Ludwig-Maximilians-Universität München

Bachelorarbeit

Vergleich von Krylov- und Chebyshevverfahren bei der Berechnung der Zeitentwicklung im Heisenberg-Modell

Autor: Wolfgang Bliemetsrieder

Betreuer: Prof. Dr. Ulrich Schollwöck

Vorgelegt am: 6. Juli 2017

Ludwig-Maximilians-Universität München

Bachelor thesis

Comparison of the Krylov and Chebyshev method for calculating the time evolution in the Heisenberg model

Author: Wolfgang Bliemetsrieder

Supervisor: Prof. Dr. Ulrich Schollwöck

Date: 2017-07-06

Inhaltsverzeichnis

1. Einleitung

Ziel dieser Arbeit ist es, die Zeitentwicklung, also eine Lösung der Schrödingergleichung

$$\hat{H}\left|\psi(t)\right\rangle = i\partial_t \left|\psi(t)\right\rangle, \tag{1}$$

für einen Ausgangszustand $|\psi(0)\rangle$ im isotropen Heisenberg-Modell auf einer eindimensionalen geschlossenen Spinkette der Länge L mithilfe zweier numerischer Verfahren zu berechnen und diese dann bezüglich ihrer Laufzeit zu vergleichen. In der Schrödingergleichung wurde $\hbar \equiv 1$ gesetzt. Das Heisenberg-Modell wird durch den Hamiltonoperator

$$\hat{H} = J\sum_{i=1}^{L-1} \hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_{i+1} + J\hat{\mathbf{S}}_L \cdot \hat{\mathbf{S}}_1. \tag{2}$$

beschrieben, wobei J die Stärke der Wechselwirkung der nächsten Nachbarn beschreibt. Falls $\hat{H}$ zeitunabhängig ist, ist die formale Lösung dieses Problems durch

$$\left|\psi(t)\right\rangle = \exp\left(-i\hat{H}t\right)\left|\psi(0)\right\rangle = U(t)\left|\psi(0)\right\rangle \tag{3}$$

gegeben. Hier bezeichnet

$$U(t) = \exp\left(-i\hat{H}t\right) \tag{4}$$

den Zeitentwicklungsoperator. Um mit dieser formalen Lösung rechnen zu können, wird für den Hilbertraum $\mathscr{H} = \{\mathbb{C}^2\}^{\otimes L}$ die Basis $\{\left|\uparrow\right\rangle, \left|\downarrow\right\rangle\}^{\otimes L}$ verwendet. So erhält man eine Matrixdarstellung H des Hamiltonoperators $\hat{H}$. Da der Hamiltonoperator nur Wechselwirkungen zwischen nächsten Nachbarn beinhaltet ist die Matrix in dieser Basis in Tridiagonalform, das heißt nur die Diagonale und die beiden Nebendiagonalen sind durch Werte ungleich von Null besetzt. Mithilfe dieser Darstellung ist es möglich Gleichung 3 mit dem Krylov- bzw. dem Chebyshevverfahrens zu lösen. Diese beiden Verfahren werden im nächsten Kapitel erläutert und im darauffolgenden Kapitel wird die explizite Umsetzung in C++ Code beschrieben. Im letzten Kapitel wird die Effizienz und die Genauigkeit der Verfahren analysiert und verglichen.

2. Verfahren

2.1. Krylov-Verfahren

Um dieses Verfahren, wie es in [Sch16] beschrieben wird, zu entwickeln startet man mit dem Ansatz den Zeitentwicklungsoperator mit der Reihendarstellung der Exponentialfunktion zu berechnen. Man schreibt also

$$\exp(-iHt) = \sum_{k=0}^{\infty} (-i)^k t^k \frac{H^k}{k!}. \tag{5}$$

Diese Form ist noch wenig nützlich, da sie zeitaufwendige Matrix-Matrix Multiplikationen erfordert, deshalb betrachtet man die Wirkung dieses Operators auf den Anfangszustand und erhält so

$$|\psi(t)\rangle = \exp\Big(-i\hat{H}t\Big)\,|\psi(0)\rangle = \sum_{k=0}^{\infty}(-i)^k t^k \frac{\hat{H}^k}{k!}\,|\psi(0)\rangle\,. \tag{6}$$

In dieser Form kann man nun Matrix-Matrix Multiplikationen vermeiden, indem man $H^k\,|\psi(0)\rangle$ durch rekursive Matrix-Vektor Multiplikationen bestimmt. Eine erste naive Näherung für $|\psi(t)\rangle$ erhält man durch die m-te Partialsumme dieser Reihe. Da diese in dem Unterraum des Hibertraums liegt, welcher durch die $\hat{H}^k\,|\psi(0)\rangle$ für $k = 0, \ldots, m-1$ aufgespannt wird, kann man die Berechnung auf diesen Unterraum beschränken. Dies hat den Vorteil, dass dieser Raum eine wesentlich kleinere Dimension hat und es möglich ist, die Matrixexponentialfunktion durch Diagonalisierung exakt zu bestimmen. Dies motiviert die folgende Definition.

Für eine natürliche Zahl m, einen Operator $\hat{H}$ und einen Ausgangszustand $|\psi(0)\rangle$ ist der Krylov-Raum ist definiert als der Span

$$\mathbb{K}_m(\hat{H}, |\psi(0)\rangle) := span(|\psi(0)\rangle\,, \hat{H}\,|\psi(0)\rangle\,, ..., \hat{H}^{m-1}\,|\psi(0)\rangle). \tag{7}$$

Im folgenden wird immer angenommen, dass die Dimension dieses Raumes gleich m ist, also $\dim \mathbb{K}_m(\hat{H}, |\psi(0)\rangle) = m$. Dies ist im Allgemeinen nicht korrekt, da aber m stets sehr viel kleiner als die Dimension des Hilbertraums gewählt wird, ist für einen zufällig gewählten Anfangszustand $|\psi(0)\rangle$ diese Annahme fast immer erfüllt. Für diesen Raum wird eine Orthonormalbasis benötigt, welche mit dem Gram-Schmidt-Verfahren bestimmt wird. Hierbei wird ausgenutzt, dass H eine tridiagonale Matrix ist, wodurch sich das Gram-Schmidt-Verfahren zur drei-Term-Rekursion

$$\beta_k\,|q_{k+1}\rangle = H\,|q_k\rangle - \langle q_k|H|q_k\rangle\,|q_k\rangle - \langle q_{k-1}|H|q_k\rangle\,|q_{k-1}\rangle \tag{8}$$

vereinfacht. So erhält man die Matrizen

$$Q_m := [|q_1\rangle\,|q_2\rangle\,...\,|q_m\rangle] \tag{9}$$

und T_m mit

$$(T_m)_{ij} := \langle q_i|H|q_j\rangle\,. \tag{10}$$

Dann ist $T_m = Q_m^\dagger H Q_m$ die Projektion von H auf den m-ten Krylov-Raum. Außerdem gilt für alle $|k\rangle \in \mathbb{K}_m(\hat{H}, |\psi(0)\rangle)$ dass $Q_m Q_m^\dagger\,|k\rangle = |k\rangle$ und $|\psi(0)\rangle = \|\,|\psi(0)\rangle\,\|_2\,|q_1\rangle = \|\,|\psi(0)\rangle\,\|_2 Q_m\,|e_1\rangle$, wobei $|e_1\rangle$ den ersten Einheitsvektor bezeichnet. So erhält man

$$|\psi(t)\rangle = \|\,|\psi(0)\rangle\,\|_2 Q_m (Q_m^\dagger \exp(-iHt) Q_m)\,|e_1\rangle\,. \tag{11}$$

