $k = 1$ to $m - 1$ do
    $q_{k+1} = Aq_k$
    $\alpha_k = q_k^T q_k$
    $q_{k+1} = q_{k+1} - \alpha_k q_k$

    if $k > 1$ then
        $q_{k+1} = q_{k+1} - \beta_{k-1} q_{k-1}$
    end if

    reorthogonalize
    $\beta_k = \|q_{k+1}\|$

    if $\beta_k = 0$ then
        $q_{k+1} = \frac{q_{rand}^{new}}{\|q_{rand}^{new}\|}$
        reorthogonalize
    end if

    $q_{k+1} = \frac{q_{k+1}}{\beta_k}$
end for

[17] Alexei Nikolaevich Krylov (1863-1945), ruski mornarički inženjer i primenjeni matematičar.

Funkcija *reorthogonalize* ortonormira skup vektora q_k standardnim Gram[18]-Šmitovim[19] postupkom ortonormalizacije [16].

Pogodnost Lancoš algoritma u odnosu na potpuni algoritam je u tome što se matrica ne menja tokom iteracija, pa njeno čuvanje u memoriji nije neophodno. U slučaju kada postoji funkcija za računanje članova matrice, kao alternativa čuvanju matrice u memoriji, ova funkcija se može pozivati svaki put kada je potrebna vrednost nekog elementa matrice tokom iteracija. Na ovaj način se mogu dijagonalizovati i matrice jako velikih dimenzija, koje ne mogu da se smeste u dostupnu memoriju na računaru.

Naravno, čuvanje u memoriji, odnosno računanje matričnih elemenata, utiče na potrebno vreme i memoriju za izvršavanje algoritma.

Preciznost Lancoš algoritma zavisi, osim od numeričke preciznosti računara, takođe i od veličine matrice evolucionog operatora, kao i traženog broja svojstvenih vrednosti. Algoritam ne konvergira za sve vrednosti parametara i najbolje se ponaša za probleme u kojima je potrebno izračunati mali broj svojstvenih vrednosti, $n_{eig} \lesssim 100$. I pored ovih negativnih strana algoritma, osobine kao što su brza konvergencija i mogućnost rada sa velikim matricama ga favorizuju u mnogim praktičnim primenama.

2.3 Implementacija

Svi opisani algoritmi su implementirani u programskom jeziku C/C++.

Za potpunu dijagonalizaciju[20] (FD) korišćena je biblioteka LAPACK [17] napisana za rešavanje problema linearne algebre u programskom jeziku Fortran 77. Odgovarajući kod je prikazan u Dodatku A.1. Vremenska kompleksnost ovog algoritma je

$$\tau^{FD} = O(D^3),$$

gde je D linearna dimenzija matrice koja se dijagonalizuje. Memorijska kompleksnost algoritma je

$$M^{FD} = O(D^2),$$

a glavni faktori ove zavisnosti dolaze od potrebe čuvanja matrice koja se dijagonalizuje, kao i svojstvenih vektora koji se računaju.

Za dijagonalizaciju Lancoš algoritmom razvijen je C/C++ program prikazan u Dodatku A.2, koji ima dva ključna moda koji su razmatrani zasebno:

1. Evoluciona matrica se računa na početku algoritma i u celosti čuva u memoriji tokom izvršavanja algoritma (LDm).
2. Evoluciona matrica se ne čuva u memoriji, već se njeni elementi izračnavaju po potrebi tokom iteracija algoritma ($LDnm$).

[18] Jørgen Pedersen Gram (1850-1916), danski statističar i matematičar.
[19] Erhard Schmidt (1876-1959), nemački matematičar.
[20] Na engleskom jeziku: Full Diagonalization.

U obe implementacije Lancoš algoritma procenjena vremenska kompleksnost je

$$\tau^{LD} = O(n_{eig}^3 + n_{eig}D^2) \approx O(n_{eig}D^2),$$

ali u drugom slučaju ($LDnm$), prefaktor je značajno veći i zavisi od složenosti funkcije za računanje amplitude verovatnoće, odnosno nivoa p efektivnog dejstva. U Dodatku A.3 prikazan je kod funkcije jednodimenzionog efektivnog dejstva nivoa $p = 6$. Memorijska kompleksnost u slučaju sa čuvanjem matrice je

$$M^{LDm} = O(D^2).$$

Ovde memorijom dominira matrica koja se dijagonalizuje, dok je u slučaju bez čuvanja matrice upotreba memorije mnogo manja, pa najviše resursa odlazi na čuvanje vektora q_k, što čini memorijsku kompleksnost značajno nižom

$$M^{LDnm} = O(n_{eig}^2 + n_{eig}D) \approx O(n_{eig}D).$$

Pored gore navedenih teorijskih ocena kompleksnosti algoritama, sve ove zavisnosti ćemo pokazati u sledećem poglavlju na osnovu rezultata numeričkih simulacija, odnosno merenjem njihovog vremena izvršavanja i zauzete memorije na računaru.

3

Rezultati

Rezultati su dobijeni pokretanjem simulacija u Linux okruženju na računarima sa četvorokornim 64-bitnim procesorima Intel Xeon E5345 i dvokornim 64-bitnim Intel Core 2 Duo T5500 sa 32-bitnim OS. Sve jednačine su reskalirane tako da se operiše bezdimenzionim jedinicama.

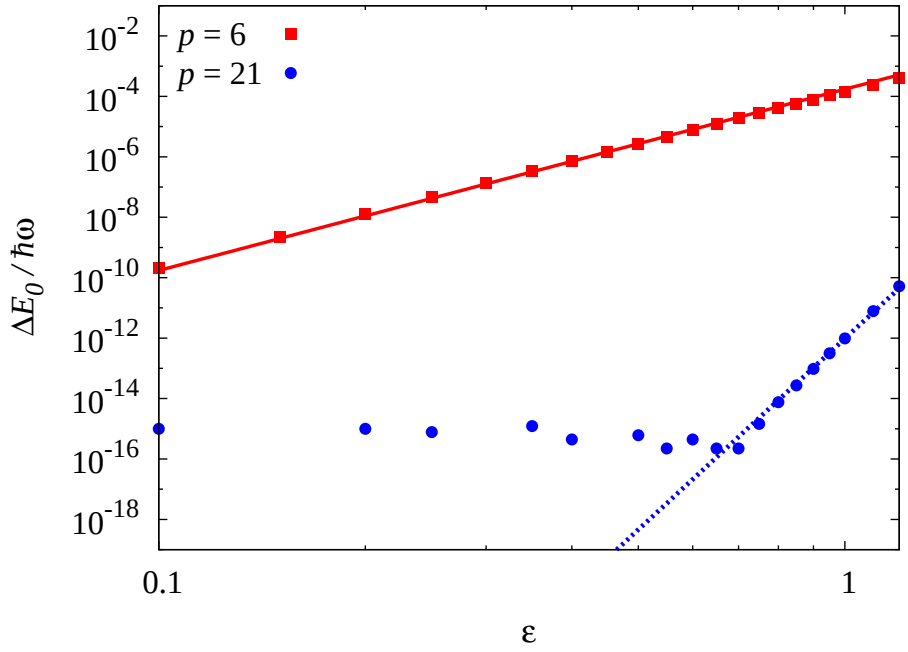
Prikazani rezultati prvo ispituju greške koje uvodi korišćeni pristup: greške računanja matričnih elemenata, greške diskretizacije i numeričke greške. Zatim analiziraju vremensku kompleksnost algoritama i na kraju pokazuju globalne i lokalne osobine dobijene za sistem gasa idealnih bozona ^{87}Rb u rotirajućoj magnetno-optičkoj zamci u BAK fazi.

Da bismo proverili preciznost metoda efektivnih dejstava koje koristimo za računanje elemenata matrice evolucionog operatora i odredili oblast vremenske propagacije ϵ za koju se dobija zadovoljavajuća preciznost, posmatrali smo greške koje ovaj metod uvodi pri računanju energetskih nivoa harmonijskog oscilatora u odnosu na analitički poznato rešenje. Slika 3.1 pokazuje grešku procene energije osnovnog stanja metoda efektivnih dejstava koja stepeno zavisi od vremena propagacije i srazmerna je sa ϵ^p, gde je p odabrani nivo efektivnog dejstva. Za vrednost $p = 21$ u levom delu grafika vidimo da numerička preciznost računara ograničava preciznost rezultata više nego primenjeni aproksimativni metod, kada greška padne na vrednost reda veličine 10^{-15}.

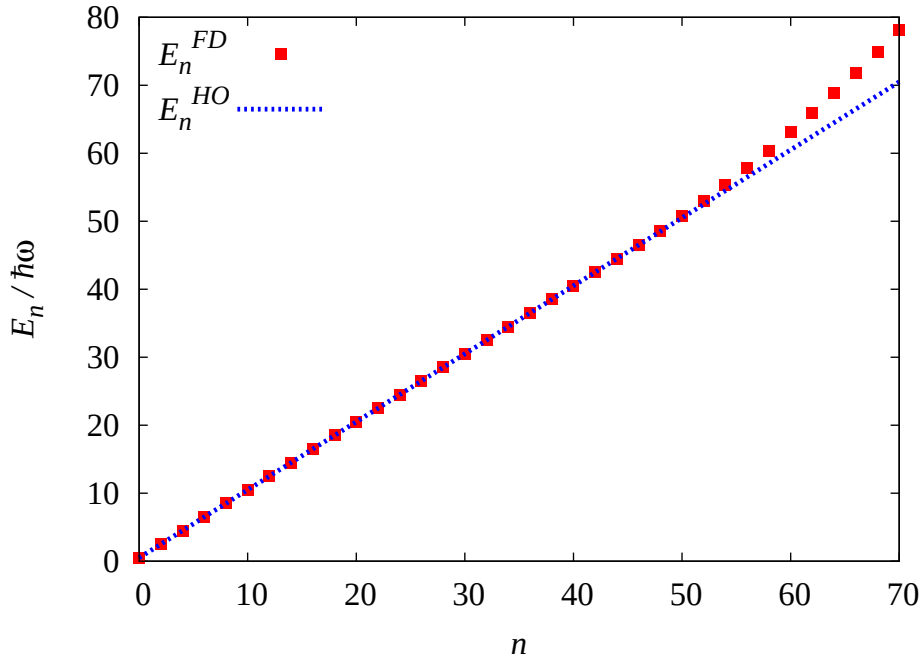
Zbog uvedene diskretizacije prostora i numeričke greške pri računanju svojstvenih vrednosti i vektora matrice A u stanju smo da precizno odredimo samo deo svojstvenih stanja. Slike 3.2 i 3.3 analiziraju tačnost određivanja energetskih nivoa (svojstvenih vrednosti), dok slike 3.4 i 3.5 pokazuju tačnost određivanja vektora stanja (svojstvenih vektora). Kroz ove grafike ilustrovana je srazmernost tačnosti algoritama potpune i Lancoš dijagonalizacije.

U numeričkom pristupu posmatranom problemu, dijagonalizacija matrica predstavlja najskuplji korak pri korišćenju računarskih resursa. Slika 3.6 pokazuje da je vremenska kompleksnost potpunog algoritma dijagonalizacije srazmerna trećem stepenu dimenzije matrice.

