Computational Physics II

Andreas Honecker

Institut für Theoretische Physik

Technische Universität Braunschweig

Sommersemester 2005

`http://www.tu-bs.de/~honecker/cphys2`

Organisatorische Bemerkungen

- ☞ Die Übungen finden Freitags 14:00-15:30 in HS 65.2 des Rechenzentrums statt.
- ☞ Bitte besorgen Sie sich hierzu einen Account am Rechenzentrum (Y-Nummer), falls noch nicht vorhanden !
- ☞ Sie lernen nur durch Mitmachen – scheuen Sie sich also nicht vor Fragen !
- ☞ Die neueste Version dieses Skripts finden Sie immer auf der Web-Seite

  `http://www.tu-bs.de/~honecker/cphys2`

  Es wird im Verlauf der Vorlesung aktualisiert, so daß sich ein früher Ausdruck nicht unbedingt empfiehlt.
- ☞ Ein wesentlicher Bestandteil des Skripts sind die Übungsaufgaben. Aufgaben, die nicht wesentlich sind, werden mit Sternchen * markiert (je mehr Sternchen, desto anspruchsvoller bzw. unwichtiger). Alle anderen Aufgaben sollte jeder bearbeiten und (mit Hilfe) lösen.

Vorbemerkungen

„Computational Physics" ist ein vergleichweises neues Teilgebiet der Physik, das manchmal neben Experiment und Theorie als ein drittes Standbein der Physik bezeichnet wird [1]. Zu betonen ist jedoch, daß Computational Physics als Bestandteil der Physik das Verständnis der Natur zum Ziel hat, wofür in diesem Fall eine numerische Methode verwendet wird. Eine grafische Darstellung ist oft unabdingbar, jedoch sollten bunte Bilder nicht mit Einsicht verwechselt werden. Auch kommt man ohne analytische Hilfsmittel nicht aus, um zum Beispiel die Probleme für eine Bearbeitung mit dem Computer geeignet auzubereiten, oder um die korrekte Funktion der Programme zu prüfen.

Da Computational Physics inzwischen ein breites Gebiet ist, ist eine Themenauswahl unumgänglich. Methodisch soll im folgenden der Schwerpunkt auf eine numerische Behandlung partieller Differentialgleichungen gelegt werden. Inhaltlich ergeben sich somit Themen aus der Hydrodynamik, Quantenmechanik, oder z.B. der Elektrodynamik.

0 Partielle Differentialgleichungen im Rechner

Partielle Differentialgleichungen müssen für eine numerische Behandlung geeignet aufbereitet werden. Im folgenden werden verschiedene Möglichkeiten angerissen bzw. ausführlicher diskutiert.

0.1 Entwicklung nach Funktionensystemen

Eine Möglichkeit ist die Entwicklung nach geeigneten Funktionensystemen. So bietet sich z.B. für periodische Systeme die Fourier-Transformation bzw. -Reihe an, die dann für die numerische Behandlung bei hohen Frequenzen abgeschnitten wird.

Auch allgemeinere orthogonale Funktionensystem können je nach Fragestellung verwendet werden. Ein Beispiel für partielle Differentialgleichungen liefert die Schrödingergleichung für einfache Atome bzw. Moleküle. Das Helium-Atom ist z.B. gegeben durch den Hamilton-Operator

$$\mathcal{H} = \mathcal{H}_{H,1}^{Z=2} + \mathcal{H}_{H,2}^{Z=2} + \mathcal{H}_{1,2}^{C}\,, \tag{0.1}$$

wobei $\mathcal{H}_{i,1}^{Z=2}$ ($i = 1,\ 2$) für beide Elektronen jeweils ein Zentralproblem mit Kernladungszahl $Z = 2$ beschreibt, und $\mathcal{H}_{1,2}^{C}$ die Coulomb-Wechselwirkung der beiden Elektronen darstellt. Der Hamilton-Operator (0.1) kann nach wasserstoffartigen Wellenfunktionen

$$\Psi_{n,l,m}(\vec{x}_1)\Psi_{n',l',m'}(\vec{x}_2) - \Psi_{n,l,m}(\vec{x}_2)\Psi_{n',l',m'}(\vec{x}_1) \tag{0.2}$$

entwickelt werden, wobei das Minuszeichen das Pauli-Prinzip für die beiden identischen Elektronen berücksichtigt. In der Basis (0.2) ist $\mathcal{H}_{H,1}^{Z=2} + \mathcal{H}_{H,2}^{Z=2}$ diagonal. Beschränkt man sich nun für beide Elektronen jeweils auf die N niedrigsten $\Psi_{n,l,m}$, so sind $\mathcal{H}_{1,2}^{C}$ und $\mathcal{H}$ in (0.1) Matrizen der Größe $\frac{N(N-1)}{2} \times \frac{N(N-1)}{2}$. Diese können dann numerisch behandelt werden, allerdings führt dies bereits für ein moderates $N = 40$ bereits auf 780×780 Matrizen.

Manchmal bietet sich auch die Entwicklung nach nicht-orthogonalen Funktionensystemen an, so z.B. beim H_2-Molekül mit dem Hamilton-Operator

$$\mathcal{H} = \mathcal{H}_{H,\vec{r}_1}^{Z=1} + \mathcal{H}_{H,\vec{r}_2}^{Z=1} + \mathcal{H}_{1,2}^{C}\,. \tag{0.3}$$

Hier stehen nun $\mathcal{H}_{H,\vec{r}_i}^{Z=1}$ für Zentral-Probleme bzgl. der beiden Wasserstoff-Kerne, die bei $\vec{r}_i$ ($i = 1,\ 2$) als fest angenommen werden. $\mathcal{H}_{1,2}^{C}$ beschreibt

wieder die Coulomb-Wechselwirkung zwischen den beiden Elektronen. Auch hier liegt es nahe, von wasserstoffartigen Wellenfunktionen $\Psi_{n,l,m}(\vec{x}-\vec{r}_1)$ und $\Psi_{n,l,m}(\vec{x}-\vec{r}_2)$ auszugehen. Nun ist aber zu beachten, daß die an verschiedenen Kernen lokalisierten Wellenfunktionen nicht zu einander orthogonal sind. Dies erschwert bereits die Berechnung der Matrix-Elemente von (0.3) in einer solchen Basis. Dennoch bilden solche Überlegungen die Grundlagen für die quantenmechanische Berechnung von Molekülen bzw. Festkörpern (siehe z.B. [2, 3]). Auch wenn dies hier nicht weiter verfolgt werden soll, sei angemerkt, daß die Betrachtung von nur 5 Elektronen in jeweils etwa 16 Orbitalen bereits auf Probleme der Größenordnung $16^5 = 2^{20} \approx 10^6$ führt.

0.2 Diskretisierung der Variablen

Eine andere Vorgehensweise basiert auf der Betrachtung der Funktion $f(\vec{x})$ an Stützstellen $\vec{x}_i$. Solche Stützstellen können recht allgemein gewählt werden, wodurch insbesondere bei adaptiver Wahl auch eine gute Anpassung an das Problem möglich ist. Allerdings ist eine allgemeine Stützstellen-Verteilung schwer zu handhaben.

Im folgenden soll ein reguläres Gitter von Punkten zur Diskretisierung einer partiellen Differentialgleichung für die Funktion $f(\vec{x})$ verwendet werden. Als Gitter wählen wir ein quadratisches bzw. (hyper-)kubisches Gitter

$$\vec{x}_{\vec{r}} = \vec{x}_0 + \sum_{i=1}^{d} \Delta x_i \; r_i \; \vec{e}_i \tag{0.4}$$

mit ganzzahligen r_i. Für die Funktionswerte an diesen Punkten schreiben wir

$$f_{\vec{r}} = f\left(\vec{x}_{\vec{r}}\right) . \tag{0.5}$$

Wir betrachten nun die partiellen Ableitungen $\frac{\partial^m}{\partial x_i^m} f(\vec{x})$ und tun der Einfachheit halber so, als ob dies die einzige Variable wäre. Für die erste partielle Ableitung an der Stelle $\vec{x}_r$ liegen zwei Definitionen nahe:

$$\begin{aligned} f_{x_i}(\vec{x}_r) = \frac{\partial}{\partial x_i} f(\vec{x}_r) &= \frac{f_r - f_{r-1}}{\Delta x_i} + \mathcal{O}(\Delta x_i) \\ &= \frac{f_{r+1} - f_r}{\Delta x_i} + \mathcal{O}(\Delta x_i) . \end{aligned} \tag{0.6}$$

Durch eine geschickte symmetrische Kombination der beiden Möglichkeiten (0.6) kann man die Ordnung verbessern:

$$f_{x_i}(\vec{x}_r) = \frac{f_{r+1} - f_{r-1}}{2\,\Delta x_i} + \mathcal{O}(\Delta x_i^2) . \tag{0.7}$$

Analog kann man die zweite partielle Ableitung konstruieren. Eine geschickte Kombination ist

$$f_{x_i x_i}(\vec{x}_r) = \frac{\partial^2}{\partial x_i^2} f(\vec{x}_r) = \frac{f_{r+1} - 2\, f_r + f_{r-1}}{\Delta x_i^2} + \mathcal{O}(\Delta x_i^2)\,. \tag{0.8}$$

Wir müssen nun noch unsere Randbedingungen unterbringen. Bei *Dirichletschen Randbedingungen* ist die Funktion f auf dem Rand des zu untersuchenden Gebiets bekannt. Dem läßt sich leicht Rechnung tragen, indem man für die Ableitungen direkt neben dem Rand in (0.7) oder (0.8) einfach für die entsprechenden $f_{r\pm1}$ die vorgegebenen Werte einsetzt. Bei *Neumannschen Randbedingungen* ist hingegen die Ableitung der Funktion f in Normalenrichtung des Randes auf diesem bekannt. Auch dies läßt sich unterbringen, z.B. indem man $\frac{f_r - f_{r-1}}{\Delta x_i}$ in (0.8) an den entsprechenden Randpunkten durch die vorgegebene Funktion ersetzt.

Krummlinige Koordinatensystemen lassen sich durch Variablentransformationen mit der geschilderten Vorgehensweise bearbeiten. So würde man z.B. in Kugelkoordinaten r, θ, ϕ diese Variablen jeweils äquidistant diskretisieren.

0.3 Potentialproblem

Die Diskretisierung (0.8) führt auf einem d-dimensionalen hyperkubischen Gitter mit Gitterabstand $\Delta x_i = \Delta x$ zu folgender Diskretisierung des Laplace-Operators

$$\begin{aligned} \Delta\, \Phi\left(\vec{x}_{\vec{r}}\right) &= \frac{1}{\Delta x^2} \left(-2d\, \Phi\left(\vec{x}_{\vec{r}}\right) + \sum_{i=1}^{d} \left[\Phi\left(\vec{x}_{\vec{r}} + \Delta x\, \vec{e}_i\right) + \Phi\left(\vec{x}_{\vec{r}} - \Delta x\, \vec{e}_i\right) \right] \right) \\ &\quad + \mathcal{O}(\Delta x^2)\,. \end{aligned} \tag{0.9}$$

Die Laplace-Gleichung

$$\Delta\, \Phi = 0 \tag{0.10}$$

ist einen bekannte Gleichung. In der Elektrodynamik ist Φ z.B. ein elektrostatisches Potential. Die Gleichung (0.10) beschreibt für $d = 2$ auch Minimalflächen auf einem endlichen Gebiet, wobei Φ nun als dritte Koordinate interpretiert werden kann.

Die Diskretisierung (0.9) führt auf folgende Gitter-Variante der Laplace-Gleichung (0.10)

$$\Phi\left(\vec{x}_{\vec{r}}\right) = \frac{1}{2\, d} \sum_{i=1}^{d} \left\{ \Phi\left(\vec{x}_{\vec{r}} + \Delta x\, \vec{e}_i\right) + \Phi\left(\vec{x}_{\vec{r}} - \Delta x\, \vec{e}_i\right) \right\} + \mathcal{O}(\Delta x^4)\,. \tag{0.11}$$

Dies bedeutet, daß das Potential am Platz $\vec{x}_{\vec{r}}$ gleich dem Mittelwert über seine Werte an den $2\,d$ Nachbarplätzen ist !

Diese Beobachtung legt ein einfaches Lösungsverfahren für die Laplace-Gleichung (0.10) nahe, das unter dem Namen Jacobi-Verfahren bekannt ist [1, 4]:

1. Wähle eine beliebige Start-Potential-Konfiguration $\Phi\left(\vec{x}_{\vec{r}}\right)$, die allerdings die geforderten Randbedingungen erfüllt.

2. Berechne

$$\Phi_{\text{neu}}\left(\vec{x}_{\vec{r}}\right) = \frac{1}{2\,d}\sum_{i=1}^{d}\left\{\Phi\left(\vec{x}_{\vec{r}} + \Delta x\,\vec{e}_i\right) + \Phi\left(\vec{x}_{\vec{r}} - \Delta x\,\vec{e}_i\right)\right\}\,, \tag{0.12}$$

   bzw. bestimme $\Phi_{\text{neu}}\left(\vec{x}_{\vec{r}}\right)$ für Randplätze $\vec{x}_{\vec{r}}$ aus den Randbedingungen.

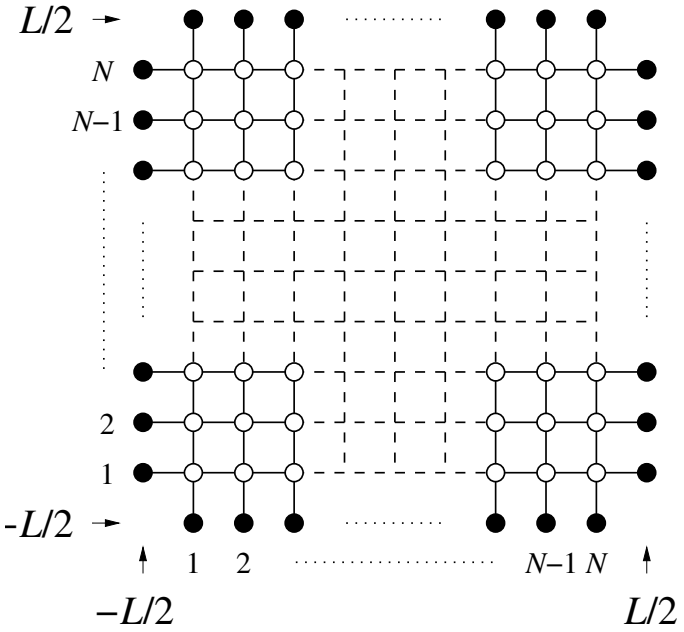
3. Berechne

$$\delta\Phi = \max_{\vec{x}_{\vec{r}}}\left|\Phi_{\text{neu}}\left(\vec{x}_{\vec{r}}\right) - \Phi\left(\vec{x}_{\vec{r}}\right)\right|\,. \tag{0.13}$$

4. Ersetze $\Phi\left(\vec{x}_{\vec{r}}\right)$ durch $\Phi_{\text{neu}}\left(\vec{x}_{\vec{r}}\right)$.

5. Fahre bei 2. fort, wenn $\delta\Phi$ eine vorgegebene Schwelle überschreitet. Andernfalls höre auf.

0.4 Aufgaben



A0.1 Wir betrachten die Laplace-Gleichung (0.10) auf einem Quadrat der Größe $L \times L$. Das Potential sei auf dem Rand vorgegeben (Dirichletsche Randbedingungen). Zur Diskretisierung verwenden wir das oben skizzierte Gitter.
Schreiben Sie ein Programm, das das allgemeine Randwertproblem (d.h. allgemeine Randbedingungen) mit dem Jacobi-Verfahren für die $N \times N$ inneren Gitterpunkte löst ! Am besten verwenden Sie C++ (zwecks späterer Übersetzung in Java – oder gleich Java).

A0.2 Ein Architekt gebe zur Überdachung des Quadrats

$$-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}\,, \qquad -\frac{\pi}{2} \leq y \leq \frac{\pi}{2}$$

($L = \pi$) mit Hilfe von Bögen am Rand die Höhe

$$\Phi\left(x = \pm\frac{\pi}{2}, y\right) = \cos(y)\,, \qquad \Phi\left(x, y = \pm\frac{\pi}{2}\right) = \cos(x)$$

vor.

a. Berechnen Sie mit Hilfe des Programms aus A0.1 wie das mit Stoff bespannte Dach aussieht, wenn Sie Diskretisierungen mit $N =$

$11, 21, 41, \ldots$ verwenden ! Starten Sie im Inneren mit $\Phi\left(\vec{x}_{\vec{r}}\right) = 0$! Wie oft müssen Sie das Jacobi-Verfahren durchlaufen, um eine Genauigkeit von $\delta\Phi = 10^{-5}$ zu erreichen ?

b. Untersuchen Sie, wie Sie N und $\delta\Phi$ wählen müssen, damit die Abweichung von der exakten Lösung

$$\Phi\left(x, y\right) = \frac{1}{\cosh(\pi/2)} \left(\cos(x)\cosh(y) + \cosh(x)\cos(y)\right) \qquad (0.14)$$

kleiner als 10^{-4} wird !

A0.3 Wir betrachten nun ein Quadrat der Größe $L = 2$. Der Rand ist bei $x = \pm L/2 = \pm 1$ geerdet ($\Phi = 0$) und bei $y = \pm L/2 = \pm 1$ sei ein Potential $\Phi(x, \pm 1) = \Phi_0 = 1$ vorgegeben.
Berechnen Sie $\Phi(\vec{x}_{\vec{r}})$ mit Hilfe des Programms aus A0.1 ! Starten Sie im Inneren mit $\Phi\left(\vec{x}_{\vec{r}}\right) = \Phi_0/2 = 1/2$ und betrachten Sie die Fälle $N = 11, 25, 51$!
Vergleichen Sie Ihre Ergebnisse für die verschiedenen N und diskutieren Sie das Verhalten an den Ecken, wobei Sie Φ als elektrostatisches Potential interpretieren sollten !

A0.4* Konvertieren Sie Ihr Programm aus A0.1 und A0.2 nach Java, sofern noch nicht geschehen !

a. Geben Sie das Ergebnis für Φ in eine Datei aus !

b. Wir wollen Φ nun graphisch darstellen. Eine Möglichkeit ist, zuerst ein Fenster mit (mindestens) $N \times N$ Bildpunkten zu öffnen. Die Funktionswerte können dann über die Bildpunkte dargestellt werden. Am einfachsten ist die Verwendung von Graustufen, z.B. weiß für $\Phi \geq 1$ und schwarz für $\Phi \leq 0$ (die korrekte Lösung sollte im Bereich $0 \leq \Phi \leq 1$ liegen).
Aktualisieren Sie die Ausgabe nach jeder Jacobi-Iteration ! So können Sie sehen, wie sich der Stoff von der 'Erde' ($\Phi = 0$) über das Dach 'spannt'.

0.5 Lineare Gleichungssysteme

Als kleine Verallgemeinerung der Laplace-Gleichung (0.10) betrachten wir die Poisson-Gleichung, die in CGS-Einheiten lautet

$$-\Delta\,\Phi = 4\,\pi\,\rho\,, \tag{0.15}$$

Mit der Diskretisierung (0.8) nimmt diese die Form

$$\sum_{i=1}^{d} \frac{\Phi\,(\vec{x}_{\vec{r}} + \Delta x_i\,\vec{e}_i) + \Phi\,(\vec{x}_{\vec{r}} - \Delta x_i\,\vec{e}_i) - 2\,\Phi\,(\vec{x}_{\vec{r}})}{\Delta x_i^2} = -4\,\pi\,\rho\,(\vec{x}_{\vec{r}}) \tag{0.16}$$

an. Bringt man alle bekannte Information in (0.16) auf die rechte Seite, so erhält man eine Gleichung der Form

$$A\,\vec{\Phi} = \vec{b}\,. \tag{0.17}$$

Dies ist ein Spezialfall der allgemeineren Aussage, daß die Diskretisierung einer linearen partiellen Differentialgleichung auf ein lineares Gleichungssystem führt. Diese Beobachtung wollen wir zum Anlaß nehmen, Verfahren zur Lösung linearer Gleichungssysteme zu diskutieren (siehe z.B. Kapitel 2 von [5] oder Kapitel 6 von [6] für eine allgemeinere Diskussion). Oft hat A in (0.17) zusätzliche nützliche Eigenschaften, insbesondere

1. Für viele Differentialgleichungen ist A symmetrisch (bzw. hermitesch)

$$A = A^{\dagger}\,. \tag{0.18}$$

2. A ist dünn besetzt, d.h. die meisten Matrixelemente a_{rs} von A verschwinden. Im Fall von (0.16) gilt in jeder Spalte s, daß $a_{rs} \neq 0$ nur für höchstens $2\,d+1$ Werte von r, und zwar unabhängig von der Größe des gewählten Gitters.

3. A ist positiv definit, d.h.

$$\vec{x}\cdot A\vec{x} > 0 \qquad \text{für} \quad \vec{x} \neq \vec{0}\,. \tag{0.19}$$

   Diese Eigenschaft geht für die Diskretisierung A von $-\Delta$ darauf zurück, daß dieser Differentialoperator positiv definit ist (man sieht schnell, daß $-\int \mathrm{d}^d x\, f(\vec{x})^\star\,\Delta\, f(\vec{x}) = \int \mathrm{d}^d x\, \left|\vec{\nabla}\, f(\vec{x})\right|^2 > 0$ für geeignete Funktionen $f(\vec{x})$, wenn man Randterme ignoriert).

Bevor wir uns nun linearen Gleichungssystemen zuwenden, sei allerdings angemerkt, daß die *direkte* Lösung des Gleichungssystems (0.17) zumindest für die Diskretisierung der Laplace-Gleichung (0.10) auf einem hyperkubischen Gitter keineswegs das effizienteste Verfahren ist, sondern für diesen Fall bereits das Jacobi-Verfahren aus Kapitel 0.3 und dessen Verallgemeinerungen deutlich besser sind.

