5 Quantenmechanik

In diesem Kapitel wollen wir in eine Welt eintauchen, die sich erst im letzten Jahrhundert aufgetan hat: der Welt der Quanten. Diese Welt blieb deshalb so lange unentdeckt, weil sie sich meistens erst erschließt, wenn man extrem kleine Objekte betrachtet bzw. es mit extrem niedrigen Energien zu tun hat. Diese Welt des Allerkleinsten scheint sich oft dem gesunden Menschenverstand zu entziehen – was darauf zurückzuführen ist, dass letzterer aus unserer Erfahrungswelt mit normalen Dimensionen entstammt. Trotz seiner scheinbaren Widersprüche existiert ein – einigermaßen – sauberes mathematisches Modell, das in der Lage ist, alle Experimente mit ihren scheinbaren Widersprüchen zu beschreiben: die Quantenmechanik.

5.1 Die mathematische Struktur der Quantenmechanik

Die mathematische Struktur der Quantenmechanik könnte bereits ein Buch für sich füllen – aus diesem Grund beschränken wir uns an dieser Stelle auf das Nötigste.

Ein quantenmechanischer Zustand wird mathematisch durch einen sogenannten *Zustandsvektor* $|\psi\rangle$ repräsentiert. Hierbei haben wir gleich die in der Quantenmechanik übliche Schreibweise mit einem senkrechten Strich auf der einen Seite und einem nach außen gerichteten Winkel auf der anderen Seite eingeführt. Der zu diesen Zustandsvektoren zugehörige Vektorraum heißt *Hilbertraum* $\mathcal{H}$ und kann je nach Problem sehr unterschiedlich aussehen. Seine mathematische Struktur kann zum Beispiel diejenige des C^N sein, also des Raumes der Vektoren mit N komplexen Zahlen; sie kann aber auch äquivalent zu der der normierbaren komplexen Funktionen über den reellen Zahlen sein.

5.2 Operationen im Hilbertraum

In diesem Abschnitt wollen wir in Erinnerung rufen, welche Operationen mit Zustandsvektoren definiert sind. Zum einen kann man zwei beliebige Vektoren des Hilbertraums addieren und diese Summe ist dann wieder ein Vektor des Hilbertraums. Diese Addition ist kommutativ, es gilt also:

$$|\psi_1\rangle + |\psi_2\rangle = |\psi_2\rangle + |\psi_1\rangle \ . \tag{5.1}$$

Ebenso kann man jeden Vektor des Hilbertraumes mit komplexen Zahlen multiplizieren, und das Ergebnis dieser Multiplikation liegt wieder im Hilbertraum.

$$\alpha|\psi\rangle \in \mathcal{H} \ . \tag{5.2}$$

Diese Multiplikation ist ebenfalls kommutativ und darüber hinaus assoziativ, d.h. bei aufeinanderfolgenden Multiplikationen ist die Reihenfolge irrelevant und wir können auch alternativ zuerst die beiden Multiplikatoren miteinander multiplizieren und anschließend das Produkt dieses Ergebnisses mit dem Zustandsvektor bilden:

$$\alpha\beta|\psi\rangle = \beta\alpha|\psi\rangle = (\alpha\beta)|\psi\rangle \ . \tag{5.3}$$

Wir können aber auch zwei Vektoren miteinander multiplizieren und erhalten auf diese Weise eine komplexe Zahl, das sogenannte Skalarprodukt. An dieser Stelle möchten wir die in Abschn. 5.1 eingeführte Schreibweise um die für den dazugehörigen hermitesch transponierten Vektor erweitern: $\langle\psi|$. Hiermit schreibt sich das Skalarprodukt zweier Vektoren besonders einfach:

$$\langle\psi_1|\psi_2\rangle \in \mathcal{C} \ . \tag{5.4}$$

Mit der Einführung des Skalarproduktes können wir zwei Begriffe aus der Linearen Algebra übertragen:

– Die Länge l (oder auch der Betrag) eines Vektors $|\psi\rangle$ definiert sich durch

$$l^2 = \langle\psi|\psi\rangle \ . \tag{5.5}$$

– Zwei Vektoren, deren Längen von null verschieden sind und deren Skalarprodukt null ist, bezeichnen wir als orthogonal.

An dieser Stelle nun können wir eine Einschränkung bezüglich Zustandsvektoren im Hilbertraum machen: Es ist nämlich nicht jeder Vektor des Hilbertraums ein erlaubter Zustandsvektor – vielmehr sind ausschließlich normierte Vektoren als Zustandsvektoren erlaubt.

Abschließend wollen wir nun noch lineare Abbildungen innerhalb des Hilbertraumes einführen. Eine solche Abbildung lässt sich durch einen (linearen) Operator $\hat{A}$ beschreiben, der auf einen beliebigen Vektor des Hilbertraums angewendet einen neuen Vektor ebenfalls aus dem Hilbertraum ergibt:

$$\hat{A}|\psi\rangle \in \mathcal{H} \ . \tag{5.6}$$

Der Operatorcharakter wird durch das kleine Hütchen über dem A explizit zum Ausdruck gebracht. Linear bedeutet in diesem Fall, dass für beliebige λ

$$\hat{A}(\lambda|\psi\rangle) = \lambda(\hat{A}|\psi\rangle) \tag{5.7}$$

$$\hat{A}(|\psi\rangle + |\chi\rangle) = \hat{A}|\psi\rangle + \hat{A}|\chi\rangle \tag{5.8}$$

gilt.

Für das Folgende müssen wir außerdem festhalten, dass die Operatoren im Hilbertraum nicht unbedingt kommutativ sind, d.h. dass zwischen dem Produkt $\hat{A}\hat{B}$ und $\hat{B}\hat{A}$ unterschieden werden muss, wie man es z.B. auch von Matrizen kennt. Diese Nichtvertauschbarkeit von Operatoren spielt in der Quantenmechanik eine große Rolle, und weil derartige Ausdrücke sehr oft vorkommen, bekommt die Differenz

$$[\hat{A}, \hat{B}] = \hat{A}\hat{B} - \hat{B}\hat{A} \tag{5.9}$$

ein eigenes Symbol – eben die eckigen Klammern auf der linken Seite der Gleichung – und wird Kommutator genannt.

Bis zu diesem Punkt haben wir gegenüber der klassischen Physik noch nichts Neues eingeführt, da wir noch keinerlei Aussagen über diese Operatoren getroffen haben. Der entscheidende Punkt und der endgültige Schritt zur Quantenmechanik wird nun durch ein Postulat (also eine prinzipiell nicht beweisbare Behauptung) gemacht: Der Kommutator von Operatoren, die in der klassischen Dynamik kanonisch konjugierte Größen sind, ist $i\hbar$, also z.B.

$$[\hat{x}, \hat{p}_x] = \mathrm{i}\hbar \; . \tag{5.10}$$

Der entsprechende Kommutator bei nicht kanonisch konjugierten Größen hingegen soll null sein:

$$[\hat{x}, \hat{p}_y] = 0 \; . \tag{5.11}$$

Abschließend wollen wir noch in Erinnerung rufen, was unter der Funktion eines Operators, z.B. unter $\sin(\mathrm{d}/\mathrm{d}x)$ zu verstehen ist. Die Festlegung ist die, dass solche Funktionen über ihre Potenzreihen definiert sind. In unserem Beispiel des Sinus benötigen wir also zunächst die Potenzreihe der Sinusfunktion

$$\sin(z) = \sum_{n=0}^{\infty} (-1)^n \frac{1}{n!} z^n \; . \tag{5.12}$$

Wenn wir nun für z den Operator $\mathrm{d}/\mathrm{d}x$ einsetzen, erhalten wir den gesuchten Sinus der ersten Ableitung nach x. Beachten Sie bitte, dass der so gewonnene Operator beliebig hohe Ableitungen nach x enthält!

5.3 Eigenzustände und ihre Verwendung als Koordinatensysteme

Operatoren und Zustände sind zunächst sehr abstrakte Begriffe – zumindest wenn wir konkrete Probleme numerisch lösen wollen, müssen wir diese in eine in Zahlen gegossene Form bringen. Hierzu wollen wir zunächst auf spezielle Zustände im Hilbertraum eingehen, nämlich die *Eigenzustände* zu einem vorgegebenen Operator $\hat{A}$. Dabei handelt es sich um Zustände, die sich bei Anwendung dieses Operators bis auf einen Faktor reproduzieren, also:

$$\hat{A}|\lambda\rangle = \lambda|\lambda\rangle \; . \tag{5.13}$$

Der Faktor λ wird in diesem Zusammenhang *Eigenwert* und (5.13) Eigenwertgleichung genannt, und wir haben die in der Quantenmechanik übliche Schreibweise benutzt, in der der Eigenzustand $|\lambda\rangle$ durch seinen Eigenwert charakterisiert wird. Trotz der ähnlichen Schreibweise ist genau zwischen der *Zahl* λ und dem *Zustand* $|\lambda\rangle$ zu unterscheiden! Die Situation ist übrigens vollkommen analog zu Eigenvektoren und Eigenwerten zu einer Matrix M, falls Ihnen dies aus der Linearen Algebra vertraut ist.

Zunächst ohne Beweis ein paar Fakten über Eigenzustände und Eigenwerte, die wir kennen müssen, um das folgende nachvollziehen zu können. Wenn Sie an den Beweisen dieser Behauptungen interessiert sind, finden Sie diese z.B. in [7] oder [28].