Als letzte Näherung ersetzt man nun $Q_m^\dagger \exp(-iHt)Q_m$, die Projektion der Exponentialfunktion von H, durch die Exponentialfunktion der Projektion von H. Das zusammengefasste Ergebnis lautet nun

$$|\psi(t)\rangle = \|\,|\psi(0)\rangle\,\|_2 Q_m \exp(-iT_m t)\,|e_1\rangle\,. \tag{12}$$

Hier kann die Matrixexponentialfunktion durch diagonalisieren von T_m bestimmt werden.

2.2. Chebyshev-Verfahren

Dieses Verfahren, welches in [H T84] und [Jad15] beschrieben wird, benutzt spezielle Polynome zur Approximation glatter Funktionen.

Die Chebyshev Polynome erster Art sind durch

$$T_0(x) = 1, \tag{13}$$
$$T_1(x) = x, \tag{14}$$
$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \qquad \text{für } n > 1 \tag{15}$$

definiert. Aus der Numerik ist bekannt, dass diese im Raum der glatten Funktionen auf dem Intervall $[-1, 1]$ eine orthogonale Basis bezüglich des Skalarprodukts

$$\langle f, g\rangle = \int_{-1}^{1} \frac{f(x)g(x)}{\pi\sqrt{1-x^2}}\, dx \tag{16}$$

bilden. Genauer gesagt gilt

$$\langle T_n, T_m\rangle = \begin{cases} 0 & \text{für } n \neq m \\ 1 & \text{für } n = m = 0 \\ \frac{1}{2} & \text{für } n = m \neq 0. \end{cases} \tag{17}$$

Damit lässt sich eine beliebige glatte Funktion $f : [-1, 1] \to \mathbb{C}$ durch

$$f(x) = \langle T_0, f\rangle + \sum_{n=1}^{\infty} 2\langle T_n, f\rangle T_n(x) \tag{18}$$

in dieser Basis darstellen. Für eine natürliche Zahl N ist die N-te Näherung f_N einer solchen Funktion durch die N-te Partialsumme von Gleichung 18 definiert.

Um dieses Verfahren auf den Zeitentwicklungsoperator $U(t)$ anwenden zu können, sind noch ein paar Anpassungen nötig. Da die Chebyshev Polynome nur auf dem Intervall $[-1, 1]$ eine Approximation liefern, außerhalb aber divergieren, muss der Hamiltonoperator $\hat{H}$ zunächst reskaliert werden, damit das Spektrum des neuen Operators in diesem Intervall enthalten ist. Die maximale Energie ist, als Eigenwert zum Eigenvektor $|\uparrow\rangle^{\otimes L}$, einfach zu bestimmen und somit durch $E_s = 0,25L$ gegeben. Die minimale Energie kann durch $E_g = -0,75L$ abgeschätzt werden. Dies definiert die Bandbreite $W = E_s - E_g$. Um sicher zu stellen, dass berechnete Werte auch mit numerischen Fehlern im Intervall $[-1, 1]$ liegen, wird die Bandbreite nicht nur auf dieses skaliert sondern auf das etwas kleinere Intervall $[-W', W']$, wobei $W' = 1 - \frac{\epsilon_t}{2}$, mit dem Sicherheitsfaktor $\epsilon_t = 0,025$. Der reskalierte Operator ist dann gegeben durch

$$\hat{H}' = \frac{\hat{H} - b}{a}, \tag{19}$$

mit $a = W/(2 - \epsilon_t)$ und $b = (E_g + E_s)/2$.

Durch Umformen des Zeitentwicklungsoperators erhält man

$$U(t) = \exp\big(-i(aH' + b)t\big) = \exp\big(-i(E_g + aW')t\big)\exp(-iaHt). \tag{20}$$

Hier kann die letzte Funktion $f(x) = \exp(-iatx)$ durch das Chebyshev-Verfahren angenähert werden und man erhält für den gesamten Ausdruck die Approximation

$$U_N(t) = \exp\big(-i(E_g + aW')t\big)\sum_{n=0}^{N-1}\phi_n(t)T_n(H'), \tag{21}$$

wobei $\phi_0(t) = c_0(t)$, $\phi_n(t) = 2c_n(t)$ für $n > 0$ und

$$c_n(t) = \int_{-1}^{1}\frac{\exp(-iat\omega)T_n(\omega)}{\pi\sqrt{1-\omega^2}}\,d\omega = (-i)^n J_n(at). \tag{22}$$

J_n ist die Besselfunktion der ersten Art von Ordnung n.

Wie zuvor schon beim Krylovverfahren ist die Form von Gleichung 21 zur numerischen Berechnung ungeeignet, da sie zeitintensive Matrix-Matrix Multiplikationen enthält. Deshalb wird auch hier die Wirkung auf den Anfangszustand $|\psi(0)\rangle$ betrachtet, woraus man

$$|\psi(t)\rangle = U_N(t)\,|\psi(0)\rangle \tag{23}$$

$$= \exp(-i(E_g + aW)t)\sum_{n=0}^{N-1}\phi_n(t)T_n(H')\,|\psi(0)\rangle \tag{24}$$

erhält. Die letzte Vereinfachung erhält man, indem man den n-ten Chebyshev Vektor als $|t_n\rangle := T_n(\hat{H}')\,|\psi(0)\rangle$ definiert. Die rekursive Definition der Chebyshev Polynome überträgt sich dann auf diese Vektoren und somit gilt

$$|t_0\rangle = |\psi(0)\rangle\,, \tag{25}$$

$$|t_1\rangle = H'\,|\psi(0)\rangle\,, \tag{26}$$

$$|t_n\rangle = 2H'\,|t_{n-1}\rangle - |t_{n-2}\rangle \qquad \text{für } n > 1. \tag{27}$$

Letztendlich ist damit die gesuchte Zeitentwicklung gegeben durch

$$|\psi_N(t)\rangle = \exp(-i(E_g + aW)t)\sum_{n=0}^{N-1}\phi_n(t)\,|t_n\rangle\,. \tag{28}$$

3. Umsetzung

In diesem Kapitel werden die beiden Algorithmen beschrieben, wie sie im Code umgesetzt wurden. Der vollständige Code findet sich in Anhang A.