0.5.1 Gauß-Elimination

Ein lineares Gleichungssystem

$$A\,\vec{x} = \vec{b}\,. \tag{0.20}$$

hat u.a. folgende bekannte Eigenschaften:

1. Das Vertauschen zweier beliebiger *Zeilen* von A und der entsprechenden 'Zeilen' von $\vec{b}$ hat keinen Einfluß auf die Lösung $\vec{x}$. Eine solche Vertauschung entspricht lediglich einer Vertauschung der Reihenfolge beim Aufschreiben der linearen Gleichungen für $\vec{x}$.

2. Die Lösung $\vec{x}$ bleibt genauso unverändert, wenn man eine Zeile von A durch eine Linearkombination ihrer selbst und einer beliebigen anderen Zeile ersetzt, so lange die gleiche Linearkombination in den 'Zeilen' von $\vec{b}$ gebildet wird.

Diese Eigenschaften werden von dem wohl bekanntesten Lösungsverfahren für lineare Gleichungssysteme ausgenutzt, der sogenannten Gauß-Elimination[1] (siehe z.B. Kapitel 5 von [4] oder Kapitel 2 von [5]).

Sei A eine $n \times n$-Matrix mit Matrixelementen a_{rs}. Dann besteht eine Variante der Gauß-Elimination aus folgenden Schritten:

1. Für alle Spalten $s = 1, \ldots, n$

   (a) (*Pivotisierung*) Suche das Matrixelement mit dem größten $|a_{rs}|$. Vertausche dann die Zeilen r und s von A und $\vec{b}$.

   (b) (*Elimination*) Für alle Zeilen $r = 1, \ldots, n$ mit $r \neq s$ und $a_{rs} \neq 0$ ersetze

$$\begin{aligned} a_{rt} &\leftarrow a_{rt} - \frac{a_{rs}}{a_{ss}}\, a_{st} \qquad \text{für } t = s, \ldots, n\,, \\ b_r &\leftarrow b_r - \frac{a_{rs}}{a_{ss}}\, b_s\,. \end{aligned} \tag{0.21}$$

2. Die Lösung von (0.20) ist nun gegeben durch

$$x_r = \frac{b_r}{a_{rr}}\,. \tag{0.22}$$

[1] Löst man ein lineares Gleichungssystem mit Papier und Bleistift, so verwendet man meistens dieses oder zumindest ein ähnliches Verfahren, auch wenn man nicht bewußt den Gauß-Algorithmus anwendet.

Bemerkungen:

1. Es sind drei geschachtelte Schleifen über s, r und t jeweils der Länge n abzuarbeiten. Zwar kann man optimieren, z.B. derart, daß man bereits erzeugte Nullen nicht noch einmal betrachtet. Dies ändert aber nichts daran, daß das komplette Verfahren $\mathcal{O}(n^3)$ Operationen benötigt.

2. Für die Diskretisierung der Poisson-Gleichung (0.16) steht das größte Matrixelement bereits auf der Diagonalen $r = s$. Wir können also in diesem Fall auf die Pivot-Suche verzichten[2].

3. Der Speicherbedarf für die Matrix A wächst mit n^2. Zum Beispiel für $n = 10\,000$ braucht man fast 800MByte Speicher um die Matrixelemente als `double`-Zahlen (zu 8 Byte) zu speichern.

0.5.2 Conjugate Gradient Verfahren

Die direkte Lösung eines linearen Gleichungssystems (0.20) z.B. mit dem Gauß-Verfahren führt für Matrizen moderater Größe (n einige Tausend) zu den bereits erwähnten Problemen eines stark wachsenden CPU-Zeit- und Speicher-Bedarfs. Auf einem modernen, gut ausgebauten PC ist z.B. $n \approx 20\,000$ mit `double`-Genauigkeit gerade noch realisierbar. Ferner wird die Lösung durch viele aufeinanderfolgende Rechenoperationen bestimmt. Somit sammeln sich Rundungsfehler an. Aus diesem Grund sind direkte Lösungsverfahren für große Matrizen weder besonders stabil noch genau. Alle diese Probleme können mit iterativen Verfahren zumindest in speziellen Fällen gleichzeitig gelöst werden. Die endliche Maschinengenauigkeit kann man z.B. so ausnutzen, daß man mit deutlich weniger als n^3 Operationen eine Lösung der gewünschten Genauigkeit approximiert. Vermeidet man dabei Transformationen der Matrix, so kann man auch die Speicherung der ganzen Matrix für dünn besetzte Ursprungsmatrizen vermeiden und so den Speicherbedarf erheblich reduzieren.

Das Conjugate Gradient Verfahren (Konjugierte Gradienten-Verfahren) ist ein solches iteratives Verfahren (siehe z.B. Kapitel 2 von [5], Kapitel 6.13 von [6], Kapitel 2 von [7] oder Kapitel 8.7 von [8]). Es basiert auf der Idee, die Funktion

$$f(\vec{x}) = \frac{1}{2}\,\vec{x} \cdot A\,\vec{x} - \vec{b} \cdot \vec{x} \tag{0.23}$$

[2] Da das Diagonal-Element verschwinden kann, ist die Pivotisierung im allgemeinen zwingend erforderlich, um in (0.21) die Division durch $a_{ss} = 0$ zu vermeiden.

zu minimieren. In einem Extremal- oder Sattelpunkt gilt für symmetrische Matrizen A

$$\vec{0} = \vec{\nabla} f(\vec{x}) = A\,\vec{x} - \vec{b}\,, \tag{0.24}$$

d.h. ein Extremal- oder Sattelpunkt $\vec{x}$ der Funktion (0.23) ist automatisch eine Lösung des linearen Gleichungssystems (0.20).

Beim Conjugate Gradient Verfahren nimmt man nun an, daß die Matrix A symmetrisch und positiv definit ist (beides gilt für Diskretisierungen des Operators $-\Delta$). Sei $\vec{x}_0$ die Lösung von (0.24), d.h. $A\,\vec{x}_0 - \vec{b} = \vec{0}$. Dann kann man unter Ausnutzung der Symmetrie von A zeigen, daß

$$f(\vec{x}_0 + \vec{x}') = \frac{1}{2}\,\vec{x}' \cdot A\,\vec{x}' + c \tag{0.25}$$

mit

$$c = \frac{1}{2}\,\vec{x}_0 \cdot A\,\vec{x}_0 - \vec{b} \cdot \vec{x}_0\,. \tag{0.26}$$

Da A positiv definit ist, gilt $\vec{x}' \cdot A\,\vec{x}' > 0$ für $\vec{x}' \neq \vec{0}$. Folglich ist $\vec{x}' = \vec{0}$ das eindeutige Minimum der Funktion (0.25).

Wir haben somit gezeigt, daß für symmetrische positiv definite Matrizen A die Lösung des Gleichungssystems (0.20) äquivalent ist zur Bestimmung des Minimums der Funktion (0.23).

Dieses Minimum kann nun iterativ bestimmt werden. Dies soll zuerst mit einem sehr einfachen Verfahren illustriert werden, der sogenannten Methode des *steilsten Abstiegs*. Hier läuft man jeweils die Funktion f entlang ihrer maximalen Steigung nach unten. Die Richtung der maximalen Steigung an einem Punkt $\vec{x}_k$ ist durch den Gradienten

$$\vec{\nabla} f(\vec{x}_k) = A\,\vec{x}_k - \vec{b} =: -\vec{r}_k \tag{0.27}$$

gegeben. Man bestimmt nun einen neuen Punkt

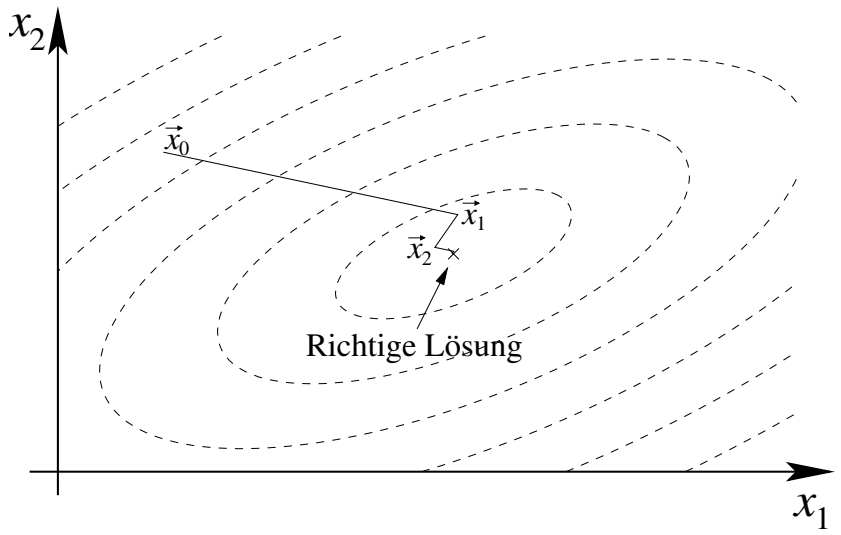
$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k\,\vec{r}_k \tag{0.28}$$

so, daß $f(\vec{x}_{k+1})$ bzgl. α_k minimal ist. Aus der Bedingung $\partial f/\partial\alpha_k = 0$ leitet man leicht her, daß

$$\alpha_k = \frac{\vec{r}_k \cdot \vec{r}_k}{\vec{r}_k \cdot A\vec{r}_k} \tag{0.29}$$

zu wählen ist. Man beachte, daß aufeinanderfolgende Schritte voneinander unabhängig sind. Rundungsfehler, die unweigerlich immer auftreten, sammeln sich also nicht an, sondern werden in nachfolgenden Schritten wieder eliminiert.

Das folgende Bild illustriert das Verfahren des steilsten Abstiegs für ein zweidimensionales Beispiel:



Man sieht, daß sich die Vektoren $\vec{x}_k$ allmählich dem Minimum entlang einer Zick-Zack-Bahn annähren, wobei auch bereits in Richtung des Minimums gelaufene Schritte teilweise wieder rückwärts gelaufen werden.

Eine bessere Wahl ist, wenn man nicht einfach immer in Richtung des Gradienten läuft, sondern eine Folge von Vektoren $\vec{p}_k$ wählt, die diese steilste Abstiegsrichtung möglichst gut unter der Nebenbedingung approximieren, daß die Vektoren $\vec{p}_k$ paarweise konjugiert sind. Hierbei bedeutet 'konjugiert', daß die Vektoren bzgl. A paarweise orthogonal sind:

$$\vec{p}_i \cdot A\,\vec{p}_j = 0 \tag{0.30}$$

für $i \neq j$.

Man kann zeigen [7], daß die folgende Iterationsvorschrift des *Conjugate Gradient* (CG) Verfahrens [6] die gewünschten Eigenschaften hat:

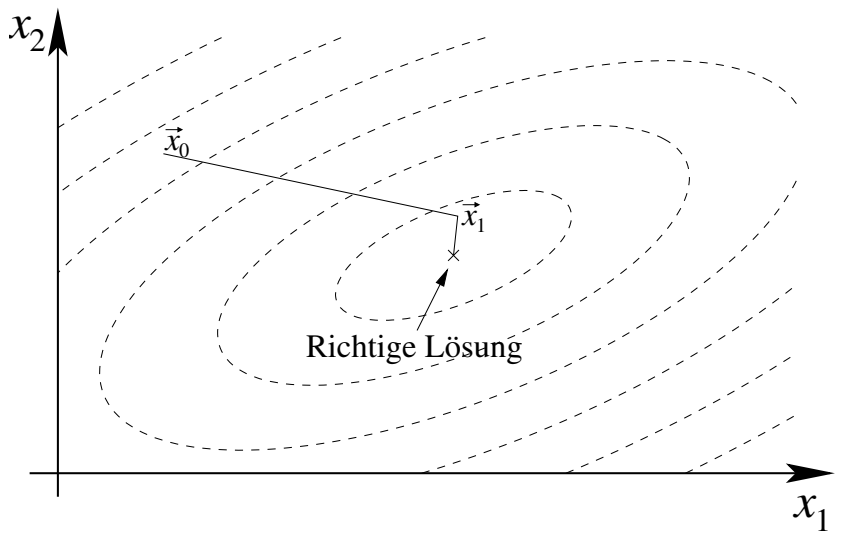
$$\begin{aligned}
\vec{u}_k &= A\,\vec{p}_k\,, \\
\alpha_k &= \frac{\vec{r}_k \cdot \vec{r}_k}{\vec{p}_k \cdot \vec{u}_k}\,, \\
\vec{x}_{k+1} &= \vec{x}_k + \alpha_k\,\vec{p}_k\,, \\
\vec{r}_{k+1} &= \vec{r}_k - \alpha_k\,\vec{u}_k\,, \\
\beta_k &= \frac{\vec{r}_{k+1} \cdot \vec{r}_{k+1}}{\vec{r}_k \cdot \vec{r}_k}\,, \\
\vec{p}_{k+1} &= \vec{r}_{k+1} + \beta_k\,\vec{p}_k\,.
\end{aligned} \tag{0.31}$$

Für einen gegebenen Startvektor $\vec{x}_0$ sind dabei die folgenden Anfangswerte zu wählen:

$$\vec{p}_0 = \vec{r}_0 = \vec{b} - A\,\vec{x}_0\,. \tag{0.32}$$

Beim CG Verfahren gilt außer (0.30) auch, daß die $\vec{r}_k$ paarweise orthogonal sind, d.h. $\vec{r}_i \cdot \vec{r}_j = 0$ für $i \neq j$ [7].

Aufgrund der Orthogonalitätsrelation (0.30) der $\vec{p}_k$ ist bei exakter Arithmetik $\vec{p}_n = \vec{0}$, d.h. das CG Verfahren findet theoretisch die exakte Lösung in n Schritten. Zur Veranschaulichung sei das Verfahren an dem obigen zweidimensionalen Beispiel ($n = 2$) mit demselben $\vec{x}_0$ illustriert:



In diesem Fall haben wir die richtige Lösung bereits nach dem 2. Schritt !

Für große Matrizen kommt man sogar meistens mit deutlich weniger als n Schritten aus, um die Lösung mit einer gewünschten Genauigkeit zu approximieren. Man kann zeigen [6], daß sich das Quadrat der Fehlerfunktion im Schritt $k + 1$ um den folgenden Betrag reduziert:

$$\alpha_k\,\vec{r}_k \cdot \vec{r}_k\,. \tag{0.33}$$

Das CG Verfahren kann somit abgebrochen werden, wenn (0.33) eine vorgegebene Fehlerschwelle unterschreitet.

Bemerkungen:

1. Auch beim CG Verfahren summieren sich die Fehler im Verlauf der Iteration. Dies hat zur Folge, daß die Orthogonalitätsrelation (0.30) umso

schlechter erfüllt ist, je größer $|i-j|$ ist. Damit es diese Rundungsfehler vergißt, sollte man das CG Verfahren nach einigen Iterationen j 'neu starten', d.h. $\vec{x}_j$ als Startvektor betrachten und mit diesem neue Anfangsbedingungen nach (0.32) berechnen.

2. Die Matrix A wird im CG Verfahren nicht verändert – man braucht lediglich eine Matrix-Vektor Multiplikation. Deswegen kann man spezielle Algorithmen für dünn besetzte Matrizen verwenden, wie sie z.B. bei der Diskretisierung partieller Differentialgleichungen auftreten. Eine Möglichkeit ist, auf die Speicherung der Matrix A zu verzichten und das Produkt $A\,\vec{x}$ für das gegebene Problem als Funktion zu programmieren.

3. Ein gut implementiertes CG Verfahren ist für die Lösung der Diskretisierung einer partiellen Differentialgleichung bei gegebener Genauigkeit normalerweise deutlich schneller als das Jacobi-Verfahren aus Kapitel 0.3. Das CG Verfahren und seine Verallgemeinerungen können darüber hinaus auch bei zahlreichen anderen Problemen verwendet werden.

Das CG Verfahren kann im Vergleich zur hier vorgestellten Variante durchaus noch verbessert werden, z.B. durch den Einsatz eines Vorkonditionierers.

Am Ende dieses Kapitels sei nur kurz erwähnt, daß beachtliche Leistungssteigerungen für die diskutierten Randwertprobleme durch den Einsatz sogenannter Mehrgitter-Verfahren erzielt werden können (siehe z.B. Kapitel 19.6 von [5], Kapitel 5.6 von [9], sowie [10]). Der Grundgedanke dieser Verfahren ist, schnellere Konvergenz durch die Kombination von Gittern mit unterschiedlicher Auflösung zu erzielen. Das Jacobi-Verfahren benötigt z.B. viele Iterationen, bis durch wiederholte Mittelung die Randbedingungen in das Innere des Gebiets propagiert werden. Es liegt somit nahe, daß man deutlich schneller zum Ziel gelangt, wenn man zuerst auf einem groben Gitter eine genäherte Lösung für das Gesamtsystem bestimmt, diese dann auf ein feineres Gitter überträgt, wo dann nur noch lokale Feinkorrekturen durchzuführen sind. Ähnliches gilt auch für das CG Verfahren: Lösungen auf einem Gitter anderer Auflösung können als Startvektoren $\vec{x}_0$ verwendet werden. Da diese bereits eine gute Näherung dastellen, konvergiert das CG Verfahren nun mit wenigen Iterationen gegen die gesuchte Lösung. Tatsächlich ist die Kombination von Mehrgitter- und CG Verfahren besonders leistungsfähig [10].

0.6 Aufgaben

A0.5 Formulieren Sie das Randwertproblem aus A0.2 (oder A0.3) als lineares Gleichungssystem für die $n = N^2$ inneren Gitterpunkte !
Achtung: Die Randterme tauchen auf der rechten Seite von (0.20) auf.

a. Lösen Sie das Gleichungssystem mit Hilfe der Gauß-Elimination ! Betrachten Sie nacheinander die Fälle $N = 11$, 21 und 51 und vergleichen Sie die Programmlaufzeit mit dem Jacobi-Algorithmus !

b. Lösen Sie das Gleichungssystem mit Hilfe des CG Verfahrens[3] ! Geben Sie also Fehlerschranke dabei eine Genauigkeit (`precision`) von 10^{-8} vor, d.h. brechen Sie das Verfahren ab, sobald (0.33) kleiner als 10^{-16} wird. Betrachten Sie nacheinander die Fälle $N = 21$, 51, 101 und 201 und vergleichen Sie die Programmlaufzeit mit dem Jacobi- bzw. Gauß-Algorithmus !
Achtung: A muß nun die Diskretisierung von $-\Delta$ sein (warum ?).

Hinweis: Unter Unix kann die Laufzeit eines Programms auf der Befehlszeile mit

`time` Kommando

gemessen werden.

A0.6* Lösen Sie die Poisson-Gleichung (0.15) mit Hilfe des CG Verfahrens für die umseitig skizzierte Konfiguration zweier Platten mit Ladung $\pm q$! Am linken und rechten Rand des Rechtecks seien dabei jeweils Neumannsche Randbedingungen vorgegeben

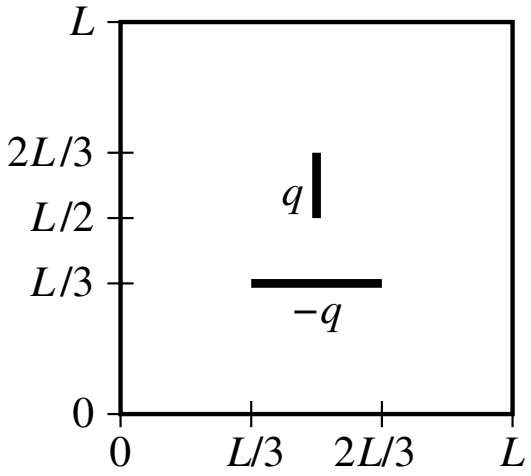
$$\frac{\partial \Phi}{\partial y}(x, 0) = \frac{\partial \Phi}{\partial y}(x, L) = 0\,,$$

sowie am oberen bzw. unteren Rand Dirichletsche Randbedingungen:

$$\Phi(0, x) = \Phi(L, y) = 0\,.$$

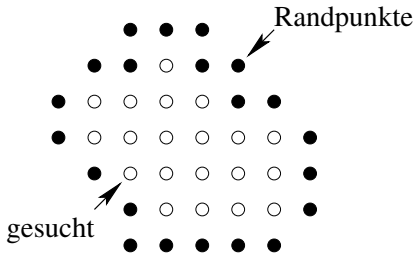
Hinweis: Verwenden Sie (0.7) um Funktionswerte am linken und rechten Rand zu eliminieren. In (0.8) kann dann z.B. der Funktionswert f_{r-1} am Rand durch den Funktionswert f_{r+1} im Inneren ersetzt werden. Leider reduziert sich hierbei die Genauigkeit der Näherung am Rand auf $\mathcal{O}(\Delta x)$ (warum ?).

[3]Das Verfahren steht als C++- und Java-Programm-Fragment auf der Kurs-Homepage zum Herunterladen zur Verfügung.