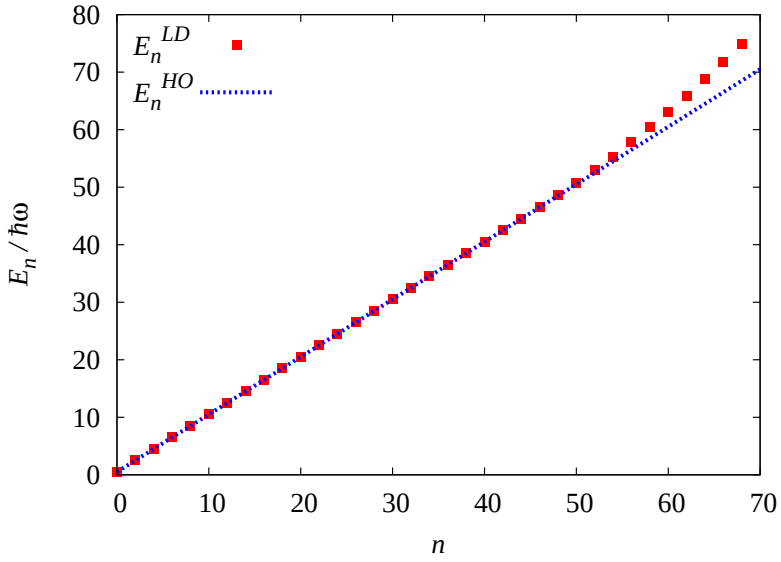
Slike 3.7, 3.8 i 3.9 pokazuju da je vremenska kompleksnost za sve implementacije Lancoš algoritma dijagonalizacije (sa i bez čuvanja matrice koja se dijagonalizuje, za različiti broj traženih svojstvenih vrednosti i vektora, za različite složenosti funkcija evolucionog operatora, što odgovara različitim nivoima efektivnih dejstava p) srazmerna sa drugim stepenom dimenzije matrice D^2, a da konstanta koja množi D^2 raste sa brojem traženih svojstvenih



Slika 3.1: Greška procene energije osnovnog stanja jednodimenzionog harmonijskog oscilatora ($E_0 = \frac{1}{2}\hbar\omega$) računate metodom efektivnih dejstava za slučaj $p = 6$ i $p = 21$ u zavisnosti od korišćenog vremena propagacije ϵ. Prikazani su i rezultati fita na stepene funkcije $\Delta E_0^{p=6}(\epsilon)/\hbar\omega = 1.730(9) \cdot 10^{-4}\ \epsilon^6$ i $\Delta E_0^{p=21}(\epsilon)/\hbar\omega = 9.67(9) \cdot 10^{-13}\ \epsilon^{21}$. U simulaciji je korišćena matrica dimenzije $D = 2000$ za oblast $|x| \leq 10$, $\Delta = 0.01$ parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



Slika 3.2: Prvih 70 energija jednodimenzionog harmonijskog oscilatora dobijenih pomoću potpunog algoritma dijagonalizacije (FD) i analitičko rešenje (HO), dato sa $E_n = (n + \frac{1}{2})\hbar\omega$. U simulaciji je korišćena matrica dimenzije $D = 2000$ za oblast $|x| \leq 10$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



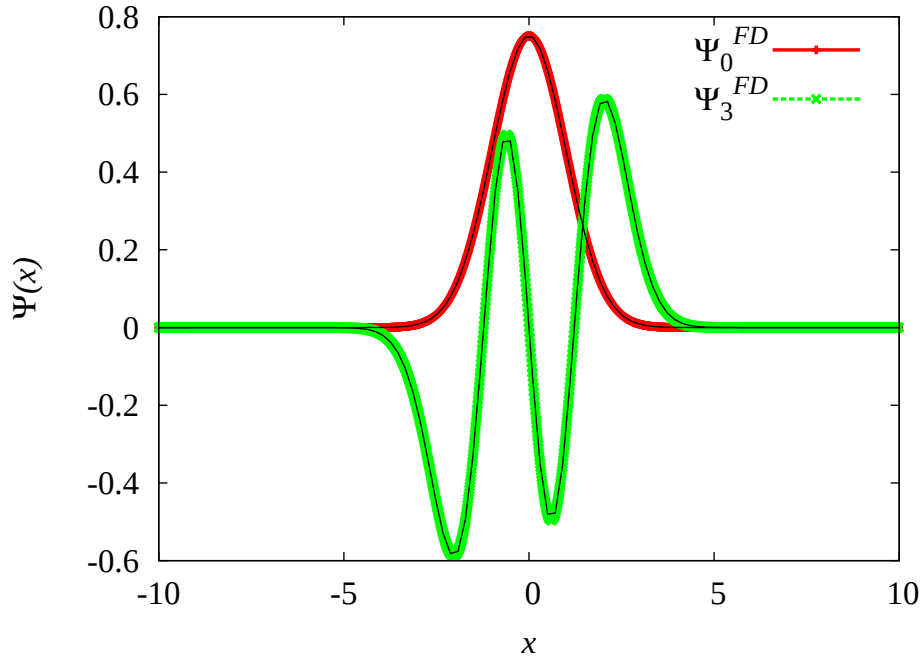
Slika 3.3: Prvih 70 energija jednodimenzionog harmonijskog oscilatora dobijenih pomoću Lancoš algoritma dijagonalizacije (LD) i analitičko rešenje (HO), dato sa $E_n = (n + \frac{1}{2})\hbar\omega$. U simulaciji je korišćena matrica dimenzije $D = 2000$ za oblast $|x| \leq 10$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.

vrednosti i složenošću računanja matričnih elemenata evolucionog operatora.

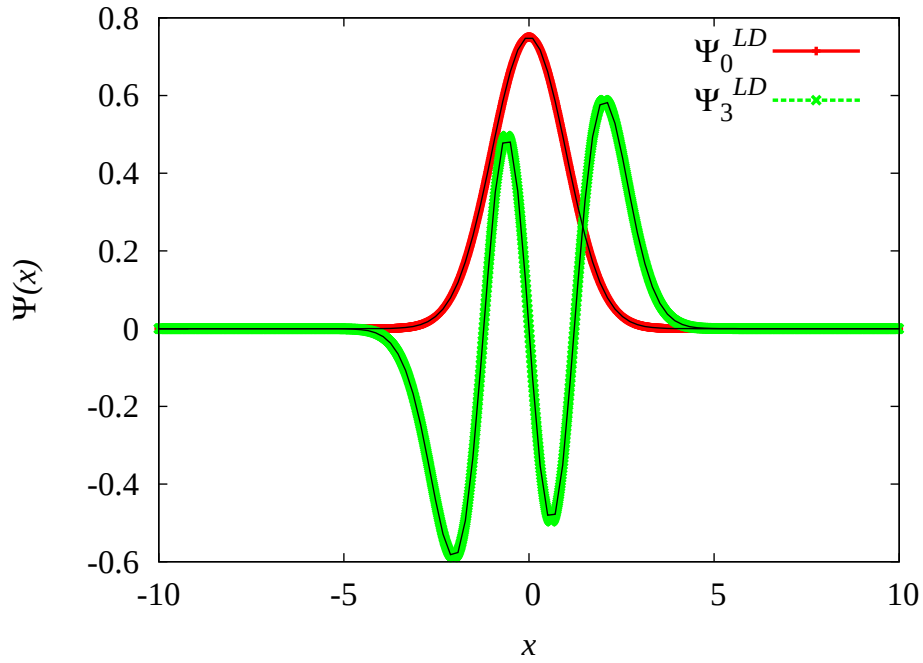
Slike 3.10 i 3.11 pokazuju da je u obe implementacije Lancoš algoritma vremenska kompleksnost linearno srazmerna broju traženih svojstvenih vrednosti. Udružene, slike 3.7-3.11 verifikuju da je vremenska kompleksnost Lancoš algoritma jednaka $O(n_{eig}D^2)$, kao što smo smo izneli u prethodnom poglavlju.

Numerička dijagonalizacija je i memorijski zahtevna. Na slici 3.12 vidimo da je memorijska kompleksnost algoritma potpune dijagonalizacije srazmerna drugom stepenu linearne dimenzije matrice D. Ista memorijska kompleksnost je dobijena i na slici 3.13 za Lancoš algoritam sa čuvanjem matrice, dok je značajno poboljšanje u vidu korišćenja memorijskih resursa, postignuto Lancoš algoritmom bez čuvanja matrice, čije su memorijske potrebe linearno srazmerne dimenziji matrice D, kao što se vidi na grafiku 3.14.

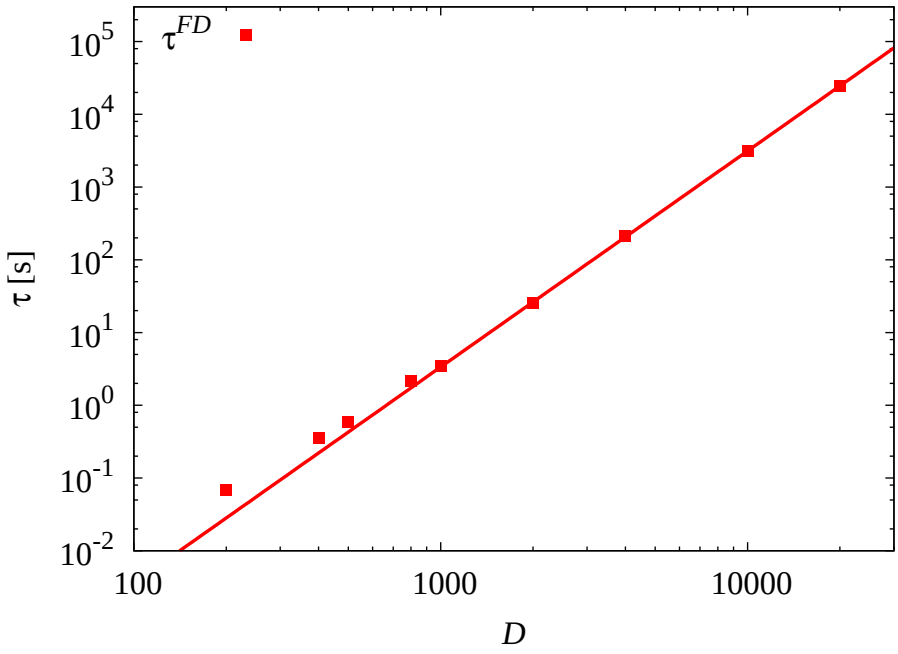
Razvijeni algoritam Lancoš dijagonalizacije matrice A evolucionog operatora smo primenili na proučavanje gasa idealnih bozona ^{87}Rb zarobljenih u rotirajućoj magnetno-optičkoj zamci. Na slici 3.15 prikazana je naseljenost osnovnog stanja takvog sistema za temperature manje ili jednake kondenzacionoj temperaturi T_c, za koju naseljenost osnovnog stanja postaje praktično jednaka nuli. Ovaj oblik zavisnosti je dobro poznat u literaturi [1, 2, 3] i može se koristiti za određivanje T_c. Na slici 3.16 pokazano je kako kondenzaciona temperatura raste sa brojem bozona u sistemu. Teorijski [1, 2, 3], za harmonijski oscilator ($k_n = 0$) ova zavisnost ima oblik $T_c \sim N^{1/2}$ za kondenzat u harmonijskoj zamci. Za posmatrani sistem sa malim anharmonicitetom ($k_n = 0.00195$) dobija se slična zavisnost što je i pokazano odgovarajućim fitom stepene funkcije. Slika 3.17 prikazuje gustinu čestica u zavisnosti od položaja u $x - y$ ravni na temperaturi ispod T_c, bliskoj apsolutnoj nuli. Dobijeni izraženi pik ima karakterističnu oblik koji se dobija u BAK eksperimentima [1, 2, 3] i predstavlja jedan od načina potvrde da je sistem doživeo fazni prelaz.