Für das Krylov-Verfahren werden zunächst die Matrizen Q_m und T_m aus Gleichung 9 und Gleichung 10 wie im Algorithmus 1 berechnet. Die komplexe hermitesche Matrix T

wird dann mit der Funktion *zhetrd* in eine reelle symmetrische Matrix in Tridiagonalform umgewandelt. Aus dieser werden dann mit der Funktion *dstev* die Eigenwerte und Eigenvektoren berechnet. Dann wird die Exponentialfunktion der Diagonalform bestimmt und diese schließlich wieder zurücktransformiert in die ursprüngliche Basis. Die dabei auftretenden Matrixmultiplikationen werden mit der Funktion *zgemm* durchgeführt. Ausgegeben wird dann die erste Spalte der resultierenden Matrix. Die hier benutzten LAPACK Funktionen kann man in [And+99] nachschlagen.

Algorithmus 1 $|\psi(t)\rangle$ für einen Ausgagangszustand $|\psi(0)\rangle$ und Parametern t, L und N mit dem Krylov-Verfahren berechnen. Hier bezeichnet Q_{*n} die n-te Spalte der Matrix Q.

$|\psi(0)\rangle \leftarrow \frac{|\psi(0)\rangle}{\||\psi(0)\rangle\|_2}$
$Q_{*,0} = |\psi(0)\rangle$
$|\phi\rangle = H(|\psi(0)\rangle, L)$
$d = \langle\phi, \psi(0)\rangle$
$T_{0,0} = d$
$Q_{*,1} = \frac{|\phi\rangle - d|\psi(0)\rangle}{\||\phi\rangle - d|\psi(0)\rangle\|_2}$
for $k = 1$ to $m - 2$ **do**
  $|q_k\rangle = Q_{*,k}$
  $|q_{k-1}\rangle = Q_{*,k-1}$
  $|\phi\rangle = H(|q_k\rangle, L)$
  $d = \langle q_k, \phi\rangle$
  $T_{k,k} = d$
  $d = \langle q_{k-1}, \phi\rangle$
  $T_{k-1,k} = d$
  $T_{k,k-1} = d^*$
  $Q_{*,k+1} = \frac{|\phi\rangle - T_{k,k}|q_k\rangle - T_{k-1,k}|q_{k-1}\rangle}{\||\phi\rangle - T_{k,k}|q_k\rangle - T_{k-1,k}|q_{k-1}\rangle\|_2}$
end for
$|q_k\rangle = Q_{*,m-1}$
$|q_{k-1}\rangle = Q_{*,m-2}$
$|\phi\rangle = H(|q_k\rangle, L)$
$d = \langle q_k, \phi\rangle$
$T_{k,k} = d$
$d = \langle q_{k-1}, \phi\rangle$
$T_{k-1,k} = d$
$T_{k,k-1} = d^*$

Das Chebyshev-Verfahren wurde wie im Algorithmus 2 beschrieben umgesetzt. Die Besselfunktion wurde hier aus der Boost C++ Bibliothek, siehe [Joh], entnommen.

Algorithmus 2 $|\psi(t)\rangle$ für einen Ausgagangszustand $|\psi(0)\rangle$ und Parametern t, L und N mit dem Chebyshev-Verfahren berechnen

$|t_{n-2}\rangle = |\psi(0)\rangle$
$|\psi(t)\rangle = J_0(at)\,|t_{n-2}\rangle$
$|t_{n-1}\rangle = H'(|t_{n-2}\rangle, L, a, b)$
$|\psi(t)\rangle \mathrel{+}= -2iJ_1(at)\,|t_{n-1}\rangle$
for $n = 2$ to $N-1$ **do**
  $|t_n\rangle = 2H'\,|t_{n-1}\rangle - |t_{n-2}\rangle$
  $|\psi(t)\rangle \mathrel{+}= (-i)^n 2J_n(at)\,|t_n\rangle$
  $|t_{n-2}\rangle \leftarrow |t_{n-1}\rangle$
  $|t_{n-1}\rangle \leftarrow |t_n\rangle$
end for
$|\psi(t)\rangle \leftarrow \exp\big(-i(\lambda_{min} + a(1-\frac{\epsilon}{2}))t\big)\,|\psi(t)\rangle$
return $|\psi(t)\rangle$

4. Auswertung

Zunächst verwenden wir für die Auswertung stets $L = 8$ als Systemparameter, einen Néelzustand als Startwert und berechnen mit einem Zeitschritt von $\Delta t = 0.1\frac{1}{J}$ iterativ alle Werte bis zur Zeit $T = 20\frac{1}{J}$.

Um die Genauigkeit der Verfahren bewerten zu können, benötigen wir ein Maß, das es uns ermöglicht auf die Abweichung zum exakten Ergebnis zu schließen. Hierzu gibt es verschiedene Möglichkeiten, zum einen ist die Energie eine Erhaltungsgröße, da der Hamiltonoperator zeitunabhängig ist, also kann die Abweichung von der Anfangsenergie als Fehlermaß benutzt werden. Da die totale Magnetisierung $\hat{M} = \sum_{i=1}^{L} \hat{S}_z(i)$ mit dem Hamiltonoperator kommutiert $\left[\hat{H}, \hat{M}\right] = 0$ ist der zugehörige Erwartungswert eine weitere Erhaltungsgröße des Systems und ebenso als Fehlermaß nutzbar. Zudem gilt, dass sich die Lösungen für große Parameter der Verfahren an die exakte Lösung annähern. Daraus folgt, dass die Norm der Differenz zweier aufeinanderfolgender Vektoren $\||\psi_{m+1}(t)\rangle - |\psi_m(t)\rangle\|_2$ beziehungsweise $\||\psi_{N+1}(t)\rangle - |\psi_N(t)\rangle\|_2$, für konstantes t, klein werden muss. Also können wir diese Normdifferenz als Fehlermaß benutzen. Diese Idee können wir auch für beliebige Observablen benutzen, indem wir die Differenz zwischen zwei Erwartungswerten bezüglich dieser Observablen bilden. Für $\hat{S}_z(1)$ berechnen wir also

$$\langle\psi_{m+1}(t)|\hat{S}_z(1)|\psi_{m+1}(t)\rangle - \langle\psi_m(t)|\hat{S}_z(1)|\psi_m(t)\rangle\,. \tag{29}$$

Vergleichen wir zunächst die Normdifferenz mit der $\langle S_z(1)\rangle$-Differenz über die gesamte Zeit. Abbildung 1 zeigt, dass der Normfehler annähernd linear mit t wächst. Der Fehler im Erwartungswert der $S_z(1)$ hingegen fällt immer wieder zurück auf Null. Um diese beiden Größen über die gesamte Zeit zu betrachten wird jeweils der Maximalwert genommen. Abbildung 2 und Abbildung 3 zeigt diese maximalen Fehler, sowie die maximale Abweichung von der Energie bzw der totalen Magnetisierung zum Startzeitpunkt in Abhängigkeit der jeweiligen Parameter.