L
2L/3
L/2
L/3
0
0
L/3
2L/3
L
q
−q

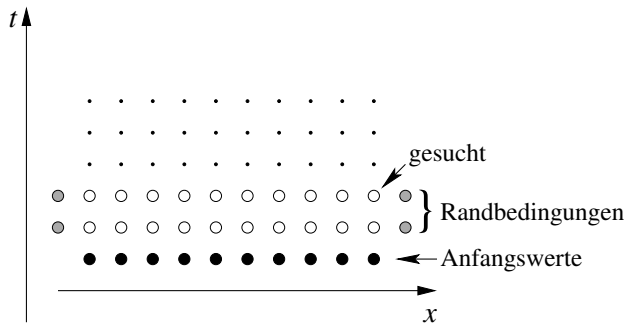
1 Zeitentwicklung für partielle Differentialgleichungen

Im vorhergehenden Kapitel haben wir uns kurz mit dem *Randwertproblem* beschäftigt, dessen Aufgabenstellung durch folgende Skizze verdeutlicht wird:



Beim Randwertproblem sind Funktionswerte oder Ableitungen auf den Rändern eines Gebiets vorgegeben. Mit diesen Vorgaben sucht man für alle inneren Gittepunkte simultan nach einer Lösung der gegebenen Diskretisierung einer partiellen Differentialgleichungen.

Wir wollen uns nun dem *Anfangswertproblem* zuwenden, das mit folgender Skizze verdeutlicht wird:



Beim Anfangswertproblem sind nun Anfangswerte für eine Funktion $\vec{u}$ gegeben und die Diskretisierung der partiellen Differentialgleichungen diktiert die Entwicklung der Funktion zu späteren Zeiten t. Auch hier benötigt man Randbedingungen, allerdings nur beim jeweils nächsten Zeitpunkt, um dort das Verhalten der Funktion festzulegen. Das Anfangswertproblem kann nacheinander für diskrete Zeitschritte gelöst werden, so daß andere Algorithmen als für Randwertprobleme zum Einsatz kommen. Auch zu diesem Thema gibt es umfangreiche Literatur, insbesondere sei wieder auf Kapitel 19 von [5], sowie [9, 11] und Kapitel 6 von [12] verwiesen.

Wir setzen eine lineare Zeitabhängigkeit voraus und schreiben allgemein

$$\frac{\partial \vec{u}}{\partial t} = \mathcal{L}\left(\vec{u}(\vec{x},t), \vec{x}, t\right) . \tag{1.1}$$

Hierbei ist $\mathcal{L}$ ein (möglicherweise nicht-linearer) partieller Differentialoperator bzgl. der räumlichen Komponenten $\vec{x}$.

Tatsächlich können allgemeine Probleme auch höherer Ordnung in der Zeit t auf die Form (1.1) gebracht werden, sofern sie linear in der Zeit sind. Die wollen wir an der Wellengleichung

$$\frac{\partial^2 \Psi(\vec{x},t)}{\partial t^2} = c^2\, \Delta \Psi(\vec{x},t) \tag{1.2}$$

illustrieren. Wir schreiben

$$\vec{u} = \begin{pmatrix} \Psi \\ \frac{\partial}{\partial t}\, \Psi \end{pmatrix}$$

und können (1.2) dann tatsächlich in die Form

$$\frac{\partial}{\partial t}\, \vec{u} = \begin{pmatrix} 0 & 1 \\ c^2\, \Delta & 0 \end{pmatrix} \vec{u} = \mathcal{L}(\vec{u}) \tag{1.3}$$

bringen. Man beachte, daß aufgrund der zweiten Ableitung in (1.2) Anfangsbedingungen sowohl für die Funktion Ψ als auch für ihre Ableitung $\frac{\partial}{\partial t}\, \Psi$ vorgegeben werden müssen. Alternativ kann man bei der Wellengleichung (1.2) die Werte von Ψ zu einem Anfangszeitpunkt t_a und einem Endzeitpunkt t_e vorgeben. Man gelangt dann zurück zu einem Randwertproblem für $t_a \leq t \leq t_e$. Im folgenden wollen wir uns jedoch mit dem Anfangswertproblem beschäftigen.

Zur numerischen Behandlung diskretisieren wir die räumlichen Variablen wieder gemäß (0.4). Die Zeit t wird analog an diskreten Punkten

$$t_n = t_0 + n\, \Delta t \tag{1.4}$$

mit ganzzahligen $n = 0,\, 1,\, 2,\, \ldots$ betrachtet. Ferner führen wir die Notation

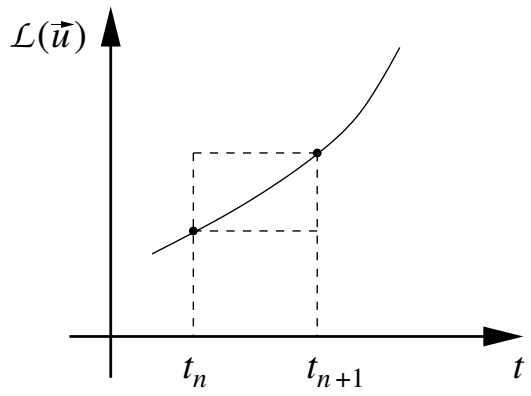
$$\vec{u}_{\vec{r}}^{\,(n)} = \vec{u}(\vec{x}_{\vec{r}}, t_n) \tag{1.5}$$

ein.

Das Ziel ist nun die Berechnung von $\vec{u}_{\vec{r}}^{\,(n+1)}$ aus bekannten $\vec{u}_{\vec{r}}^{\,(n)}$ (sowie ggfs. $\vec{u}_{\vec{r}}^{\,(n-1)}$, $\vec{u}_{\vec{r}}^{\,(n-2)}$, $\ldots$). Formal wird das Problem durch Integration von (1.1) gelöst:

$$\vec{u}_{\vec{r}}^{\,(n+1)} = \vec{u}_{\vec{r}}^{\,(n)} + \int_{t_n}^{t_{n+1}} \mathrm{d}t\, \mathcal{L}(\vec{u}) \tag{1.6}$$

Diese Integration sei mit folgender Skizze veranschaulicht:



Das Integral in (1.6) muß näherungsweise ausgewertet werden. Einige einfache Möglichkeiten sind:

i. Man nähert das Integral durch ein Rechteck mit der Höhe des linken Randpunkts:

$$\int_{t_n}^{t_{n+1}} \mathrm{d}t\, \mathcal{L}(\vec{u}) \approx \Delta t\, \mathcal{L}(\vec{u}^{(n)})\,.$$

Dies führt auf das *explizite Euler-Verfahren*. Nach (1.6) ist $\vec{u}^{(n+1)}$ mit dieser Näherung direkt („explizit") für gegebenes $\vec{u}^{(n)}$ berechenbar

$$\vec{u}^{(n+1)} = \vec{u}^{(n)} + \Delta t\, \mathcal{L}(\vec{u}^{(n)})\,. \tag{1.7}$$

Dieses Verfahren ist erster Ordnung in der Zeit, wie man leicht einsieht, wenn man $\vec{u}$ um t_n Taylor-entwickelt:

$$\left.\frac{\partial \vec{u}}{\partial t}\right|_n = \frac{\vec{u}^{(n+1)} - \vec{u}^{(n)}}{\Delta t} - \frac{1}{2}\,\Delta t\, \left.\frac{\partial^2 \vec{u}}{\partial t^2}\right|_n + \mathcal{O}(\Delta t^2)\,. \tag{1.8}$$

ii. Man kann auch den rechten Randpunkt verwenden, bzw. $\vec{u}$ um t_{n+1} Taylor-entwickeln:

$$\left.\frac{\partial \vec{u}}{\partial t}\right|_{n+1} = \frac{\vec{u}^{(n+1)} - \vec{u}^{(n)}}{\Delta t} + \frac{1}{2}\,\Delta t\, \left.\frac{\partial^2 \vec{u}}{\partial t^2}\right|_{n+1} + \mathcal{O}(\Delta t^2)\,. \tag{1.9}$$

Dies führt nun auf das *implizite Euler-Verfahren*

$$\vec{u}^{(n+1)} = \vec{u}^{(n)} + \Delta t\, \mathcal{L}(\vec{u}^{(n+1)})\,. \tag{1.10}$$

Das unbekannte $\vec{u}^{(n+1)}$ taucht nun auf beiden Seiten der Gleichung auf, es handelt sich also um eine implizite Gleichung für $\vec{u}^{(n+1)}$, d.h. ein (nicht-)lineares Gleichungssystem für die unbekannten $\vec{u}_{\vec{r}}^{(n+1)}$. Auch dieses Verfahren ist erster Ordnung in Δt.

iii. Schließlich kann man durch Kombination von (1.8) und (1.9) die Ordnung verwenden. Unter Berücksichtigung von

$$\left.\frac{\partial^2 \vec{u}}{\partial t^2}\right|_{n+1} = \left.\frac{\partial^2 \vec{u}}{\partial t^2}\right|_{n} + \mathcal{O}(\Delta t)$$

ist nämlich

$$\frac{1}{2}\left(\left.\frac{\partial \vec{u}}{\partial t}\right|_{n+1} + \left.\frac{\partial \vec{u}}{\partial t}\right|_{n}\right) = \frac{\vec{u}^{(n+1)} - \vec{u}^{(n)}}{\Delta t} + \mathcal{O}(\Delta t^2)\,. \tag{1.11}$$

Man erhält damit das *implizite Euler-Verfahren 2. Ordnung*

$$\vec{u}^{(n+1)} = \vec{u}^{(n)} + \frac{\Delta t}{2}\left(\mathcal{L}(\vec{u}^{(n)}) + \mathcal{L}(\vec{u}^{(n+1)})\right)\,. \tag{1.12}$$

1.1 Explizite Lösungsverfahren

In diesem Unterkapitel werden wir zuerst das explizite Euler-Verfahren genauer untersuchen und weitere explizite Lösungsverfahren vorstellen, die Nachteile des Euler-Verfahrens beheben.

1.1.1 Explizites Euler-Verfahren

Das explizite Euler-Verfahren (1.7) ist leicht zu implementieren und sehr schnell. Es hat aber leider einen großen Nachteil: Es ist unbrauchbar.

Die Probleme des Euler-Verfahrens werden durch eine Stabilitätsanalyse offengelegt. Als Beispiel verwenden wir die partielle Differentialgleichung

$$\frac{\partial u}{\partial t} = -v\,\frac{\partial u}{\partial x} \tag{1.13}$$

Die Lösung dieser Differentialgleichung zu gegebenen Anfangsbedingungen $u(x, t_0) = u_0(x)$ kann sofort angegeben werden:

$$u(x,t) = u_0\left(x - v\,(t - t_0)\right)\,. \tag{1.14}$$

Dies bedeutet, daß sich ein vorgegebenes Profil der Funktion u mit konstanter Geschwindigkeit v unter Beibehaltung seiner Form nach rechts bewegt.

Die partielle Ableitung nach x in (1.13) diskretisieren wir gemäß (0.7) und erhalten für diesen Fall des expliziten Euler-Verfahrens

$$u_r^{(n+1)} = u_r^{(n)} - \frac{v\,\Delta t}{2\,\Delta x}\left(u_{r+1}^{(n)} - u_{r-1}^{(n)}\right) . \tag{1.15}$$

Man betrachtet nun Eigenmoden des Differenzenoperators mit festem k

$$u_r^{(n)} = g(n,k)\,\mathrm{e}^{i\,k\,r\,\Delta x} . \tag{1.16}$$

Durch Einsetzen in (1.15) überzeugt man sich, daß (1.16) tatsächlich auf eine Lösung führt, sofern die Koeffizienten $g(n,k)$ die folgende Rekursionsrelation erfüllen:

$$g(n+1,k) = g(n,k)\,\left(1 - i\frac{v\,\Delta t}{\Delta x}\sin(k\,\Delta x)\right) . \tag{1.17}$$

Diese Rekursionsrelation wird gelöst durch

$$g(n,k) = g(k)^n\,g(0,k) \qquad \text{mit} \qquad g(k) = 1 - i\frac{v\,\Delta t}{\Delta x}\sin(k\,\Delta x) . \tag{1.18}$$

Offensichtlich ist für $k \neq 0$

$$|g(k)|^2 = 1 + \left(\frac{v\,\Delta t}{\Delta x}\sin(k\,\Delta x)\right)^2 > 1 . \tag{1.19}$$

Die Amplitude $g(n,k)$ wächst also für alle $k \neq 0$. Dies ist in krassem Gegensatz zur bekannten Lösung (1.14) und bedeutet, daß das Verfahren immer instabil ist ! Natürlich liefert (1.15) für kurze Zeiten trotzdem vernünftige Lösungen. Die genaue Zeitspanne hängt davon ab, wie groß der zweite Term in (1.19) ist, d.h. insbesondere wie groß k bzw. wie glatt die Funktion $u_0(x)$ ist. Früher oder später wird man aber immer eine nicht mehr sinnvolle Näherung erhalten.

Im allgemeinen kann man eine ,,von Neumann-Stabilitätsanalyse" durchführen. Für eine gegebene nicht-lineare Differentialgleichung in $\vec{u}$ schreibt man $\vec{u} = \vec{u}_0 + \delta\vec{u}$ und linearisiert durch Entwicklung bis zur ersten Ordnung in $\delta\vec{u}$, wobei man annimmt, daß $\vec{u}_0$ bereits eine exakte Lösung der Differenzengleichung ist. Die linearisierte Gleichung für $\delta\vec{u}$ ist vom Typ (1.13), wobei der Koeffizient v natürlich vom Entwicklungspunkt $\vec{u}_0$ abhängt. Zu beantworten ist dann die Frage, ob eine instabile Eigenmode $\delta\vec{u}$ existiert. Die Existenz einer Instabilität bedeutet z.B., daß sich Rundungsfehler aufschaukeln werden und ist somit unerwünscht. Das oben vorgeführte Argument zeigt demnach, daß das explizite Euler-Verfahren immer (d.h. für allgemeine Differentialoperatoren $\mathcal{L}$ und allgemeine Anfangsbedingungen) instabil ist.

1.1.2 Lax-Verfahren

Die Instabilität des expliziten Euler-Verfahrens kann darauf zurückgeführt werden, daß sich insbesondere Schwankungen mit großen k schnell aufschaukeln. Dies legt eine Regularisierung des Verfahrens durch Glättung nahe. Man ersetzt in der Entwicklungsvorschrift (1.7) den Wert $\vec{u}_r^{(n)}$ durch den Mittelwert seiner Nachbarpunkte. In einer räumlichen Dimension ersetzt man also $\vec{u}_r^{(n)}$ in (1.7) durch $\frac{1}{2}\left(\vec{u}_{r+1}^{(n)} + \vec{u}_{r-1}^{(n)}\right)$ und erhält so das *explizite Lax-Verfahren*

$$\vec{u}_r^{(n+1)} = \frac{1}{2}\left(\vec{u}_{r+1}^{(n)} + \vec{u}_{r-1}^{(n)}\right) + \Delta t\, \mathcal{L}(\vec{u}^{(n)})_r\,. \tag{1.20}$$

Die Stabilitätsanalyse führen wir wieder repräsentativ für die einfache Differentialgleichung (1.13) durch. Das Lax-Verfahren führt nun auf

$$u_r^{(n+1)} = \frac{1}{2}\left(u_{r+1}^{(n)} + u_{r-1}^{(n)}\right) - \frac{v\,\Delta t}{2\,\Delta x}\left(u_{r+1}^{(n)} - u_{r-1}^{(n)}\right)\,. \tag{1.21}$$

Wir betrachten wieder die Eigenmoden (1.16), setzen diese in (1.21) ein und erhalten mit $g(n,k) = g(k)^n\, g(0,k)$ folgende Lösung

$$g(k) = \cos(k\,\Delta x) - i\frac{v\,\Delta t}{\Delta x}\sin(k\,\Delta x)\,. \tag{1.22}$$

Man beachte, daß der erste Summand im Gegensatz zu (1.18) nun kleiner als eins ist. Für den Betrag gilt nun

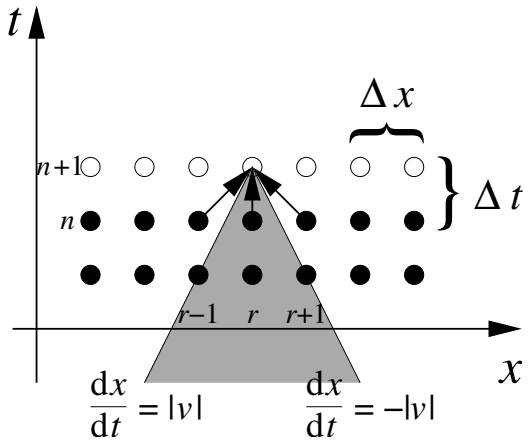
$$|g(k)|^2 = 1 + \left(1 - \left(\frac{v\,\Delta t}{\Delta x}\right)^2\right)\,\sin^2(k\,\Delta x)\,. \tag{1.23}$$

Damit die Lösung nicht anwächst, sollte $|g(k)| \leq 1$ für alle k sichergestellt sein. Aus (1.23) liest man ab, daß in der Tat $|g(k)| \leq 1$ gilt, wenn die *Courant-Friedrichs-Lewy* Stabilitätsbedingung

$$|v|\,\frac{\Delta t}{\Delta x} \leq 1 \tag{1.24}$$

erfüllt ist.

Das Lax-Verfahren und insbesondere die Stabilitätsbedingung (1.24) ist in der folgenden Skizze veranschaulicht:



Der Informationsfluß in (1.20) ist hier durch Pfeile angedeutet. Die Bedingungn (1.24) bedeutet nun, daß der durch die Pfeile definierte Kegel den grau schaffierten physikalischen Ausbreitungskegel enthält, d.h. daß die numerische Ausbreitungsgeschwindigkeit $\frac{\Delta x}{\Delta t}$ größer ist als der Betrag der physikalischen Geschwindigkeit v: $\frac{\Delta x}{\Delta t} \geq |v|$.

Es ist ferner instruktiv, (1.20) in die Form des expliziten Euler-Verfahrens (1.7) umzuschreiben:

$$\begin{aligned} \vec{u}_r^{(n+1)} &= \vec{u}_r^{(n)} + \frac{1}{2}\left(\vec{u}_{r+1}^{(n)} - 2\vec{u}_r^{(n)} + \vec{u}_{r-1}^{(n)}\right) + \Delta t\, \mathcal{L}(\vec{u}^{(n)})_r \\ &=: \vec{u}_r^{(n)} + \Delta t\, \widetilde{\mathcal{L}}(\vec{u}^{(n)})_r\,. \end{aligned} \tag{1.25}$$

Den Übergang von (1.7) zu (1.20) kann man folglich durch Addition eines Terms zu dem partiellen Differentialoperator $\mathcal{L}(\vec{u})$ interpretieren, dessen Kontinuumsform

$$\widetilde{\mathcal{L}}(\vec{u}) = \mathcal{L}(\vec{u}) + \frac{1}{2}\,\frac{(\Delta x)^2}{\Delta t}\,\Delta\vec{u} \tag{1.26}$$

lautet. Dieser Zusatzterm hat die Form eines Diffusionsterms ! Beim Übergang vom expliziten Euler- zum Lax-Verfahren wurde also eine numerische Dämpfung eingeführt (natürlich mit einem in Abhängigkeit von der Diskretisierung geeignet gewählten Koeffizienten). Dieser Zusatzterm dämpft insbesondere hohe k-Moden und verhindert auf diese Weise deren Anwachsen. Zwar werden auch die kleinen k-Moden gedämpft, so daß auch hier bei langen Zeiten Artefakte auftreten. Die Dämpfung ist bei kleinen k jedoch weniger stark. Mit dem Lax-Verfahren kann man somit langwellige Eigenschaften (kleine k) über Zeiträume verfolgen, bei denen im expliziten Euler-Verfahren die hohen k-Moden, die sowieso im Bereich von Diskretisierungs-Artefakten

liegen, bereits stark angewachsen wären und die Lösung unbrauchbar gemacht hätten.

Tatsächlich handelt es sich um eine Standard-Vorgehensweise, numerische Verfahren durch Addition geeignet gewählter Dämpfungsterme zu stabilisieren (siehe z.B. [11]). Solche numerischen Verfahren können durchaus Lösungen liefern, die von einer mathematisch exakten Lösung der ursprünglichen partiellen Differentialgleichung abweichen. Solche Abweichungen mögen jedoch vertretbar sein, denn die Natur regularisiert Singularitäten ebenfalls durch Einführung von Dissipation. Aus diesem Grund kann man sich vorstellen, daß ein regularisiertes numerisches Verfahren ein gegebenes physikalisches Problem sogar genauer beschreibt als die ursprüngliche idealisierte Beschreibung mit einer partiellen Differentialgleichung.

1.1.3 Leapfrog-Verfahren

Das Lax-Verfahren (1.20) ist ein erstes Verfahren, das stabil und somit brauchbar ist. Leider ist es nur von erster Ordnung in Δt. Arbeitet man mit zweiter Ordnung Genauigkeit in Δx muß man also Δt entsprechend klein wählen, d.h. sehr viele Zeit-Schritte ausführen um die Entwicklung über einen Zeitraum zu berechnen, der einem räumlichen Gitterabstand entspricht. Dieses Verhalten wird im ,,Leapfrog"- (Bocksprung)-Verfahren verbessert. Wir verwenden nun auch für die partielle Ableitung nach der Zeit t die Näherung zweiter Ordnung (0.7):

$$\left.\frac{\partial \vec{u}}{\partial t}\right|_n = \frac{\vec{u}^{(n+1)} - \vec{u}^{(n-1)}}{2\Delta t} + \mathcal{O}(\Delta t^2)\,. \tag{1.27}$$

Dies führt auf das *Leapfrog-Verfahren*

$$\vec{u}^{(n+1)} = \vec{u}^{(n-1)} + 2\Delta t\,\mathcal{L}(\vec{u}^{(n)})\,. \tag{1.28}$$

Dieses Verfahren ist nach Konstruktion genau bis zur zweiten Ordnung in Δt. Allerdings benötigt man zur Berechnung von $\vec{u}^{(n+1)}$ nun neben $\vec{u}^{(n)}$ offensichtlich auch die Funktion $\vec{u}^{(n-1)}$ aus einem Schritt zuvor[4].