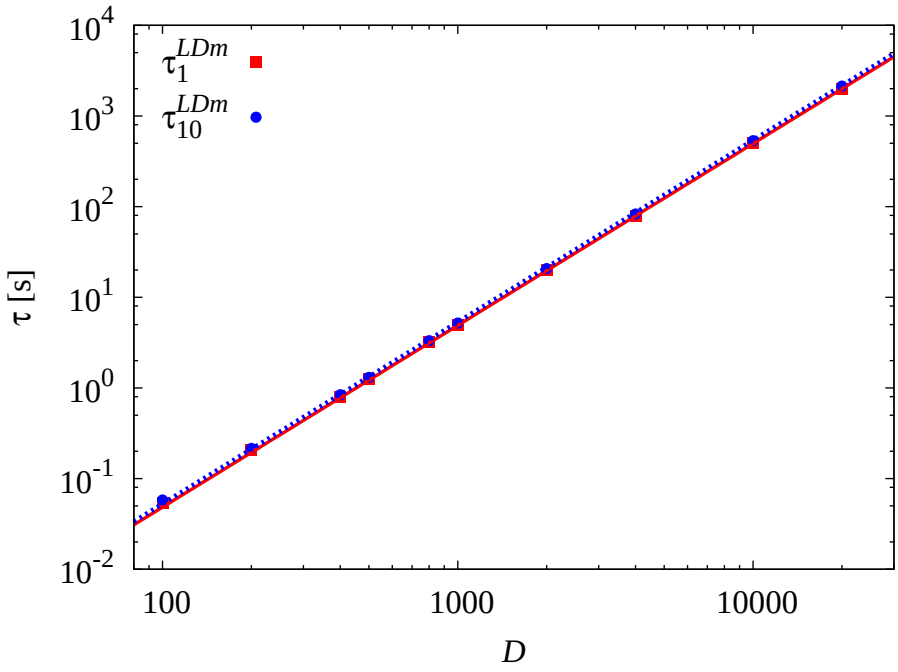
Slika 3.4: Svojstvena stanja jednodimenzionog harmonijskog oscilatora za osnovni i treći energetski nivo dobijena pomoću potpunog algoritma dijagonalizacije (FD). Linijama su prikazana analitička rešenja $\Psi_n(x) = \sqrt{\frac{1}{2^n n!}}\pi^{-\frac{1}{4}} e^{-\frac{x^2}{2}} H_n(x)$, gde H_n predstavlja Hermitov polinom. U simulaciji je korišćena matrica dimenzije $D = 2000$ za oblast $|x| \leq 10$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



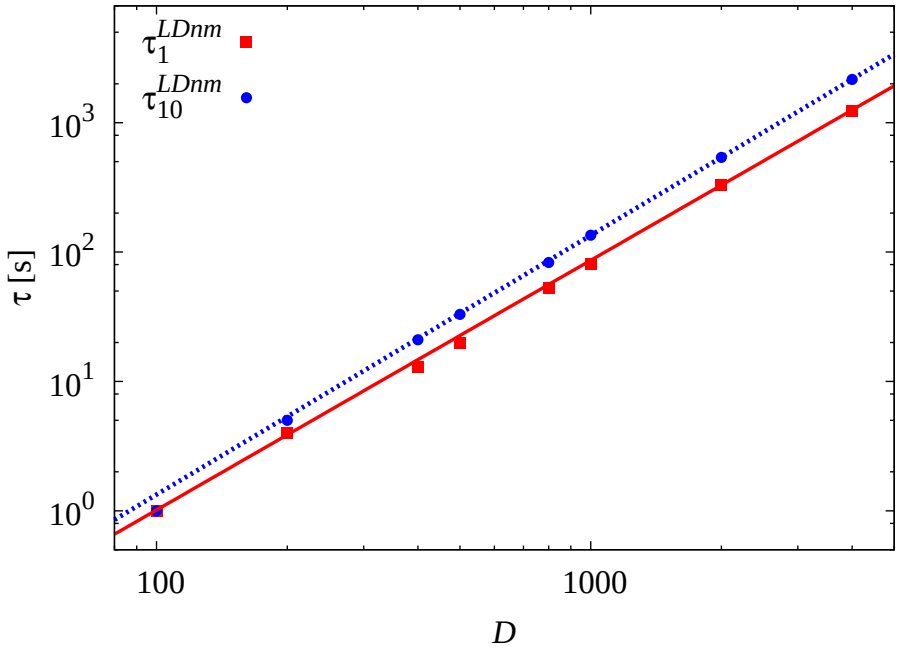
Slika 3.5: Svojstvena stanja jednodimenzionog harmonijskog oscilatora za osnovni i treći energetski nivo dobijena pomoću Lancoš algoritma dijagonalizacije (LD). Analitičko rešenje, kao i svi ostali parametri su isti kao na prethodnoj slici.



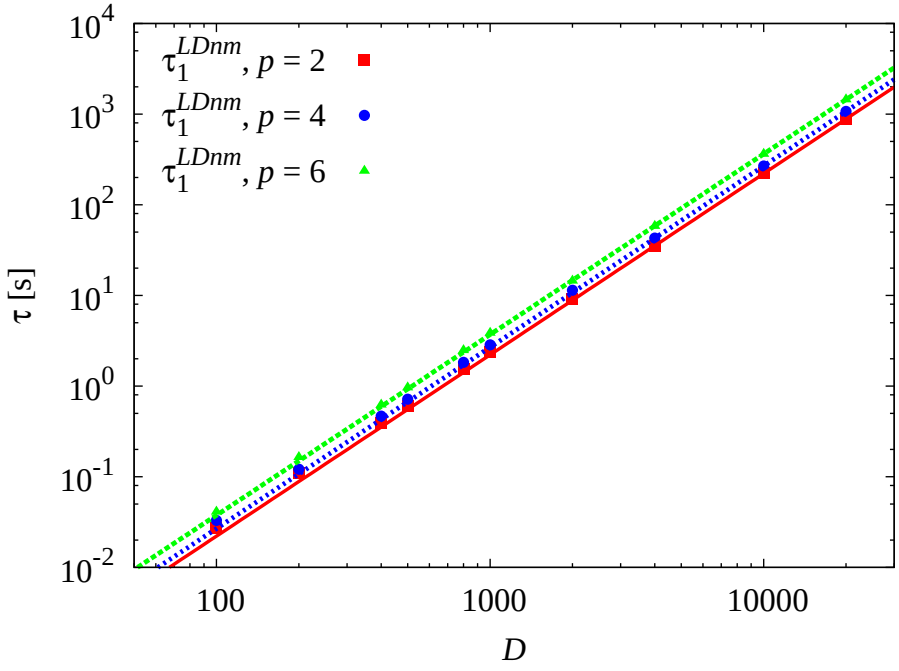
Slika 3.6: Vreme izvršavanja potpunog algoritma dijagonalizacije (FD) u zavisnosti od veličine matrice koja se dijagonalizuje fitovana na stepenu zavisnost $\tau^{FD}(D) = 4.13(2) \cdot 10^{-9}\text{s} \cdot D^{2.9704(5)}$. Dijagonalizovan je jednodimenzionalni BAK potencijal sa anharmonicitetom $k_n = 10^{-3}$ za posmatranu oblast $|x| \leq 10$, $\Delta = \frac{20}{D}$, vreme propagacije $\epsilon = 0.1$, brzine rotacije $r = 0$ i nivoa efektivnog dejstva $p = 21$.



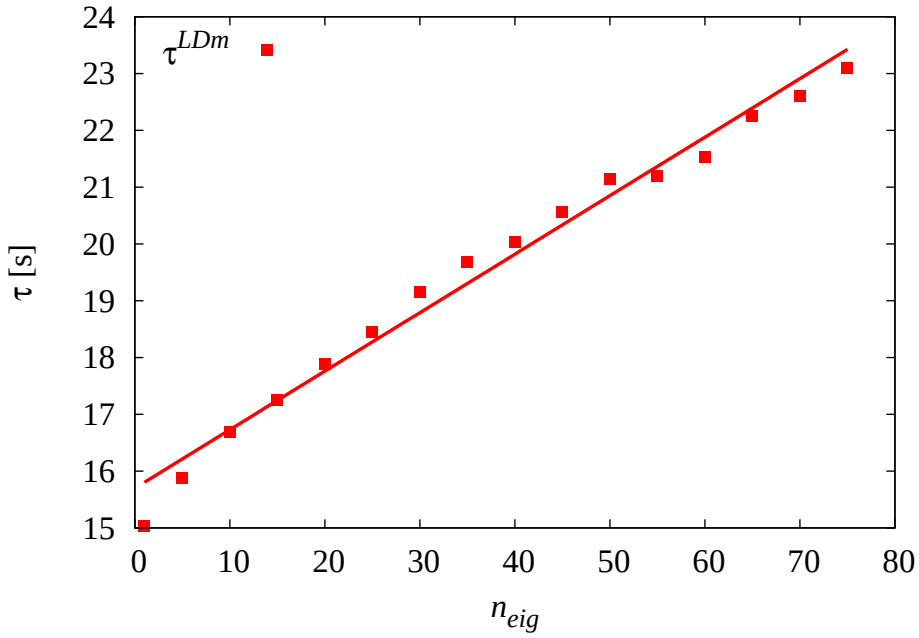
Slika 3.7: Vreme izvršavanja Lancoš algoritma dijagonalizacije sa čuvanjem matrice koja se dijagonalizuje (LDm) kada se računa jedna, odnosno 10, svojstvenih vrednosti u zavisnosti od veličine matrice, fitovano na stepene zavisnosti $\tau^{LDm}_{n_{eig}=1}(D) = 4.7(3) \cdot 10^{-6}\text{s} \cdot D^{2.006(5)}$ i $\tau^{LDm}_{n_{eig}=10}(D) = 5.1(2) \cdot 10^{-6}\text{s} \cdot D^{2.006(2)}$. Dijagonalizovan je jednodimenzionalni BAK potencijal sa anharmonicitetom $k_n = 1$ za posmatranu oblast $|x| \leq 10$, $\Delta = \frac{20}{D}$,vreme propagacije je $\epsilon = 0.1$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