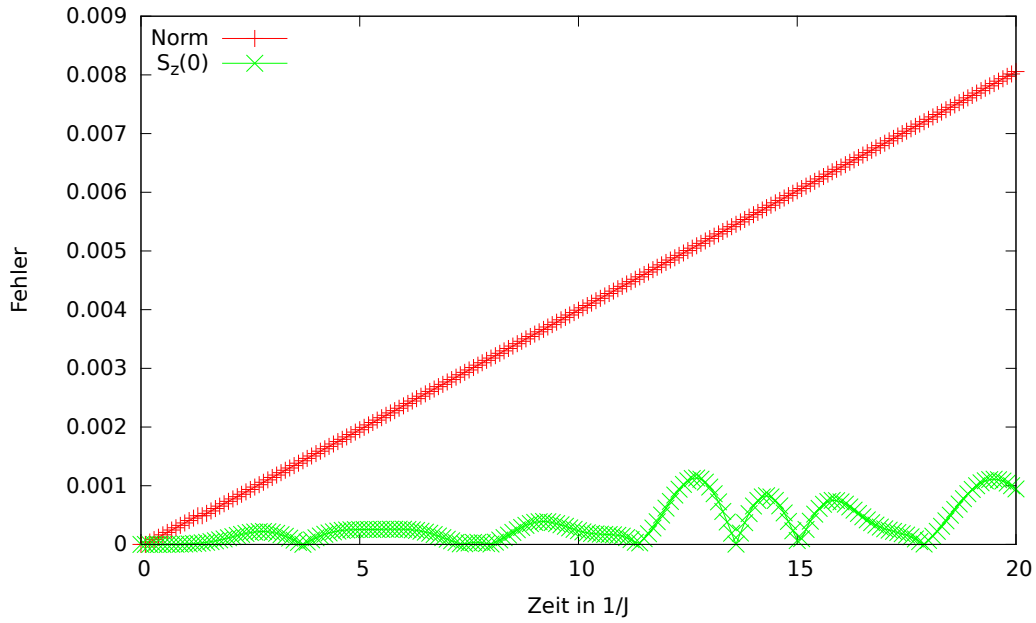

Abbildung 1: Fehler in Abhängigkeit der Zeit für das Krylov-Verfahren mit $m = 5$ und $L = 8$