Wir führen wieder die bekannte Stabilitätsanalyse für unsere einfache Differentialgleichung (1.13) durch, die nun auf folgende Berechnungsvorschrift führt

$$u_r^{(n+1)} = u_r^{(n-1)} - v\,\frac{\Delta t}{\Delta x}\left(u_{r+1}^{(n)} - u_{r-1}^{(n)}\right)\,. \tag{1.29}$$

[4]Im ersten Schritt tritt ein unbekanntes $\vec{u}^{(-1)}$ auf. Dessen Bestimmung kann z.B. vermieden werden, wenn man im ersten Schritt $\vec{u}^{(1)}$ mit dem expliziten Euler-Verfahren (1.7) aus dem gegebenen $\vec{u}^{(0)}$ bestimmt und erst dann das Leapfrog-Schema (1.28) anwendet.

Der Ansatz (1.16) führt nun auf eine zweistufige Rekursion:

$$g(n+1,k) = g(n-1,k) - \frac{2i\,v\,\Delta t}{\Delta x}\sin(k\,\Delta x)\,g(n,k)\,. \tag{1.30}$$

Mit $g(n,k) = g(k)^n\,g(0,k)$ erhält man folgende quadratische Gleichung

$$g(k)^2 + 2i\,A(k)\,g(k) - 1 = 0 \qquad \text{mit} \qquad A(k) = \frac{v\,\Delta t}{\Delta x}\sin(k\,\Delta x)\,. \tag{1.31}$$

Für die beiden Lösungen dieser Gleichung $g(k) = -i\,A(k) \pm \sqrt{1 - A(k)^2}$ gilt nun

$$|g(k)|^2 = \begin{cases} A(k)^2 + |1 - A(k)^2| = 1 & \text{für } |A(k)| \le 1, \\ \left|A(k) \pm \sqrt{A(k)^2 - 1}\right|^2 & \text{für } |A(k)| > 1. \end{cases} \tag{1.32}$$

Im zweiten Fall ist der Betrag einer der beiden Lösungen größer eins, das Verfahren also instabil. Hingegen ist das Leapfrog-Verfahren stabil, wenn $|A(k)| \le 1$ gilt. Dies ist wieder für alle k simultan sichergestellt, wenn die Courant-Bedingung (1.24) erfüllt ist.

Wir fassen die wesentlichen Eigenschaften des Leapfrog-Verfahrens (1.28) zusammen:

- $\oplus$ Das Leapfrog-Verfahren ist stabil für

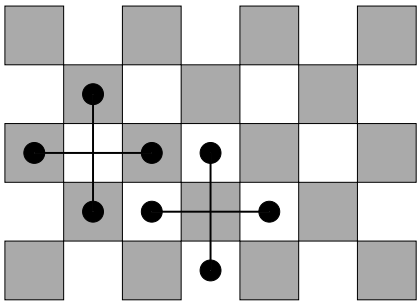
$$|v|\,\frac{\Delta t}{\Delta x} \le 1\,. \tag{1.33}$$

- $\oplus$ Im stabilen Fall (1.33) tritt keine Dämpfung auf ($|g(k)| = 1$ – siehe (1.32)).

- $\oplus$ Das Verfahren (1.28) ist zweiter Ordnung in Δt.

- $\bullet$ Aufgrund seiner guten Eigenschaften kommt das Leapfrog-Verfahren u.a. auch in modernen Simulationen von Tsunamis zum Einsatz, die den Anspruch einer quantitativen Modellierung haben [13].

- $\ominus$ Leider ist in vielen Fällen eine Untergitter-Entkopplung möglich. Zur Verdeutlichung dieses Punktes diene folgende Skizze für die Entwicklungsvorschrift (1.29):



Man beobachtet, daß die Entwicklung für die Gitterpunkte (r, n) schachbrettartig entkoppelt ist, d.h. Werte von u auf dem einen Untergitte (z.B. die weißen Felder) tragen nicht zu der Entwicklung auf dem anderen Untergitter (z.B. graue Felder) bei und umgekehrt. Dies birgt das Risiko des Auseinanderdriftens benachbarter Funktionswerte auf verschiedenen Untergittern im Verlauf der Simulation.
Die Skizze mag auch den Namen des Verfahrens („Leapfrog" bzw. Bocksprung) erklären.

Am Endes dieses Kapitel seien zwei allgemeine Bemerkungen zu expliziten Lösungsverfahren ergänzt. Zuerst sei darauf hingewiesen, daß die Untergitter-Entkopplung des Leapfrog-Verfahrens mit anderen expliziten Verfahren unter Beibehaltung der Genauigkeit zweiter Ordnung behoben werden kann. Ein solches Verfahren ist das *Lax-Wendroff-Schema*. Diese Verbesserung erkauft man sich allerdings auf Kosten einer (kleinen) numerischen Dämpfung. Das Lax-Wendroff-Schema ist somit nicht unbedingt besser und soll deswegen hier auch nicht näher besprochen werden, sondern es sei an dieser Stelle auf die Literatur verwiesen (z.B. Kapitel 19.1 von [5], sowie [9, 11]).

Ferner ist darauf hinzuweisen, daß nicht alle möglichen Fehlerquellen diskutiert wurden. Insbesondere wurde nicht auf die Dispersion der Verfahren eingegangen. Dies bedeutet, daß Fehler nicht nur in der Amplitude sondern auch in der Phasenbeziehung auftreten können. Für die vorgestellten Verfahren gilt, daß für kleine k näherungsweise die gewünschte lineare Dispersion vorliegt, für große k jedoch Abweichungen auftreten.

1.2 Aufgaben

Die vorgestellten Verfahren wollen wir an zwei Modell-Beispielen ausprobieren:

(A) Die einfache Ausbreitungsgleichung (1.13) dient zum Test der Verfahren.

(B) Als einfache nicht-lineare Gleichung betrachten wir die Burgers-Gleichung

$$\frac{\partial u}{\partial t} = -u \frac{\partial u}{\partial x} . \tag{1.34}$$

Diese Gleichung ist (1.13) sehr ähnlich, nur wurde die konstante Geschwindigkeit v durch die Funktion $u(x,t)$ ersetzt. Die Gleichung (1.34) ist ein Beispiel für Transportprozesse, bei denen die Ausbreitungsgeschwindigkeit abhängig vom Wert der Funktion ist (man stelle sich z.B. eine Dichteverteilung vor, bei der die Ausbreitungsgeschwindigkeit abhängig von der Dichte ist).

In beiden Fällen betrachten wir ein Intervall $0 \leq x \leq L$ der Länge L mit periodischen Randbedigungen, d.h.

$$u(x+L,t) = u(x,t) . \tag{1.35}$$

Als Anfangsbedingungen zur Zeit $t_0 = 0$ betrachten wir ferner in beiden Fällen

$$u(x, t=0) = u_0(x) = 1 + \sin\left(\frac{2\pi}{L}\, x\right) . \tag{1.36}$$

A1.1 Verwenden Sie das explizite Euler-Verfahren zur Berechnung von $u^{(n+1)}$ aus $u^{(n)}$!

a. Berechnen Sie für das Modellproblem (A) die Zeitentwicklung nach (1.15) mit $\Delta x = L/128$ für $v\,\Delta t/\Delta x = 1$, $1/10$ und $1/100$! Untersuchen Sie, bis zu welcher Zeit t Sie entwickeln können, bevor die Lösung $u^{(n)}$ unbrauchbar wird !

b. Berechnen Sie nun für die Burgers-Gleichung (1.34) (Modellproblem (B)) die Zeitentwicklung mit $\Delta x = L/256$! Wählen Sie Δt so, daß Sie aufgrund der Ergebnisse aus Aufgabenteil a mit einer stabilen Zeitenentwicklung bis $t = L$ rechnen ! Was beobachten Sie nun in der Zeitentwicklung der Anfangsbedingung (1.36) für $t \leq L$? Interpretieren Sie Ihr Ergebnis !

A1.2 Betrachten Sie das Modellproblem (A) mit $\Delta x = L/1024$ und $\Delta t = \Delta x/(2v)$. Berechnen Sie die Zeitentwicklung bis $t = 2L/v$ bzw. $t = 8L/v$ sowohl mit dem Lax-Verfahren (1.21) als auch mit dem Leapfrog-Verfahren (1.29) ! Bestimmen Sie die Genauigkeit des Ergebnisses durch Vergleich mit der bekannten Lösung (1.14) und diskutieren Sie mögliche Abweichungen !

A1.3* Betrachten Sie das Modellproblem (B) mit $\Delta x = L/1024$ und $\Delta t = \Delta x/2$. Berechnen Sie die Zeitentwicklung bis $t = L/2$ sowohl mit dem Lax-Verfahren (1.20) als auch mit dem Leapfrog-Verfahren (1.28) !
Zur Kontrolle der Genauigkeit nutzen wir die Tatsache, daß die Burgers-Gleichung (1.34) eine integrable Entwicklungsgleichung ist. Dies bedeutet, daß es unendlich viele Erhaltungssätze gibt. Tatsächlich prüft man für hinreichend glatte Lösungen $u(x,t)$ von (1.34) leicht nach, daß

$$\int \mathrm{d}x\, u(x,t)^m = C_m \tag{1.37}$$

konstant, d.h. unabhängig von t ist. Berechnen Sie insbesondere die Integrale C_1, C_2 sowie C_3 und diskutieren Sie aufgrund deren zeitlichen Verlaufes die Verläßlichkeit des numerischen Ergebnisses !

1.3 Transport I: Verkehrsfluß

Die in den vorhergehenden Kapiteln vorgestellten Verfahren sollen nun zu einer ersten Anwendung kommen: Dem Straßenverkehr. Dazu betrachtet man eine einspurige Straße bei großen Abständen und geht zu einer Kontinuums-Beschreibung über (siehe z.B. [14, 15]). Das Problem wird nun durch eine Dichte $\rho(x,t)$ der Autos sowie den Fluß $q(x,t)$ durch eine Stelle x beschrieben. Die Autos bewegen sich unter diesen Bedingungen mit einer (mittleren) Geschwindigkeit

$$v = \frac{q}{\rho} \tag{1.38}$$

Da Autos auf der Straße nicht verloren gehen, gilt ein Erhaltungssatz. Die Anzahl der Autos $N = \int_{x_1}^{x_2} \mathrm{d}x\, \rho(x,t)$ auf der Strecke $[x_1, x_2]$ kann sich nur durch Zu- bzw. Abfluß an der Rändern ändern:

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{x_1}^{x_2} \mathrm{d}x\, \rho(x,t) + q(x_2,t) - q(x_1,t) = 0\,. \tag{1.39}$$

Im Grenzfall $x_2 \to x_1$ führt (1.39) auf die Kontinuitätsgleichung

$$\frac{\partial \rho}{\partial t} + \frac{\partial q}{\partial x} = 0\,. \tag{1.40}$$

Dies ist eine erste Gleichung für den Straßenverkehr. Da zwei Variablen (ρ, q) vorliegen, benötigt man eine weitere Beziehung um die Zeitentwicklung festzulegen. Wir nehmen an, daß eine Zustandsgleichung

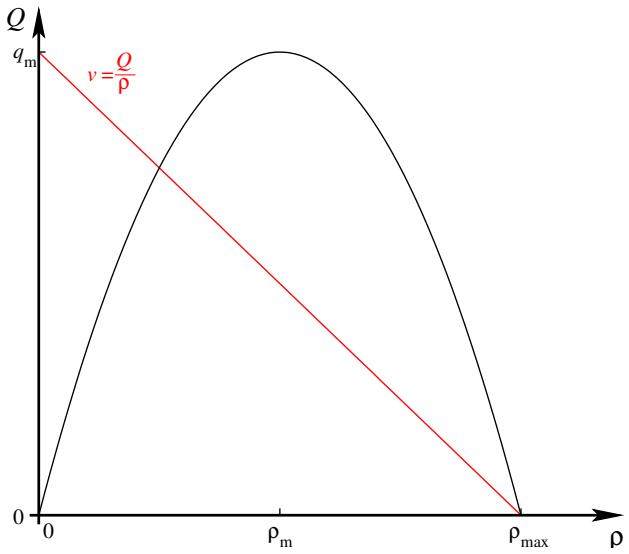
$$q = Q(\rho) \tag{1.41}$$

existiert, d.h. daß der Fluß q eine Funktion der Dichte ρ ist. Einsetzen von (1.41) in (1.40) führt auf folgende partielle Differentialgleichung (man beachte die Ähnlichkeit zur Burgers-Gleichung (1.34)):

$$\frac{\partial \rho}{\partial t} + \frac{\partial Q(\rho)}{\partial \rho} \frac{\partial \rho}{\partial x} = 0\,. \tag{1.42}$$

Empirische Daten und mikroskopische Modelle führen auf das auf der nächsten Seite skizzierte ,,fundamentale Diagramm" für $Q(\rho)$. Bei einer maximalen Dichte $\rho_{\max}$ bricht der Straßenverkehr zusammen und es findet kein Transport mehr statt, d.h. $Q(\rho) = 0$ für $\rho \geq \rho_{\max}$. Das Maximum q_{m} des Flusses wird für eine Dichte ρ_{m} zwischen 0 und $\rho_{\max}$ beobachtet. Die Geschwindigkeit einzelner Autos (1.38) ist in der Skizze rot dargestellt. Man beachte, daß diese für $\rho \to 0$ maximal ist und dann für $\rho \to \rho_{\max}$ abnimmt. Davon zu

unterscheiden ist die Ausbreitungsgeschwindigkeit von Dichteschwankungen, die nach (1.42) durch $c(\rho) = \frac{\partial Q(\rho)}{\partial \rho}$ gegeben ist. Diese ist für $\rho > \rho_{\mathrm{m}}$ negativ, d.h. Dichteschwankungen breiten sich bei hohen Dichten entgegen der Fahrtrichtung aus !



Wir führen nun eine Ampel bei x_0 ein [16]. Während der Grünphase liegt x_0 einfach im Inneren des Intervals und wird nicht gesondert behandelt. Eine rote Ampel hingegen zwingt die Auto zum Anhalten. Dies kann dadurch erreicht werden, daß während der Rot-Phase für $x < x_0$ die Randbedingung

$$\rho(x_0, t)|_{\mathrm{links}} = \rho_{\mathrm{max}} \tag{1.43}$$

gefordert wird. Aufgrund von $Q(\rho_{\mathrm{max}}) = 0$ erzwingt dies $v = 0$. Die Autos müssen also vor der Ampel anhalten. Jenseits der Ampel beobachtet man während einer Rotphase, daß keine Autos mehr kommen. Die Randbedingung lautet also für $x > x_0$

$$\rho(x_0, t)|_{\mathrm{rechts}} = 0\,. \tag{1.44}$$

1.3.1 Projekt

Eine einfache Näherung für das skizzierte fundamentale Diagramm ist

$$Q(\rho) = q_{\mathrm{m}} \left(\frac{2\,\rho}{\rho_{\mathrm{m}}} - \left(\frac{\rho}{\rho_{\mathrm{m}}} \right)^2 \right)\,. \tag{1.45}$$

Sinnvolle mit Beobachtungen verträgliche [15] Parameter sind hierbei

$$q_{\mathrm{m}} = 3000\,\frac{\text{Autos}}{\text{h}}\,, \qquad \rho_{\max} = 100\,\frac{\text{Autos}}{\text{km}}\,. \tag{1.46}$$

Diese Annahmen führen für $\rho \to 0$ auf eine plausible maximale Geschwindigkeit $v = \frac{2q_{\mathrm{m}}}{\rho_{\mathrm{m}}} = 120\,\frac{\mathrm{km}}{\mathrm{h}}$ (für (1.45) gilt $\rho_{\mathrm{m}} = \rho_{\max}/2$)[5].

Für ein endliches Straßenstück $[x_L, x_R]$ $(x_L < x_0 < x_R)$ sind weitere Randbedingungen anzugeben. Am linken Rand x_L modellieren wir einen konstanten Zufluß mit der Randbedingung

$$\rho|_{x_L} = \rho_0\,, \tag{1.47}$$

mit $0 < \rho_0 < \rho_{\max}$. Am rechten Rand x_R sollen die Autos ungehindert die Straße verlassen können, was auf die Randbedingung

$$\left.\frac{\partial \rho}{\partial x}\right|_{x_R} = 0 \tag{1.48}$$

führt.

A1.4** Wählen Sie geeignete Anfangsbedingungen bei $t = 0$ (möglich sind z.B. $\rho(x,0) = 0$ oder $\rho(x,0) = \rho_0$) sowie ein geeignetes Verfahren um die zeitliche Entwicklung des geschilderten Straßenverkehr-Modells zu berechnen ! Betrachten Sie zuerst eine Grün-Phase, schalten Sie dann Ihre Ampel auf Rot (und wieder zurück) ! Diskutieren Sie

(i) die numerische Stabilität Ihrer Simulation, sowie

(ii) die beobachteten Phänomene im Straßenverkehrs-Modell !

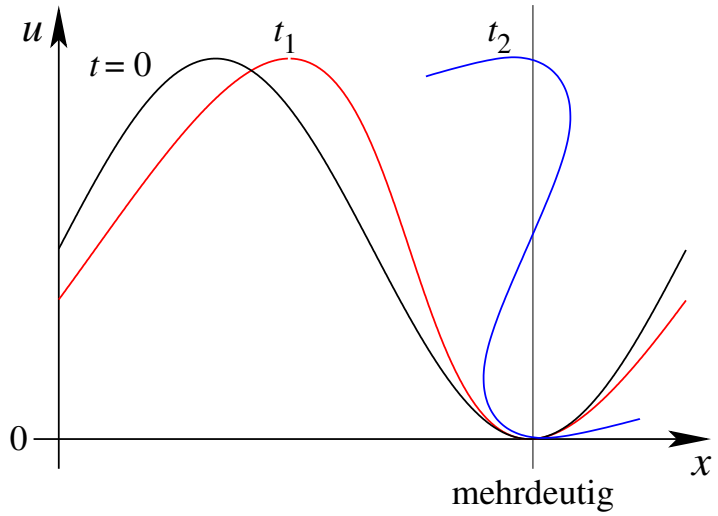
1.3.2 Schocks

Sowohl die Burgers-Gleichung (1.34) als auch die Verkehrsflußgleichung (1.42) sind vom Typ

$$\frac{\partial u}{\partial t} + c(u)\,\frac{\partial u}{\partial x} = 0\,. \tag{1.49}$$

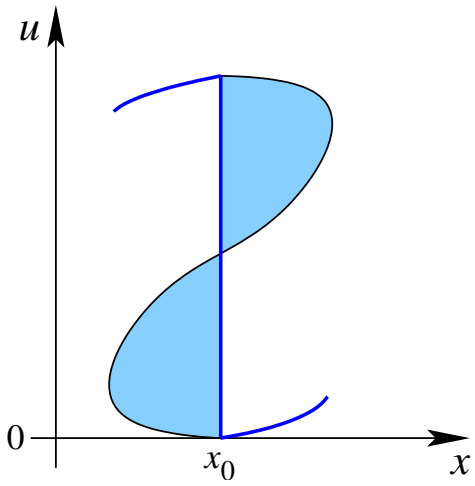
Insbesondere hängt also die Ausbreitungsgeschwindigkeit $c(u)$ von u ab; in der Burgers-Gleichung (1.34) hat man z.B. $c(u) = u$. Dies führt zu der im folgenden skizzierten Situation:

[5]Unter diesen Bedingungen wird der maximale Fluß bei einer relativ niedrigen Geschwindigkeit $v = 60\,\frac{\mathrm{km}}{\mathrm{h}}$ der einzelnen Autos erreicht.



Die Maxima eines bei $t = 0$ vorgegebenen Profils bewegen sich schneller als kleinere Funktionswerte ($u = 0$ bewegt sich bei der Burgers-Gleichung (1.34) gar nicht). Bei der Zeitentwicklung bis zum Zeitpunkt t_1 führt das zu einer Aufsteilung der Funktion u. Zu einem hypotethischen Zeitpunkt t_2 hat dann das Maximum das Minimum 'überholt'. Zum Zeitpunkt t_2 treten also Mehrdeutigkeiten auf, so daß das Profil von u keine Funktion von x mehr ist ! Je nach Fragestellung kann diese Situation unphysikalisch sein. Hat u insbesondere die Bedeutung einer Dichte am Ort x (wie z.B. ρ in der Verkehrsflußgleichung (1.42)), so sind solche Mehrdeutigkeiten unerwünscht.

Dieses Problem kann durch eine Maxwell-artige Konstruktion behoben werden, wie in folgender Skizze dargestellt:



Man folgt links z.B. erst dem oberen Zweig der 'Lösung' u und springt dann

an einer Stelle x_0 direkt auf den unteren Zweig, der im weiteren zu folgen ist. Die Position dieses Sprungs ist nicht eindeutig und kann nicht aus der Differentialgleichung (1.49) abgeleitet werden, sondern wird durch die Physik, d.h. Erhaltungssätze, diktiert. Für die Verkehrsflußgleichung (1.42) ist z.B. Erhaltung der Anzahl der Autos zu berücksichtigen. Die Stelle x_0 ist also so zu wählen, daß die beiden schraffierten Flächen in der Skizze die gleiche Anzahl Autos enthalten.