- Wenn Sie einen Eigenzustand $|\lambda\rangle$ zu einem Operator gefunden haben, erhalten Sie durch Multiplikation mit einer beliebigen (von null verschiedenen) Zahl α weitere Eigenzustände $\alpha|\lambda\rangle$.
- Hat man zu einem Eigenwert zwei Eigenvektoren, so ist auch die Summe dieser beiden wieder ein Eigenvektor zum selben Eigenwert.
- für selbstadjungierte Operatoren stehen Eigenzustände zu verschiedenen Eigenwerte stehen immer senkrecht aufeinander, d.h. ihr Produkt ist null:

$$\langle\lambda_1|\lambda_2\rangle = 0 \; . \tag{5.14}$$

- Die Eigenzustände einer großen Klasse von Operatoren, nämlich der Hermiteschen Operatoren, bilden ein vollständiges System, d.h. jeder beliebige Zustand lässt sich als Linearkombination dieser Eigenzustände schreiben:

$$|\psi\rangle = \sum_{\lambda} \psi_\lambda |\lambda\rangle \; . \tag{5.15}$$

- Die Operatoren zu allen physikalischen Größen sind Hermitesch.

An dieser Stelle wollen wir noch kurz erklären, was unter *Entartung* zu verstehen ist. Die ersten beiden der oben angeführten Punkte besagen, dass die Eigenvektoren zu einem Eigenwert unter Hinzunahme des Nullvektors einen (mindestens eindimensionalen) Untervektorraum des Hilbertraums bilden. Falls dieser Raum nun eine Dimension größer als eins hat, spricht man von Entartung. Da die Berücksichtigung von Entartung in den meisten Fällen nur einen erhöhten Schreibaufwand darstellt, werden wir dieses Phänomen, wenn nicht explizit erforderlich, vernachlässigen.

Wieviele Eigenvektoren (und zugehörige Eigenwerte) gibt es nun zu einem vorgegebenen Operator? Das hängt ganz vom Operator ab – es gibt Operatoren, zu denen überhaupt keine Eigenvektoren existieren, es gibt aber auch Operatoren, die sozusagen die maximal mögliche Zahl von Eigenvektoren haben, deren Eigenvektoren nämlich eine Basis bezüglich des Hilbertraums

darstellen. Insbesondere gehören alle Hermiteschen Operatoren in diese Kategorie. Auf diese Weise liefern uns Hermitesche Operatoren eine Basis, in die wir alle Vektoren des Hilbertraums entwickeln können.

Gegeben sei ein Hermitescher Operator $\hat{A}$ mit Eigenwerten a_1, a_2, ... Da die zugehörigen Eigenvektoren eine Basis bilden ist

$$|\psi\rangle = \sum_i \langle a_i|\psi\rangle |a_i\rangle \tag{5.16}$$

$$= \sum_i \psi_i |a_i\rangle \ . \tag{5.17}$$

Hierbei haben wir die Komponenten ψ_i durch

$$\psi_i = \langle a_i|\psi\rangle \tag{5.18}$$

definiert.

Wir können jedoch nicht nur Zustände nach einer Basis entwickeln, sondern auch Operatoren:

$$\hat{O} = \sum_{ij} \langle a_i|\hat{O}|a_j\rangle |a_i\rangle\langle a_j| \tag{5.19}$$

$$= \sum_{ij} O_{ij} |a_i\rangle\langle a_j| \ . \tag{5.20}$$

Zusammenfassend sind wir nun in der Lage, sowohl Zustände als auch Operatoren des Hilbertraums durch Zahlenwerte, nämlich durch Koordinaten bzgl. einer Basis, darzustellen.

5.4 Orts- und Impulsdarstellung

Ein wichtiger Spezialfall des letzten Abschnittes ist die Darstellung bzgl. der Eigenzustände des Orts- bzw. des Impulsoperators. Im Vorgriff auf den nächsten Abschnitt stellen wir hier fest, dass die Eigenwerte der beiden Operatoren die Orts- bzw. die Impulsvektoren sind und damit ein dreidimensionales Kontinuum bilden. Unter Anwendung des vorhergehenden Abschnitts können wir einen beliebigen Zustand also nach Ortseigenzuständen

$$|\psi\rangle = \int \mathrm{d}^3\boldsymbol{x}\, \langle x|\psi\rangle |\boldsymbol{x}\rangle \tag{5.21}$$

$$= \int \mathrm{d}^3\boldsymbol{x}\, \psi(\boldsymbol{x})\, |\boldsymbol{x}\rangle \tag{5.22}$$

oder Impulseigenzuständen

$$|\psi\rangle = \int \mathrm{d}^3\boldsymbol{p}\, \langle p|\psi\rangle |\boldsymbol{p}\rangle \tag{5.23}$$

$$= \int \mathrm{d}^3\boldsymbol{p}\, \psi(\boldsymbol{p})\, |\boldsymbol{p}\rangle \tag{5.24}$$

entwickeln, wobei wir jedoch die Summation über die Eigenzustände durch entsprechende Integrale ersetzen müssen, da diese ein Kontinuum bilden.

Wie sieht nun der Ortsoperator in der Ortsdarstellung aus? Dazu entwickeln wir

$$\langle \boldsymbol{x}_1|\boldsymbol{x}|\boldsymbol{x}_2\rangle = \boldsymbol{x}_2\langle \boldsymbol{x}_1|\boldsymbol{x}_2\rangle \tag{5.25}$$

$$= \boldsymbol{x}_2\delta(\boldsymbol{x}_1-\boldsymbol{x}_2)\;. \tag{5.26}$$

Der Ortsoperator selber ist also

$$\hat{x} = \int \mathrm{d}^3\boldsymbol{x}_1 \int \mathrm{d}^3\boldsymbol{x}_2\; \boldsymbol{x}_2\delta(\boldsymbol{x}_1-\boldsymbol{x}_2)|\boldsymbol{x}_1\rangle\langle\boldsymbol{x}_2| \tag{5.27}$$

$$= \int \mathrm{d}^3\boldsymbol{x}\; \boldsymbol{x}|\boldsymbol{x}\rangle\langle\boldsymbol{x}|\;. \tag{5.28}$$

Wenn wir dies nun auf einen ebenfalls in Ortsdarstellung gegebenen Zustand $\psi(\boldsymbol{x})$ anwenden, erhalten wir

$$\langle \boldsymbol{x}|\hat{x}|\psi\rangle = \int \mathrm{d}^3\boldsymbol{x'}\; \boldsymbol{x'}\langle\boldsymbol{x}|\boldsymbol{x'}\rangle\langle\boldsymbol{x'}|\psi\rangle \tag{5.29}$$

$$= \boldsymbol{x}\,\psi(\boldsymbol{x})\;. \tag{5.30}$$

Der Ortsoperator $\hat{x}$ wird also lediglich zu einem Faktor $\boldsymbol{x}$.

Entsprechendes gilt natürlich für den Impulsoperator in Impulsdarstellung. Wie aber sieht der Impulsoperator in Ortsdarstellung aus? Das einzige, was wir über die Beziehung zwischen Orts- und Impulsoperator wissen, ist der Kommutator (5.10). Wenn dieser erfüllt sein soll, muss der Impulsoperator durch $-\mathrm{i}\hbar\mathrm{d}/\mathrm{d}\boldsymbol{x}$ gegeben sein, denn für jede beliebige Funktion $\psi(\boldsymbol{x})$ gilt:

$$\left(-\boldsymbol{x}\mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}} + \mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}\boldsymbol{x}\right)\psi(\boldsymbol{x}) = \mathrm{i}\hbar\psi(\boldsymbol{x})\;. \tag{5.31}$$

Entsprechend ist der Ortsoperator in Impulsdarstellung $\mathrm{i}\hbar\mathrm{d}/\mathrm{d}\boldsymbol{p}$.

Aus der Darstellung des Impulsoperators in Ortsdarstellung und umgekehrt kann man auch entnehmen, dass die Transformation von der einen in die andere Darstellung eine Fouriertransformation (siehe auch Anhang F) sein muss:

$$\psi(\boldsymbol{p}) = \left(\frac{1}{\sqrt{2\pi\hbar}}\right)^3 \int \mathrm{d}^3\boldsymbol{x}\;\psi(\boldsymbol{x})\exp\left(\frac{\mathrm{i}}{\hbar}\boldsymbol{x}\boldsymbol{p}\right) \tag{5.32}$$

und entsprechend

$$\psi(\boldsymbol{x}) = \left(\frac{1}{\sqrt{2\pi\hbar}}\right)^3 \int \mathrm{d}^3\boldsymbol{p}\;\psi(\boldsymbol{p})\exp\left(-\frac{\mathrm{i}}{\hbar}\boldsymbol{x}\boldsymbol{p}\right)\;. \tag{5.33}$$

5.5 Die Kopenhagener Interpretation der Quantenmechanik

Bis zu diesem Punkt haben wir jede Menge mathematischer Begriffe eingeführt (Zustandsvektor, Skalarprodukt, Operator, Kommutator, ...), aber noch keinen physikalischen Bezug hergestellt. Letztendlich aber muss die Quantenmechanik wie jede andere physikalische Theorie den Ausgang von Experimenten vorhersagen. Diesen Zusammenhang zwischen dem mathematischen Formalismus und dem Ergebnis von Messvorgängen liefert die sogenannte *Kopenhagener Interpretation*, die sich in drei Postulate zusammenfassen lässt.

Das erste Postulat macht eine Aussage darüber, welche Ergebnisse die Messung haben kann. Wird an einem quantenmechanischen System die Messung einer Observablen O vorgenommen, so wird (Messfehler außer Acht gelassen) immer ein Eigenwert o des zugehörigen Operators $\hat{O}$ gemessen.

Das zweite Postulat konkretisiert nun das Gesagte, indem es eine Aussage über die Wahrscheinlichkeit für das Eintreffen der möglichen Ergebnisse macht: Die Wahrscheinlichkeit $P(o)$ für einen konkreten Eigenwert o ist gleich dem Absolutbetrag der Projektion des Zustandsvektors auf den entsprechenden Eigenvektor:

$$P(o) = |\langle o|\psi\rangle|^2 \; . \tag{5.34}$$

An dieser Stelle führen wir also eine Zufallskomponente ein – auch wenn wir den Zustand des Systems genau kennen, können wir den Ausgang eines konkreten Experiments nicht vorhersagen – es sei denn, dieser Zustand ist ein Eigenzustand der Messobservablen. Dies hat eine andere Qualität als der statistische Charakter aus der klassischen Physik (z.B. bei der Brownschen Bewegung), der daher rührt, dass wir das System eben nicht vollständig kennen.