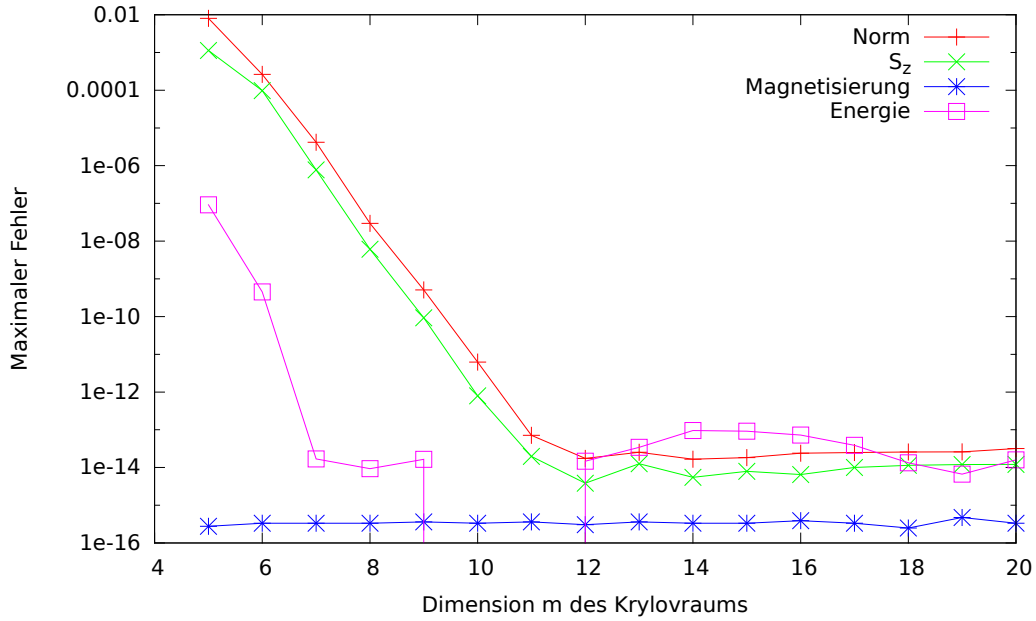

Abbildung 2: Fehler beim Krylov-Verfahren in Abhängigkeit von m für $L = 8$

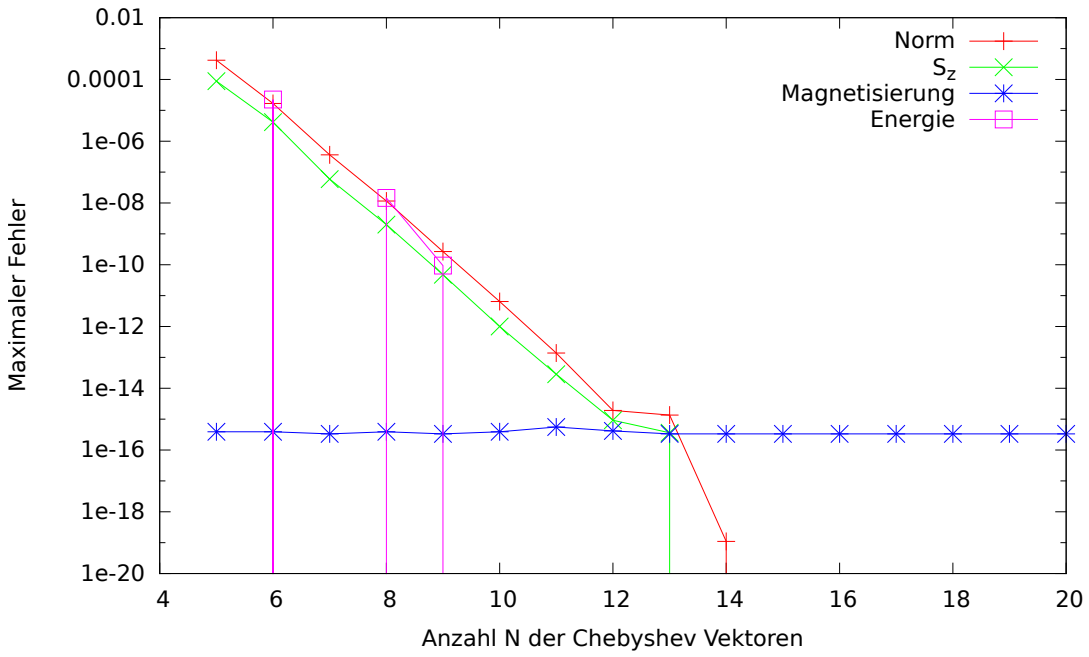

Abbildung 3: Fehler beim Chebyshev-Verfahren in Abhängigkeit von N für $L = 8$

Man erkennt, dass sowohl die Abweichung von der Anfangsenergie als auch die Abweichung von der Anfangsmagnetisierung für die Bewertung der Genauigkeit ungeeignet sind, da sie nicht mit wachsenden Parametern kleiner werden. Außerdem erkennt man, dass die anderen beiden Größen jeweils die selbe Größenordnung haben, also sind sie gleichermaßen für die Analyse geeignet. Deshalb werden wir im weiteren stets die Normdifferenz verwenden. Darüber hinaus kann man aus Abbildung 2 erkennen, dass der Fehler beim Krylov-Verfahren irgendwann nicht mehr kleiner wird, wir können also erwarten, dass ab diesem m keine Verbesserung im Ergebnis möglich ist und dass das wir das exakte Ergebnis nur bis auf diese Unsicherheit erreichen können.

Nun wollen wir die beiden Verfahren für $L = 8$ und $L = 10$ in ihrer Laufzeit vergleichen. Hierzu ermitteln wir zunächst die Eingabeparameter die mindestens benötigt werden, um eine bestimmte Fehlerschranke nicht zu überschreiten. Die erforderlichen Parameter sind in Tabelle 1 aufgelistet und die mit diesen Werten gemessenen Zeiten in Tabelle 2 und Tabelle 3.

Da die Programmlaufzeit natürlich auch von der Rechenleistung des Computers abhängt, auf dem das Programm ausgeführt wird, ist neben den Laufzeiten, die für die Berechnung der Daten bis zur Zeit $T = 20\frac{1}{J}$ benötigt wurden, auch das Verhältnis dieser beiden Zeiten angegeben. Dieses Ergebnis sollte möglichst universell die Leistungsfähigkeit der implementierten Methoden vergleichen und es zeigt, dass das Chebyshev-Verfahren gegenüber dem Krylov-Verfahren in diesem Test etwa 30 bis 40 Prozent effizienter ist.

Geforderte Genauigkeit	Erforderliches m bei Krylov		Erforderliches N bei Chebyshev	
	L=8	L=10	L=8	L=10
10^{-3}	6	6	5	6
10^{-4}	7	7	6	6
10^{-5}	7	8	7	7
10^{-6}	8	8	7	8
10^{-7}	8	9	8	8
10^{-8}	9	9	9	9
10^{-9}	9	10	9	10
10^{-10}	10	10	10	10
10^{-11}	10	11	10	11
10^{-12}	11	12	11	11
10^{-13}	11	14	12	11

Tabelle 1: Erforderliche Parameter für $L = 8$ und $L = 10$

Geforderte Genauigkeit	Zeit Krylov t_k in s	Zeit Chebyshev t_c in s	$\frac{t_c}{t_k}$
10^{-3}	0.710193	0.418659	0.5895
10^{-4}	0.78877	0.474533	0.601611
10^{-5}	0.790223	0.532894	0.674359
10^{-6}	0.888434	0.54237	0.610479
10^{-7}	0.889301	0.59435	0.668334
10^{-8}	0.999296	0.655833	0.656295
10^{-9}	0.991252	0.655713	0.6615
10^{-10}	1.09365	0.715365	0.65411
10^{-11}	1.09312	0.715803	0.654826
10^{-12}	1.19977	0.77664	0.647322
10^{-13}	1.19596	0.837368	0.700162

Tabelle 2: Gemessene Zeiten für $L = 8$

Geforderte Genauigkeit	Zeit Krylov t_k in s	Zeit Chebyshev t_c in s	$\frac{t_c}{t_k}$
10^{-3}	3.04383	2.17665	0.715102
10^{-4}	3.4261	2.16735	0.632601
10^{-5}	3.83946	2.44022	0.635564
10^{-6}	3.90814	2.72489	0.697235
10^{-7}	4.25593	2.71618	0.638211
10^{-8}	4.2536	2.99176	0.703348
10^{-9}	4.66857	3.26742	0.699875
10^{-10}	4.67081	3.2719	0.7005
10^{-11}	5.09079	3.54323	0.696008
10^{-12}	5.51294	3.54203	0.642493
10^{-13}	6.35694	3.81067	0.599451

Tabelle 3: Gemessene Zeiten für $L = 10$

Geforderte Genauigkeit	Krylov		Chebyshev	
	m	Δt in $\frac{1}{\mathrm{J}}$	N	Δt in $\frac{1}{\mathrm{J}}$
10^{-3}	6	0.140	5	0.124
10^{-4}	7	0.189	6	0.143
10^{-5}	7	0.119	7	0.174
10^{-6}	8	0.180	7	0.118
10^{-7}	8	0.119	8	0.136
10^{-8}	9	0.154	9	0.157
10^{-9}	9	0.103	9	0.117
10^{-10}	10	0.149	10	0.135
10^{-11}	10	0.100	10	0.104
10^{-12}	11	0.163	11	0.121
10^{-13}	11	0.107	12	0.138

Tabelle 4: Erforderliche Parameter für $L = 8$

Nun wollen wir für $L = 8$ ideale Parameter für eine bestimmte Fehlerschranke finden und dann die Laufzeit für diese Parameter vergleichen. Neben den Parametern m bzw N können wir auch die Schrittweite Δt verändern und so über die Anzahl der nötigen Iterationsschritte auch die Programmlaufzeit verkürzen. Wir benutzen die selben Parameter wie in Tabelle 1 und vergrößern die Zeitschritte um den vorgeschriebenen Fehler möglichst exakt zu erreichen. Diese Werte sind in Tabelle 4 aufgelistet und Tabelle 5 zeigt dann die gemessenen Zeiten. Auch hier gilt wie zuvor, dass das Chebyshev-Verfahren stets effizienter ist als das Krylov-Verfahren.