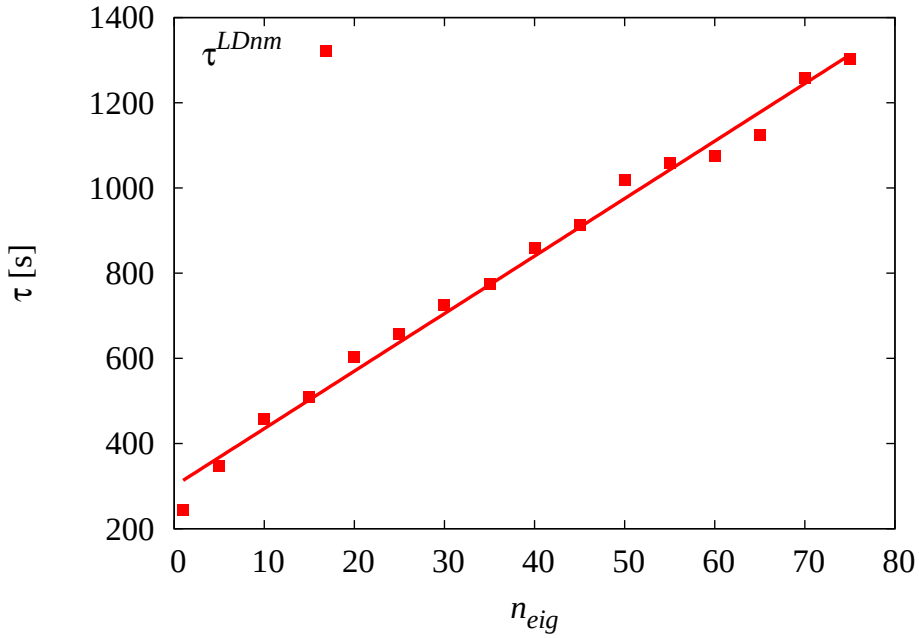
Slika 3.8: Vreme izvršavanja Lancoš algoritma dijagonalizacije bez čuvanja matrice koja se dijagonalizuje ($LDnm$) kada se računa jedna, odnosno 10, svojstvenih vrednosti u zavisnosti od veličine matrice, fitovano na stepene zavisnosti $\tau^{LDnm}_{n_{eig}=1}(D) = 1.4(1)\cdot 10^{-4}\text{s}\cdot D^{1.93(1)}$ i $\tau^{LDnm}_{n_{eig}=10}(D) = 1.34(3)\cdot 10^{-4}\text{s}\cdot D^{2.004(3)}$. Ostali parametri sistema su isti kao na prethodnoj slici.



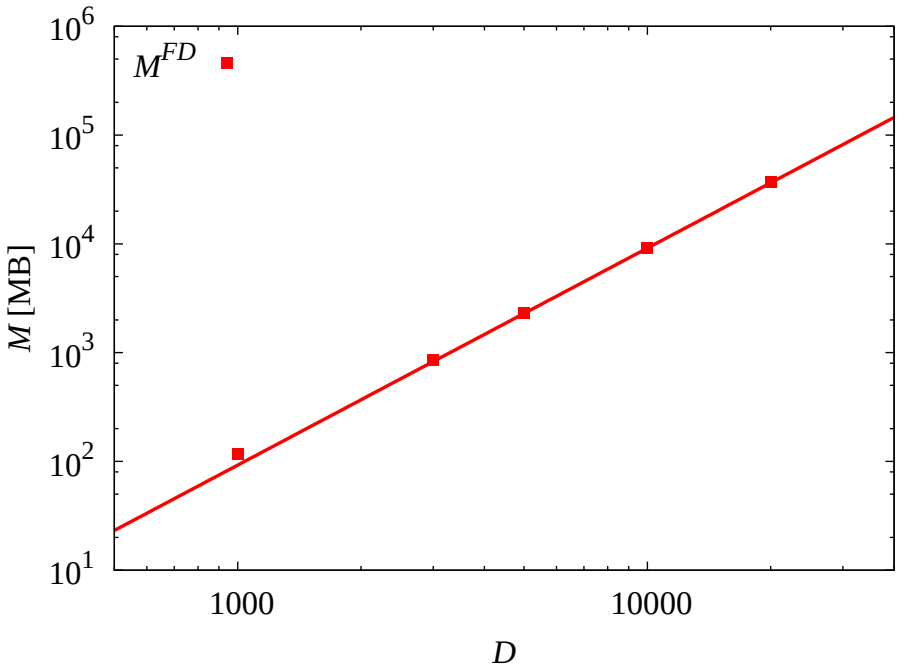
Slika 3.9: Vreme izvršavanja Lancoš algoritma dijagonalizacije bez čuvanja matrice koja se dijagonalizuje ($LDnm$) kada se računa jedna svojstvena vrednost, za funkcije sa različitom vrednošću nivoa p efektivnog dejstva, u zavisnosti od veličine matrice fitovano na stepene zavisnosti $\tau^{LDnm}_{p=2}(D) = 2.21(6)\cdot 10^{-6}\text{s}\cdot D^{2.000(3)}$, $\tau^{LDnm}_{p=4}(D) = 2.68(4)\cdot 10^{-6}\text{s}\cdot D^{2.000(2)}$ i $\tau^{LDnm}_{p=6}(D) = 3.88(3)\cdot 10^{-6}\text{s}\cdot D^{1.993(1)}$. Dijagonalizovan je jednodimenzionalni BAK potencijal za posmatranu oblast $|x| \leq 10$, $\Delta = \frac{20}{D}$, vreme propagacije je $\epsilon = 0.1$, anharmonicitet $k_n = 1$, a parametar rotacije $r = 0$.



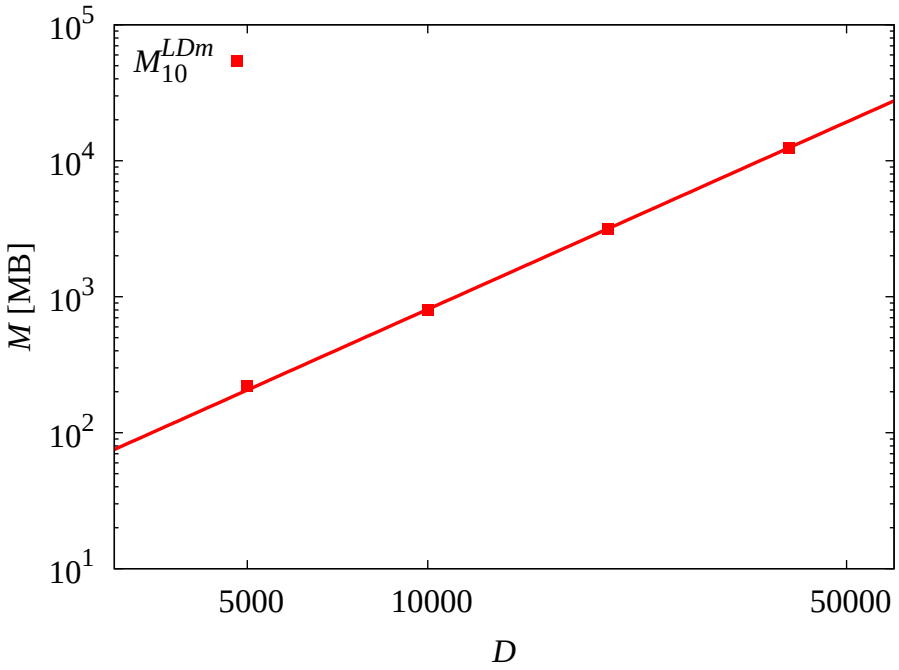
Slika 3.10: Vreme izvršavanja Lancoš algoritma sa čuvanjem matrice koja se dijagonalizuje (LDm) u zavisnosti od traženog broja svojstvenih vrednosti fitovana na linearnu zavisnost $\tau^{LDm}(n_{eig}) = 0.103(4)\text{s} \cdot n_{eig} + 15.7(2)\text{s}$. Dijagonalizovan je jednodimenzionalni BAK potencijal, korišćena je matrica dimenzije $D = 2000$ za oblast $|x| \leq 10$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 10^{-3}$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



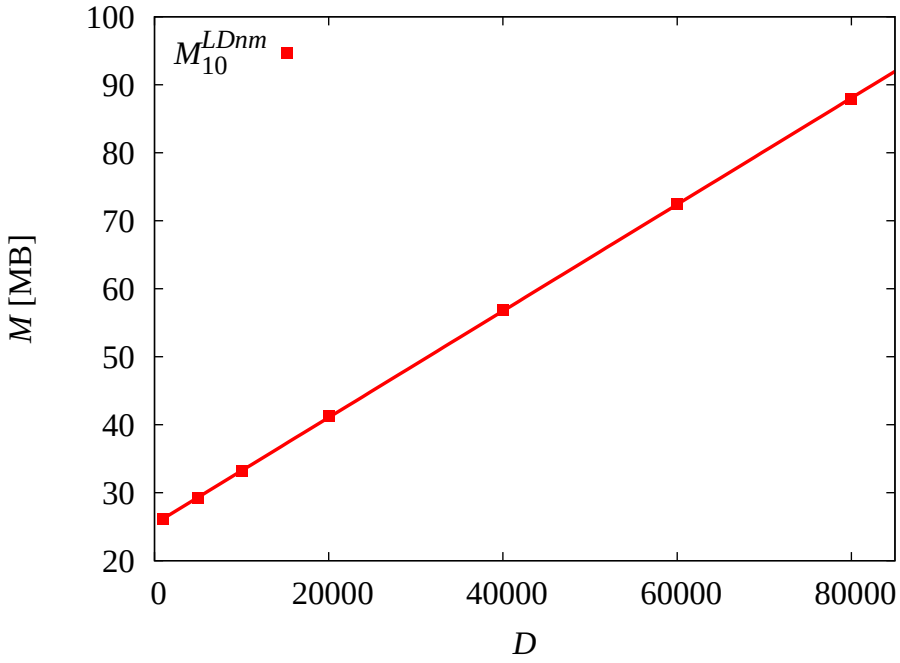
Slika 3.11: Vreme izvršavanja Lancoš algoritma bez čuvanja matrice koja se dijagonalizuje ($LDnm$) u zavisnosti od traženog broja svojstvenih vrednosti fitovana na linearnu zavisnost $\tau^{LDnm}(n_{eig}) = 13.5(4)\text{s} \cdot n_{eig} + 300(20)\text{s}$. Dijagonalizovan je jednodimenzionalni BAK potencijal, korišćena je matrica dimenzije $D = 2000$ za oblast $|x| \leq 10$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 10^{-3}$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