Das letzte Postulat schließlich legt fest, in welchem Zustand sich das System nach der Messung befindet, nämlich in der Projektion des ursprünglichen Zustands $|\psi\rangle$ auf den Eigenraum zum Eigenwert o.

Mittels dieser Postulate lässt sich der Mittelwert einer Observablen berechnen:

$$\langle O\rangle = \sum_n P(o_n) o_n \tag{5.35}$$

$$= \sum_n \langle\psi|o_n\rangle o_n \langle o_n|\psi\rangle \tag{5.36}$$

$$= \langle\psi|\hat{O}|\psi\rangle \; . \tag{5.37}$$

5.6 Schrödingergleichung

Wodurch wird nun die Dynamik in der Quantenmechanik festgelegt? Die Grundgleichung – also das, was in der klassischen Mechanik die Newtonschen

Axiome und in der Elektrodynamik die Maxwellschen Gleichungen sind – ist hier die Schrödingergleichung, die festlegt, wie sich ein Zustand mit der Zeit verändert:

$$\mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}t}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle \ . \tag{5.38}$$

Hierbei ist $\hat{H}$ der Hamiltonoperator, das quantenmechanische Pendant zur Hamiltonfunktion, die in der klassischen Mechanik – im skleronomen Fall – die Energie eines Systems als Funktion aller Orte und Impuls angibt.

Dabei haben wir uns auf das sogenannte *Schrödingerbild* beschränkt, in dem die Zustandsvektoren zeitabhängig sind und die Operatoren nur eine explizite Zeitabhängigkeit haben können. Auf die umgekehrte Sichtweise des *Heisenbergbildes* werden wir in diesem Buch nicht eingehen.

5.7 Bestimmung des Hamilton-Operators

Wie wir aus (5.38) entnehmen, müssen wir zunächst den Hamiltonoperator eines Problems bestimmen, ehe wir uns daran machen können, seine Bewegungsgleichung zu lösen. Dieser Schritt entspricht dem Bestimmen der Bewegungsgleichungen im Fall der klassischen Physik. Für Probleme, die eine klassische Entsprechung haben, also zum Beispiel die Bewegung eines Teilchens in einem Potential, kann der Hamiltonoperator der Quantenmechanik aus der Hamiltonfunktion der klassischen Physik bestimmt werden – daher auch die Namensgebung *Hamiltonoperator*. Dies geschieht einfach dadurch, dass man die klassischen Größen durch die entsprechenden Operatoren ersetzt – also die Ortsvariable x durch den Ortsoperator $\hat{x}$, die Impulsvariable p durch den Impulsoperator $\hat{p}$ und so weiter.

5.8 Das freie Teilchen

Bevor wir uns komplizierteren Problemen zuwenden, wollen wir das in den beiden vorangegangenen Abschnitten Zusammengefasste festigen, indem wir es auf ein sehr einfaches Problem anwenden. Das einfachste Problem der klassischen Physik ist ein freies Teilchen ohne äußere Kräfte, das sich in einem Inertialsystem bekanntlich gleichförmig bewegt. Seine Hamiltonfunktion ist durch

$$H = \frac{1}{2m}\boldsymbol{p}^2 = \frac{1}{2m}\left(p_x^2 + p_y^2 + p_z^2\right) \tag{5.39}$$

gegeben. Daraus ergibt sich der Hamiltonoperator des entsprechenden quantenmechanischen Problems:

$$\hat{H} = \frac{1}{2m}\left(\hat{p}_x^2 + \hat{p}_y^2 + \hat{p}_z^2\right) \ , \tag{5.40}$$

das heißt, sowohl H als auch die Impulskomponenten p_x, p_y und p_z haben nun Operatorcharakter. Das gleiche gilt für die Komponenten des Ortsvektors x, y und z, die in (5.40) aber nicht auftreten.

Da der Hamiltonoperator eine Funktion des Impulsoperators ist, ist dieses Problem am Einfachsten in der Impulsdarstellung zu lösen. Dazu entwickeln wir die Schrödingergleichung nach den Impulseigenzuständen $|p_x, p_y, p_z\rangle = |\boldsymbol{p}\rangle$ (die analytische Betrachtung können wir dreidimensional durchführen, während wir uns bei der nachfolgenden numerischen Behandlung auf eine Dimension beschränken müssen):

$$\mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}t}\int \mathrm{d}^3\boldsymbol{p}\,\langle\boldsymbol{p}|\psi(t)\rangle|\boldsymbol{p}\rangle = \frac{1}{2m}\int \mathrm{d}^3\boldsymbol{p}\,\boldsymbol{p}^2\langle\boldsymbol{p}|\psi(t)\rangle|\boldsymbol{p}\rangle\ . \tag{5.41}$$

Da die Impulseigenzustände $|\boldsymbol{p}\rangle$ linear unabhängig sind, kann dies nur erfüllt sein, wenn die beiden Integranden gleich sind:

$$\mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}t}\langle\boldsymbol{p}|\psi(t)\rangle = \mathrm{i}\hbar\frac{\mathrm{d}}{\mathrm{d}t}\psi(\boldsymbol{p},t) \tag{5.42}$$

$$= \frac{1}{2m}\boldsymbol{p}^2\psi(\boldsymbol{p},t)\ . \tag{5.43}$$

Diese Differentialgleichung lässt sich direkt lösen und wir erhalten als Lösung:

$$\psi(\boldsymbol{p},t) = \psi(\boldsymbol{p},0)\exp\left(-\mathrm{i}\frac{1}{2m\hbar}\boldsymbol{p}^2 t\right)\ . \tag{5.44}$$

Nachdem wir die Lösung im Impulsraum haben, können wir diese mittels (5.33) in den Ortsraum transformieren:

$$\psi(\boldsymbol{x},t) = \left(\frac{1}{\sqrt{2\pi\hbar}}\right)^3\int \mathrm{d}^3\boldsymbol{p}\ \psi(\boldsymbol{p},t)\exp\left(-\frac{\mathrm{i}}{\hbar}\boldsymbol{x}\boldsymbol{p}\right)\ . \tag{5.45}$$

Schreiten wir nun zur numerischen Implementation des Problems und beginnen mit der vielleicht naheliegendsten Lösung: Die Schrödingergleichung ist im Ortsraum eine partielle Differentialgleichung, die Ableitungen erster Ordnung nach der Zeit enthält:

$$\mathrm{i}\hbar\frac{\partial}{\partial t}\psi(x) = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\psi(x)\ . \tag{5.46}$$

Diese Differentialgleichung können wir direkt einer entsprechenden Bibliotheksroutine übergeben. Das Ergebnis könnte dann in ungefähr so aussehen (im Folgenden setzen wir $\hbar = m = 1$):

```
1 /*************************************************************************
2 * Name:      frei1.cpp                                                   *
3 * Zweck:     Simuliert ein quantenmechanisches freies Teilchen           *
4 * Gleichung: Schroedingergleichung ohne Potential                        *
5 * verwendete Bibiliothek: GSL                                           *
```

```
6  ******************************************************************************/
7
8  #include <iostream>
9  #include <fstream>
10 #include <gsl/gsl_errno.h>
11 #include <gsl/gsl_odeiv.h>
12 #include <math.h>
13 #include "tools.h"
14
15 using namespace std;
16
17 //-- Zahl der Stuetzstellen
18 const int nmax = 8192;
19
20 //-- globale Variablen
21 double dx;
22
23 //------------------------------------------------------------------------
24
25 int f(double t, const double psi[], double dpsi_dt[], void *params)
26
27 {
28 int n;
29 double norm, norm1, norm2;
30
31 //-- Realteil
32 for (n=1; n<nmax-1; n++)
33   dpsi_dt[n] = -(psi[nmax+n+1]+psi[nmax+n-1]-2*psi[nmax+n]) / sqr(dx);
34
35 dpsi_dt[0]      = -(psi[nmax+1]+psi[2*nmax-1]-2*psi[nmax]) / sqr(dx);
36 dpsi_dt[nmax-1] = -(psi[nmax]+psi[2*nmax-2]-2*psi[2*nmax-1])/ sqr(dx);
37
38 //-- Imaginaerteil
39 for (n=nmax+1; n<2*nmax-1; n++)
40   dpsi_dt[n] = +(psi[n+1-nmax]+psi[n-1-nmax]-2*psi[n-nmax]) / sqr(dx);
41
42 dpsi_dt[nmax]     = +(psi[1]+psi[nmax-1]-2*psi[0]) / sqr(dx);
43 dpsi_dt[2*nmax-1] = +(psi[0]+psi[nmax-2]-2*psi[nmax-1]) / sqr(dx);
44
45 return GSL_SUCCESS;
46 }
47
48 //----------------------------------------------------------------------
49
50 int jac(double t, const double x[], double *dfdx, double dxdt[],
51         void *params)
52
53 {
54 return GSL_SUCCESS;
55 }
56
57 //------------------------------------------------------------------------
58
59 main( int argc, char *argv[] )
60
61 {
62 //-- Definition der Variablen
63 int    n, n1, n_start, n_out, max_fct, status;
64 double psi[2*nmax], dpsi_dt[2*nmax], t, t1, t2, tend, dt, rtol, atol;
```

```
65  double alpha, p_0, x, h, hmax, daux, sum;
66  double mu = 10;
67  ifstream in_stream;
68  ofstream out_stream;
69
70  //-- Fehlermeldung, wenn Input- und Outputfilename nicht uebergeben wurden
71  if (argc<3)
72    {
73    cout << " Aufruf: frei1 infile outfile\n";
74    exit(1);
75    }
76
77  //-- Einlesen der Parameter --
78  in_stream.open(argv[1]);
79  out_stream.open(argv[2]);
80  out_stream << "! Ergebnis-Datei generiert von frei1.cpp\n";
81  inout(tend,"tend");        inout(dx,"dx");
82  inout(n_out,"n_out");      inout(alpha,"alpha");
83  inout(p_0,"p_0");          inout(rtol,"rtol");
84  inout(atol,"atol");
85  in_stream.close();
86
87  //-- Berechnung einiger benoetigter Parameter --
88  dt = tend / n_out;
89  h = 1.e-6;
90
91  //-- Anfangsbedingungen
92  t = 0;
93  max_fct = 1000000;
94  for (n=0; n<nmax; n++)
95    {
96    x = (n-nmax/2) * dx;
97    psi[n] = exp(-sqr(x/alpha)/2);
98    sum  = sum + sqr(psi[n]);
99    }
100 sum = 1 / sqrt(sum);
101
102 for (n=0; n<nmax; n++)
103   {
104    x = (n-nmax/2) * dx;
105    psi[n+nmax] = -psi[n] * sum * sin(p_0*x);   // Imaginaerteil
106    psi[n] = psi[n] * sum * cos(p_0*x);         // Realteil
107   }
108
109 out_stream << t << "\n";
110 for (n=0; n<nmax; n++)
111   out_stream << psi[n] << " " << psi[n+nmax] << "\n";
112
113 const gsl_odeiv_step_type *T = gsl_odeiv_step_rkf45;
114 gsl_odeiv_step    *s = gsl_odeiv_step_alloc(T, 2*nmax);
115 gsl_odeiv_control *c = gsl_odeiv_control_y_new(atol, rtol);
116 gsl_odeiv_evolve  *e = gsl_odeiv_evolve_alloc(2*nmax);
117 gsl_odeiv_system  sys = {f, jac, 2*nmax, &mu};
118
119 //-- Zeitschleife --
120 t = 0;
121 for (n=1; n<=n_out; n++)
122   {
123   t1 = n * dt;
```

```
124   cout << n << " " << t << "\n";
125   while (t<t1)
126     {
127     status = gsl_odeiv_evolve_apply(e, c, s, &sys, &t, t1, &h, psi);
128     if (status != GSL_SUCCESS) break;
129     }
130   out_stream << t << "\n";
131   for (n1=0; n1<nmax; n1++)
132     out_stream << psi[n1] << " " << psi[n1+nmax] << "\n";
133   }
134
135 gsl_odeiv_evolve_free(e);
136 gsl_odeiv_control_free(c);
137 gsl_odeiv_step_free(s);
138
139 out_stream.close();
140 }
```