Geforderte Genauigkeit	Zeit Krylov t_k in s	Zeit Chebyshev t_c in s	$\frac{t_c}{t_k}$
10^{-3}	0.513026	0.346306	0.675026
10^{-4}	0.433129	0.344011	0.794246
10^{-5}	0.700999	0.318726	0.454674
10^{-6}	0.517702	0.470126	0.908102
10^{-7}	0.777491	0.456694	0.587395
10^{-8}	0.666707	0.435887	0.653791
10^{-9}	0.999093	0.58106	0.581587
10^{-10}	0.762898	0.553525	0.725556
10^{-11}	1.13636	0.717198	0.631136
10^{-12}	0.764666	0.670662	0.877065
10^{-13}	1.15916	0.632141	0.545345

Tabelle 5: Gemessene Zeiten für angepasste Schrittweiten und $L = 8$

A. Vollständiger Code

Dies ist der Code, wie er zur Zeitmessung in Abschnitt 4 benutzt wurde. Für zuvor durchgeführte Fehlermessung wurde ein abgeändertes Programm benutzt, mit einer angepassten *main* Funktion.

```
#include <boost/math/special_functions/bessel.hpp>
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
#include <complex>
#include <vector>
#include <lapacke.h>
#include <cblas.h>
#include <fstream>
#include <sstream>
#include <time.h>

using namespace std;

vector<complex<double> > H(vector<complex<double> > psi,int
   L){
  int j=0,l=pow(2,L);
  vector<complex<double> > phi;
  phi.resize(l);
  //Schleife fuer alle Summanden des Hamilton
  for(int k=0; k<L-1; ++k){
    //Schleife fuer die Vektorkomponenten
    for(int i=0; i<l; ++i){
      switch((i>>k)&3){
        case 0: phi[i]+=0.25*psi[i];
                break;
        case 1: j = i+(1<<k);
                phi[j]+=0.5*psi[i];
                phi[i]+=(-0.25)*psi[i];
                break;
        case 2: j = i-(1<<k);
                phi[j]+=0.5*psi[i];
                phi[i]+=(-0.25)*psi[i];
                break;
        case 3: phi[i]+=0.25*psi[i];
                break;
        default: cout<<"Fehler"<<endl; break;
```

```
      }
    }
  }
  //Periodische Randbedingungen fuer gerades L
  if((L&1)==0){
    //Schleife fuer die Vektorkomponenten
    for(int i=0; i<l; ++i){
      if(((i&1)==0)&&(((i>>(L-1))&1)==0)) phi[i]+=0.25*psi[i
         ];
      if(((i&1)==0)&&(((i>>(L-1))&1)==1)){
        j=i-(l>>1)+1;
        phi[j]+=0.5*psi[i];
        phi[i]+=(-0.25)*psi[i];
      }
      if(((i&1)==1)&&(((i>>(L-1))&1)==0)){
        j=i+(l>>1)-1;
        phi[j]+=0.5*psi[i];
        phi[i]+=(-0.25)*psi[i];
      }
      if(((i&1)==1)&&(((i>>(L-1))&1)==1)) phi[i]+=0.25*psi[i
         ];
    }
  }
  return phi;
}

vector<complex<double> > S_z(vector<complex<double> > psi,
   int L,int i){
  int l=pow(2,L);
  vector<complex<double> > phi;
  phi.resize(l);
  //Schleife fuer die Vektorkomponenten
  for(int k=0; k<l; ++k){
    switch((k>>i)&1){
      case 0: phi[k]+=(-0.5)*psi[k];
        break;
      case 1: phi[k]+=0.5*psi[k];
        break;
      default: cout<<"Fehler"<<endl; break;
    }
  }
  return phi;
}
```

```
vector<complex<double> > H_rescaled(vector<complex<double> >
    psi,int L,double a, double b){
  int l=pow(2,L);
  vector<complex<double> > phi;
  phi.resize(l);
  phi=H(psi,L);
  for(int i=0; i<l; ++i){
    phi[i]=(phi[i]-b*psi[i])/a;
  }
  return phi;
}

vector<complex<double> > krylov(vector<complex<double> > psi
   ,double t,int L, int m){
  int l=pow(2,L),i=0,j=0,k=0;
  vector<complex<double> > q_m,q_m1,phi;
  phi.resize(l);
  q_m.resize(l);
  q_m1.resize(l);
  complex<double> Q[l*m]={},d=0;
  double Norm=0,norm=0;
  //Variablen fuer zhetrd
  char uplo='U';
  complex<double> T[m*m]={};
  double D[m]={},E[m-1]={},TAU[m-1]={};
  complex<double> WORK_C[m*m]={};
  int info=0;
  //Variablen fuer dstev
  char v='V';
  double Z[m*m]={},WORK[2*m-2]={};

  //Normierung
  Norm=0;
  for(i=0; i<l; ++i) Norm+=real(psi[i]*conj(psi[i]));
  Norm=sqrt(Norm);
  for(i=0; i<l; ++i){
    Q[i*m]=psi[i]/Norm;
    psi[i]=psi[i]/Norm;
  }

  //Gram-Schmidt
  phi=H(psi,L);
```

```
d=0;
for(i=0; i<l; ++i) d+=conj(psi[i])*phi[i];
T[0]=d;
norm=0;
for(i=0; i<l; ++i) norm+=real((phi[i]-d*psi[i])*conj(phi[i
   ]-d*psi[i]));
norm=sqrt(norm);
for(i=0; i<l; ++i) Q[i*m+1]=(phi[i]-d*psi[i])/norm;

for(k=1; k<m-1; ++k){
  for(i=0; i<l; ++i){
    q_m[i]=Q[i*m+k];
    q_m1[i]=Q[i*m+k-1];
  }

  phi=H(q_m,L);

  d=0;
  for(i=0; i<l; ++i) d+=conj(q_m[i])*phi[i];
  T[k*m+k]=d;

  d=0;
  for(i=0; i<l; ++i) d+=conj(q_m1[i])*phi[i];
  T[(k-1)*m+k]=d;
  T[k*m+k-1]=conj(d);
  norm=0;
  complex<double> normC=0;
  for(i=0; i<l; ++i) normC+=(phi[i]-T[k*m+k]*q_m[i]-T[(k
     -1)*m+k]*q_m1[i])*
                        conj(phi[i]-T[k*m+k]*q_m[i]-T[(k
                           -1)*m+k]*q_m1[i]);
  norm=real(normC);
  norm=sqrt(norm);
  for(i=0; i<l; ++i) Q[i*m+k+1]=(phi[i]-T[k*m+k]*q_m[i]-T
     [(k-1)*m+k]*q_m1[i])/norm;
}
//T berechnen fuer k=m-1
for(i=0; i<l; ++i){
  q_m[i]=Q[i*m+k];
  q_m1[i]=Q[i*m+k-1];
}
phi=H(q_m,L);
d=0;
```

```
for(i=0; i<l; ++i) d+=conj(q_m[i])*phi[i];
T[k*m+k]=d;
d=0;
for(i=0; i<l; ++i) d+=conj(q_m1[i])*phi[i];
T[(k-1)*m+k]=d;
T[k*m+k-1]=conj(d);

//Komplexe Matrix in reelle Tridiagonalform umwandeln mit
   zhetrd
zhetrd_(&uplo,&m,reinterpret_cast<double __complex__*>(&T
   [0]),&m,
                reinterpret_cast<double*>(&D),
                reinterpret_cast<double*>(&E),
                reinterpret_cast<double __complex__*>(&
                   TAU),
                reinterpret_cast<double __complex__*>(&
                   WORK_C),&m,&info);

//Diese neue Matrix diagonalisieren mit dstev
dstev_(&v,&m,D,E,Z,&m,WORK,&info);

//Komplexe Matrizen
complex<double> D_complex[m*m]={};
complex<double> Z_complex[m*m]={};
complex<double> Q_complex[l*m]={};
complex<double> H[m*m]={};
complex<double> C1[m*m]={},C2[m*m]={},C3[m*m]={},C4[l*m
   ]={},alpha=1,beta=1;

//Werte von D auf D_complex uebertragen und Exp berechnen
for(i=0; i<m;++i){
  D_complex[i*(m+1)]=-t*D[i]*1.0i;
  D_complex[i*(m+1)]=exp(D_complex[i*(m+1)]);
}

//Werte von Z auf Z_complex uebertragen
for(i=0; i<m*m;++i) Z_complex[i]=Z[i];

//Werte von Q auf Q_complex uebertragen
for(i=0; i<l*m;++i) Q_complex[i]=Q[i];

//Matrixmultiplikationen
cblas_zgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,m,m,m,
```

```
                &alpha,&D_complex,m,&Z_complex,m,&beta,&C1,m);
    cblas_zgemm(CblasRowMajor,CblasConjTrans,CblasNoTrans,m,m,