Aufgrund des Sprungs sind solche ,,Schock"-Lösungen unstetig und somit auch nicht im klassischen Sinn differenzierbar. Dementsprechend benötigt können solche Funktionen nur in einem verallgemeinerten Sinn Lösungen der Differentialgleichung (1.49) sein. Man integriert (1.49) bzgl. x und t:

$$\int \mathrm{d}t \int \mathrm{d}x\, \phi(x,t) \left(\frac{\partial u}{\partial t} + c(u) \frac{\partial u}{\partial x} \right) = 0\,, \tag{1.50}$$

wobei $\phi(x,t)$ eine beliebige 'Testfunktion' ist, Diese Testfunktionen werden als hinreichend differenzierbar und mit Nullrandbedingungen angenommen, so daß die Ableitungen nach partieller Integration in (1.50) auf $\phi(x,t)$ wirken. Die präzise Begriffsbildung hängt von der genauen Situation ab und soll hier nicht weiter diskutiert werden, sondern wir verweisen an dieser Stelle auf die mathematische Literatur (z.B. [17]). Lösungen von (1.49) im verallgemeinerten Sinn (1.50) werden als 'schwache Lösungen' bezeichnet.

Ein wichtiges technisches Hilfsmittel sind die sogenannten ,,Charakteristischen Kurven" $\mathcal{C}$, die für die partielle Differentialgleichung (1.49) als Lösungen $x(t)$ der gewöhnlichen Differentialgleichung

$$\frac{\mathrm{d}x}{\mathrm{d}t} = c(u(x,t)) \tag{1.51}$$

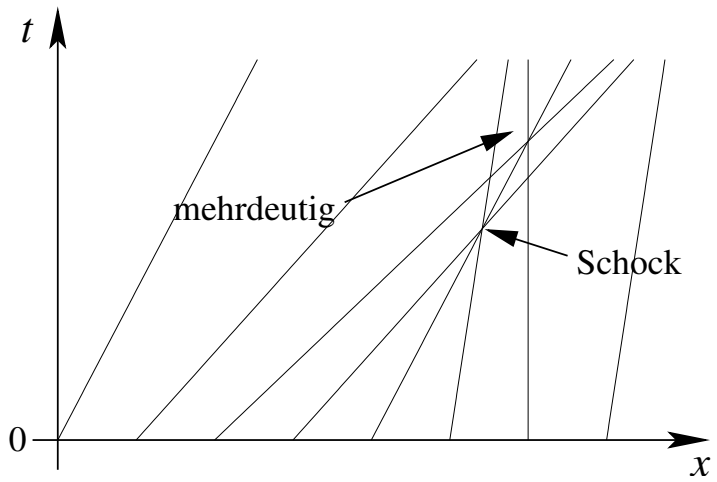
definiert werden (vgl. z.B. [14]). Für eine Lösung $u(x,t)$ von (1.49) gilt offensichtlich

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \frac{\partial u}{\partial t} + \frac{\mathrm{d}x}{\mathrm{d}t} \frac{\partial u}{\partial x} = 0\,. \tag{1.52}$$

Dies bedeutet, daß u konstant auf jeder charakteristischen Kurve $\mathcal{C}$ ist. Mit (1.51) folgt also, daß die charakteristischen Kurven $\mathcal{C}$ Geraden sind, deren Steigung bereits durch die Anfangsbedingungen $u(x,t_0)$ festgelegt sind.

Für die oben skizzierte Situation der Burgers-Gleichung sehen die charakteristischen Kurven z.B. wie auf der nächsten Seite skizziert aus. Schnittpunkte der charakteristischen Kurven weisen auf Mehrdeutigkeiten hin. Diese Überlegung zeigt also, daß für eine nicht-lineare partielle Differentialgleichung vom Typ (1.49) im allgemeinen ab einem bestimmten Zeitpunkt Schocks auftreten. Wie bereits oben erwähnt, ist die Auflösung der Mehrdeutigkeiten

(d.h. das Abschneiden der charakteristischen Kurven beim Aufeinanderstoßen) nicht eindeutig, sondern von der Physik bestimmt.



Aus der Perspektive numerischer Verfahren ist es beim Auftreten eines Schocks besonders schwer, Stabilität zu gewährleisten. Die numerische Dämpfung im Lax-Verfahren verhindert das Auftreten des Schocks komplett. Je nach Problematik ist dieses Verhalten unerwünscht, so beobachtet man z.B. im Straßenverkehr tatsächlich Staus. Der Mach-Kegel beim Überschallflug ist ein weiteres beobachtetes Schock-Phänomen.

Eine Simulation mit dem Leapfrog-Verfahren ist stabil bis zum Eintreten des Schocks, sofern die Courant-Bedingung (1.33) beachtet wird. Kommt es jedoch zum Schock, werden die Disktretisierungfehler in der räumlichen Ableitung u_x groß. Dies führt zu einem Überschwingen von u und dies wiederum zu einer Verletzung der Courant-Bedingung. Somit wird eine Simulation mit dem Leapfrog-Verfahren typischer Weise bald nach dem Eintreten eines Schocks instabil.

Eine mögliche alternative Vorgehensweise zur numerischen Behandlung von Schock-Lösungen ist ,,Godunov's Verfahren" (siehe z.B. Kapitel 19.1 von [5] und Kapitel 2 von [18]). Dieses Verfahren wird z.B. auch in der Verkehrsflußsimulation [16] verwendet. Es berücksichtigt Nichtlinearitäten insofern explizit, als es auf der expliziten Lösung für zwei Bereiche konstanten u aufbaut, die durch einen Schock getrennt sind. Die Lösung wird dann durch Zellen genähert, in denen die Funktion u jeweils konstant ist. Die Anpassung an den Rändern der Zellen wird schließlich mit Hilfe der bekannten Schock-Lösung vorgenommen.

1.4 Implizites Euler-Verfahren 2. Ordnung

Zu Beginn dieses Kapitels hatten wir auch das implizite Euler-Verfahren 2. Ordnung (1.12) vorgestellt. Dieses wollen wir im folgenden ein wenig genauer diskutieren (siehe auch [11]).

Wir führen wieder eine Stabilitätsanalyse anhand der Differentialgleichung (1.13) durch. Für diese Gleichung führt (1.12) auf

$$u_r^{(n+1)} = u_r^{(n)} - \frac{v\,\Delta t}{4\,\Delta x}\left(u_{r+1}^{(n)} - u_{r-1}^{(n)} + u_{r+1}^{(n+1)} - u_{r-1}^{(n+1)}\right) . \tag{1.53}$$

Wie gewohnt setzt man die Eigenmoden (1.16) in (1.53) ein und findet

$$g(n+1,k) = g(n,k) - i\frac{v\,\Delta t}{2\,\Delta x}\sin(k\Delta x)\left(g(n,k) + g(n+1,k)\right) . \tag{1.54}$$

Auflösung nach $g(n+1,k)$ führt auf

$$g(n+1,k) = \frac{1 - i\frac{v\,\Delta t}{2\,\Delta x}\sin(k\Delta x)}{1 + i\frac{v\,\Delta t}{2\,\Delta x}\sin(k\Delta x)}\, g(n,k)\,. \tag{1.55}$$

Da der Vorfaktor vom Betrag eins ist, ändert sich die Amplitude $g(n,k)$ nicht:

$$|g(n+1,k)| = |g(n,k)| . \tag{1.56}$$

Diese Beobachtung erlaubt bereits eine Auflistung wichtiger Eigenschaften des impliziten Euler-Verfahrens 2. Ordnung:

- $\oplus$ Es ist immer stabil (die Amplitude $g(n,k)$ wächst nicht).
- $\oplus$ Es tritt keine Dämpfung auf (die Amplitude $g(n,k)$ nimmt auch nicht ab – siehe (1.56)).
- $\oplus$ Es ist nach Konstruktion 2. Ordnung in Δt.
- $\oplus$ $\vec{u}^{(n+1)}$ ist eindeutig durch $\vec{u}^{(n)}$ bestimmt, die Start-Probleme des Leapfrog-Verfahrens liegen hier also nicht vor.
- $\ominus$ Die guten Eigenschaften basieren auf einer simultanen Propagation der Information an allen Plätzen in einem Zeitschritt. Dies bedeutet umgekehrt eine unendliche numerische Ausbreitungsgeschwindigkeit von Information. Störungen an einer Stelle können also instantan Auswirkungen an beliebig weit entfernten Orten haben. Je nach Fragestellung kann ein solches Verhalten unphysikalisch sein.

$\ominus$ $\vec{u}^{(n+1)}$ ist als Lösung einer impliziten Gleichung zu bestimmen. Dies macht implizite Verfahren aufwendiger als explizite Verfahren.

Für eine effiziente Implementation benötigt man also insbesondere ein schnelles Verfahren zur Lösung linearer Gleichungssysteme. Es liegt nahe, an die Verwendung des aus Abschnitt 0.5.2 bekannten CG Verfahrens zu denken. Dies ist jedoch nicht direkt möglich, wie z.B. die folgende Umordnung von (1.53) bereits im linearen Fall zeigt:

$$-\frac{v\,\Delta t}{4\,\Delta x}\,u_{r-1}^{(n+1)}+u_r^{(n+1)}+\frac{v\,\Delta t}{4\,\Delta x}\,u_{r+1}^{(n+1)}=\frac{v\,\Delta t}{4\,\Delta x}\,u_{r-1}^{(n)}+u_r^{(n)}-\frac{v\,\Delta t}{4\,\Delta x}\,u_{r+1}^{(n)}\,. \quad (1.57)$$

Diese Gleichung hat die Form $A\vec{u}^{(n+1)} = A^\dagger\vec{u}^{(n)} = \vec{b}$. Leider ist die Matrix A nicht-hermitesch (man beachte die unterschiedlichen Vorzeichen in (1.57) vor $u_{r-1}^{(n+1)}$ und $u_{r+1}^{(n+1)}$) ! Eine der Voraussetzungen für die Anwendung des CG Verfahrens ist also nicht erfüllt.

1.4.1 Bi-Conjugate Gradient Verfahren

Ist A in dem linearen Gleichungssystem (0.20) nicht hermitesch, so führt Multiplikation mit $A^\dagger$ auf

$$\tilde{A}\,\vec{x} = A^\dagger A\,\vec{x} = A^\dagger\vec{b} = \tilde{\vec{b}}\,, \quad (1.58)$$

d.h. auf ein lineares Gleichungssystem $\tilde{A}\,\vec{x} = \tilde{\vec{b}}$ mit hermiteschem $\tilde{A} = A^\dagger A$. $\tilde{A}$ ist außerdem positiv (semi-)definit, d.h. $\vec{x}\cdot\tilde{A}\vec{x} \geq 0$ für beliebige Vektoren $\vec{x}$.

Zur Lösung eines linearen Gleichungssystems (0.20) mit nicht-hermiteschem A könnte man also das CG Verfahren auf $\tilde{A} = A^\dagger A$ und $\tilde{\vec{b}} = A^\dagger\vec{b}$ anwenden. Gegen diese Vorgehensweise sprechen jedoch Konvergenz-Eigenschaften. Zu diesem Punkt ist zu ergänzen, daß die Konvergenz des CG Verfahrens durch die 'Konditionszahl' von A

$$\kappa_A = \frac{\lambda_{\max}}{\lambda_{\min}} \quad (1.59)$$

bestimmt wird (vgl. Kapitel 5.3.6 von [12]), wobei $\lambda_{\max}$ und $\lambda_{\min}$ der größte bzw. kleinste Eigenwert von A sind. Dies kann man sich dadurch plausibel machen, daß für $\kappa_A = 1$ alle Eigenwerte von A gleich sind. Dann muß $A = \lambda\mathbb{1}$ sein; der erste Abstiegsvektor $\vec{p}_0$ in (0.32) zeigt direkt auf das Minimum und das CG Verfahren liefert die gesuchte Lösung mit nur einem Schritt.

Je größer die Konditionszahl κ_A ist (d.h. umso mehr sie von eins abweicht), desto mehr weichen die Abstiegsrichtungen $\vec{p}_k$ vom Minimum ab und umso mehr Iterationen sind erforderlich. Nun ist

$$\kappa_{\tilde{A}} = \kappa_A^\star \kappa_A \,. \tag{1.60}$$

Dies führt zu entsprechend ungünstigen Konvergenz-Eigenschaften des CG Verfahrens bei Verwendung der Matrix $\tilde{A}$.

Das Bi-Conjugate Gradient Verfahren (siehe z.B. Kapitel 2.7 von [5] und Kapitel 5.3.7 von [12]) liefert hier günstigeres Verhalten. Es handelt sich um eine Verallgemeinerung des CG Verfahrens, das mit einem doppelten Satz Vektoren $\vec{p}_i$, $\vec{r}_i$, $\vec{u}_i$, $\hat{\vec{p}}_i$, $\hat{\vec{r}}_i$, $\hat{\vec{u}}_i$ arbeitet. Die Vektoren $\hat{\vec{r}}_i$ und $\vec{r}_i$ sind paarweise 'biorthogonal':

$$\hat{\vec{r}}_i \cdot \vec{r}_j = 0 \qquad \text{für } i \neq j. \tag{1.61}$$

Ferner gilt als Verallgemeinerung von (0.30), daß die Abstiegsrichtungen $\hat{\vec{p}}_i$ und $\vec{p}_i$ zueinander 'bikonjugiert' sind:

$$\hat{\vec{p}}_i \cdot A\,\vec{p}_j = 0 \qquad \text{für } i \neq j. \tag{1.62}$$

Die Iterationsvorschrift des Bi-Conjugate Gradient Verfahrens lautet:

$$\begin{aligned}
\vec{u}_k &= A\,\vec{p}_k \,, & \hat{\vec{u}}_k &= A^\dagger\,\hat{\vec{p}}_k \,, \\
\alpha_k &= \frac{\hat{\vec{r}}_k \cdot \vec{r}_k}{\hat{\vec{p}}_k \cdot \vec{u}_k} \,, & & \\
\vec{x}_{k+1} &= \vec{x}_k + \alpha_k\,\vec{p}_k \,, & & \\
\vec{r}_{k+1} &= \vec{r}_k - \alpha_k\,\vec{u}_k \,, & \hat{\vec{r}}_{k+1} &= \hat{\vec{r}}_k - \alpha_k^\star\,\hat{\vec{u}}_k \,, \\
\beta_k &= \frac{\hat{\vec{r}}_{k+1} \cdot \vec{r}_{k+1}}{\hat{\vec{r}}_k \cdot \vec{r}_k} \,, & & \\
\vec{p}_{k+1} &= \vec{r}_{k+1} + \beta_k\,\vec{p}_k \,, & \hat{\vec{p}}_{k+1} &= \hat{\vec{r}}_{k+1} + \beta_k^\star\,\hat{\vec{p}}_k \,.
\end{aligned} \tag{1.63}$$

Schließlich sind für einen gegebenen Startvektor $\vec{x}_0$ Anfangswerte gemäß der folgenden Verallgemeinerung von (0.32) zu wählen:

$$\hat{\vec{p}}_0 = \hat{\vec{r}}_0 = \vec{p}_0 = \vec{r}_0 = \vec{b} - A\,\vec{x}_0 \,. \tag{1.64}$$

Auf folgende Eigenschaften dieses Verfahrens sei besonders hingewiesen:

- Für hermitesche Matrizen $A^\dagger = A$ ist das Bi-Conjugate Gradient Verfahren äquivalent zum CG Verfahren.

- Die Konditionierung und damit die Konvergenz ist dem CG Verfahren gleich.

- Die zusätzlichen Operationen in (1.63) führen in etwa zu einer Verdopplung des Rechenaufwands im Vergleich zum CG Verfahren. Insbesondere benötigt man neben der Matrix-Vektor-Multiplikation für A auch eine für $A^\dagger$.

- Es ist möglich, die Konvergenz durch eine sogenannte ,,Vorkonditionierung" zu verbessern. Dazu multipliziert man das Gleichungssystem (0.20) mit einer Matrix K^{-1}

$$K^{-1} A \, \vec{x} = K^{-1} \vec{b} \,. \tag{1.65}$$

  Gelingt es, $K^{-1}A$ hinreichend nah an der Identität zu wählen, konvergiert das Verfahren bei Verwendung von $K^{-1}A$ entsprechend schneller. Im Zweifelsfall kann K einfach als diagonaler Anteil von A gewählt werden. Man beachte, daß $K^{-1}A$ selbst für hermitesches A im allgemeinen nicht-hermitesch ist, die Vorkonditierung also im Bi-Conjugate Gradient Verfahren entsprechend leichter durchzuführen ist.

Bei der Anwendung im impliziten Euler-Verfahren bietet es sich offensichtlich an, $\vec{u}^{(n)}$ als Startvektor zur Bestimmung von $\vec{u}^{(n+1)}$ zu wählen. Ist die Änderung in einem Zeitschritt nicht groß, konvergiert das Verfahren entsprechend mit wenigen Iterationen auf die gesuchte Lösung.

Zur Illustration der Behandlung von nicht-Linearitäten betrachten wir die Burgers-Gleichung (1.34), bei der $\mathcal{L}(u) = -u\, u_x$ ist. Nähert man in $\mathcal{L}(u^{(n+1)})$ den Funktionswert $u^{(n+1)} \approx u^{(n)}$, d.h. $\mathcal{L}(u^{(n+1)}) \approx -u^{(n)} \frac{\partial}{\partial x} u^{(n+1)}$, führt das implizite Euler-Verfahren (1.12) wieder auf eine lineare Gleichung[6]. Diese Vorgehensweise kann z.B. iterativ verbessert werden. Dazu betrachtet man eine Folge von Näherungen $\mathcal{L}(u^{(n+1,l+1)}) \approx -u^{(n+1,l)} \frac{\partial}{\partial x} u^{(n+1,l+1)}$ und beginnt mit $u^{(n+1,0)} = u^{(n)}$. Die skizzierte Vorgehensweise bedeutet Abbruch nach der Bestimmung von $u^{(n+1,1)} \approx u^{(n+1)}$. Man kann jedoch die Lösung des linearen Gleichungssystems mehrfach iterieren, bis die Approximationen $u^{(n+1,l)}$ eine gewünschte Genauigkeit erreicht haben.

[6]Da die Änderungen in den Ableitungen größer sind als in der Funktion selbst, führt man diese Näherung besser für den Wert der Funktion als ihre Ableitung durch.

1.4.2 Aufgabe

A1.5* Betrachten Sie das Modellproblem (B) (d.h. die Burgers-Gleichung (1.34)) mit $\Delta x = L/1024$ und $\Delta t = \Delta x/2$. Berechnen Sie die Zeitentwicklung bis $t = L/2$ mit dem impliziten Euler-Verfahren 2. Ordnung (1.12). Lösen Sie die auftretenden linearen Gleichungssysteme mit dem Bi-Conjugate Gradient Verfahren ! Vergleichen Sie Stabilität, Genauigkeit und Programm-Laufzeit mit dem Leapfrog-Verfahren aus Aufgabe A1.3 !

2 Transport II: Tsunamis

Aus aktuellem Anlaß wollen wir uns in diesem Kapitel mit Tsunamis beschäftigen, d.h. den gefürchteten Flutwellen, die durch Seebeben ausgelöst werden können. Auf dem offenen Meer liegt die Höhe der Tsunamis im Meter- und ihre Wellenlänge im Kilometerbereich, so daß sie kaum zu beobachten sind. Allerdings breiten sie sich sehr schnell aus (Geschwindigkeiten von einigen hundert km/h). Bei abnehmender Wassertiefe steilen sich Tsunamis stark auf und können dann an der Küste verheerende Schäden anrichten. Offensichtlich spielt die Meerestiefe bei diesem Phänomen eine zentrale Rolle.

Eine quantitativ genaue Simulation von Tsunamis ist ein wesentliches Anliegen beim Schutz vor solchen Katastrophen [13]. Auch sind Daten zu weltweiten Meerestiefen zumindest mit einer Auflösung von 2 Minuten öffentlich zugänglich [19]. Allerdings wollen wir uns hier mit einem einfachen Modell begnügen, das die wesentlichen qualitativen Eigenschaften illustriert. Auch die ursächliche Bewegung des Meeresbodens soll nicht simuliert werden, sondern wir betrachten die Ausbreitung einer vorgegebenen Anhebung der Meeresoberfläche.

Ausgangspunkt sind die Grundgleichungen der Hydrodynamik. Als erstes führt die Massenerhaltung mit Hilfe des Gaußschen Satzes auf die *Kontinuitätsgleichung* für eine Flüssigkeit:

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot (\rho \, \vec{u}) = 0 \,. \tag{2.1}$$

Hierbei ist ρ die Dichte und $\vec{u}$ die Geschwindigkeit der Flüssigkeit am Ort $\vec{r}$ zur Zeit t.

Die zweite Grundgleichung ist die „*Navier-Stokes-Gleichung*"

$$\frac{\partial \vec{u}}{\partial t} + \left(\vec{u} \cdot \vec{\nabla} \right) \vec{u} = \vec{K} - \frac{1}{\rho} \vec{\nabla} p + \nu \, \Delta \vec{u} \,. \tag{2.2}$$

Zur Veranschaulichung der linken Seite dieser Gleichung betrachten wir die Geschwindigkeit eines Teilchens, das als Bestandteil der Flüssigkeit transportiert wird. Nach der Kettenregel gilt

$$\frac{\mathrm{d}\, u_j}{\mathrm{d}t} = \frac{\partial u_j}{\partial t} + \sum_{k=1}^{3} \frac{\mathrm{d}\, r_k}{\mathrm{d}t} \, \frac{\partial u_j}{\partial r_k} \,. \tag{2.3}$$

Da das Teilchen Bestandteil der Flüssigkeit ist, gilt $\frac{\mathrm{d}\, r_k}{\mathrm{d}t} = u_k$. Somit kann (2.3) auch in der Form

$$\frac{\mathrm{d}\, \vec{u}}{\mathrm{d}t} = \frac{\partial \vec{u}}{\partial t} + \left(\vec{u} \cdot \vec{\nabla} \right) \vec{u} \tag{2.4}$$

geschrieben werden. Die linke Seite der Navier-Stokes-Gleichung (2.2) ist somit nichts anderes als die totale Ableitung der Geschwindigkeit $\vec{u}$ nach der Zeit t. Insbesondere basiert die Nichtlinearität des zweiten Terms auf der dualen Rolle der Teilchen, die einerseits Konstituenten der Flüssigkeit sind, andererseits von ihr mittransportiert werden.