Besonderes Augenmerk verdient hier die Berechnung der zweiten Ableitung der Wellenfunktion ψ nach der sogenannten Dreipunktformel in Zeilen 33 und 40:

$$\frac{\mathrm{d}^2}{\mathrm{d}^2 x}\psi(x_n) \approx \frac{1}{(\Delta x)^2}\left(\psi(x_{n+1}) - 2\psi(x_n) + \psi(x_{n-1})\right) \ . \tag{5.47}$$

Am Rand des abgedeckten Intervalls, also bei x_0 und $x_{\mathrm{nmax}-1}$, lässt sich diese Formel nicht direkt anwenden, da $\psi(x_{-1})$ und $\psi(x_{\mathrm{nmax}})$ nicht bekannt sind. Für's erste haben wir hier periodische Randbedingungen angenommen, bei denen diese beiden fehlenden Werte durch $\psi(x_{\mathrm{nmax}-1})$ und $\psi(x_0)$ ersetzt werden. Wir werden diesen Punkt noch eingehender beleuchten und verschiedene Alternativen zu dieser Wahl erläutern.

Ein zweiter Punkt, der der Erläuterung bedarf, ist die Konstruktion des Ausgangszustandes in Zeile 91 bis 107. Wir könnten diesen Ausgangszustand zwar auch direkt aus der – entsprechend modifizierten – Eingabedatei einlesen; wir werden uns aber an dieser Stelle auf eine kleine Klasse von Ausgangszuständen beschränken, die durch lediglich zwei Parameter festgelegt werden. Dabei handelt es sich um Gauß-förmige Wellenpakete, die sich zum Zeitpunkt $t = 0$ am Ursprung befinden und sich mit einem mittleren Impuls $\bar{p}$ bewegen. Außer diesem mittleren Impuls braucht man nur noch die anfängliche Breite α des Wellenpakets im Ortsraum.

Zunächst aber wollen wir uns die Frage stellen, ob unser Programm vom Standpunkt der Performance und der Stabilität eine gute Implementation des gegebenen Problems darstellt. Die Antwort auf diese Frage muss – zumindest in der gegenwärtigen Form – eindeutig *nein* lauten; wir wollen jedoch hier schon anmerken, dass wir auf das Programm `frei1.cpp` noch zurückgreifen werden. Der Grund dafür, dass `frei1.cpp` in puncto Performance nicht zufriedenstellen kann, ist die Tatsache, dass wir keinerlei Gebrauch davon gemacht haben, dass uns im Impulsraum die analytische Lösung bekannt ist (5.44), weswegen wir eine hohe Zahl von Teilschritten Δt in Kauf nehmen, in die die Bibliotheksroutine zur Lösung der Differentialgleichung die

Zeit unterteilt. Ein auf der analytischen Lösung im Impulsraum aufbauendes Programm könnte folgendermaßen vorgehen:

- Transformation des Ausgangszustandes im Ortsraum $\psi(x, t = 0)$ in den Impulsraum $\psi(p, t = 0)$ durch Fouriertransformation.
- Berechnung des Zustandes $\psi(p, t_n)$ zu den Zeiten t_n, zu denen die Lösung erwünscht ist mittels (5.44).
- Rücktransformation dieser Zustände in den Ortsraum durch inverse Fouriertransformation.

Diese Vorgehensweise ist in dem folgenden Programm realisiert:

```
/*************************************************************************
* Name:      frei2.cpp                                                   *
* Zweck:     Simuliert ein quantenmechanisches freies Teilchen           *
* Gleichung: Schroedingergleichung ohne Potential                        *
* verwendete Bibiliothek: GSL                                            *
*************************************************************************/

#include <iostream>
#include <fstream>
#include <math.h>
#include <gsl/gsl_fft_complex.h>
#include "tools.h"

using namespace std;

//-- Zahl der Stuetzstellen
const int nmax = 8192;

//------------------------------------------------------------------------

main( int argc, char *argv[] )

{
//-- Definition der Variablen
int    n, n1, n_start, n_out, max_fct, status;
double psi[2*nmax], psi_p[2*nmax], t, t1, t2, tend, dt, alpha;
double x, dx, p, dp, p_0, arg, sum;
double mu = 10;
ifstream in_stream;
ofstream out_stream;
gsl_fft_complex_wavetable * wavetable;
gsl_fft_complex_workspace * workspace;

//-- Fehlermeldung, wenn Input- und Outputfilename nicht uebergeben wurden
if (argc<3)
  {
  cout << " Aufruf: frei2 infile outfile\n";
  exit(1);
  }

//-- Einlesen der Parameter --
in_stream.open(argv[1]);
out_stream.open(argv[2]);
out_stream << "! Ergebnis-Datei generiert von frei2.cpp\n";
inout(tend,"tend");       inout(dx,"dx");
```

```
46 inout(n_out,"n_out");     inout(alpha,"alpha");
47 inout(p_0,"p_0");
48 in_stream.close();
49
50 //-- Berechnung einiger benoetigter Parameter --
51 dt = tend / n_out;
52 dp = 2 * pi / dx / nmax;
53
54 //-- Anfangsbedingungen
55 t = 0;
56 for (n=0; n<nmax; n++)
57   {
58   x = (n-nmax/2) * dx;
59   psi[2*n] = exp(-sqr(x/alpha)/2);
60     sum  = sum + sqr(psi[2*n]);
61   }
62 sum = 1 / sqrt(sum);
63 for (n=0; n<nmax; n++)
64   {
65   x = (n-nmax/2) * dx;
66   psi[2*n+1] = -psi[2*n] * sum * sin(p_0*x);   // Imaginaerteil
67   psi[2*n]   =  psi[2*n] * sum * cos(p_0*x);   // Realteil
68   }
69
70 //--Anfangszustand im Impulsraum
71 for (n=0; n<=2*nmax; n++) psi_p[n] = psi[n];
72 gsl_fft_complex_radix2_forward (psi_p, 1, nmax);
73
74 //-- Zeitschleife --
75 t = 0;
76 for (n=1; n<=n_out; n++)
77   {
78   t1 = n * dt;
79   cout << n << " " << t1 << " " << t1*dp*nmax/2 << "\n";
80   for (n1=0; n1<nmax/2; n1++)
81     {
82     p = n1 * dp;
83     arg = sqr(p)*t1/2;
84     psi[2*n1]   = psi_p[2*n1] * cos(arg) - psi_p[2*n1+1] * sin(arg);
85     psi[2*n1+1] = psi_p[2*n1] * sin(arg) + psi_p[2*n1+1] * cos(arg);
86     }
87   for (n1=nmax/2; n1<nmax; n1++)
88     {
89     p = (n1-nmax) * dp;
90     arg = sqr(p)*t1/2;
91     psi[2*n1]   = psi_p[2*n1] * cos(arg) - psi_p[2*n1+1] * sin(arg);
92     psi[2*n1+1] = psi_p[2*n1] * sin(arg) + psi_p[2*n1+1] * cos(arg);
93     }
94   gsl_fft_complex_radix2_inverse (psi, 1, nmax);
95   out_stream << t << "\n";
96   for (n1=0; n1<nmax; n1++)
97     out_stream << (n1-nmax/2)*dx << " " << psi[2*n1] << " " << psi[2*n1+1]
98                << " " << sqr(psi[2*n1])+sqr(psi[2*n1+1]) << "\n";
99   }
100
101 out_stream.close();
102 }
```