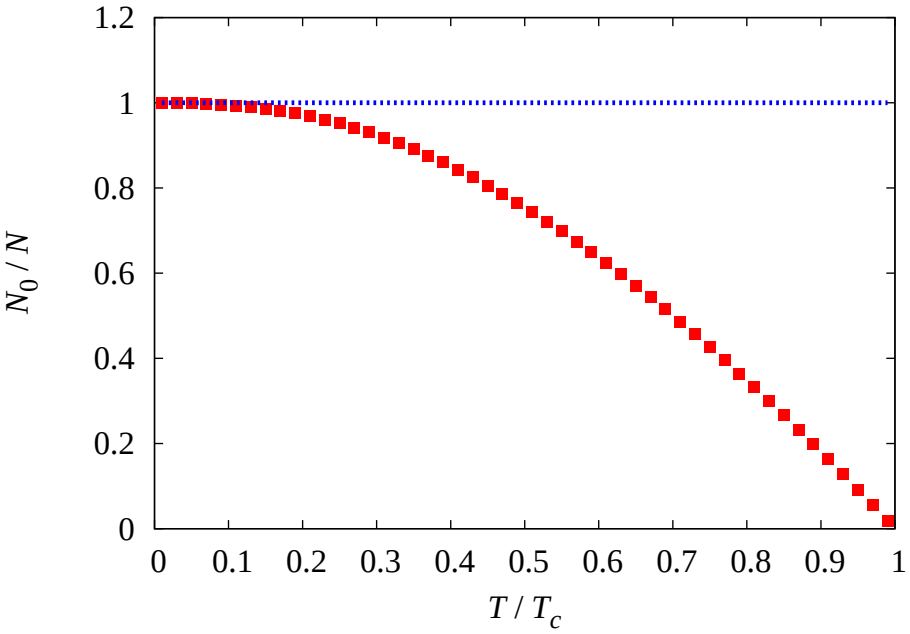
Slika 3.12: Memorija potrebna za izvršavanje programa koji koristi algoritam potpune dijagonalizacije (FD) za rešavanje svojstvenog problema matrice A evolucionog operatora jednodimenzionog sistema u zavisnosti od linearne dimenzije D matrice koja se dijagonalizuje, fitovana na stepenu zavisnost $M^{FD}(D) = 9.6(3) \cdot 10^{-5}\text{MB} \cdot D^{1.995(3)}$. Dijagonalizovan je jednodimenzionalni BAK potencijal sa za oblast $|x| \leq 10$, $\Delta = \frac{20}{D}$, vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 0$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



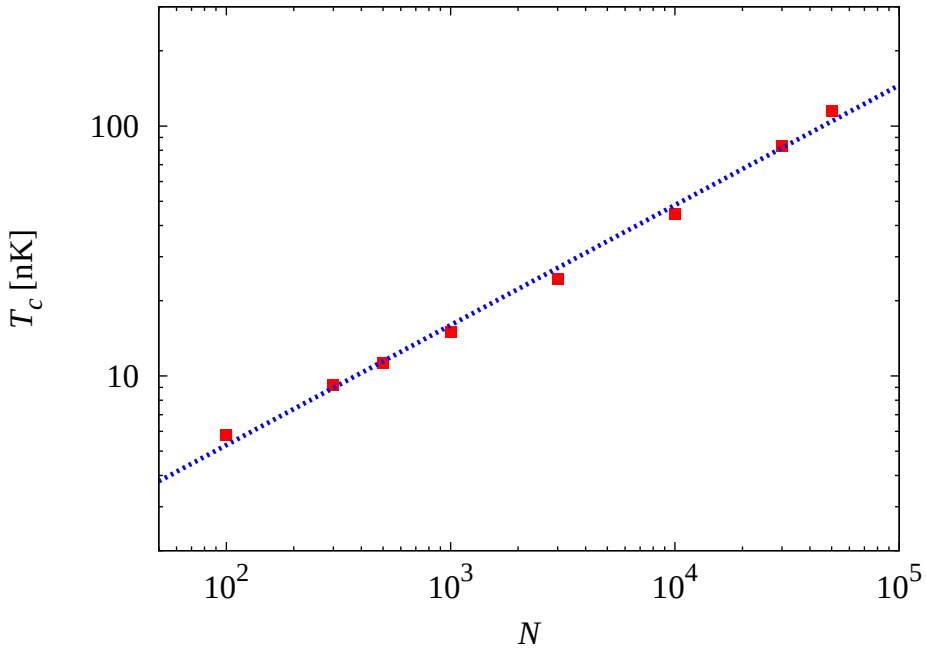
Slika 3.13: Memorija potrebna za izvršavanje programa koji koristi Lancoš algoritam dijagonalizacije sa čuvanjem matrice koja se dijagonalizuje (LDm) za rešavanje svojstvenog problema matrice A evolucionog operatora jednodimenzionog sistema u zavisnosti od linearne dimenzije D matrice koja se dijagonalizuje, fitovana na stepenu zavisnost $M^{LDm}(D) = 1.05(6) \cdot 10^{-5}\text{MB} \cdot D^{1.971(6)}$. Ostali parametri sistema su $\Delta = 0.1$, $|x| \leq D\Delta/2$, vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 0$, parametar rotacije $r = 0$ i nivo efektivnog dejstva $p = 21$.



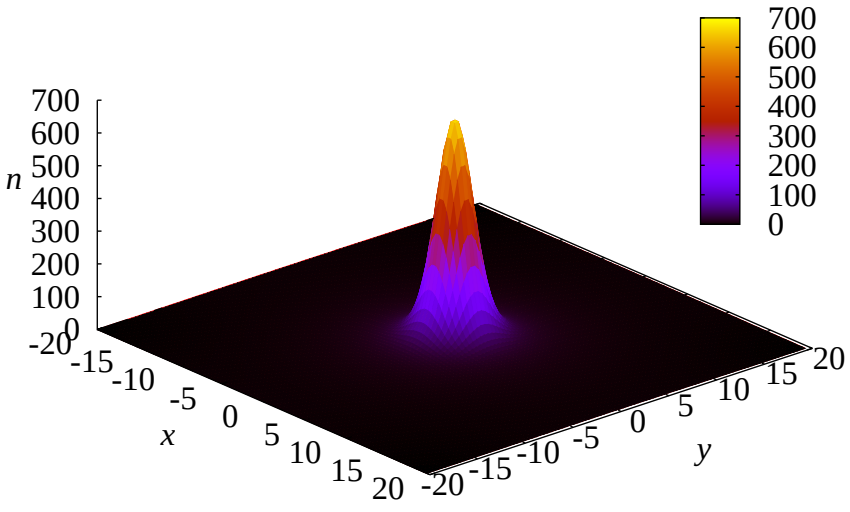
Slika 3.14: Memorija potrebna za izvršavanje programa koji koristi Lancoš algoritam dijagonalizacije bez čuvanja matrice koja se dijagonalizuje (LDnm) za rešavanje svojstvenog problema matrice A evolucionog operatora jednodimenzionog sistema u zavisnosti od linearne dimenzije D matrice koja se dijagonalizuje, fitovana na linearnu zavisnost $M^{LDnm}(D) = 7.83(3) \cdot\ 10^{-4}\text{MB} \cdot D + 25.4(9)\text{MB}$. Ostali parametri sistema su isti kao na prethodnoj slici.



Slika 3.15: Udeo od $N = 10000$ čestica koje se nalaze u osnovnom stanju u zavisnosti od temperature koja je manje od kondenzacione. U ovom slučaju kondenzaciona temperatura iznosi $T_c = 44.567$ nK. Dijagonalizovan je dvodimenzionalni BAK potencijal. U simulaciji je korišćena matrica dimenzije $D = 10000$ za oblast $|x|, |y| \leq 20$, $\Delta = 0.04$ vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 0.00195$, parametar rotacije $r = 0.95833$ i nivo efektivnog dejstva $p = 21$.



Slika 3.16: Zavisnost kondenzacione temperature T_c od broja bozona u sistemu N fitovana na stepenu funkciju $T_c(N) = 0.58(6)\text{nK}\cdot N^{0.48(2)}$. Dijagonalizovan je dvodimenzionalni BAK potencijal. U simulaciji je korišćena matrica dimenzije $D = 10000$ za oblast $|x|, |y| \leq 20$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 0.00195$, parametar rotacije $r = 0.95833$ i nivo efektivnog dejstva $p = 21$.



Slika 3.17: Gustina čestica n u sistemu u zavisnosti od položaja u $x-y$ ravni za gas $N = 10000$ idealnih bozona u rotirajućem anharmoniskom potencijalu na temperaturi $T = 0.01T_c$, $T_c = 44.567$nK. Dijagonalizovan je dvodimenzionalni BAK potencijal. U simulaciji je korišćena matrica dimenzije $D = 10000$ za oblast $|x|, |y| \leq 20$, $\Delta = 0.01$, vreme propagacije $\epsilon = 0.1$, anharmonicitet $k_n = 0.00195$, parametar rotacije $r = 0.95833$ i nivo efektivnog dejstva $p = 21$.

4

Diskusija

Numerički rezultati prezentovani u prethodnom poglavlju ilustruju i verifikuju teorijski izvedene zavisnosti kompleksnosti proučavanihh algoritama, kao i ranije izvedene zavisnosti grešaka primenjenog metoda.

Sa slike 3.1 vidimo da metod efektivnih dejstava za vrednost $\epsilon < 1$ i različite vrednosti parametra p daje visoku preciznost proporcionalnu sa ϵ^p. U većini simulacija u ovom radu smo koristili vrednosti $p = 21$ i $\epsilon = 0.1$. Na ovom grafiku vidimo da za te vrednosti metod unosi manju grešku od uobičajene, koja potiče od reprezentacije realnih (double) brojeva na računaru, pa zbog toga za vrednost $p = 21$, dobijamo saturaciju greške oko vrednosti $\Delta E_0 \sim 10^{-15}$. To nam pokazuje da izabrani metod računanja elemenata evolucione matrice ne unosi značajnu grešku, odnosno da možemo da smatramo da su matrični elementi izračunati egzaktno.

Slika 3.2 pokazuje izračunate vrednosti energetskih nivoa potpunim algoritmom dijagonalizacije. S obzirom da računamo uvodeći određenu diskretizaciju i uz dvostruku preciznost (oko 15 značajnih cifara), dobijamo prikazano odstupanje usled greške diskretizacije i numeričke greške. Na slici 3.3 su prikazane vrednosti energetskih nivoa izračunati Lancoš algoritmom. Ako uporedimo odstupanje ovih vrednosti sa odstupanjem vrednosti na prethodnom grafiku, zaključujemo da su vrlo slične. Stoga se, za potrebe ovog rada, tačnost Lancoš algoritma može smatrati podjednakom potpunom algoritmu. Za praktične primene ovo se uvek mora proveriti. Dobar kriterijum je poređenje numerički dobijene gustine stanja sa semiklasičnom aproksimacijom gustine stanja [8].

Na slikama 3.4 i 3.5 vidimo da računanje funkcija stanja (kao svojstvenih vektora evolucione matrice) i potpunim i Lancoš algoritmom daju rezultate koji se odlično poklapaju sa analitičkim rešenjima, kada su ona poznata[21]. Ovo nam pokazuje da Lancoš algoritam uspešno računa i svojstvena stanja čestice.