Auf der rechten Seite der Navier-Stokes-Gleichung (2.2) steht als erstes die äußere Kraft pro Masse

$$\vec{K} = \frac{\mathrm{d}\,\vec{F}}{\mathrm{d}m} = \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} , \tag{2.5}$$

wobei bereits die spezielle hier relevante Form einer in negativer z-Richtung zeigenden Gravitationskraft angegeben wurde. Ferner stehen auf der rechten Seite der Navier-Stokes-Gleichung (2.2) der Druck p und eine Viskosität ν.

Wasser kann in guter Näherung als *inkompressible Flüssigkeit* betrachten werden, d.h. ρ ist konstant: $\frac{\partial \rho}{\partial t} = 0$, $\vec{\nabla}\rho = \vec{0}$. Die Kontinuitätsgleichung führt also für eine inkompressible Flüssigkeit auf

$$\vec{\nabla} \cdot \vec{u} = 0\,. \tag{2.6}$$

Insgesamt stellen (2.2) und (2.6) vier Gleichungen für die vier Unbekannten $\vec{u}$ und p dar.

Zum letzten Term der Navier-Stokes-Gleichung (2.2) beachte man ferner, daß in diesem Fall[7]

$$\Delta\vec{u} = \vec{\nabla}\left(\vec{\nabla}\cdot\vec{u}\right) + \vec{\nabla}\times\vec{\nabla}\times\vec{u} = \vec{\nabla}\times\vec{\nabla}\times\vec{u}\,, \tag{2.7}$$

wobei in der zweiten Identität die Gleichung (2.6) für eine inkompressible Flüssigkeit eingeht. Dieses Ergebnis zeigt, daß Viskosität für eine inkompressible Flüssigkeit genau dann wichtig ist, wenn Wirbel auftreten ($\vec{\nabla}\times\vec{u} \neq \vec{0}$). Dies tritt z.B. bei turbulenter Strömung auf, jedoch nicht im Fall der hier zu diskutierenden Tsunamis. Wir beschränken uns deswegen auf $\nu = 0$. In diesem Fall wird (2.2) auch ,,Euler-Gleichung" genannt.

Es ist im Prinzip möglich, die Zeitentwicklung mit Hilfe der Gleichungen (2.2), (2.6) zu berechnen. Eine zeitabhängige dreidimensionale Simulation mit Ausdehnungen zumindest einiger Kilometer bei der erforderlichen Auflösung im Meterbereich für die Oberfläche stellt jedoch offensichtlich eine

[7]Häufig wird eine allgemeinere Gleichung als Navier-Stokes-Gleichung bezeichnet, bei der $\vec{\nabla}\left(\vec{\nabla}\cdot\vec{u}\right)$ und $\vec{\nabla}\times\vec{\nabla}\times\vec{u}$ verschiedene Koeffizienten besitzen.

rechentechnische Herausforderung dar. Ferner gilt es noch das konzeptionelle Problem zu lösen, daß man am Verhalten der Meeresoberfläche interessiert ist, d.h. an der Grenzfläche zwischen Wasser und Luft. Bei einer hydrodynamischen Simulation wäre dies mit Hilfe geeigneter Randbedingungen zu implementieren[8]. Eine direkte Beschreibung der Dynamik der Oberfläche ist sowohl aus rechentechnischer als auch konzeptioneller Sicht eine attraktive Alternative.

2.1 Seichtwassergleichungen

Viele Tsunami-Simulationen [13] basieren auf den *Seichtwassergleichungen* (auch als *Flachwassergleichungen* bezeichnet). Diese Gleichungen stellen eine effektive Beschreibung der Dynamik einer Meeres- oder Flußoberfläche dar. Im folgenden sollen diese Gleichungen aus dem System (2.2), (2.6) abgeleitet werden. Die Argumentationsweise ist folgt dem Anhang von [20], für eine alternative Sichtweise im Sinne einer Störungsentwicklung um eine hydrostatisches System vergleiche man z.B. [21] sowie dort angegebene Referenzen. Vor der Rechnung sei nochmals daran erinnert, daß es sich bei Tsunamis um die Ausbreitung einer lokalen Störung handelt (d.h. um keine Oberflächenwellen), und daß die Wassertiefe wesentlich ist.

Zur Vereinfachung beschränken wir uns auf zwei räumliche Koordinaten x und z, d.h. wir identifizien

$$\vec{r} \to \begin{pmatrix} x \\ z \end{pmatrix}, \qquad \vec{u} \to \begin{pmatrix} u \\ w \end{pmatrix}. \tag{2.8}$$

Für diesen Fall kann die Inkompressibilitäts-Bedingung (2.6) als

$$\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z} = 0 \tag{2.9}$$

geschrieben werden. Die Navier-Stokes-Gleichung (2.2) mit $\nu = 0$ ist nun in Komponenten

$$\frac{\partial u}{\partial t} + u\,\frac{\partial u}{\partial x} + w\,\frac{\partial u}{\partial z} = -\frac{1}{\rho}\,\frac{\partial p}{\partial x}\,, \tag{2.10}$$

$$\frac{\partial w}{\partial t} + u\,\frac{\partial w}{\partial x} + w\,\frac{\partial w}{\partial z} = -\frac{1}{\rho}\,\frac{\partial p}{\partial x} - g\,. \tag{2.11}$$

Eine weitere nützliche Gleichung ist die Bedingung von Wirbelfreiheit (man beachte, daß verschwindede Viskosität $\nu = 0$ angenommen wurde)

$$\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x} = 0\,. \tag{2.12}$$

[8] Solche Simulationen werden z.B. in [12] beschrieben.

Die Wasseroberfläche wird als

$$\vec{h} = \begin{pmatrix} x \\ h(x,t) \end{pmatrix} \tag{2.13}$$

parametrisiert. Eine wesentliche Komponente der folgenden Diskussion wird die richtige Wahl der Randbedingungen an der Wasseroberfläche $h(x,t)$ sowie am Meeresboden $b(x,t)$ sein.

Zuerst betrachtet man die Geschwindigkeitskomponenten u_s, w_s an der Oberfläche. Nach (2.13) gilt

$$\begin{pmatrix} u_s \\ w_s \end{pmatrix} = \frac{\mathrm{d}\,\vec{h}}{\mathrm{d}t} = \frac{\partial\,\vec{h}}{\partial t} + \left(u_s \frac{\partial}{\partial x} + w_s \frac{\partial}{\partial z} \right) \vec{h} = \begin{pmatrix} 0 \\ \frac{\partial\, h}{\partial t} \end{pmatrix} + \begin{pmatrix} u_s \\ u_s \frac{\partial}{\partial x} h(x,t) \end{pmatrix} . \tag{2.14}$$

Wir nehmen nun an, daß die Horizontalkomponente der Geschwindigkeit an der Oberfläche verschwindet

$$u_s = 0 \tag{2.15}$$

Dies bedeutet, daß die Ausbreitung der Störung der Oberfläche durch Anheben und Absenken erfolgt und direkter Transport durch Oberflächenströmungen vernachlässigt werden kann. Glg. (2.14) führt dann auf die einleuchtende Identität

$$w_s = \frac{\partial\, h}{\partial t} \,. \tag{2.16}$$

Für die weitere Argumentation werden Strömungsgeschwindigkeiten durch Integration vom Meeresboden $b(x,t)$ bis zur Wasseroberfläche $h(x,t)$ gemittelt (hier wird die Bedeutung der Wassertiefe deutlich), d.h. man führt gemittelte Größen ein:

$$\bar{f}(x,t) = \frac{1}{h-b} \int\limits_{b(x,t)}^{h(x,t)} \mathrm{d}z\, f(x,z,t) \,. \tag{2.17}$$

Zuerst zeigt man, daß

$$\begin{aligned} \frac{\partial}{\partial x}\, (h-b)\, \bar{u} &= \frac{\partial}{\partial x} \int\limits_b^h \mathrm{d}z\, u(x,z,t) = \int\limits_b^h \mathrm{d}z\, \frac{\partial}{\partial x}\, u(x,z,t) \\ &= -\int\limits_b^h \mathrm{d}z\, \frac{\partial w}{\partial z} = -w_s = -\frac{\partial\, h}{\partial t} \,. \end{aligned} \tag{2.18}$$

Man kann sich überzeugen, daß die Randterme, die beim Vertauschen der partiellen Ableitung und der Integration auftreten, verschwinden (vgl. die genauere Diskussion für die nächste Gleichung). Im folgenden Schritt wurde (2.9) verwendet. Danach verwenden wir, daß die Strömung am Meeresboden keine Verktikalkomponent haben kann[9] $w_b = 0$. Im letzten Schritt geht schließlich (2.16) ein.

Glg. (2.18) ist die erste gesuchte Seichtwassergleichung

$$\frac{\partial h(x,t)}{\partial t} + \frac{\partial}{\partial x}\left(\left(h(x,t) - b(x,t)\right)\,\bar{u}(x,t)\right) = 0\,. \tag{2.19}$$

Hierbei handelt es sich um eine erste Gleichung für die beiden unbekannten Funktionen $h(x,t)$, $\bar{u}(x,t)$. Bei der Herleitung wurde nur die Inkompressibilitäts-Bedingung (2.6) benutzt. Für weitere Gleichungen ist nun die Navier-Stokes-Gleichung heranzuziehen.

Als erstes integrieren wir die x-Komponente der Navier-Stokes-Gleichung (2.10)

$$\begin{aligned}
0 &= \int_b^h \mathrm{d}z\,\left(\frac{\partial u}{\partial t} + u\,\frac{\partial u}{\partial x} + w\,\frac{\partial u}{\partial z} + \frac{1}{\rho}\,\frac{\partial p}{\partial x}\right) \\
&= \frac{\partial}{\partial t}\int_{b(x,t)}^{h(x,t)} \mathrm{d}z\,u - u_s\,\frac{\partial}{\partial t}\,h + u_b\,\frac{\partial}{\partial t}\,b \\
&\quad + \frac{1}{2}\int_b^h \mathrm{d}z\,\frac{\partial}{\partial x}\,u^2 \\
&\quad - \int_b^h \mathrm{d}z\,u\,\frac{\partial w}{\partial z} + w_s\,u_s - w_b\,u_b \\
&\quad + \frac{1}{\rho}\int_b^h \mathrm{d}z\,p - \frac{1}{\rho}\,p_s\,\frac{\partial h}{\partial x} + \frac{1}{\rho}\,p_b\,\frac{\partial b}{\partial x}\,.
\end{aligned} \tag{2.20}$$

In der ersten Zeile der rechten Seite wurden die Randterme ausgeschrieben, die bei Vertauschen der Integration und der partiellen Ableitung auftreten. In der dritten Zeile wurde partiell integriert. Das dort auftretende Integral kann mit (2.9) weiter umgeformt werden: $-\int_b^h \mathrm{d}z\,u\,\frac{\partial w}{\partial z} = \int_b^h \mathrm{d}z\,u\,\frac{\partial u}{\partial x} =$

[9] Hier geht genaugenommen die Näherung eines horizontalen Meeresbodens ein $\frac{\partial b}{\partial x} \approx 0$. Man beachte, daß Herleitungen [20, 21] gerne die Annahme $b = 0$ verwenden, was insbesondere $\frac{\partial b}{\partial x} = \frac{\partial b}{\partial t} = 0$ sicherstellt.

$\frac{1}{2}\int_b^h \mathrm{d}z\,\frac{\partial}{\partial x}\,u^2$. Wir wählen nun den Nullpunkt des Drucks so, daß an der Oberfläche $p_s = 0$ gilt. Ferner betrachten wir einen statischen Meeresboden ($\frac{\partial b}{\partial t} = 0$), der sich nur sehr langsam ändert ($\frac{\partial b}{\partial x} \approx 0$). Beachtet man ferner $u_s = 0 = w_b$, führt (2.20) auf

$$\begin{aligned} 0 &\approx \frac{\partial}{\partial t}\int_b^h \mathrm{d}z\,u + \int_b^h \mathrm{d}z\,\frac{\partial}{\partial x}\,u^2 + \frac{1}{\rho}\,\frac{\partial}{\partial x}\int_b^h \mathrm{d}z\,p \qquad (2.21)\\ &= \frac{\partial}{\partial t}\,(h-b)\,\bar{u} + \frac{\partial}{\partial x}\,(h-b)\,\overline{u^2} - u_s^2\,\frac{\partial h}{\partial x} + u_b^2\,\frac{\partial b}{\partial x} + \frac{1}{\rho}\,\frac{\partial}{\partial x}\,(h-b)\,\bar{p}\,. \end{aligned}$$

Mit Hilfe von $\frac{\partial b}{\partial x} \approx 0$, $u_s = 0$ kann (2.21) schließlich wie folgt geschrieben werden

$$0 \approx \frac{\partial}{\partial t}\,(h-b)\,\bar{u} + \frac{\partial}{\partial x}\left((h-b)\left(\overline{u^2} + \frac{1}{\rho}\,\bar{p}\right)\right)\,. \qquad (2.22)$$

Es gilt nun, den mittleren Druck $\bar{p}$ zu eliminieren. Dazu verwendet man die z-Komponente der Navier-Stokes-Gleichung (2.11) in der Form

$$\frac{\mathrm{d}\,w}{\mathrm{d}t} = -\frac{1}{\rho}\,\frac{\partial p}{\partial x} - g\,. \qquad (2.23)$$

Der Druck in Tiefe z' kann nun durch Integration dieser Gleichung von der Oberfläche $h(x,t)$ bis z' bestimmt werden:

$$\frac{1}{\rho}\,p(x,z',t) = \frac{1}{\rho}\,p_s - \int_h^{z'} \mathrm{d}z\,\left(g + \frac{\mathrm{d}\,w}{\mathrm{d}t}\right) = \int_{z'}^h \mathrm{d}z\,\left(g + \frac{\mathrm{d}\,w}{\mathrm{d}t}\right)\,, \qquad (2.24)$$

wobei wieder $p_s = 0$ eingesetzt wurde. Eine weitere Integration führt auf

$$\begin{aligned} \frac{1}{\rho}\,(h-b)\,\bar{p} &= \int_b^h \mathrm{d}z'\int_{z'}^h \mathrm{d}z\,\left(g + \frac{\mathrm{d}\,w}{\mathrm{d}t}\right)\\ &= \frac{1}{2}\,g\,(h-b)^2 + \int_b^h \mathrm{d}z\,(z-b)\,\frac{\mathrm{d}\,w}{\mathrm{d}t}\,. \qquad (2.25) \end{aligned}$$

Hierbei wurden die Integrationen im ersten Term einfach ausgeführt, im zweiten waren sie zuerst unter Beachtung von $z' \le z$ zu vertauschen.

Einsetzen von (2.25) in (2.22) führt schließlich auf

$$\frac{\partial}{\partial t}\,(h-b)\,\bar{u} + \frac{\partial}{\partial x}\left((h-b)\,\overline{u^2} + \frac{1}{2}\,g\,(h-b)^2 + \int_b^h \mathrm{d}z\;(z-b)\,\frac{\mathrm{d}\,w}{\mathrm{d}t}\right) \approx 0\,. \tag{2.26}$$

Hierbei handelt es sich um eine erste Gleichung, in der der Druck nicht mehr explizit auftritt. Zur weiteren Vereinfachung sind Näherungen erforderlich [20]. Erstens kann $\int_b^h \mathrm{d}z\;(z-b)\,\frac{\mathrm{d}\,w}{\mathrm{d}t}$ mit Hilfe von Identitäten wie (2.9) und (2.12) umgeschrieben werden auf höhere Ableitungen von $\bar{u}$ und h. In erster Näherung kann dieser Term dann vernachlässigt werden

$$\int_b^h \mathrm{d}z\;(z-b)\,\frac{\mathrm{d}\,w}{\mathrm{d}t} \approx 0\,. \tag{2.27}$$

Analog unterscheidet sich $\overline{u^2}$ von $\bar{u}^2$ lediglich durch höhere Ableitungen von $\bar{u}$, so daß $\overline{u^2} \approx \bar{u}^2$ genähert werden kann. Anschaulich kann diese Näherung über Bewegung in Scheiben gerechtfertigt werden (ist $u(x,z,t)$ unabhängig von z, gilt offensichtlich $\overline{u^2} = \bar{u}^2$).

Mit diesen Näherungen führt (2.26) auf die zweite Seichtwassergleichung

$$\frac{\partial}{\partial t}\left((h-b)\;\bar{u}\right) + \frac{\partial}{\partial x}\left((h-b)\;\bar{u}^2 + \frac{1}{2}\,g\,(h-b)^2\right) = 0\,. \tag{2.28}$$

Zusammen mit (2.19) bildet (2.28) eine Form des Seichtwassersystems. Zum Vergleich mit [13] schreiben wir (2.28) noch ein wenig um

$$\begin{aligned} 0 &= \bar{u}\frac{\partial}{\partial t}\,h + (h-b)\;\frac{\partial \bar{u}}{\partial t} + \bar{u}\,\frac{\partial}{\partial x}\left((h-b)\;\bar{u}\right) + (h-b)\;\bar{u}\,\frac{\partial \bar{u}}{\partial x} \\ &\qquad + g\,(h-b)\;\frac{\partial}{\partial x}\,(h-b) \\ &= (h-b)\;\frac{\partial \bar{u}}{\partial t} + (h-b)\;\bar{u}\,\frac{\partial \bar{u}}{\partial x} + (h-b)\;g\,\frac{\partial}{\partial x}\,(h-b)\;, \end{aligned} \tag{2.29}$$

wobei in der zweiten Zeile die erste Seichtwassergleichung (2.19) eingesetzt wurde. Mit der Näherung $\frac{\partial b}{\partial x} \approx 0$ erhält man die folgende alternative Form der zweiten Seichtwassergleichung

$$\frac{\partial \bar{u}}{\partial t} + \bar{u}\,\frac{\partial \bar{u}}{\partial x} + g\,\frac{\partial h}{\partial x} = 0\,. \tag{2.30}$$

In (2.19) zusammen mit (2.30) erkennt man den Spezialfall verschwindender y-Abhängigkeit, d.h. insbesondere mittlere y-Geschwindigkeit $\bar{v} = 0$,

des allgemeinen Seichtwassersystems wieder, das den Ausgangspunkt von [13] bildet:

$$\begin{aligned} \frac{\partial \bar{u}}{\partial t} + \bar{u}\,\frac{\partial \bar{u}}{\partial x} + \bar{v}\,\frac{\partial \bar{u}}{\partial y} + g\,\frac{\partial h}{\partial x} &= 0\,, \\ \frac{\partial \bar{v}}{\partial t} + \bar{u}\,\frac{\partial \bar{v}}{\partial x} + \bar{v}\,\frac{\partial \bar{v}}{\partial y} + g\,\frac{\partial h}{\partial y} &= 0\,, \\ \frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\left((h-b)\;\bar{u}\right) + \frac{\partial}{\partial y}\left((h-b)\;\bar{v}\right) &= 0\,. \end{aligned} \tag{2.31}$$

Partielle Differentialgleichungen dieses Typs besitzen spezielle Lösungen, sogenannte „Solitonen". Hierbei handelt es sich um lokale Störungen, die sich unter Beibehaltung ihrer Form ausbreiten (im konkreten Fall für konstante Meerestiefe b). Bei Tsunamis handelt es sich um solche Solitonen. Ein erste wichtige Frage ist die Bestimmung der Ausbreitungsgeschwindigkeit. Eine langsam abnehmende Wassertiefe $h - b$ führt schließlich zu einer Formveränderung, d.h. insbesondere einem Anwachsen der Wellenhöhe. Gleichzeitig verlangsamt sich der Tsunami glücklicher Weise in Küstennähe. Diese Kopplung von Höhe und Ausbreitungsgeschwindigkeit kann leicht mit Erhaltung der Energie der Störung der Wasseroberfläche begründet werden.

2.2 Projekt

Mit Hilfe der Seichtwassergleichungen soll nun das Tsunami-Phänomen simuliert werden. Dazu betrachten wir ein kleines Meeresbecken zwischen $-x_0$ und x_0, das wir als Parabel annehmen:

$$b(x) = b_0\,\left(\frac{x^2}{x_0^2} - 1\right)\,. \tag{2.32}$$

Bei $x = \pm x_0$ können Null-Randbedingungen angenommen werden

$$h(x = \pm x_0, t) = 0\,, \qquad \bar{u}(x = \pm x_0, t) = 0\,. \tag{2.33}$$

Wir betrachten eine anfängliche statische Störung in der Mitte des Meeres. Die Anhebung der Oberfläche wird als Gaußkurve mit Höhe h_0 und Breite D angesetzt

$$h(x, t = 0) = h_0\,\mathrm{e}^{-x^2/(2\,D)}\,, \tag{2.34}$$

die mittlere Anfangsgeschwindigkeit wird zu Null angenommen

$$\bar{u}(x, t = 0) = 0\,. \tag{2.35}$$

A2.1* Simulieren Sie die Zeitentwicklung für die Seicherwassergleichungen (2.19), (2.30) mit (2.33) sowie (2.35) mit Hilfe des Leapfrog-Verfahrens (1.28) ! Wählen Sie in (2.32) die Meeresbreite zu $x_0 = 10\text{km}$ und die maximale Meerestiefe zu $b_0 = 1\text{km}$, sowie in (2.34) die Höhe der Anfangs-Auslenkung $h_0 = 1\text{m}$ bei einer Breite $D = 1\text{km}^2$. Für die Erbeschleunigung kann $g = 10\frac{\text{m}}{\text{s}^2}$ verwendet werden. Geeignete Parameter für die Diskretisierung sind $\Delta x = 1\text{m}$ und $\Delta t = 0.001\text{s}$.
Verfolgen Sie das Maximum $h(x_{\max}, t)$ bei $x = x_{\max}$ und betrachten Sie die Zeitentwicklung seines Wertes ! Schätzen Sie aus $x_{\max}(t)$ ferner die Geschwindigkeit c des Tsunamis ! Was beobachten Sie für $x_{\max}(t) \to \pm x_0$?
Bemerkung: Die Simulation sollte spätestens dann abgebrochen werden, wenn $x_{\max}(t)$ sich der Küste bei $\pm x_0$ auf weniger als einen Gitterabstand Δx genähert hat.