Betrachten wir nun die Lösungen, die uns dieses oder das vorangegangene Programm liefert, so scheint sich das Wellenpaket zunächst wie erwartet zu verhalten:

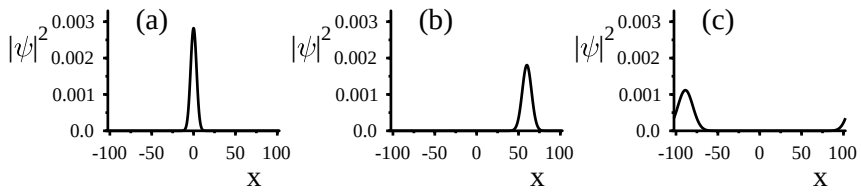

Abb. 5.1. Zeitentwicklung eines Wellenpakets berechnet mit `frei2.cpp`. Zunächst wandert das Paket nach rechts, wenn es den Rand des abgedeckten Ortsbereichs erreicht, verschwindet es dort und taucht am entgegengesetzten Ende wieder auf

Das Wellenpaket wandert aufgrund seines positiv gewählten Anfangsimpulses p_0 nach rechts. Gleichzeit läuft das Wellenpaket auseinander, d.h. es verliert an Höhe und wird breiter. Interessant wird es, wenn es den Rand des betrachteten Ortsbereichs erreicht: Während ein Teil des Pakets scheinbar nach rechts das Intervall verlässt, kommt ein Double unseres Wellenpakets am anderen Ende des Intervalls ins Spiel. Diese implizite Periodizität können wir schon am Programmcode an einigen Punkten festmachen:

- Im Programm `frei1.cpp` werden die Randbedingungen in den Zeilen 35/36 sowie in 42/43 explizit festgelegt. Indem wir z.B. das eigentlich benötigte `psi[-1]` (links vom x-Bereich) durch `psi[nmax-1]` (am rechten Rand des x-Bereichs) ersetzt haben, haben wir uns für periodische Randbedingungen entschieden. An dieser Stelle lassen sich also die Randbedingungen verhältnismäßig einfach ändern.
- Im Programm `frei2.cpp` sehen wir, dass die Zeitentwicklung sich in Impulsdarstellung lediglich in einer Phase $\exp(-\mathrm{i}p^2t/2)$ äußert (Zeile 84/85 und 91/92). Es ist also offensichtlich, dass die Norm der Wellenfunktion erhalten bleibt und ein Wellenpaket nicht einfach an den Enden des Ortsbereichs verschwinden kann.
- Die Periodizität im Ortsbereich liegt im Programm `frei2.cpp` in der Verwendung der FFT, die eigentlich keine Fourier*transformation* berechnet, sondern eine Fourier*reihenentwicklung* (siehe Anhang F).

Versuchen wir es zunächst mit einer Änderung der Randbedingungen in den Zeilen 35/36 und 42/43 von `frei1.cpp`, indem wir diese durch

```
dpsi_dt[0]      = - (psi[nmax+1]-2*pis[nmax]) / sqr(dx);
dpsi_dt[nmax-1] = - (psi[2*nmax-2]-2*psi[2*nmax-1]) / sqr(dx)
```

bzw. durch

```
dpsi_dt[nmax]     = + (psi[1]-2*psi[0]) / sqr(dx);
dpsi_dt[2*nmax-1] = + (psi[nmax-2]-2*psi[nmax-1]) / sqr(dx)
```

ersetzen. Dadurch dass wir ψ jenseits des Intervalls ersatzlos haben wegfallen lassen, implizieren wir, dass die Wellenfunktion außerhalb des betrachteten Ortsbereichs null ist – was bei einem unendlich tiefen Potentialtopf der Fall ist. Entsprechend bekommen wir durch diese Änderung ein Verhalten analog zu dem beim Potentialtopf: das Wellenpaket wird beim Erreichen des Randes des Potentialtopfs an demselben reflektiert – auch nicht das Verhalten, das wir im Normalfall haben wollen. Vielmehr wünschen wir in der Mehrheit der Fälle, dass das Teilchen nach dem Erreichen des Randes den betrachteten Ortsbereich verlässt ohne irgendwo anders wiederaufzutauchen.

Um zu verstehen, wie wir ein solches Verhalten realisieren können, müssen wir uns zunächst Gedanken über die Normerhaltung machen – denn genau diese müssen wir aushebeln, wenn wir wollen, dass das Teilchen irgendwo verschwindet. Betrachten wir hierzu die Dynamik eines Teilchens in einem Potential $V(x)$. Die Schrödingergleichung in Ortsdarstellung lautet in skalierten Einheiten

$$\mathrm{i}\frac{\mathrm{d}}{\mathrm{d}t}\psi(x,t) = -\frac{1}{2}\frac{\mathrm{d}^2}{\mathrm{d}x^2}\psi(x,t) + V(x)\psi(x)\ . \tag{5.48}$$

Damit können wir bestimmen, ob sich die Norm der Wellenfunktion ändert:

$$\frac{\mathrm{d}}{\mathrm{d}t}N = \frac{\mathrm{d}}{\mathrm{d}t}\int \mathrm{d}x\ \psi^\star(x)\psi(x) \tag{5.49}$$

$$= \frac{\mathrm{i}}{2}\int \mathrm{d}x\ \left[\left(\psi^\star(x)\frac{\mathrm{d}^2}{\mathrm{d}x^2}\psi(x) - \text{c.c.}\right)\right. \tag{5.50}$$

$$\left. - \left(\psi^\star(x)(V(x) - V^\star(x))\psi(x)\right)\right]\ . \tag{5.51}$$

Hierbei bedeutet c.c. den zum Vorausgegangenen konjugierten Ausdruck. Durch partielle Integration sieht man, dass der erste Term zu keiner Änderung der Norm führen kann. Bei normalen, also reellen Potentialen verschwindet auch der zweite Term – wie es auch sein muss, da in Wirklichkeit kein Teilchen einfach verschwinden kann. Gleichzeitig liefert der Ausdruck (5.51) aber einen Hinweis, was man tun muss, um absorbierende Randbedingungen zu implementieren: Man muss in diesem Bereich ein imaginäres Potential $V(x)$ hinzufügen. Da solche Potentiale zuerst in der klassischen Optik verwendet wurden, werden diese auch als *optische* Potentiale bezeichnet.

Ein solches – zwangsweise ortsabhängiges Potential – führt jedoch dazu, dass das Problem nicht mehr durch Transformation in den Impulsraum gelöst werden kann. Das Programm `frei2.cpp` kann also so nicht mehr verwendet werden. Andererseits haben wir festgestellt, dass `frei1.cpp` sehr viele Teilschritte und damit enorme Rechenzeiten benötigt. Wir müssen also nach einem Weg suchen, die analytische Lösung für das Ausgangsproblem (ohne Potential) mit einem ortsabhängigen Potential zu verknüpfen. Eine Möglichkeit besteht in der Näherung

$$\exp(\mathrm{i}(\hat{H}_1 + \hat{H}_2)\Delta t) \approx \exp(\mathrm{i}\hat{H}_1\mathrm{d}t)\exp(\mathrm{i}\hat{H}_2\Delta t)\ , \tag{5.52}$$

die für beliebige Operatoren H_1 und H_2 möglich ist. In unserem Fall wählen wir

$$H_1 = -\frac{1}{2}\mathrm{d}^2\mathrm{d}x^2 \tag{5.53}$$

$$H_2 = V(x) \tag{5.54}$$

und trennen die Dynamik damit in zwei Abschnitte, die diagonal im Impulsraum bzw. im Ortsraum sind. Den Fehler, den wir dabei machen, ist von der Ordnung