       m,
                &alpha,&Z_complex,m,&C1,m,&beta,&C2,m);

    for(i=0; i<m*m; ++i) C1[i]=0;

    //Faktoren von TAU auf die Matrix anwenden
    for(i=0; i<m-1; ++i) T[i*(m+1)+1]=1;

    for(k=m-2; k>-1; --k){
      for(i=0; i<m; ++i) H[i*(m+1)]=1;
      for(i=0; i<k+1; ++i){
        for(j=0; j<k+1; ++j){
          H[i*m+j]-=TAU[k]*T[i*m+k+1]*conj(T[j*m+k+1]);
        }
      }
      cblas_zgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,m,m,
         m,
                  &alpha,&C2,m,&H,m,&beta,&C1,m);
      cblas_zgemm(CblasRowMajor,CblasConjTrans,CblasNoTrans,m,
         m,m,
                  &alpha,&H,m,&C1,m,&beta,&C3,m);
      for(i=0; i<m; ++i){
        H[i]=0;
        C2[i]=C3[i];
        C1[i]=0;
        C3[i]=0;
      }
    }

    //Multiplikation mit Q
    cblas_zgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,l,m,m,
                &alpha,&Q_complex,m,&C2,m,&beta,&C4,m);

    //Ausgabe
    vector<complex<double> > psi_aus;
    psi_aus.resize(l);
    for(i=0; i<l; ++i) psi_aus[i]=Norm*C4[i*m];

    return psi_aus;
}
```

```
vector<complex<double> > chebyshev(vector<complex<double> >
   psi,double t,int L, int N){
  int l=pow(2,L),i=0;
  double lambda_min=-0.75*(L-1),lambda_max=0.25*(L-1),
     epsilon=0.025;
  double a=(lambda_max-lambda_min)/(double(2.0)-epsilon),b=(
     lambda_max+lambda_min)/2.0;
  complex<double> bessel=0,faktor=0,i_n=-1.0i,i_minus=-1.0i;
  vector<complex<double> > t_n_1,t_n_2,t_n,psi_t,Ht_n_1;
  t_n_1.resize(l);
  t_n_2.resize(l);
  t_n.resize(l);
  psi_t.resize(l);
  Ht_n_1.resize(l);

  //Werte fuer den nullten und ersten Chebyshevvektor setzen
  t_n_2=psi;
  bessel=boost::math::cyl_bessel_j(0,a*t);
  for(i=0; i<l; ++i){
    psi_t[i]+=bessel*t_n_2[i];
  }
  t_n_1=H_rescaled(t_n_2,L,a,b);
  bessel=-2.0i*boost::math::cyl_bessel_j(1,a*t);
  for(i=0; i<l; ++i){
    psi_t[i]+=bessel*t_n_1[i];
  }
  //Rekursive Definition der restlichen Chebyshevvektoren
  for(int n=2; n<N; ++n){
    Ht_n_1=H_rescaled(t_n_1,L,a,b);
    for(i=0; i<l; ++i){
      t_n[i]=2.0*Ht_n_1[i]-t_n_2[i];
    }
    i_n*=i_minus;
    bessel=2.0*i_n*boost::math::cyl_bessel_j(n,a*t);
    for(i=0; i<l; ++i){
      psi_t[i]+=bessel*t_n[i];
    }
    t_n_2=t_n_1;
    t_n_1=t_n;
  }

  faktor=-1.0i*(lambda_min+a*(1.0-epsilon/2.0))*t;
  faktor=exp(faktor);
```

```
  for(i=0; i<l; ++i){
    psi_t[i]*=faktor;
  }

  return psi_t;
}


int main(int argn, char* argv[]){
  int L,l;
  int m,N,number;
  double t_k,t_c,T;
  char meth;
  int Zeitschritte_k,Zeitschritte_c;
  //Parameter einlesen
  if (argn!=9){
    cout<<"Eingabe:␣L␣m␣N␣t_k␣t_c␣T␣Methode␣#"<<endl;
    return 0;
  }
  istringstream ss1(argv[1]);
  if (!(ss1>>L)) cerr << "Invalid␣number␣" << argv[1] << '\n
     ';
  istringstream ss2(argv[2]);
  if (!(ss2>>m)) cerr << "Invalid␣number␣" << argv[1] << '\n
     ';
  istringstream ss3(argv[3]);
  if (!(ss3>>N)) cerr << "Invalid␣number␣" << argv[1] << '\n
     ';
  istringstream ss4(argv[4]);
  if (!(ss4>>t_k)) cerr << "Invalid␣number␣" << argv[1] << '
     \n';
  istringstream ss5(argv[5]);
  if (!(ss5>>t_c)) cerr << "Invalid␣number␣" << argv[1] << '
     \n';
  istringstream ss6(argv[6]);
  if (!(ss6>>T)) cerr << "Invalid␣number␣" << argv[1] << '\n
     ';
  istringstream ss7(argv[7]);
  if (!(ss7>>meth)) cerr << "Invalid␣char␣" << argv[1] << '\
     n';
  istringstream ss8(argv[8]);
  if (!(ss8>>number)) cerr << "Invalid␣number␣" << argv[1]
     << '\n';
```

```
Zeitschritte_k=T/t_k+1;
Zeitschritte_c=T/t_c+1;

complex<double> betrag=0;
l=pow(2,L);
vector<complex<double> > psi,phi;
vector<complex<double> > psi_1,psi_2;
psi.resize(l);
phi.resize(l);
psi_1.resize(l);
psi_2.resize(l);
//Neel-Zustand setzen
int index=0;
for(int k=1; k<L; k+=2){
  index+=pow(2,k);
}
psi_1[index]=1;
psi_2[index]=1;

//Zeitmessung
double time=0.0,time_c=0.0,time_k=0.0,tstart,zeitfaktor
   =0.0;


//Textausgabe
stringstream dateiname_k,dateiname_c;
dateiname_k<<"data/krylov_L"<<L<<"_m"<<m<<"_t"<<t_k<<"_T"
   <<T<<".data";
dateiname_c<<"data/chebyshev_L"<<L<<"_N"<<N<<"_t"<<t_c<<"
   _T"<<T<<".data";

ofstream fileout;
//Krylov aufrufen
if(meth=='k'||meth=='b'){
tstart=clock();
fileout.open((dateiname_k.str()).c_str());
fileout<<"#Krylov␣L␣=␣"<<L<<"␣m␣=␣"<<m<<endl;
for(int k=0; k<Zeitschritte_k; ++k){
  //Energie <psi|H|psi>
  phi=H(psi_1,L);
  betrag=0;
  for(int i=0; i<l; ++i){
```

```
    betrag+=conj(psi_1[i])*phi[i];
  }
  fileout<<t_k*k<<"␣␣␣"<<real(betrag);
  //<psi|S_z(j)|psi>
  for(int j=0; j<L; ++j){
    phi=S_z(psi_1,L,j);
    betrag=0;
    for(int i=0; i<l; ++i){
      betrag+=conj(psi_1[i])*phi[i];
    }
    fileout<<"␣␣␣"<<real(betrag);
  }
  fileout<<endl;
  psi_1=krylov(psi_1,t_k,L,m);
}
fileout.close();
time=clock()-tstart;
time=time/CLOCKS_PER_SEC;
cout<<"Zeit␣Krylov:"<<time<<endl;
time_k=time;
fileout.open("Zeiten.data",ios::app);
fileout<<number<<"␣"<<L<<"␣kr␣"<<m<<"␣"<<time<<"␣"<<t_k<<"
   ␣"<<T<<endl;
fileout.close();
}

//Chebyshev aufrufen
if(meth=='c'||meth=='b'){
tstart=clock();
fileout.open((dateiname_c.str()).c_str());
fileout<<"#Chebyshev␣L␣=␣"<<L<<"␣N␣=␣"<<N<<endl;
for(int k=0; k<Zeitschritte_c; ++k){
  //Energie <psi|H|psi>
  phi=H(psi_2,L);
  betrag=0;
  for(int i=0; i<l; ++i){
    betrag+=conj(psi_2[i])*phi[i];
  }
  fileout<<t_c*k<<"␣␣␣"<<real(betrag);
  //<psi|S_z(j)|psi>
  for(int j=0; j<L; ++j){
    phi=S_z(psi_2,L,j);
    betrag=0;
```

```
      for(int i=0; i<l; ++i){
        betrag+=conj(psi_2[i])*phi[i];
      }
      fileout<<"␣␣␣"<<real(betrag);
    }
    fileout<<endl;
    psi_2=chebyshev(psi_2,t_c,L,N);
  }
  fileout.close();
  time=clock()-tstart;
  time=time/CLOCKS_PER_SEC;
  cout<<"Zeit␣Chebyshev:"<<time<<endl;
  time_c=time;
  fileout.open("Zeiten.data",ios::app);
  fileout<<number<<"␣"<<L<<"␣ch␣"<<N<<"␣"<<time<<"␣"<<t_c<<"
     ␣"<<T<<endl;
  fileout.close();
  }

  //LaTeX Ausgabe
  if(meth=='b'){
    zeitfaktor=time_c/time_k;
    fileout.open("latex/Tabelle_Zeiten_L_10.tex",ios::app);
    fileout<<"␣␣$10^{-"<<number+2<<"}$␣&␣"<<time_k<<"␣&␣"<<
       time_c<<"␣&␣"<<zeitfaktor<<"␣\\\\"<<endl;
    fileout.close();
  }

  return 0;
}
```