Kao što se vidi sa slike 3.6, vremenska kompleksnost potpunog algoritma dijagonalizacije je $\tau^{FD} = O(D^3)$, dok je kod Lancoš algoritma data mnogo povoljnija zavisnost $\tau^{LD} = O(n_{eig}D^2)$ (slike 3.7–3.11). U slučaju kada je potrebno naći sve svojstvene vrednosti, oba algortima su iste klase vremenske kompleksnosti. U praksi se najčešće javlja potreba za računanjem samo prvih nekoliko svojstvenih vrednosti, stoga Lancoš algoritam može značajno ubrzati rešavanje problema. Ako je broj energetskih vrednosti koje treba izračunati

[21] Primetimo da su funkcije stanja računate za nivoe čije su energije precizno određene ovim algoritmima.

konstantan (npr. $n_{eig} = 10$), važi relacija $\tau^{LD} \sim (\tau^{FD})^{2/3}$, a vremenske složenosti algoritama za različiti broj dimenzija sistema su date u Tabeli 4.1.

vremenska kompleksnost τ	$d = 1$	$d = 2$	$d = 3$
algoritam potpune dijagonalizacije (FD)	$O(L^3)$	$O(L^6)$	$O(L^9)$
Lancoš algoritam (LD)	$O(L^2)$	$O(L^4)$	$O(L^6)$

Tabela 4.1: Vremenska kompleksnost potpunog i Lancoš algoritma dijagonalizacije pri konstantnom broju traženih svojstvenih vrednosti, u jednoj, dve i tri dimenzije, u kojima koordinate uzimaju diskretne vrednosti $\{-\frac{L}{2}\Delta, -(\frac{L}{2}-1)\Delta, ..., (\frac{L}{2}-2)\Delta, (\frac{L}{2}-1)\Delta\}$, gde Δ predstavlja prostorni diskretizacioni korak.

Memorijska kompleksnost za potpuni algoritam je $M^{FD} = O(D^2)$, kao i za Lancoš algoritam sa čuvanjem matrice koja se dijagonalizuje $M^{LDm} = O(D^2)$, dok je za Lancoš algoritam bez čuvanja matrice mnogo povoljnija, $M^{LDnm} = O(n_{eig}D)$. Ovo se u slučaju kada je potrebno izračunati sve svojstvene vrednosti svodi na prethodih $O(D^2)$. Lancoš algoritam bez čuvanja matrice je, iako iste klase vremenske kompleksnosti, u principu sporiji od analogona sa čuvanjem matrice, jer je neophodno uvek nanovo računati matrične elemente. Ipak, u slučaju kada je memorija ograničavajući resurs, Lancoš algoritam bez čuvanja matrice može da predstavlja jedino odgovarajuće rešenje za problem. Pri fiksiranom broju traženih svojstvenih vrednosti (npr. $n_{eig} = 10$), važi relacija $M^{LDnm} \sim (M^{FD})^{1/2} \sim (M^{LDm})^{1/2}$, što je potvrđeno slikama 3.12–3.14. Memorijske složenosti algoritama za različiti broj dimenzija sistema su date u Tabeli 4.2.

memorijska kompleksnost M	$d = 1$	$d = 2$	$d = 3$
algoritam potpune dijagonalizacije (FD)	$O(L^2)$	$O(L^4)$	$O(L^6)$
Lancoš algoritam sa čuvanjem matrice (LDm)	$O(L^2)$	$O(L^4)$	$O(L^6)$
Lancoš algoritam bez čuvanja matrice (LDnm)	$O(L)$	$O(L^2)$	$O(L^3)$

Tabela 4.2: Memorijska kompleksnost potpunog i Lancoš algoritama dijagonalizacije pri konstantnom broju traženih svojstvenih vrednosti, u jednoj, dve i tri dimenzije, u kojima koordinate uzimaju diskretne vrednosti $\{-\frac{L}{2}\Delta, -(\frac{L}{2}-1)\Delta, ..., (\frac{L}{2}-2)\Delta, (\frac{L}{2}-1)\Delta\}$, gde Δ predstavlja stepen diskretizacije.

Proučavanje vremenske i memorijske složenosti algoritama i njihovih praktičnih implementacija je ključan element za primenu u numeričkim simulacijama. Na osnovu rezultata prikazanih u ovom poglavlju moguće je odabrati optimalan algoritam za proučavanje BAK (ali i drugih fenomena koji zahtevaju egzaktnu dijagonalizaciju) u zavisnosti od parametara sistema i dostupnih računarskih resursa. Pored principijenih stepenih zakona koje smo pokazali, jedan od ključnih elemenata u optimizaciji su i konkretne vrednosti konstantnih prefaktora, koji mogu u nekom opsegu parametara da favorizuju određeni algoritam, iako bi se na osnovu opšte analize (npr. $O(L^2)$ nasuprot $O(L^3)$) moglo očekivati drugačije.

Uspešnost primene razvijenog algoritma Lancoš dijagonalizacije na proučavani sistem gasa idealnih bozona ^{87}Rb prikazana je na slikama 3.14–3.16, gde vidimo zavisnosti koje opisuju ponašanje sistema poznato u literaturi: naseljenost osnovnog stanja pri hlađenju sistema ispod kondenzacione temperature, porast kondenzacione temperature pri povećanju broja bozona u sistemu, oštar skok gustine čestica u centru sistema za temperature ispod kondenzacione.

5

Zaključak

U ovom radu proučavane su numeričke simulacije Boze-Ajnštajn kondenzata u rotirajućim magnetno-optičkim zamkama, kao i njihov ključni numerički korak pri određivanju kondenzacione temperature, u dijagonalizaciji prostorno diskretizovanog evolucionog operatora. Uporedili smo preciznost, vremensku i memorijsku kompleksnost algoritma potpune dijagonalizacije i Lancoš algoritma u dve varijante, sa (LDm) i bez ($LDnm$) čuvanja matrice koja se dijagonalizuje. Za potpunu dijagonalizaciju koristili smo funkcije LAPACK biblioteke (kod je dat u okviru Dodatka A.1), dok je za Lancoš dijagonalizaciju razvijen program u programskom jeziku C/C++, sa dve opisane opcije (odgovarajući program je dat u okviru Dodatka A.2). U Dodatku A.3 prikazan je kod funcije za računanje efektivnog dejstva reda $p = 6$. Razvijeni Lancoš algoritam smo primenili na primeru gasa idealnih bozona ^{87}Rb i dobili zavisnosti u skladu sa teorijskim predviđanjima i dosadašnjim eksperimentalnim rezultatima.

Za posmatrani problem (dijagonalizacija harmonijskog potencijala sa kvartičnim anharmonicitetom) videli smo da je tačnost Lancoš dijagonalizacije slična tačnosti potpunog algoritma, kao i da u oba slučaja tačnost zavisi od finoće diskretizacije (odnosno od vrednosti diskretizacionih parametara) i numeričke preciznosti računara. Lancoš algoritam je najpovoljniji u primenama u kojima je potrebno naći mali broj svojstvenih vrednosti. Tada ovaj algoritam ima nižu klasu vremenske kompleksnosti u odnosu na potpuni algoritam, a veza je data relacijom

$$\tau^{LD} \sim (\tau^{FD})^{2/3}.$$

U slučaju kada je memorija ograničavajući resurs, Lancoš algoritam bez čuvanja matrice koja se dijagonalizuje ima veliku prednost, jer je njegova memorijska složenost značajno manja od drugih algoritama,

$$M^{LDnm} \sim (M^{FD})^{1/2}.$$

Analitički i numerički rezultati koji su dobijeni u ovom radu se mogu direkno koristiti za optimizaciju izbora algoritma za rešavanje različitih problema koji zahtevaju egzaktnu dijagonalizaciju velikih matrica. Pored toga veliki broj interesantnih pitanja se nameće za dalja istraživanja, kao što su analiza greške računanja energetskih nivoa u zavisnosti od vrednosti diskretizacionih parametara i konvergencija problema, posebno kod Lancoš algoritma dijagonalizacije. Jednostavnim izmenama u kodu simulacije, Lancoš algoritam se lako može primeniti i na druge probleme u kojima je potrebna efikasna dijagonalizacija.

Dodatak A

Programski kod

A.1 C/C++ implementacija algoritma potpune dijagonalizacije

Implementacija algoritma potpune dijagonalizacije za rešavanje svojstvenog problema matrice evolucionog operatora A u $d = 1$ koristi funkcije paketa LAPACK [17]. Za pokretanje programa potrebno je zadati parametre diskretizacije L i $L\Delta$, vreme evolucije ϵ, kao i parametre BAK potencijala $A = 1 - r^2$ i anharmonicitet k_n. Funkcija za računanje efektivnog dejstva `S_eff()` je izostavljena i treba je definisati u okviru programa. Primer ovakve funkcije je dat u Dodatku A.3.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <time.h>

int main(int argc, char **argv)
{

   double S_eff(double, double, double, double, double);
   int *ivector(long, long);
   double *dvector(long, long);
   void free_ivector(int *, long, long);
   void free_dvector(double *, long, long);

   int *IWORK, INFO;
   long NF, LWORK, LIWORK, L, i, k, maxenlev;
   double delta, xi, xk, *M, *d, *v, A, kn, eps, Pi, temp, r, g;
   FILE *psi;
   char JOBZ, UPLO;
   time_t t_exec;

   if(argc != 7)
   {
```

```
   fprintf(stderr, "Usage: %s psi L L*delta eps A kn\n\n\n", argv[0]);
   exit(EXIT_FAILURE);
}

t_exec = time(NULL);
psi = fopen(argv[1], "w+");
Pi = 4 * atan(1);
L = atol(argv[2]);
delta = atof(argv[3]) / L;
eps = atof(argv[4]);
A = atof(argv[5]);
kn = atof(argv[6]);
M = dvector(0, 2 * L * 2 * L - 1);
JOBZ = 'V';
UPLO = 'U';
NF = 2 * L;
LIWORK = 3 + 5 * NF;
LWORK = 1 + 6 * NF + 2 * NF * NF;
d = dvector(0, NF - 1);
v = dvector(0, LWORK - 1);
IWORK = ivector(0, LIWORK - 1);
INFO = 0;

for(i = 0; i < 2 * L; i++)
{
   xi = (i - L) * delta;

   for(k = i; k < 2 * L; k++)
   {
      xk = (k - L) * delta;
      M[i + 2 * L * k] = exp(-S_eff(0.5 * (xi + xk), xk - xi, A, kn, eps));
   }
}

t_exec -= time(NULL);
t_exec = time(NULL);
dsyevd_(&JOBZ, &UPLO, &NF, M, &NF, d, v, &LWORK, IWORK, &LIWORK, &INFO);
t_exec -= time(NULL);
free_ivector(IWORK, 0, LIWORK - 1);
free_dvector(v, 0, LWORK - 1);

maxenlev = 0;

for(i = 0; i < NF / 2; i++)
{
   temp = d[i];
   temp = temp < 0 ? DBL_MAX : (maxenlev++, - log(temp * delta
            / sqrt(2 * Pi * eps)) / eps);
   d[i] = d[NF - 1 - i];
   d[i] = d[i] < 0 ? DBL_MAX : (maxenlev++, - log(d[i] * delta
            / sqrt(2 * Pi * eps)) / eps);
```