$$[\hat{H}_1, \hat{H}_2](\Delta t)^2 \ , \tag{5.55}$$

d.h. im Gegensatz zu der Situation ohne Potential müssen wir die Zeitentwicklung in viele kleine Zeitschritte unterteilen, um den Fehler gering zu halten, was zunächst keine Verbesserung gegenüber `frei1.cpp` verspricht. Außerdem stellen wir mit etwas Enttäuschung fest, dass der Fehler wesentlich größer ist als bei den Verfahren zur Integration der Schrödingergleichung im Ortsraum, die wir im Programm `frei1.cpp` eingesetzt haben (dort war der Fehler in der Größenordnung $(\Delta t)^5$). Zunächst ist also nicht einzusehen, warum ein auf dieser Näherung basierendes Programm eine Verbesserung darstellen sollte. Einen Punkt wollen wir jedoch noch erwähnen, bevor wir daran gehen, die Näherung (5.52) in ein konkretes Computerprogramm umzusetzen: Eine wichtige Eigenschaft bleibt dieser Näherung nämlich exakt erhalten: wenn $\hat{H}_1$ und $\hat{H}_2$ hermitesch sind, bleibt die Norm der Wellenfunktion erhalten.

```
/*************************************************************************
* Name:      frei3.cpp                                                   *
* Zweck:     Simuliert ein quantenmechanisches freies Teilchen           *
* Gleichung: Schroedingergleichung mit optischem Potential               *
* verwendete Bibiliothek: GSL                                            *
*************************************************************************/

#include <iostream>
#include <fstream>
#include <math.h>
#include <gsl/gsl_fft_complex.h>
#include "tools.h"

using namespace std;

//-- Zahl der Stuetzstellen
const int nmax = 8192;

//-- globale Variablen
double psi[2*nmax], V[nmax];
double dp;
gsl_fft_complex_wavetable * wavetable;
gsl_fft_complex_workspace * workspace;

//------------------------------------------------------------------------

int timestep(double dt)

{
```

```
int    n, n1;
double psi_p[2*nmax], arg, p;

//-- Zustand im Impulsraum
for (n=0; n<=2*nmax; n++) psi_p[n] = psi[n];
gsl_fft_complex_radix2_forward (psi_p, 1, nmax);

//-- Zeitentwicklung (freie Dynamik)
for (n1=0; n1<nmax/2; n1++)
  {
  p = n1 * dp;
  arg = sqr(p)/2 * dt;
  psi[2*n1]   = psi_p[2*n1] * cos(arg) - psi_p[2*n1+1] * sin(arg);
  psi[2*n1+1] = psi_p[2*n1] * sin(arg) + psi_p[2*n1+1] * cos(arg);
  }
for (n1=nmax/2; n1<nmax; n1++)
  {
  p = (n1-nmax) * dp;
  arg = sqr(p)/2 * dt;
  psi[2*n1]   = psi_p[2*n1] * cos(arg) - psi_p[2*n1+1] * sin(arg);
  psi[2*n1+1] = psi_p[2*n1] * sin(arg) + psi_p[2*n1+1] * cos(arg);
  }

//-- Ruecktransformation in den Ortsraum
gsl_fft_complex_radix2_inverse (psi, 1, nmax);

//-- Zeitentwicklung (optisches Potential)
for (n1=0; n1<nmax; n1++)
  {
  arg = - V[n1] * dt;
  psi[2*n1]   = psi[2*n1] * exp(arg);
  psi[2*n1+1] = psi[2*n1+1] * exp(arg);
  }

return 0;
}

//-------------------------------------------------------------------------

main( int argc, char *argv[] )

{
//-- Definition der Variablen
int    n, n1, n_start, n_out, n_interm, max_fct, status;
double t, t1, t2, tend, dt, rtol, atol;
double alpha, x, dx, p, p_0, a_opt, b_opt, arg, sum;
double mu = 10;
ifstream in_stream;
ofstream out_stream;

//-- Fehlermeldung, wenn Input- und Outputfilename nicht uebergeben wurden
if (argc<3)
  {
  cout << " Aufruf: frei3 infile outfile\n";
  exit(1);
  }

//-- Einlesen der Parameter --
in_stream.open(argv[1]);
```

```
89  out_stream.open(argv[2]);
90  out_stream << "! Ergebnis-Datei generiert von frei3.cpp\n";
91  inout(tend,"tend");      inout(dx,"dx");
92  inout(n_out,"n_out");    inout(n_interm,"n_interm");
93  inout(alpha,"alpha");    inout(p_0,"p_0");
94  inout(rtol,"rtol");      inout(atol,"atol");
95  inout(a_opt,"a_opt");    inout(b_opt,"b_opt");
96  in_stream.close();
97
98  //-- Berechnung einiger benoetigter Parameter --
99  dt = tend / n_out / n_interm;
100 dp = 2 * pi / dx / nmax;
101
102 //-- Berechnung des optischen Potentials
103 for (n=0; n<nmax; n++)
104   {
105   if (n < nmax/2) x = n * dx;
106               else x = (nmax-n) * dx;
107   V[n] = a_opt*exp( -sqr(b_opt*x) );
108   }
109
110 //-- Anfangsbedingungen
111 for (n=0; n<nmax; n++)
112   {
113   x = (n-nmax/2) * dx;
114   psi[2*n] = exp(-sqr(x/alpha)/2);
115   sum  = sum + sqr(psi[2*n]);
116   }
117 sum = 1 / sqrt(sum);
118 for (n=0; n<nmax; n++)
119   {
120   x = (n-nmax/2) * dx;
121   psi[2*n+1] = -psi[2*n] * sum * sin(p_0*x);   // Imaginaerteil
122   psi[2*n]   =  psi[2*n] * sum * cos(p_0*x);   // Realteil
123   }
124
125 t = 0;
126 out_stream << n*dt << "\n";
127 for (n1=0; n1<nmax; n1++)
128   out_stream << psi[2*n1] << " " << psi[2*n1+1] << "\n";
129
130 //-- Zeitschleife --
131 t = 0;
132 for (n=1; n<=n_out; n++)
133   {
134   for (n1=0; n1<n_interm; n1++) status = timestep(dt);
135   cout << n << " " << n_out << "\n";
136   out_stream << n*dt << "\n";
137   for (n1=0; n1<nmax; n1++)
138     out_stream << psi[2*n1] << " " << psi[2*n1+1] << "\n";
139   }
140
141 out_stream.close();
142 }
```

Zu den bisherigen Parametern kommen noch die Höhe `a_opt` und die inverse Breite `b_opt` des (Gaußschen) optischen Potentials hinzu, sowie `n_interm`, das angibt, in wieviele intermediäre Zeitschritte jedes Zeitinter-

vall zwischen zwei Zeiten, zu denen der Zustand abgespeichert wird, eingeteilt wird. Bei der Wahl der beiden Parameter `a_opt` und `b_opt` ist ein wenig Fingerspitzengefühl gefragt:

- Ein zu kleines `b_opt` schränkt den Bereich ein, in dem sich das Teilchen tatsächlich wie ein freies Teilchen verhält.
- Ein zu großes `b_opt` hingegen wirkt wie eine Wand: das Teilchen dringt nur zu einem Teil in den Potentialwall ein, wo es absorbiert wird, während der Rest reflektiert wird.
- Ein zu großes `a_opt` hat ebenfalls den Effekt, dass ein nennenswerter Teil der Wellenfunktion am optischen Potential reflektiert wird, anstatt einzudringen.
- Ein zu kleines `a_opt` schließlich erlaubt dem Teilchen, das Potential zu durchdringen, und so auf der anderen Seite des Ortsbereichs wieder aufzutauchen.

Zur Verdeutlichung illustrieren wir die Zeitentwicklung für einige Parameter von `a_opt` in Abb. 5.2. Sie sehen, dass in der oberen Reihe der Wert von

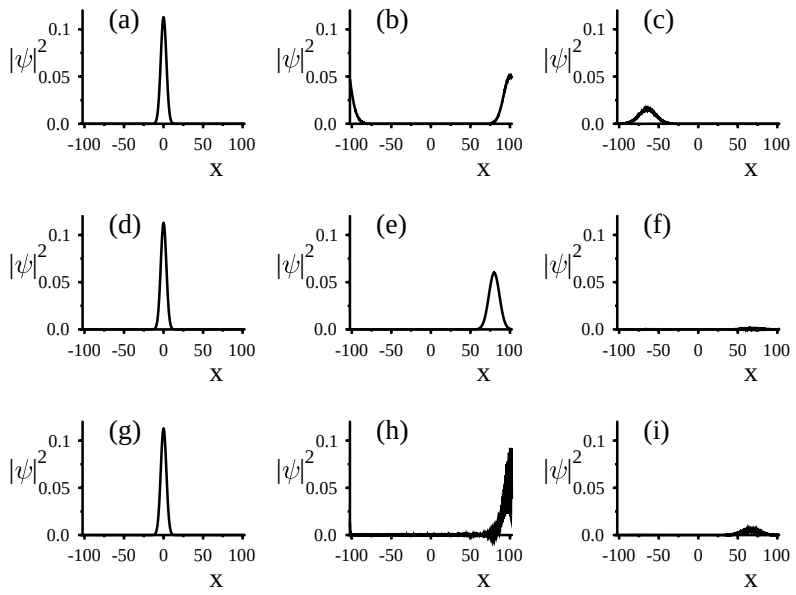

Abb. 5.2. Zeitentwicklung eines Wellenpakets und dessen Absorption sowie Reflektion an einem optischen Potential, das an den beiden Enden des dargestellten Ortsbereichs lokalisiert ist. In der oberen Bildreihe ist `a_opt` = 0.1, in der mittleren Reihe ist `a_opt` = 0.8 und in der unteren Reihe ist `a_opt` = 3. Gezeigt wird jeweils der Ausgangszustand des Wellenpakets (linkes Bild), das Wellenpaket im Bereich des optischen Potentials (mittleres Bild) und das Wellenpaket nach Durchlaufen des optischen Potentials (rechtes Bild)

`a_opt` zu klein ist – das Teilchen durchdringt rechts den Bereich des optischen Potentials, erreicht den Rand und taucht auf der linken Seite auf. In der mittleren Reihe ist der Parameter `a_opt` so gewählt, dass das Teilchen das optische Potential nicht durchdringen kann, aber auch nicht an demselben reflektiert wird. In der unteren Reihe schließlich ist `a_opt` zu groß gewählt, so dass das Wellenpaket nur zu einem Teil absorbiert wird, während der verbleibende Teil reflektiert wird.

Ein vorteilhafter Aspekt des nun entwickelten Programms `frei3.cpp` ist, dass es durch minimale Modifikationen auch die Entwicklung einer Wellenfunktion in einem beliebigen und sogar zeitabhängigen Potential berechnen kann. Dazu müssen lediglich die Zeilen 56–62 auf ein reelles Potential angepasst und die Zeilen 102–105 entsprechend geändert werden. Dies werden wir uns im übernächsten Abschnitt zu Nutze machen, vorher jedoch müssen wir noch lernen, wie wir zu einem vorgegebenen Potential die zugehörigen Eigenzustände und deren Energieniveaus berechnen können.