3 Zeitentwicklung in der Quantenmechanik

In diesem Kapitel soll die zeitabhängige *Schrödingergleichung*

$$i\hbar \frac{\partial \Psi(\vec{r},t)}{\partial t} = \mathcal{H}\,\Psi(\vec{r},t) \tag{3.1}$$

numerisch behandelt werden. Durch Diskretisierung der Ortsvariablen wird aus dem Hamilton-Operator $\mathcal{H}$ eine hermitesche Matrix (bei der Diskretisierung ist darauf zu achten, daß die Hermitizität erhalten bleibt). $\mathcal{H}$ besitzt somit einen vollständigen Satz von Eigenfunktionen Ψ_j zu Energie E_j:

$$\mathcal{H}\,\Psi_j = E_j\,\Psi_j\,. \tag{3.2}$$

Die Anzahl dieser Eigenvektoren ist endlich und durch die Dimension des diskretisierten Problems gegeben. Ferner sind die Eigenfunktionen paarweise orthogonal[10]

$$\int \mathrm{d}\vec{r}\,\Psi_j^\star\,\Psi_k = \delta_{j,k}\,. \tag{3.3}$$

Eine erste Möglichkeit zur Berechnung der Zeitentwicklung ist die explizite Bestimmung aller Eigenvektoren (3.2), d.h. die vollständige numerische Diagonalisierung von $\mathcal{H}$. Eine gegebene Anfangswellenfunktion kann dann (z.B. mit Hilfe von (3.3)) nach Eigenfunktionen entwickelt werden

$$\Psi(\vec{r},0) = \sum_j a_j\,\Psi_j\,. \tag{3.4}$$

Mit Hilfe von (3.2) kann die Lösung von (3.1) nun leicht angegeben werden

$$\Psi(\vec{r},t) = \sum_j a_j\,\mathrm{e}^{-i\,E_j\,t/\hbar}\,\Psi_j\,. \tag{3.5}$$

Man beachte, daß in einem solchen Zugang die Zeitentwicklung nicht diskretisiert werden muß. Allerdings bedeutet die vollständige Diagonalisierung von $\mathcal{H}$ einen erheblichen Aufwand.

3.1 Differenzenverfahren

In Kapitel 1 wurden verschiedene numerische Verfahren zur Berechnung der diskretisierten Zeitentwicklung partieller Differentialgleichungen vorgestellt.

[10] Hier und im folgenden wird das Skalarprodukt symbolisch als Integral geschrieben; in der konkreten Realisierung ist jedoch eine endliche Summe auszuführen.

Diese sollen nun zuerst mit Blick auf Ihre Anwendbarkeit für die Schrödingergleichung (3.1) diskutiert werden. Zuerst stellt man fest, daß (3.1) in der Form (1.1) geschrieben werden kann

$$\frac{\partial \Psi(\vec{r},t)}{\partial t} = -\frac{i}{\hbar}\,\mathcal{H}\,\Psi(\vec{r},t) = \mathcal{L}\,(\Psi)\ . \tag{3.6}$$

Man beachte, daß die Quantenmechanik eine lineare Theorie ist, d.h. $\mathcal{L} = -\frac{i}{\hbar}\,\mathcal{H}$ ist hier ein linearer Operator ! Nun betrachten wir die Wellenfunktion zu diskreten Zeiten t_n, wir sie insbesondere durch (1.4) gegeben sind. Eine wesentliche Eigenschaft der Quantenmechanik ist die Unitarität der Zeitentwicklung, d.h. die Erhaltung von Wahrscheinlichkeiten. Für die diskretisierte Zeitentwicklung bedeutet dies, daß die Norm der Wellenfunktion erhalten bleiben muß

$$\|\Psi(t_{n+1})\| = \|\Psi(t_n)\|\ , \tag{3.7}$$

und zwar für alle $\Psi(t_n)$. Es reicht also nicht, Stabilität des Verfahrens sicherzustellen, sondern es ist die im allgemeinen stärkere Bedingung (3.7) zu erfüllen. Man beachte, daß es sich hierbei um eine nicht-lineare Bedingung handelt, da in der Quantenmechanik die Wellenfunktion selbst nicht meßbar ist, sondern nur ihr Quadrat (die Wahrscheinlichkeitsdichte).

Zuerst betrachten wir das *explizite Euler-Verfahren* (1.7)

$$\Psi(t_{n+1}) = \Psi(t_n) - \Delta t\,\frac{i}{\hbar}\,\mathcal{H}\,\Psi(t_n)\,. \tag{3.8}$$

Nun setzt man eine beliebige Eigenfunktion (3.2) ein $\Psi(t_n) = \Psi_j$. Dann gilt

$$\Psi(t_{n+1}) = \left(1 - \Delta t\,\frac{i}{\hbar}\,E_j\right)\,\Psi_j\,, \tag{3.9}$$

folglich

$$\|\Psi(t_{n+1})\|^2 = \left(1 + \left(\Delta t\,\frac{i}{\hbar}\,E_j\right)^2\right)\,\|\Psi_j\|^2\ . \tag{3.10}$$

Der Vorfaktor auf der rechten Seite ist immer größer als 1, d.h. (3.7) ist immer verletzt. Das explizite Euler-Verfahren ist also auch in der Quantenmechanik kein brauchbares Verfahren.

Das erste stabile Verfahren war das *explizite Lax-Verfahren* (1.20). Aufgrund der räumlichen Mittelung kann es nicht allgemein analysiert werden. Wir betrachten somit ein freies Teilchen

$$\mathcal{H} = -\frac{\hbar^2}{2m}\,\Delta\,. \tag{3.11}$$

Man beachte, daß dies mit (0.8) für die Diskretisierung der zweiten Ableitung in einer Dimension auf

$$(\mathcal{H}\,\Psi)_r = -\frac{\hbar^2}{2m}\,\frac{\Psi_{r+1} - 2\,\Psi_r + \Psi_{r-1}}{\Delta x^2} \tag{3.12}$$

führt. Die Eigenfunktionen lauten nun

$$\Psi_r^{(n)} = g_n\,\mathrm{e}^{i\,k\,r\,\Delta x}\,. \tag{3.13}$$

Das explizite Lax-Verfahren (1.20) führt nun auf

$$\begin{aligned}\Psi_r^{(n+1)} &= \frac{1}{2}\left(\Psi_{r+1}^{(n)} + \Psi_{r-1}^{(n)}\right) - \Delta t\,\frac{i}{\hbar}\,\mathcal{H}\,\Psi_r^{(n)} \\ &= \left(\cos\left(k\Delta x\right) - \frac{\Delta t\,i}{\hbar}\,\frac{\hbar^2}{2m}\,\frac{2\left(\cos\left(k\Delta x\right) - 1\right)}{\Delta x^2}\right)\,\Psi_r^{(n)}\,,\end{aligned} \tag{3.14}$$

wobei in der zweiten Zeile (3.12) eingesetzt wurde. Für die Normen folgt

$$\left\|\Psi^{(n+1)}\right\|^2 = \left(\cos^2\left(k\Delta x\right) + \left(\frac{\Delta t\,\hbar}{m}\,\frac{\left(\cos\left(k\Delta x\right) - 1\right)}{\Delta x^2}\right)^2\right)\,\left\|\Psi^{(n)}\right\|^2\,. \tag{3.15}$$

Der Vorfaktor auf der rechten Seite ist im allgemeinen von 1 verschieden, d.h. (3.7) ist nicht erfüllt. Da die Zeitentwicklung des expliziten Lax-Verfahrens nicht einmal für ein freies Teilchen unitär ist, ist dieses Verfahren in der Quantenmechanik unbrauchbar.

Der Favorit unter den expliziten Verfaren war das *Leapfrog-Verfahren* (1.28). Hier führt es auf

$$\Psi(t_{n+1}) = \Psi(t_{n-1}) - 2\,\Delta t\,\frac{i}{\hbar}\,\mathcal{H}\,\Psi(t_n)\,. \tag{3.16}$$

Wir betachten wieder eine beliebige Eigenfunktion (3.2) $\Psi(t_n) = g_n\,\Psi_j$. Dann führt (3.16) auf die Rekursionsrelation

$$g_{n+1} = g_{n-1} - 2\Delta t\,\frac{i}{\hbar}\,E_j\,g_n\,. \tag{3.17}$$

Setzt man $g_n = g^n\,g_0$ an, so erhält man die quadratische Gleichung

$$g^2 + 2\Delta t\,\frac{i}{\hbar}\,E_j\,g - 1 = 0\,. \tag{3.18}$$

Mit der Notation $A_j := \frac{\Delta t}{\hbar}\,E_j$ sind die Lösungen von (3.18)

$$g = -i\,A_j \pm \sqrt{-A_j^2 + 1}\,. \tag{3.19}$$

Beachtet man nun, daß A_j reell ist und daß die Unitarität (3.7) $|g|^2 = 1$ bedeutet, so kann dies nach (3.19) für $|A_j| \leq 1$ sichergestellt werden. Setzt man die Definition von A_j ein, folgt

$$\Delta t \leq \frac{\hbar}{|E_j|}\,. \tag{3.20}$$

Aufgrund der Diskretisierung hat $\mathcal{H}$ nur endlich viele Eigenwerte, $|E_j|$ ist nach oben beschränkt und (3.20) kann für alle j sichergestellt werden, wenn man Δt hinreichend klein wählt (vgl. u.a. Kapitel V von [22]). Da Unitarität $|g|^2 = 1$ sicherstellt, kann die Amplitude g_n nicht anwachsen. Das explizite Leapfrog-Verfahren ist somit automatisch stabil, wenn (3.20) erfüllt ist.

In der Literatur werden teilweise implizite Verfahren für die Anwendung auf die Zeitentwicklung in der Quantenmechanik empfohlen (vgl. z.B. Kapitel 19.2 von [5] und Kapitel 7.5 von [23]). Insbesondere führt das *implizite Euler-Verfahren 2. Ordnung* (1.12) auf

$$\left(1 + \frac{\Delta t}{2\hbar}\, i\,\mathcal{H}\right)\, \Psi(t_{n+1}) = \left(1 - \frac{\Delta t}{2\hbar}\, i\,\mathcal{H}\right)\, \Psi(t_n)\,. \tag{3.21}$$

Schreibt man dies als $\Psi(t_{n+1}) = U(\Delta t)\,\Psi(t_n)$, so erkennt man in $U(\Delta t) = \left(\mathbb{1} + \frac{\Delta t}{2\hbar}\, i\,\mathcal{H}\right)^{-1}\left(\mathbb{1} - \frac{\Delta t}{2\hbar}\, i\,\mathcal{H}\right)$ die Cayley-Transformierte von $\mathcal{H}$ wieder. Die durch (3.21) gegebene Zeitentwicklung ist somit unitär. Um dies auf einem elementaren Niveau nachzuvollziehen, betrachtet man wieder Eigenfunktionen $\Psi(t_n) = g_n\,\Psi_j$. Die Relation (3.21) führt dann mit (3.2) auf

$$\Psi(t_{n+1}) = \frac{1 - \frac{\Delta t}{2\hbar}\, iE_j}{1 + \frac{\Delta t}{2\hbar}\, i\,E_j}\, \Psi(t_n)\,. \tag{3.22}$$

Der Vorfaktor in (3.22) ist der Quotient zweier zueinander komplex konjugierter Zahlen, hat also den Betrag 1. Es folgt, daß die Vorschrift (3.21) *immer* unitär ist. Aus der Normerhaltung folgt auch hier automatisch Stabilität des Verfahrens. Glg. (3.21) führt auf ein lineares Gleichungssystem (0.20) mit $A = \mathbb{1} + \frac{\Delta t}{2\hbar}\, i\,\mathcal{H}$. Auch wenn die Diskretisierung des Hamilton-Operators $\mathcal{H}$ hermitesch ist, ist der Operator A aufgrund des Faktors i manifest nicht-hermitesch. Das Gleichungssystem (3.21) ist also mit dem in Abschnitt 1.4.1 diskutierten Bi-Conjugate Gradient Verfahren zu lösen. Ferner kann die Diskretisierung $\mathcal{H}$ zwar häufig reell gewählt werden, jedoch führt der Faktor i auf eine explizit komplexe Matrix. Dieses Verfahren wird somit naheliegender Weise mit komplexer Arithmetik implementiert.

Zusammenfassend läßt sich sagen, daß unter den bisher diskutierten Verfahren weder das explizite Euler-Verfahren noch das Lax-Verfahren unitär

sind, also für die Zeitentwicklung der Schrödingergleichung (3.1) untauglich sind. Das Leapfrog-Verfahren (3.16) kann auch in der Quantenmechanik eingesetzt werden, sofern der Zeitschritt Δt so klein gewählt wird, daß (3.20) erfüllt ist. Das implizite Euler-Verfahren 2. Ordnung (3.21) kann für beliebige Δt verwendet werden, ohne daß prinzipielle Probleme auftreten (natürlich hängt die Genauigkeit der genäherten Zeitentwicklung von Δt ab).

3.1.1 Explizites Verfahren nach Visscher

Der höhere Rechenaufwand impliziter Verfahren mag unbefriedigend sein. Ferner ist es bequem, das Verfahren so zu formulieren, daß reelle Arithmetik verwendet werden kann. So wird z.B. in Kapitel 18.4 von [1] die Verwendung eines von Visscher vorgeschlagenen expliziten Verfahrens [24] empfohlen. Dazu wird die Wellenfunktion zuerst als

$$\Psi(\vec{r},t) = R(\vec{r},t) + i\, I(\vec{r},t) \tag{3.23}$$

geschrieben, wobei R und I *reell* gewählt werden. Die Zeitentwicklung (3.6) führt nun auf

$$\frac{\partial}{\partial t} R = \frac{1}{\hbar}\mathcal{H}\, I\,, \qquad \frac{\partial}{\partial t} I = -\frac{1}{\hbar}\mathcal{H}\, R\,. \tag{3.24}$$

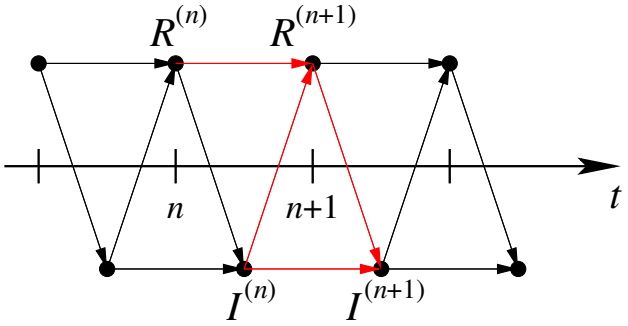
Visscher [24] schlägt nun vor, aus Stabilitätsgründen Halbschritte $\Delta t/2$ zu verwenden, sowie Realteil R und Imaginärteil I zu um $\Delta t/2$ versetzten Zeipunkten zu betrachten

$$R^{(n)} = R(t_n)\,, \qquad I^{(n)} = I(t_n + \Delta t/2)\,. \tag{3.25}$$

Die diskretisierte Version der Zeitentwicklung (3.24) lautet nun

$$\begin{aligned} R^{(n+1)} &= R^{(n)} + \frac{\Delta t}{\hbar}\mathcal{H}\, I^{(n)}\,, \\ I^{(n+1)} &= I^{(n)} - \frac{\Delta t}{\hbar}\mathcal{H}\, R^{(n+1)}\,. \end{aligned} \tag{3.26}$$

Diese Entwicklungsvorschrift wird mit folgendem Diagramm veranschaulicht:



Da R und I nicht gleichzeitig vorliegen, ist die Definition der Wahrscheinlichkeitsdichte $P = R^2 + I^2$ nicht eindeutig. Eine mögliche Wahl, die den richtigen Kontinuumsgrenzfall besitzt, ist

$$\begin{aligned} P(\vec{r}, t_n) &= \left(R^{(n)}\right)^2 + I^{(n)}\, I^{(n-1)}\,, \\ P(\vec{r}, t_n + \Delta t/2) &= R^{(n+1)}\, R^{(n)} + \left(I^{(n)}\right)^2\,. \end{aligned} \tag{3.27}$$

Diese Wahl bedeutet, für Realteil bzw. Imaginärteil das Quadrat zu nehmen, sofern es an diesem Zeitpunkt verfügbar ist, andernfalls das Produkt der Werte einen halben Zeitschreit zuvor sowie später.

Es gilt nun, Unitarität der Zeitentwicklung zu zeigen. Dazu setzt man (3.26) in (3.27) ein

$$\begin{aligned} P(\vec{r}, t_n + \Delta t/2) &= \left(R^{(n)} + \frac{\Delta t}{\hbar}\,\mathcal{H}\, I^{(n)}\right) R^{(n)} + I^{(n)} \left(I^{(n-1)} - \frac{\Delta t}{\hbar}\,\mathcal{H}\, R^{(n)}\right) \\ &= P(\vec{r}, t_n) + \left(\frac{\Delta t}{\hbar}\,\mathcal{H}\, I^{(n)}\right) R^{(n)} - \frac{\Delta t}{\hbar}\, I^{(n)}\,\mathcal{H}\, R^{(n)}\,. \end{aligned} \tag{3.28}$$

Der Übergang zur Gesamtwahrscheinlichkeit, der hier durch ein Integral angedeutet wird, sowie Hermitizität des Hamilton-Operators $\mathcal{H}$ führt nun auf

$$\begin{aligned} \int \mathrm{d}\vec{r}\, P(\vec{r}, t_n + \Delta t/2) &= \int \mathrm{d}\vec{r}\, P(\vec{r}, t_n) + \frac{\Delta t}{\hbar} \int \mathrm{d}\vec{r}\, I^{(n)}\,\mathcal{H}\left(R^{(n)} - R^{(n)}\right) \\ &= \int \mathrm{d}\vec{r}\, P(\vec{r}, t_n)\,. \end{aligned} \tag{3.29}$$

Analog kann gezeigt werden, daß

$$\int \mathrm{d}\vec{r}\, P(\vec{r}, t_{n+1}) = \int \mathrm{d}\vec{r}\, P(\vec{r}, t_n + \Delta t/2)\,, \tag{3.30}$$

und somit, daß die Zeitentwicklung mit der Definition (3.27) für beliebige Δt ,,unitär" ist.

Man beachte, daß die Wahrscheinlichkeitsdichte nach (3.27) nicht einfach die Summe von Quadraten ist, sondern Produkte zu unterschiedlichen Zeiten auftreten. ,,Unitarität" stellt also hier nicht unbedingt Stabilität sicher. So könnte z.B. das Integral über $I^{(n)}\, I^{(n-1)}$ groß und negativ sein, so daß ein großes positives Integral über $\left(R^{(n)}\right)^2$ kompensiert wird. Zur weiteren Diskussion betrachten wir ein reelles $\mathcal{H}$ und Wellenfunktionen, die jeweils Eigenfunktionen zur gleichen Energie E_j sind, d.h. $\mathcal{H}\, R_j^{(n)} = E_j\, R_j^{(n)}$ und

$\mathcal{H}\, I_j^{(n)} = E_j\, I_j^{(n)}$. Die Vorschrift (3.26) führt nun auf

$$\begin{aligned} R_j^{(n+1)} &= R_j^{(n)} + \frac{\Delta t}{\hbar}\, E_j\, I_j^{(n)}\,, \\ I_j^{(n+1)} &= I_j^{(n)} - \frac{\Delta t}{\hbar}\, E_j\, R_j^{(n+1)}\,. \\ &= \left(1 - \left(\frac{\Delta t}{\hbar}\, E_j\right)^2\right) I_j^{(n)} - \frac{\Delta t}{\hbar}\, E_j\, R_j^{(n)}\,. \end{aligned} \tag{3.31}$$

Mit der Abkürzung $a_j = \frac{\Delta t}{\hbar}\, E_j$ kann (3.31) auch als

$$\begin{pmatrix} R_j^{(n+1)} \\ I_j^{(n+1)} \end{pmatrix} = \begin{pmatrix} 1 & a_j \\ -a_j & 1 - a_j^2 \end{pmatrix} \begin{pmatrix} R_j^{(n)} \\ I_j^{(n)} \end{pmatrix} \tag{3.32}$$

geschrieben werden. Die Eigenwerte der Matrix in (3.32) sind

$$\lambda_\pm = 1 - \frac{a_j^2}{2} \pm \frac{a_j^2}{2}\sqrt{1 - \frac{4}{a_j^2}}\,. \tag{3.33}$$

Da man Δt klein wählen möchte, sind zumindest einige a_j klein. Sind alle $|a_j|^2 \leq 4$, so sind die Eigenwerte (3.33) komplex mit Betrag

$$|\lambda_\pm|^2 = 1 - 2\, a_j^2 + \frac{a_j^4}{2}\,. \tag{3.34}$$

Gilt $|\lambda_\pm|^2 \leq 1$, so wächst die Lösung nicht an, d.h. das Verfahren ist stabil. Dies gilt nach (3.34) für $|a_j|^2 \leq 4$, d.h. mit der Definition $a_j = \frac{\Delta t}{\hbar}\, E_j$, wenn für alle j die folgende Ungleichung erfüllt ist

$$\Delta t \leq \frac{2\,\hbar}{|E_j|}\,. \tag{3.35}$$

Man beachte die Ähnlichkeit von (3.35) mit der Unitaritätsbedingung (3.20) für das Leapfrog-Verfahren ! Beim Visscher-Verfahren kann Δt folglich doppelt so groß gewählt werden, wie beim Leapfrog-Verfahren, d.h. der Rechenaufwand ist entsprechend geringer. Ferner ist das Verfahren (3.26) so formuliert, daß es reell implementiert werden kann.