```
      d[NF - 1 - i] = temp;
   }

   for(i = 0; i < maxenlev; i++)
   {
      fprintf(psi, "%d\n1.16le\n", i, d[i]);
   }


   fprintf(psi, "\n");
   fprintf(psi, "Eigenvectors:\n");
   for(k = 0; k < 2 * L; k++)
   {
        fprintf(psi, "%d\t", k);
   for(i = 0; i < 10/*maxenlev; i++)
   {
        fprintf(psi, "%1.16le\t", M[k + 2 * L * (2 * L - 1 - i)]);
   }

      fprintf(psi, "\n");
   }

   fclose(psi);
   free_dvector(M, 0, 2 * L * 2 * L - 1);

   exit(EXIT_SUCCESS);
}
```

A.2 C/C++ implementacija Lancoš algoritma dijagonalizacije

Razvijeni program koristi Lancoš dijagonalizaciju za rešavanje svojstvenog problema matrice evolucionog operatora A i nudi opciju čuvanja matrice u memoriji, odnosno računanja elemenata matrice po potrebi [18]. Ulazni parametri su maksimalan broj iteracija (tipično dvostruka vrednost traženog broja svojstvenih vrednosti), broj diskretizacionih tačaka, broj traženih svojstvenih vrednosti i vektora, diskretizacioni korak Δ, vreme evolucije ϵ, parametar $A = 1 - r^2$, anharmonicitet k_n i opcija sa čuvanjem/bez čuvanja matrice. Funkcija za računanje efektivnog dejstva `S_eff()` je izostavljena i treba je definisati u okviru programa. Primer ovakve funkcije je dat u Dodatku A.3.

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_heapsort.h>
```

```
int Dimension;
//size of Hamiltonian matrix
double** QVector;
//Lanczos vectors, size = maxNbIter * Dimension
int QVsize;
//Lanczos vector size
int Index;
//iteration index
int nbIter, maxNbIter;
//number of iterations and maximal number of Lanczos iterations
double *TDdiagonal, *TDupperdiag;
//Tridiagonal matrix, size = maxNbIteration
double *tempTDdiagonal, *tempTDupperdiag;
//Temporary tridiagonal matrix, size = maxNbIteration
int TDsize;
//Tridiagonal matrix size
int nbEigenvalues;
//number of wanted egenvalues
double prevLowest;
//previous lowest eigenvalue
double dx, A, kn, eps;
//phi4 parameters
double **tempEigenvector;
//temporary matrix for Eigenvector evaluation, size = <= maxNbIter * maxNbIter
double **Eigenvectors;
//Eigenvectors, size = Dimension * nbEigenvalues
bool useMatrix;
//true = store Hamiltonian values, false = calculate on the fly Hamiltonian values
double **HamiltonianMatrix;
//Hamiltonian matrix, size = Dimension * Dimension

//function evaluating Hamiltonian value
double HamiltonianValue(int i, int j)
{
        if (useMatrix == true) return HamiltonianMatrix[i][j];
        double konst = 1./sqrt(2*eps*4*atan(1.));
        double S = S_eff((i+j-Dimension)*dx/2,(j-i)*dx,A,kn,eps);
        return -konst*exp(-S);
}


//Vector norm
double VectorNorm(double* Vector, int size)
{
        double sum = 0;
        int i;
        for (i = 0; i < size; i++)
        {
                sum += Vector[i] * Vector[i];
        }
        return sqrt(sum);
```

```
}


//normalizes vector
void NormalizeVector(double* Vector, int size)
{
        int i;
        double norm = VectorNorm(Vector, size);
        if (norm == 0) return;
        for (i=0; i<size; i++)
        {
                Vector[i] /= norm;
        }
        return;
}


//realocates for another Lanczos vector
void AddQVector(int nbAdditional)
{
        QVsize += nbAdditional;
        int i;
        for(i = nbAdditional; i > 0; i--)
                QVector[QVsize - i] = (double*) calloc(Dimension, sizeof(double));*/
        return;
}


//Lanczos initialization
void InitializeLanczos()
{
        if (useMatrix)
        {
                double konst = 1./sqrt(2*eps*4*atan(1.));
                double S;
                HamiltonianMatrix = new double* [Dimension];
                for (int i = 0; i < Dimension; i++)
                {
                        HamiltonianMatrix[i] = new double [Dimension];
                        for (int j = 0; j < Dimension; j++)
                        {
                                S = phi4d1p21((i+j-Dimension)*dx/2,(j-i)*dx,A,kn,eps);
                                HamiltonianMatrix[i][j] = -konst*exp(-S);
                        }
                }
        }

        QVector = (double**) malloc((maxNbIter + 2) * sizeof(double *));
        int i;
        QVsize = 3;
        for(i = 0; i < maxNbIter + 2; i++)
```

```
        QVector[i] = (double*) malloc(Dimension * sizeof(double));

        for (i = 0; i < Dimension; i++)
            {
                      QVector[0][i] = rand()-0.5;
        }
        NormalizeVector(QVector[0], Dimension);

        TDdiagonal = (double*) calloc(maxNbIter, sizeof(double));
        TDupperdiag = (double*) calloc(maxNbIter - 1, sizeof(double));
        tempTDdiagonal = (double*) calloc(maxNbIter, sizeof(double));
        tempTDupperdiag = (double*) calloc(maxNbIter - 1, sizeof(double));

        Index = 0;
        TDsize = 0;
        return;
}


//Printing vector
void printVector(double* Vector, int size)
{
        int i;
        for (i = 0; i < size; i++)
        {
                printf("%1.14le\n", Vector[i]);
        }
        printf("\n");

        return;
}


//retun maximal integer out of two given ones
int max(int a, int b)
{
        if (a > b) return a;
        return b;
}


//resizes Tridiagonal matrix;
void TDresize(int size)
{
        TDsize = size;

        return;
}


//p = A*q                *time consuming!
```

```
void HamiltonianTimesVector(double* q, double* p, int size)
{
        int i,j;
        for (i=0; i<size; i++)
        {
                p[i] = 0;
                for (j=0; j<size; j++)
                {
                        p[i] += HamiltonianValue(i,j) * q[j];
                }
        }
        return;
}


//Scalar product (p, q)
double ScalarProduct(double* p, double*q, int size)
{
        int i;
        double sum = 0;
        for (i=0; i<size; i++)
        {
                sum += p[i] * q[i];
        }
        return sum;
}


//q += a*p
void AddLinearCombination(double* q, double a, double* p, int size)
{
        int i;
        for (i=0; i<size; i++)
        {
                q[i] += a * p[i];
        }
}


//Lanczos iteration
void Lanczos(int nbIter)
{
        int size;

        if (Index == 0)
        {
                size = TDsize + max(nbIter, 2);
                TDresize(size);

                HamiltonianTimesVector(QVector[0], QVector[1], Dimension);
                TDdiagonal[Index] = ScalarProduct(QVector[0], QVector[1], Dimension);
```

```
        AddLinearCombination(QVector[1], -TDdiagonal[Index], QVector[0], Dimension);
        NormalizeVector(QVector[1], Dimension);
        HamiltonianTimesVector(QVector[1], QVector[2], Dimension);
        TDupperdiag[Index] = ScalarProduct(QVector[0], QVector[2], Dimension);
        TDdiagonal[Index + 1] = ScalarProduct(QVector[1], QVector[2], Dimension);
}
  else
{
        size = TDsize + nbIter;
        TDresize(size);
}

int i,j;

for (i = Index + 2; i < size; i++)
    {
AddLinearCombination(QVector[i], -TDdiagonal[Index + 1], QVector[i - 1], Dimension);
AddLinearCombination(QVector[i], -TDupperdiag[Index], QVector[i - 2], Dimension);

if (i > 2)
{
        double pom;
        for(j = 0; j < i - 2; j++)
        {
        pom = -ScalarProduct(QVector[i], QVector[j], Dimension);
        AddLinearCombination(QVector[i], pom, QVector[j], Dimension);
        }
}

double norm = VectorNorm(QVector[i], Dimension);
int errNb = 0;
     while (norm < 1e-10)
{
        printf("i  %d  norm %1.18le \n", i, norm);
        errNb++;
        if (errNb > 20)
        {
                printf("Lanczos algorithm cannot converge\n");
                exit(0);
        }

        double tmp = 0;

        for (j = 0; j < Dimension; j++)
        {
                QVector[i][j] = rand()-0.5;//gsl_rng_uniform (random) - 0.5;
        }

        NormalizeVector(QVector[i], Dimension);

        double* tmpVector;
```

```
                tmpVector = (double*) malloc(Dimension * sizeof(double));
                HamiltonianTimesVector(tmpVector, QVector[i], Dimension);
                QVector[i] = tmpVector;

                double pom;
                for(j = 0; j < i; j++)
                {
                        pom = -ScalarProduct(QVector[i], QVector[j], Dimension);
                        AddLinearCombination(QVector[i], pom, QVector[j], Dimension);
                }
                norm = VectorNorm(QVector[i], Dimension);
                }

                NormalizeVector(QVector[i], Dimension);
                Index++;
                AddQVector(1);

                HamiltonianTimesVector(QVector[i], QVector[i + 1], Dimension);

        TDupperdiag[Index] = ScalarProduct(QVector[i - 1], QVector[i + 1], Dimension);
        TDdiagonal[Index + 1] = ScalarProduct(QVector[i], QVector[i + 1], Dimension);
        }

}


//compare function for heap sort
int compare_doubles (const void * a, const void * b)
{
        double aa = *((double*)a);
        double bb = *((double*)b);
        if (aa > bb)
                return 1;
        else if (aa < bb)
                return -1;
        else
                return 0;
}


//sorts eigenvalues in diagonal matrix from the lowest to the highest
void sortDiagonal()
{
        gsl_heapsort (tempTDdiagonal, TDsize, sizeof(double), compare_doubles);

        return;
}


//Diagonalizes Tridiagonal matrix
void Diagonalize(bool vectors)
```

```
{
        int counter;

        for (counter = 0; counter < TDsize; counter++)
        {
                tempTDdiagonal[counter] = TDdiagonal[counter];
                if (counter != Dimension - 1)
                tempTDupperdiag[counter] = TDupperdiag[counter];
        }


        int ReducedDimension = TDsize - 1;
        double Cos;
        double Sin;
        double Theta;
        double T, R, P, F, B;
        int maxIter = 1000;

          for (int i = 0; i < ReducedDimension; i++)
            {
                int iter = 0;
                while (iter < maxIter)
                {
                  // find block matrices so that QL algorithm will be applied
                // on submatrix from i to j
                  int j = i;
                  bool Flag = false;
                  while ((j < ReducedDimension) && (Flag == false))
                    {
                      double d2 = fabs(tempTDdiagonal[j]) + fabs(tempTDdiagonal[j + 1]);
                      if ((d2 + fabs(tempTDupperdiag[j])) == d2)
                Flag = true;
                      else
                j++;
                    }
                  // if i != j, i-th eigenvalue has not been obtained yet,
                // apply diagonalization on submatrix
                  if (j != i)
                    {
                      iter++;
                      // evaluate shift
                      Theta = (tempTDdiagonal[i + 1] - tempTDdiagonal[i])
                        / (2.0 * tempTDupperdiag[i]);
                      R = sqrt (1.0 + Theta * Theta);
                      T = tempTDdiagonal[j] - tempTDdiagonal[i];
                      if (Theta >= 0)
                T += tempTDupperdiag[i] / (Theta + R);
                      else
                T += tempTDupperdiag[i] / (Theta - R);
                      Cos = 1.0;
                      Sin = 1.0;
```

```
                P = 0.0;
                // apply shift and conjugation with Jacobi and Givens rotations
                for (int k = j - 1; k >= i; k--)
            {
            F = Sin * tempTDupperdiag[k];
            B = Cos * tempTDupperdiag[k];
            R = sqrt (F * F + T * T);
            tempTDupperdiag[k + 1] = R;
            if (R == 0.0)
            {
            tempTDdiagonal[k + 1] -= P;
            tempTDupperdiag[j] = 0.0;
            k = i - 1;
            }
            else
            {
            Sin = 1.0 / R;
            Cos = Sin * T;
            Sin *= F;
            T = tempTDdiagonal[k + 1] - P;
            R = (tempTDdiagonal[k] - T) * Sin + 2.0 * Cos * B;
            P = Sin * R;
            tempTDdiagonal[k + 1] = T + P;
            T = Cos * R - B;
            // apply transformation to vectors
            if (vectors)
            {
            double tmp;
            for (int n = 0; n < TDsize; n++)
            {
            tmp = tempEigenvector[n][k + 1];
            tempEigenvector[n][k + 1] *= Cos;
            tempEigenvector[n][k + 1] += Sin * tempEigenvector[n][k];
            tempEigenvector[n][k] *= Cos;
            tempEigenvector[n][k] -= Sin * tmp;
            }
            }
            }
            }
                  tempTDdiagonal[i] -= P;
                  tempTDupperdiag[i] = T;
                  tempTDupperdiag[j] = 0.0;
                }
              else
                iter = maxIter;
            }
        }

    if (!vectors)
            sortDiagonal();
```

```
        return;
}


//evaluates and prints eigenvecors
void printEigenstates()
{
        Eigenvectors = new double* [nbEigenvalues];

        tempEigenvector = new double* [TDsize];

        for (int i = 0; i < TDsize; i++)
        {
                tempEigenvector[i] = new double[TDsize];
                for (int j = 0; j < TDsize; j++)
                        tempEigenvector[i][j] = 0;
                tempEigenvector[i][i] = 1.0;
        }

        Diagonalize(true);

        int ReducedDim = TDsize - 2;
          double tmp;
          int MinPos;
          double MinValue;
          for (int i = 0; i <= ReducedDim; i++)
            {
                      MinPos = TDsize - 1;
                      MinValue = tempTDdiagonal[MinPos];
                      for (int j = ReducedDim; j >= i; j--)
                if (tempTDdiagonal[j] < MinValue)
                {
                          MinValue = tempTDdiagonal[j];
                          MinPos = j;
                }
                      tmp = tempTDdiagonal[i];
                      tempTDdiagonal[i] = MinValue;
                      tempTDdiagonal[MinPos] = tmp;
                for (int ii = 0; ii < TDsize; ii++)
                {
                              tmp = tempEigenvector[ii][i];
                              tempEigenvector[ii][i] = tempEigenvector[ii][MinPos];
                              tempEigenvector[ii][MinPos] = tmp;
                }
            }

          double* TmpCoefficents;
        TmpCoefficents = new double [TDsize];
          for (int i = 0; i < nbEigenvalues; ++i)
            {
                      for (int j = 0; j < TDsize; ++j)
```

```
                        TmpCoefficents[j] = tempEigenvector[j][i];
                    Eigenvectors[i] = new double [Dimension];
                int ii;
                      for (ii = 0; ii < Dimension; ii++)
                Eigenvectors[i][ii] = QVector[0][ii] * tempEigenvector[0][i];
                for (ii = 0; ii < TDsize - 1; ii++)
                AddLinearCombination(Eigenvectors[i], TmpCoefficents[ii+1],
                                        QVector[ii+1], Dimension);
            }


        for (int ii = 0; ii < nbEigenvalues; ii++)
        {
                NormalizeVector(Eigenvectors[ii], Dimension);
        }


        for (int j = 0; j < Dimension; j++)
        {
        printf("%d\t", j);
         for (int i = 0; i < nbEigenvalues; i++)
        {
                printf("%1.18le\t", Eigenvectors[i][j]);
        }
        printf("\n");
        }

}



int main(int argc, char** argv)
{
        int i,j;

        if (argc!=9)
        {
                printf("usage: max number of iterations, matrix dimension,
                        number of eigenvalues, dx, eps, A, kn, useMatrix 1-yes 0-no\n");
                return 0;
        }
        else
        {
                maxNbIter = (int) atol(argv[1]);
                Dimension = (int) atol(argv[2]);
                nbEigenvalues = (int) atol(argv[3]);
                dx = atof(argv[4]);
                eps = atof(argv[5]);
                A = atof(argv[6]);
                kn = atof(argv[7]);
                if (argv[8][0] == '0') useMatrix = false; else useMatrix = true;
```

```
        }

        InitializeLanczos();
        Lanczos(nbEigenvalues + 2);
        double Precision = 1.0;
        int currNbIter = nbEigenvalues + 2;
        prevLowest = 1e30;
             while ((Precision > 1e-14) && (currNbIter++ < maxNbIter))
        {
                Lanczos(1);
                Diagonalize(false);
                  Precision = fabs((prevLowest - tempTDdiagonal[nbEigenvalues-1])
                                / prevLowest);
                  prevLowest = tempTDdiagonal[nbEigenvalues-1];
        }
             if (currNbIter >= maxNbIter)
        {
                  printf("too many Lanczos iterations\n");
                  exit(0);
        }
        printf("Eigenvalues:\n");
        double eig;
          for (i = 0; i < nbEigenvalues; ++i)
            {
                eig = -log(-dx*(tempTDdiagonal[i]))/eps;
                printf("%d\t%1.14le\n", i, eig);
            }

        printf("\n");

        printf("Eigenvectors:\n");
        printEigenstates();

        for (i=QVsize-1; i>=0; i--)
                free(QVector[i]);
        free(QVector);
        free(TDdiagonal);
        free(TDupperdiag);
        free(tempTDdiagonal);
        free(tempTDupperdiag);

        return 0;

}
```