5.9 Eigenzustände des Hamiltonoperators

Die zeitabhängige Schrödingergleichung (5.38) ist besonders leicht zu lösen, wenn wir als Anfangszustand einen Eigenzustand $|\psi(t=0)\rangle = |E\rangle$ des Hamiltonoperators H wählen, d.h. einen Zustand für den

$$H|E\rangle = E|E\rangle \tag{5.56}$$

gilt. In diesem Fall ist der Zustandsvektor zu einem beliebigen Zeitpunkt t durch

$$|\psi(t)\rangle = \exp(-\mathrm{i}Et)|E\rangle \tag{5.57}$$

gegeben. Die Zeitentwicklung äußert sich also lediglich in einem skalaren Faktor vom Betrag 1. Das bedeutet auch, dass sich der Erwartungswert eines beliebigen, nicht explizit zeitabhängigen Operators nicht mit der Zeit ändert, sondern konstant bleibt:

$$\langle\psi(t)|\hat{O}|\psi(t)\rangle = \exp(\mathrm{i}Et)\langle E|\hat{O}|E\rangle \exp(\mathrm{i}Et) = \langle E|\hat{O}|E\rangle \ . \tag{5.58}$$

Sie sehen also, dass den Eigenzuständen des Hamiltonoperators eine besondere Bedeutung zukommt. In diesem Abschnitt wollen wir uns deshalb mit der numerischen Berechnung dieser Eigenzustände bei beliebig vorgegebenem äußeren Potential beschäftigen.

Die Grundlage dafür haben wir bereits in (5.47) gelegt, denn wenn wir die Dreipunktsformel für die zweite Ableitung in $\hat{H}$ einsetzen, erhalten wir einen diskretsieren Hamiltonoperator in Matrixform:

$$H_{n,n} = \frac{1}{(\Delta x)^2} + V(x) \tag{5.59}$$

$$H_{n,n-1} = H_{n,n+1} = -\frac{1}{2}\frac{1}{(\Delta x)^2} \ . \tag{5.60}$$

Um die Eigenzustände des Hamiltonoperators zu erhalten, müssen wir also die Eigenvektoren dieser Matrix berechnen, wofür zum Glück in jeder Standardbibliothek fertige Routinen zur Verfügung stehen. Betrachten wir das Programm `eigen1.cpp`:

```
/*************************************************************************
* Name:      eigen1.cpp                                                  *
* Zweck:     Berechnet Eigenzustaende zu vorgegebenem Potential          *
* Gleichung: Schroedingergleichung                                       *
* verwendete Bibiliothek: GSL                                            *
*************************************************************************/

#include <iostream>
#include <fstream>
#include <math.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>
#include "tools.h"

using namespace std;

//-- Zahl der Stuetzstellen
const int nmax = 512;

//------------------------------------------------------------------------

main( int argc, char *argv[] )

{
//-- Definition der Variablen
int    n, n1, n2;
double x, dx, x2[nmax], psi[nmax], V[nmax];
ifstream in_stream;
ofstream out_stream;

//-- Fehlermeldung, wenn Input- und Outputfilename nicht uebergeben wurden
if (argc<4)
{
cout << " Aufruf: eigen1 infile outfile1 outfile2\n";
cout << "         outfile1 enthaelt das Potential\n";
cout << "         outfile2 enthaelt die Eigenwerte und Eigenzustaende\n";
return 0;
}

//-- Einlesen der Parameter --
in_stream.open(argv[1]);
out_stream.open(argv[2]);
out_stream << "! Ergebnis-Datei generiert von eigen1.cpp\n";
inout(dx,"dx");
in_stream.close();
out_stream << "! Spalte 1: n    Spalte 2: x_n    Spalte 3: V(x_n)\n";

//-- Berechnung des reellen Potentials
for (n=0; n<nmax; n++)
  {
  x = (n-nmax/2) * dx;
  V[n] = 0.5 * sqr(x);
  out_stream << n << " " << x << " " << V[n] << "\n";
  }
```

```
55 out_stream.close();
56
57 out_stream.open(argv[3]);
58 out_stream << "! Ergebnis-Datei generiert von eigen1.cpp\n";
59 out_stream << "! nmax = " << nmax << "\n";
60 out_stream << "! dx = " << dx << "\n";
61 out_stream << "! Erster Teil:  Spalte 1: n  Spalte 2: E_n"
62    << "  Spalte 3: <x^2>_n\n";
63 out_stream << "! Zweiter Teil: Spalte 1: n  Spalte 2: x"
64    << "    Spalte 3: psi_n(x)\n";
65
66 //-- Speicherplatz allokieren
67 gsl_vector *eval = gsl_vector_alloc(nmax);
68 gsl_matrix *evec = gsl_matrix_alloc(nmax, nmax);
69 gsl_matrix *H    = gsl_matrix_alloc(nmax, nmax);
70 gsl_eigen_symmv_workspace * w = gsl_eigen_symmv_alloc(nmax);
71
72 //-- Hamiltonoperator konstruieren
73 for (n1=0; n1<nmax; n1++)
74   for (n2=0; n2<nmax; n2++)
75     gsl_matrix_set(H, n1, n2, 0);
76
77 for (n1=0; n1<nmax; n1++)
78   gsl_matrix_set(H, n1, n1, 1/sqr(dx) + V[n1]);
79 for (n1=1; n1<nmax; n1++)
80   {
81   gsl_matrix_set(H, n1-1, n1, -0.5/sqr(dx));
82   gsl_matrix_set(H, n1, n1-1, -0.5/sqr(dx));
83   }
84 for (n1=0; n1<nmax-1; n1++)
85   {
86   gsl_matrix_set(H, n1+1, n1, -0.5/sqr(dx));
87   gsl_matrix_set(H, n1, n1+1, -0.5/sqr(dx));
88   }
89
90 //-- Eigenwerte und Eigenvektoren berechnen und sortieren
91 gsl_eigen_symmv(H, eval, evec, w);
92 gsl_eigen_symmv_free(w);
93 gsl_eigen_symmv_sort (eval, evec, GSL_EIGEN_SORT_VAL_ASC);
94
95 //-- Berechnung von <x*x>
96 for (n1=0; n1<nmax; n1++)
97   {
98   x2[n1] = 0;
99   for (n2=0; n2<nmax; n2++)
100    {
101    x = (n2-nmax/2) * dx;
102    x2[n1] = x2[n1] + sqr(gsl_matrix_get(evec,n2,n1)*x);
103    }
104  }
105 //-- Ausgabe der Eigenwerte und Eigenvektoren
106 for (n1=0; n1<nmax; n1++)
107   out_stream << n1 << " " << gsl_vector_get(eval, n1) << " "
108              << x2[n1] << "\n";
109 for (n1=0; n1<nmax; n1++)
110   for (n2=0; n2<nmax; n2++)
111     {
112     x = (n2-nmax/2) * dx;
113     out_stream << n1 << " " << x << " "
```

```
114                 << gsl_matrix_get(evec, n2, n1) << "\n";
115     }
116
117 gsl_vector_free(eval);
118 gsl_matrix_free(evec);
119 gsl_matrix_free(H);
120
121 out_stream.close();
122 }
```

Nach dem Einlesen der Parameter wird das Potential $V(x)$ an den Stützstellen berechnet (Zeile 48–54) – das ist die einzige Stelle, die wir modifizieren müssen, wenn wir die Eigenzustände zu anderen Potentialtypen berechnen wollen. Danach wird Speicherplatz für die Matrix des Hamiltonoperators, für deren Eigenwerte und Eigenvektoren, sowie für die Berechnung dieser Eigenvektoren alloziert. Dann erfolgt die Aufstellung des Hamiltonoperators nach (5.59), die Berechnung der Eigenwerte und -vektoren, sowie die Ausgabe der Ergebnisse.

Zur Überprüfung des Programms eignen sich besonders Potentiale, bei denen die Eigenzustände analytisch berechnet werden können, z.B. das Potential des harmonischen Oszillators:

$$V(x) = \frac{1}{2} m\omega^2 x^2 \; . \tag{5.61}$$

Dessen Eigenzustände finden sich in jedem Lehrbuch zur Quantenmechanik und lauten:

$$\psi_n(x) = N_n \exp\left(-\frac{1}{2}k^2x^2\right) H_n(k^2x) \tag{5.62}$$

mit

$$k^2 = \frac{m\omega}{\hbar} \tag{5.63}$$

$$N_n = \sqrt{\frac{k^2}{2^n n! \sqrt{\pi}}} \; . \tag{5.64}$$

Die dabei auftretenden Hermiteschen Polynome H_n geben wir für die niedrigsten Ordnungen n explizit an:

$$H_0(\xi) = 1 \tag{5.65}$$

$$H_1(\xi) = 2\xi \tag{5.66}$$

$$H_2(\xi) = 4\xi^2 - 2 \; . \tag{5.67}$$

Die zu diesen Eigenzuständen gehörenden Energieniveaus sind:

$$E_n = \hbar\omega \left(n + \frac{1}{2}\right) \; . \tag{5.68}$$

Tabelle 5.1. Numerisch bestimmte Energieeigenwerte des harmonischen Oszillators

n	E	n	E
0	0.49999	7	7.49985
1	1.49994	8	8.50359
2	2.49984	9	9.51658
3	3.49969	10	10.5515
4	4.49949	11	11.6277
5	5.49927	12	12.7666
6	6.49918	13	13.9861

Dieses analytische Ergebnis können wir nun mit dem numerischen Resultat von `eigen1.cpp` vergleichen. Für `nmax`=512 und `dx` = 0.02 erhalten wir die Eigenwerte:

Die Energieniveaus werden also für diese Parameter bis etwa $n = 10$ ganz gut wiedergegeben und weichen dann immer mehr von ihrem Sollwert $n+1/2$ ab. Was bei diesem Wert geschieht sehen wir, wenn wir die zugehörigen Eigenfunktionen betrachten (Abb. 5.3):

Wir sehen, dass die Wellenfunktion ψ für kleine n gut beim Ursprung lokalisiert ist, ihre Ausdehnung aber mit wachsendem n zunimmt. Für die-

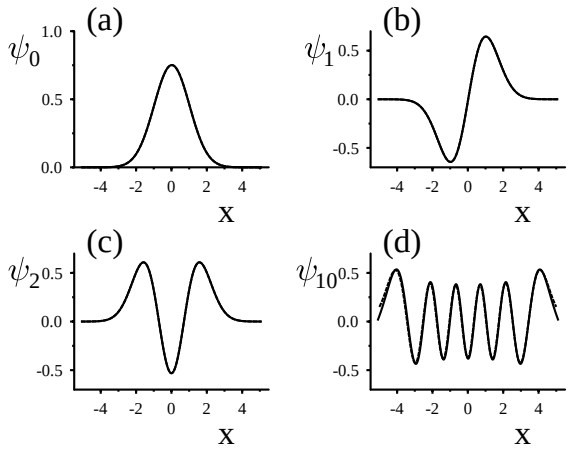

Abb. 5.3. Eigenzustände des harmonischen Oszillators: Vergleich des numerisch gewonnenen Ergebnisses (durchgezogene Linie) mit dem analytischen Ergebnis (gestrichelt) für $n = 0$, $n = 1$, $n = 2$ und $n = 10$. Nur bei $n = 10$ kann ein kleiner Unterschied zwischen diesen beiden Kurven ausgemacht werden

se n ist in Abb. 5.3 das numerische vom analytischen Ergebnis nicht mehr zu trennen. Bei etwa $n = 10$ erreicht die Wellenfunktion den Rand des abgedeckten Ortsbereichs, so dass verständlich ist, dass für höhere n unsere numerische Berechnung nicht mehr richtig sein kann. Trotzdem ist die Übereinstimmung der gewonnenen Eigenfunktion (in Abb. 5.3 rechts unten) mit dem analyitschen Resultat (5.62) erstaunlich.