Literatur

[And+99] E. Anderson u. a. *LAPACK Users' Guide*. Third.
Philadelphia, PA: Society for Industrial und Applied Mathematics, 1999.
ISBN: 0-89871-447-8 (paperback).

[H T84] R. Kosloff H. Tal-Exer. "An accurate and efficient scheme for propagating the time dependent Schrödinger equation".
In: *The Journal of Chemical Physics* 81.3967 (1984).
DOI: 10.1063/1.448136.

[Jad15] Ian P. McCulloch Jad C. Halimeh Fabian Kolley.
"Chebyshev matrix product state approach for time evolution".
In: *Physical Review B* 92.11 (2015). DOI: 10.1103/PhysRevB.92.115130.
arXiv: 1507.01226v1.

[Joh] Christopher Kormanyos John Maddock. *Boost C++ Libraries*.
URL: http://www.boost.org/doc/libs/1_64_0/boost/math/special_functions/bessel.hpp.

[Sch16] Ulrich Schollwöck. "Numerical methods in the study of non-equilibrium strongly interacting quantum many-body physics".
In: *Strongly Interacting Quantum Systems out of Equilibrium: Lecture Notes of the Les Houches Summer School*. Bd. 99.
Oxford Scholarship Online, 2016. ISBN: 978-0-19876-816-6.
DOI: 10.1093/acprof:oso/9780198768166.003.0004.

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

München, den 6. Juli 2017