A.3 C/C++ implementacija efektivnog dejstva nivoa $p = 6$

U nastavku je dat primer funkcije efektivnog dejstva za jednodimenzioni anharmonijsi oscilator $V = \frac{1}{2}Ax^2 + \frac{k_n}{24}x^4$ i nivo $p = 6$. Ulazni parametri su vrednost koordinate, diskretizacioni korak Δ, parametar $A = 1 - r^2$, anharmonicitet k_n i vreme evolucije ϵ.

```
double S_eff_1d_p6(double x, double dx, double A, double kn, double eps){

dx2 = dx*dx;
dx4 = dx2*dx2;
dx6 = dx2*dx4;
dx8 = dx4*dx4;
x2 = x*x;
x4 = x2*x2;
x6 = x3*x3;
x7 = x4*x3;
x8 = x4*x4;
eps2 = eps*eps;
eps3 = eps*eps2;
eps4 = eps2*eps2;
eps5 = eps3*eps2;
kn2 = kn*kn;
kn3 = kn*kn2;
A2 = A*A;

return (0.5*dx2)/eps + eps*(0.00052083333333333333*dx4*kn +
        0.0020833333333333333*dx2*eps*kn - 2.4112654320987654 3e-7*dx6*eps2*kn2 -
        2.8935185185185185 2e-6*dx4*eps3*kn2 + 0.5*A*x2 + 0.041666666666666667*kn*x4 +
        eps*(0.083333333333333333*A + 0.041666666666666667*kn*x2) +
        0.041666666666666667*dx2*(A + 0.5*kn*x2) +
        0.25*dx2*eps3*(-0.00079365079365079365 1*A*kn - 0.0013888888888888889*kn2*x2) +
        0.0625*dx4*eps2*(-0.00039682539682539682 5*A*kn - 0.0006944444444444444444*kn2*x2)
        + 0.25*dx2*eps2*(-0.0027777777777777778*A2 - 0.011111111111111111*A*kn*x2 -
        0.0020833333333333333*kn2*x4) +  eps3*(-0.0027777777777777778*A2 -
        0.011111111111111111*A*kn*x2 - 0.0020833333333333333*kn2*x4) +
        eps2*(0.0041666666666666667*kn - 0.041666666666666667*A2*x2 -
        0.013888888888888889*A*kn*x4 - 0.0011574074074074074 1*kn2*x6) + 0.25*dx2*eps4*
        (0.00006613756613756613 76*A3 - 0.00006448412698412698 41*kn2 +
        0.0015873015873015873*A2*kn*x2 + 0.0009424603174603174 6*A*kn2*x4 +
        0.0001157407407407407 41*kn3*x6) + eps5*(0.0001763668430335097*A3 -
        0.0000388558201058201058*kn2 + 0.0021494708994708994 7*A2*kn*x2 +
        0.0012400793650793650 8*A*kn2*x4 + 0.0001543209876543209 88*kn3*x6) +
        eps4*(-0.0005952380952380952 38*A*kn + 0.0041666666666666667*A3*x2 -
        0.0008680555555555556*kn2*x2 + 0.0034722222222222222*A2*kn*x4 +
        0.0008101851851851851 85*A*kn2*x6 + 0.0000578703703703703704*kn3*x8));
}
```

Literatura

[1] L. Pitaevski, S. Stringari, *Bose–Einstein Condensation*, Oxford University Press (2003).

[2] C. J. Pethick, H. Smith, *Bose–Einstein Condensation in Dilute Gases*, Cambridge University Press (2002).

[3] I. Bloch, J. Dalibard, W. Zwerger, Rev. Mod. Phys. **80**, 885 (2008).

[4] K. B. Davis, M.-O. Mewes, M. R. Andrews, N. J. van Druten, D. S. Durfee, D. M. Kurn, W. Ketterle, Phys. Rev. Lett. **75**, 3969 (1995).

[5] M. H. Anderson, J. R. Ensher, M. H. Matthews, C. E. Wieman, and E. A. Cornell, Science **269**, 198 (1995).

[6] S. Kling, A. Pelster, Phys. Rev. A **76**, 023609 (2007).

[7] I. Vidanović, A. Bogojević A. Balaž, Phys. Rev. E **80**, 066705 (2009).

[8] I. Vidanović, A. Bogojević, A. Balaž, A. Belić, Phys. Rev. E **80**, 066706 (2009).

[9] A. Bogojević, I. Vidanović, A. Balaž, A. Belić, Phys. Lett. A **372**, 3341 (2008).

[10] A. Bogojević, A. Balaž, A. Belić, Phys. Rev. Lett. **94**, 180403 (2005).

[11] A. Bogojević, A. Balaž, A. Belić, Phys. Rev. B **72**, 036128 (2005).

[12] A. Aftalion, *Vortices in Bose–Einstein Condensates*, Birkhäuser, Boston (2006).

[13] A. L. Fetter, Rev. Mod. Phys. **81**, 647 (2009).

[14] V. Bretin, S. Stock, Y. Seurin, J Dalibard, Phys. Rev. Lett. **92**, 050403 (2004).

[15] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press (1992).

[16] I. Milošević, *Vektorski prostori i elementi vektorske analize*, Univerzitet u Beogradu (1997).

[17] LAPACK numerical library, `http://www.netlib.org/lapack/`

[18] M. Radulaški, C/C++ implementacija Lancoš algoritma za egzaktnu dijagonalizaciju, `https://svn.scl.rs/lanczos/`