Nachdem wir nun verifiziert haben, dass das Programm korrekt funktioniert, ist es verhältnismäßig einfach, das Potential beliebig zu variieren und jeweils Energieniveaus und die zugehörigen Eigenfunktionen zu berechnen. Da wir für den übernächsten Abschnitt die Eigenzustände zum Potential

$$V(x) = -a \exp\left(-b(x - x_0)^2\right) \tag{5.69}$$

benötigen, werden wir die Eigenzustände dieses Potentials kurz diskutieren – darüber hinaus kann dieses Potential auch gut als eindimensionales Modell eines Atoms herangezogen werden.

Das Programm zur Berechnung der Eigenzustände finden Sie auf der CD als `eigen2.cpp`. Wie schon oben erwähnt müssen nur die Zeilen geändert werden, in denen das Potential V berechnet wird:

```
...
55 //-- Berechnung des reellen Potentials
56 for (n=0; n<nmax; n++)
57 {
58 x = (n-nmax/2) * dx;
59 V[n] = -a*exp(-b*sqr(x-x0));
60 }

...
```

Wenn wir die Eigenzustände und zugehörigen Eigenwerte zu diesem Potential für

$$a = 1.5 \qquad b = 0.5 \qquad x_0 = 0 \tag{5.70}$$

berechnen, stellen wir zunächst fest, dass das Spektrum der Eigenwerte aus zwei negativen und sehr vielen positiven Werten besteht. Die negativen Energieeigenwerte gehören zu gebundenen Zuständen – d.h. die Energie des Elektrons reicht nicht aus, den Atomkern zu verlassen – während die positiven Energien zu ungebundenen Zuständen gehören. Diese Aussage können wir auf zwei Weisen untermauern: Zum einen können wir die Eigenzustände direkt anschauen und werden sehen, dass tatsächlich die Eigenzustände zu den negativen Energien am Potential lokalisiert sind, was bei den übrigen Eigenzuständen nicht zutrifft. Da wir dies aber nicht für alle Eigenzustände (in unserem Beispiel immerhin 512 Stück) durchführen können, können wir alternativ für jeden Eigenzustand den Mittelwert $\langle x^2 \rangle$ berechnen und ausgeben lassen. Die beiden gebundenen Zustände haben einen kleinen Wert von $\langle x^2 \rangle$, der sich zudem bei einer Vergrößerung des betrachteten Ortsbereichs nur wenig ändert, während diese Größe bei den freien Zuständen sehr groß ist und mit anwachsendem Ortsbereich divergiert.

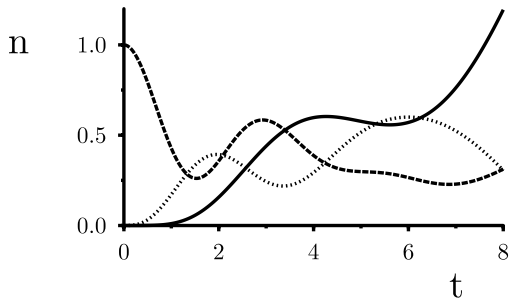

Abb. 5.13. Besetzung der Niveaus 1 und 3 (gestrichelte bzw. gepunktete Linie), eines Drei-Niveau-Atoms, das an ein Lichtfeld gekoppelt ist, sowie die Photonenzahl im Lichtfeld (durchgezogene Kurve)

bei einem Ein-Atom-Laser nicht möglich, da sich das einzige Atom nach der ersten Emission im Grundzustand befindet. Man hat also guten Grund, anzunehmen, dass die Statistik der Photonen beim Ein-Atom-Laser grundlegend anders ist als beim konventionellen Laser. Um dies genauer zu beleuchten, müssten wir leider tief in die Quantenoptik eintauchen – den Leser, der über entsprechende Kenntnisse verfügt, verweisen wir auf Übung 5.7.

Übungen

5.1 Eigenzustände des Delta-Potentials

Die Eigenzustände zu einem δ-förmigen Potential $V(x) = a\delta(x)$ können analytisch bestimmt werden. Gerade deswegen ist es aber ein gute Übung, diese numerisch zu berechnen und mit dem analytischen Ergebnis zu vergleichen. Verwenden Sie als Ausgangspunkt das Programm `eigen1.cpp` aus Abschnitt 5.9 und modifizieren Sie den Programmabschnitt, in dem das Potential festgelegt wird. Studieren Sie insbesondere die Abhängigkeit des Ergebnisses von den Parametern `xmax` und `dx`.

5.2 Eindimensionales Modell eines Atoms

Das Potential $V(x) = \beta/\sqrt{1+\alpha x^2}$ kann als einfaches Modell für ein Atom dienen, wobei die Parameter α und β die Breite und die Tiefe des vom Atomkern erzeugten Potentials festlegen. Berechnen Sie für einige Parametersätze die gebundenen Zustände und deren Bindungsenergien. Letztere können Sie mit den Ergebnissen aus Abschn. 5.10 vergleichen. Interessant ist es auch, die freien Zustände für niedrige und für große Energien zu betrachten.

5.3 Eindimensionales Modell für H_2^+

Fügen Sie zwei der Einzelpotentiale aus der vorangegangenen Aufgabe zum Potential zweier Atomkerne im Abstand a zusammen und berechnen Sie wieder die Energieniveaus der gebundenen Zustände. Variieren Sie nun den Abstand a und bestimmen Sie die Grundzustandsenergie als Funktion dieses Parameters. Was Sie erhalten ist das effektive Potential, in dem sich (aufgrund der großen Masse der Atomkerne auf einer viel größeren Zeitskala) die beiden Kerne bewegen. Aus dem Potentialverlauf kann also sowohl der Abstand der beiden Kerne im Grundzustand als auch deren Schwingungsfrequenz entnommen werden.

5.4 Berechnung von Energieniveaus nach der Variationsmethode

In Abschn. 5.10 haben wir die Eigenenergien eines Modellatoms (siehe auch Aufgabe 5.2) nach der Variationsmethode berechnet. Als Testfunktionen dienten uns dabei Funktionen vom Typ

$$f_n = \frac{1}{\sqrt{n!}}\, x^n \exp(-\mu x^2/2)\ , \tag{5.167}$$

wobei der Parameter μ fest gewählt wurde. Untersuchen Sie, was passiert, wenn Sie diesen Parameter verändern – insbesondere, wenn Sie ein sehr großes oder ein sehr kleines μ nehmen. Interessant ist es auch, mehrere solche Funktionensätze zu verschiedenen μ zu kombinieren und so einen größeren Satz an Testfunktionen zu haben, aus denen man die Eigenzustände und die dazugehörigen Eigenenergien berechnet.

5.5 Quantenmechanische Entropie

Überlegen Sie sich, wie groß die in (5.120) definierte Entropie bei einem Zwei-Niveau- bzw. Drei-Niveau-Atom werden kann. Implementieren Sie nun eine Berechnung der Entropie in die Programme `zweiniveau.cpp` sowie `dreiniveau.cpp` und untersuchen Sie den Zeitverlauf der Entropie.

5.6 Zeno-Effekt

Wenn Sie bei den Parametern zum Programm `zeno2.cpp` (auf der CD-ROM) die Zahl der Messungen im Zeitintervall $[0:100]$ auf 20 herabsetzen, werden Sie feststellen, dass die Abnahme der Besetzung des angeregten Zustandes wieder langsamer erfolgt – entgegen dem allgemeinen Trend. Woran liegt das? Betrachten Sie zum Vergleich noch einmal Abb. 5.8!

5.7 Ein-Atom-Laser

Durch Spurbildung über die atomaren Freiheitsgrade kommen Sie von der Dichtematrix $\varrho_{i,n;j,m}$ aus Abschn. 5.16 auf eine Dichtematrix nur für das Atom:

$$\varrho_{n;m} = \sum_i \varrho_{i,n;i,m} \ . \tag{5.168}$$

Aus dieser können Leser mit Kenntnissen in Quantenoptik (siehe z.B. [34]) die Quasiwahrscheinlichkeit

$$Q(\alpha) = \frac{1}{\pi}\langle\alpha|\varrho|\alpha\rangle \tag{5.169}$$

berechnen. Dabei machen wir Gebrauch von den kohärenten Zuständen

$$|\alpha\rangle = \exp\left(-\frac{1}{2}|\alpha|^2\right)\sum_n \frac{\alpha^n}{\sqrt{n!}}|n\rangle \ . \tag{5.170}$$