3.1.2 Aufgabe

A3.1 Wir betrachten den diskretisierten Hamilton-Operator für ein freies Teilchen (3.12) auf einem Intervall $[0, L]$ der Länge L mit periodischen Randbedingungen, d.h.

$$\Psi(x + L, t) = \Psi(x, t)\,. \tag{3.36}$$

Implementieren Sie für diesen Fall die Zeitentwicklung mit dem expliziten Verfahren nach Visscher (vgl. Abschnitt (3.1.1) mit einer Auflösung $\Delta x = L/256$! Als Anfangsbedingung sei ein um x_0 zentriertes Gaußsches Wellenpaket gegegeben

$$\Psi_{x_0,k_0}(x, t = 0) = C\,\mathrm{e}^{-\frac{(x-x_0)^2}{\sigma^2}}\,\mathrm{e}^{i\,k_0\,x}\,. \tag{3.37}$$

Als Parameter betrachten wir hierbei $x_0 = L/2$, $\sigma = L/8$, $k_0 = 10\pi/L$. Für den Realteil bedeutet (3.4)

$$R^{(0)}(x) = C\,\mathrm{e}^{-\frac{(x-x_0)^2}{\sigma^2}}\,\cos\left(k_0\,x\right)\,. \tag{3.38}$$

a. Überzeugen Sie sich, daß (3.37) seine Form beibehält, wobei sich der Schwerpunkt $x_0(t) = x_0 + \langle v\rangle t$ mit Geschwindigkeit

$$\langle v\rangle = \frac{\hbar\,k_0}{m}\,, \tag{3.39}$$

bewegt !

b. Zur Bestimmung von $I^{(0)}$ wird die Zeitentwicklung über einen halben Zeitschritt benötigt. Hier tritt erstens ein Phasenfaktor $\mathrm{e}^{-i\,E\,t/\hbar}$ auf. Schätzen Sie $E = E(k_0) = \frac{\hbar^2\,k_0^2}{2\,m}$! Verwenden Sie diesen Phasenfaktor sowie die Bewegung des Schwerpunktes nach (3.39) zur Bestimmung von $I^{(0)}$!

c. Testen Sie, mit welcher Wahl von Δt die Zeitentwicklung stabil ist !

d. Die Anfangsbedingungen $\tilde{\Psi}(x, t = 0) = \Psi_{x_0,-k_0}(x, t = 0)$ ($\tilde{R}^{(0)}(x) = R^{(0)}(x)$, $\tilde{I}^{(0)}(x) = -I^{(0)}(x)$) und $\bar{\Psi}(x, t = 0) = \Psi_{x_0,0}(x, t = 0)$ ($\bar{R}^{(0)}(x) = C\,\mathrm{e}^{-\frac{(x-x_0)^2}{\sigma^2}}$, $\bar{I}^{(0)}(x) = 0$) führen auf (praktisch) die gleiche Anfangswahrscheinlichkeitsdichte $P(\Delta t/2)$ wie $\Psi_{x_0,k_0}(x, t = 0)$. Führen Sie die Zeitentwicklung für diese modifizierten Anfangsbedingungen durch und diskutieren Sie die drei Simulationen im Vergleich !

3.1.3 Freies Teilchen

Wir wollen das Problem eines freien Teilchens (3.11) etwas genauer untersuchen, wobei konkret eine Dimension diskutiert wird.

Zuerst betrachten wir das Problem auf dem Gitter. Gegeben sei ein endliches Intervall $[0, L]$ der Länge L, das in Teil-Intervalle der Länge Δx unterteilt sei. Die nach (0.5) diskretisierten Funktionswerte sind dann

$$\Psi_r = \Psi(r\Delta x)\,, \qquad r = 0, \ldots, \frac{L}{\Delta x} - 1\,. \tag{3.40}$$

Ferner werden periodische Randbedigungen (3.36) gefordert, d.h. $\Psi(x+L) = \Psi(x)$. Die diskretisierte Fassung des Hamilton-Operators ist in (3.12) angegeben, die zugehörigen Eigenfunktionen sind (3.13), die wir hier als

$$\Psi_r(k) = \mathrm{e}^{i\,k\,r\Delta x}\,. \tag{3.41}$$

schreiben. Periodizität erzwingt nun $kL = n\,2\pi$ mit ganzzahligem n. Von diesen Wellenvektoren sind $L/\Delta x$ unabhängig. Die unabhängigen Werte von k kann man wie folgt wählen:

$$k = \frac{n\,2\pi}{L}\,, \qquad n \in \mathbb{Z}\,, \qquad -\frac{\pi}{\Delta x} < k \leq \frac{\pi}{\Delta x}\,. \tag{3.42}$$

Man beachte, daß der Abstand benachbarter k-Punkte nur von der Intervall-Länge L abhängt, die Grenzen hingegen vom Gitterabstand Δx (für $\Delta x \to 0$ gehen die Grenzen in $-\infty < k \leq \infty$ über). Die zu den Eigenfunktionen (3.41) gehörigen Eigenwerte wurden bereits in (3.14) verwendet: Mit der Diskretisierung (3.12) gilt

$$\mathcal{H}\,\Psi(k) = -\frac{\hbar^2}{m}\,\frac{\cos\left(k\Delta x\right) - 1}{\Delta x^2}\,\Psi(k) = E(k)\,\Psi(k) \tag{3.43}$$

mit dem Eigenwert

$$E(k) = \frac{\hbar^2}{m}\,\frac{1 - \cos\left(k\Delta x\right)}{\Delta x^2}\,. \tag{3.44}$$

Betrachtet man nun $k\Delta x \ll 1$, so gilt $\cos\left(k\Delta x\right) = 1 - \frac{1}{2}\,\left(k\Delta x\right)^2 + \mathcal{O}\left(\left(k\Delta x\right)^4\right)$, d.h.

$$E(k) = \frac{\hbar^2}{2m}\,k^2 + \mathcal{O}\left(\left(k\Delta x\right)^4\right)\,. \tag{3.45}$$

Für kleine k und kleine Δx geht der Gitter-Ausdruck (3.44) also in das bekannte Kontinuums-Ergebnis über. Betrachtet man hingegen alle auf dem Gitter erlaubten Werte von k, gilt $0 \leq 1 - \cos\left(k\Delta x\right) \leq 2$, somit

$$0 \leq E(k) \leq \frac{2\hbar^2}{m\,\Delta x^2}\,. \tag{3.46}$$

Aufgrund der Diskretisierung ist das Spektrum begrenzt; der maximale Eigenwert wächst mit Δx^{-2} bei kleiner werdendem Gitterabstand. Setzt man schließlich (3.46) in die Stabilitätsbedingung (3.35) ein, so findet man

$$\Delta t \leq \frac{m\,\Delta x^2}{\hbar} \qquad \Leftrightarrow \qquad \frac{\hbar\,\Delta t}{m} \leq \Delta x^2\,. \tag{3.47}$$

In dem Verfahren nach Visscher ist also bei feiner werdendem Δx der Zeitschritt Δt entsprechend quadratisch kleiner zu wählen ! Nebenbei sei angemerkt, daß die Kombination $\hbar\,\Delta t/m$ eine günstige Wahl für die interne Implementierung eines Zeitschritts ist.

Wir kehren nun zum Kontinuum zurück ($\Delta x = 0$) und betrachten ferner die gesamte räumliche Achse $x \in \mathbb{R}$. Für die Beschreibung von Streu-Experimenten sind die Gaußschen Wellenpakete (3.37) von besonderem Interesse. Erstens beschreiben sie ein Teilchen, daß sowohl im Ort als auch im Impuls gleichzeitig mit minimaler Unschärfe lokalisiert ist. Zweitens sind Gauß-Integrale berechenbar, so daß mit Hilfe einer quadratischen Ergänzung die Fourier-Transformierte der Anfangsbedingungen (3.37) berechnet werden kann

$$\begin{aligned}
\hat{\Psi}_{x_0,k_0}(k,t=0) &= \frac{1}{\sqrt{2\pi}} \int \mathrm{d}x\, \Psi_{x_0,k_0}(x,t=0)\, \mathrm{e}^{-i\,k\,x} \\
&= \frac{C}{\sqrt{2\pi}} \int \mathrm{d}x\, \mathrm{e}^{-\frac{(x-x_0)^2}{\sigma^2}}\, \mathrm{e}^{i\,x\,(k_0-k)} \\
&= \frac{C\,\sigma}{\sqrt{2}}\, \mathrm{e}^{-\frac{\sigma^2}{4}\,(k-k_0)^2}\, \mathrm{e}^{i\,x_0\,(k_0-k)}\,.
\end{aligned} \tag{3.48}$$

Wie gewohnt findet man als Fourier-Transformierte einer Gaußfunktion wieder eine Gaußfunktion, deren Breite invers zu der ursprünglichen ist. Nach der Fourier-Transformation ist $\mathcal{H}$ diagonal mit Eigenwerten $E(k) = \frac{\hbar^2}{2m}\,k^2$ (siehe (3.45)). Somit kann im k-Raum die Zeitentwicklung der Wellenfunktion leicht angegeben werden

$$\hat{\Psi}_{x_0,k_0}(k,t) = \mathrm{e}^{-i\,E(k)\,t/\hbar}\,\hat{\Psi}_{x_0,k_0}(k,0) = \mathrm{e}^{-i\,\hbar k^2\,t/(2m)}\,\hat{\Psi}_{x_0,k_0}(k,0)\,. \tag{3.49}$$

Die Zeitentwicklung im Ortsraum findet man schließlich mit Hilfe der inversen Fourier-Transformation

$$\begin{aligned}
\Psi_{x_0,k_0}(x,t) &= \frac{1}{\sqrt{2\pi}} \int \mathrm{d}k\, \mathrm{e}^{i\,k\,x}\, \hat{\Psi}_{x_0,k_0}(k,t) \\
&= \frac{C\,\mathrm{e}^{\frac{-\left(x-x_0-\frac{k_0\,\hbar t}{m}\right)^2+\frac{k_0^2\,\hbar^2\,t^2}{m^2}}{\sigma^2+\frac{2\,i\hbar\,t}{m}}}\, \mathrm{e}^{\,i\left(k_0 x-\frac{k_0^2\,\sigma^2\,\hbar\,t}{2\,m\left(\sigma^2+\frac{2\,i\,\hbar\,t}{m}\right)}\right)}}{\sqrt{1+\frac{2\,i\,\hbar t}{\sigma^2\,m}}}\,.
\end{aligned} \tag{3.50}$$

Nähert man nun $\frac{k_0^2\,\sigma^2\,\hbar\,t}{m\left(\sigma^2+\frac{2\,i\,\hbar\,t}{m}\right)} \approx \frac{k_0^2\,\hbar\,t}{m}$, so geht der letzte Faktor in diesem Ausdruck in $\exp\left(i\,k_0\,\left(x-\frac{k_0\,\hbar\,t}{2\,m}\right)\right)$ über. Zur Interpretation identifiziert man $\bar{E}=\frac{\hbar^2\,k_0^2}{2\,m}$ als die mittlere Energie und $\langle v\rangle$ nach (3.39) als mittlere Geschwindigkeit in dem Wellenpaket. Mit weiteren Näherungen für die Zähler und unter Vernachlässigung des Terms $\frac{k_0^2\,\hbar^2\,t^2}{m^2}$ kann (3.50) dann wie folgt geschrieben werden:

$$\begin{aligned}\Psi_{x_0,k_0}(x,t) &\approx C\,\mathrm{e}^{-\frac{\left(x-x_0-\frac{\hbar\,k_0}{m}\,t\right)^2}{\sigma^2}}\,\mathrm{e}^{i\,k_0\left(x-\frac{k_0\,\hbar}{2\,m}\,t\right)}\\ &= C\,\mathrm{e}^{-i\,\bar{E}\,t/\hbar}\,\mathrm{e}^{-\frac{(x-x_0-\langle v\rangle\,t)^2}{\sigma^2}}\,\mathrm{e}^{i\,k_0\,x}\,. \qquad (3.51)\end{aligned}$$

Setzt man schließlich $t=\Delta t/2$ in (3.51) ein, so findet man die zweite Anfangsbedingung für das Visscher-Verfahren in Aufgabe A3.1:

$$I^{(0)}(x)=C\,\mathrm{e}^{-\frac{\left(x-x_0-\frac{\hbar\,k_0\,\Delta t}{2\,m}\right)^2}{\sigma^2}}\,\sin\left(k_0\,\left(x-\frac{k_0\,\hbar\,\Delta t}{4\,m}\right)\right)\,. \qquad (3.52)$$

3.1.4 Aufgabe

A3.2 a. Modifizieren Sie das Programm aus A3.1 so, daß es ein beliebiges Potential $V(x)$ zuläßt !

b. Wählen Sie als Parameter $\Delta x = L/512$ sowie für die Anfangsbedingungen ein Gaußsches Wellenpaket (3.37) mit $x_0 = L/5$, $\sigma = L/32$, $k_0 = 64\,\pi/L$! Wir betrachten nun eine Potential-Barriere

$$V(x)=\begin{cases}0 & \text{für } x<L/2,\\ V_0 & \text{für } L/2\le x\le L/2+a,\\ 0 & \text{für } x>L/2+a,\end{cases} \qquad (3.53)$$

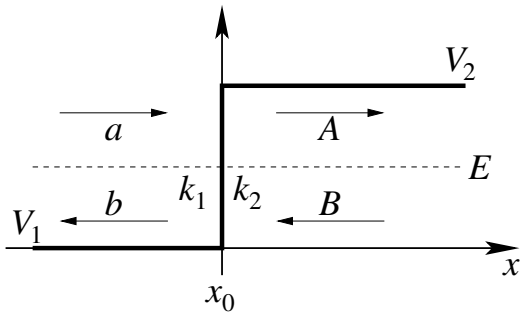
mit der Breite $a=L/64$ und der Höhe $V_0=2\cdot10^4\,\frac{\hbar^2}{m\,L^2}$. Simulieren Sie die Streuung des Wellenpaketes an dieser Rechteck-Barriere mit dem Verfahren nach Visscher bis zur Zeit $\hbar\,t/m=1/200$!

c. Berechnen Sie sowohl die Wahrscheinlichkeit $P_l(t)=\int_0^{L/2}\mathrm{d}x\,P(x,t)$ das Teilchen links von der Barriere zu finden als auch die Wahrscheinlichkeit $P_r(t)=\int_{L/2+a}^{L}\mathrm{d}x\,P(x,t)$ das Teilchen rechts von der Barriere zu finden ! Schätzen Sie die Reflektionswahrscheinlichkeit über $R=P_l(t_s)/P_l(0)$ sowie die Transmissionswahrscheinlichkeit über $T=P_r(t_s)/P_l(0)$ durch geeignete Wahl des Zeitpunkts t_s !

3.2 Stationäre Streutheorie

Dieses Kapitel soll einige Aspekte stationärer Streutheorie in einer Dimension kurz zusammenfassen. Im Gegensatz zur Zeitabhängigkeit von Wellenpaketen, die z.B. in Simulationen aber auch in realen Streuprozessen betrachtet werden, betrachtet man bei diesem Zugang stationäre Lösungen mit vorgegebener Asymptotik weit weg vom Potential, d.h. in einer Dimension ebene Wellen für $x \to \pm\infty$.

Zuerst wollen wir uns mit Stufenpotentialen beschäftigen, wobei insbesondere auch auf das Kapitel 1 von [25] verwiesen sei. Der erste Baustein für solche Potentiale ist die im folgenden skizzierte Stufe:



Die stationäre Wellenfunktion lautet in diesem Fall

$$\Psi(x) = \begin{cases} a\,\mathrm{e}^{i\,k_1\,(x-x_0)} + b\,\mathrm{e}^{-i\,k_1\,(x-x_0)} & \text{für } x \le x_0, \\ A\,\mathrm{e}^{i\,k_2\,(x-x_0)} + B\,\mathrm{e}^{-i\,k_2\,(x-x_0)} & \text{für } x \ge x_0. \end{cases} \tag{3.54}$$

Für $x \neq x_0$ ist dies eine Lösung der stationären Schrödingergleichung mit der Energie $E = \frac{\hbar^2\,k_j^2}{2\,m} + V_j$. Dies kann man auflösen nach

$$k_j = \frac{1}{\hbar}\,\sqrt{2\,m\;(E - V_j)}\,. \tag{3.55}$$

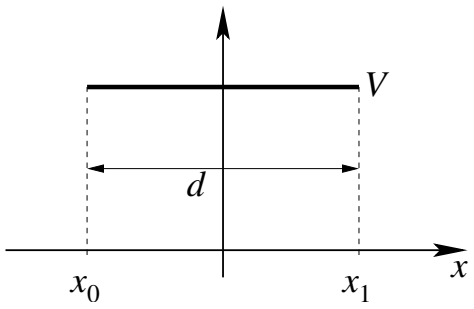
Da über das Vorzeichen von $E - V_j$ keine Annahme gemacht wurde, sind nach (3.55) sowohl die Fälle $k_j \in \mathbb{R}$ als auch $k_j \in i\,\mathbb{R}$ möglich. Bei x_0 sind nun ferner Anschlußbedingungen zu erfüllen. Aus der Stetigkeit von Ψ folgt $a+b = A+B$, aus der Stetigkeit der Ableitung $\frac{\partial\,\Psi}{\partial x}$ folgt $k_1\,(a-b) = k_2\,(A-B)$. Dies kann in Matrixschreibweise umgeschrieben werden als

$$\begin{pmatrix} a \\ b \end{pmatrix} = M_\Delta(k_1, k_2)\,\begin{pmatrix} A \\ B \end{pmatrix}\,, \tag{3.56}$$

wobei die sogenannte ,,Transfermatrix" für die Stufe bei x_0 gegeben ist durch

$$M_\Delta(k_1, k_2) = \frac{1}{2}\,\begin{pmatrix} 1 + \frac{k_2}{k_1} & 1 - \frac{k_2}{k_1} \\ 1 - \frac{k_2}{k_1} & 1 + \frac{k_2}{k_1} \end{pmatrix}\,. \tag{3.57}$$

Der zweite Baustein ist ein konstanter Abschnitt mit Potential V der Breite d:



Dies bewirkt in erster Linie eine Verschiebung des Ursprungspunktes. Die Wellenfunktion kann bezüglich des linken Punktes x_0 bzw. des rechten Punktes $x_1 = x_0 + d$ wie folgt geschrieben werden:

$$\begin{aligned} \Psi(x) &= a\,\mathrm{e}^{i\,k\,(x-x_0)} + b\,\mathrm{e}^{-i\,k\,(x-x_0)} = A\,\mathrm{e}^{i\,k\,(x-x_1)} + B\,\mathrm{e}^{-i\,k\,(x-x_1)} \\ &= A\,\mathrm{e}^{-i\,k\,d}\,\mathrm{e}^{i\,k\,(x-x_0)} + B\,\mathrm{e}^{i\,k\,d}\,\mathrm{e}^{-i\,k\,(x-x_0)}\,. \end{aligned} \tag{3.58}$$

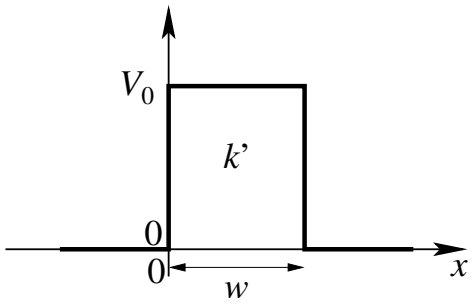
Dies kann in Matrixform gebracht werden

$$\begin{pmatrix} a \\ b \end{pmatrix} = M_d(k) \begin{pmatrix} A \\ B \end{pmatrix} \tag{3.59}$$

mit einer diagonalen Transfermatrix

$$M_d(k) = \begin{pmatrix} \mathrm{e}^{-i\,k\,d} & 0 \\ 0 & \mathrm{e}^{i\,k\,d} \end{pmatrix}\,. \tag{3.60}$$

Die Kombination dieser beiden Bausteine erlaubt es nun, kompliziertere Situationen zu diskutieren. Ein wichtiges Beispiel ist ein Kastenpotential der Breite w und Höhe V_0:



Setzt man die Wellenfunktion für $x \leq 0$ wie gewohnt als $\Psi(x) = a\,\mathrm{e}^{i\,k\,x} + b\,\mathrm{e}^{-i\,k\,x}$ und für $x \geq w$ als $\Psi(x) = A\,\mathrm{e}^{i\,k\,(x-w)} + B\,\mathrm{e}^{-i\,k\,(x-w)}$ an, führen die Anschlußbedingungen auf lineare Beziehungen zwischen a, b und A, B. Dies kann wieder in Matrixform geschrieben werden als

$$\begin{pmatrix} a \\ b \end{pmatrix} = M_V \begin{pmatrix} A \\ B \end{pmatrix} . \tag{3.61}$$

Mit Hilfe der bisherigen Ergebnisse finden wir

$$M_V = M_\Delta(k,k')\,M_w(k')\,M_\Delta(k',k) = \begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix} \tag{3.62}$$

Durch Einsetzen von (3.57) und (3.60) und ausmultiplizieren findet man insbesondere

$$\begin{aligned} m_{1,1} &= \cosh(i\,k'\,w) - \frac{1}{2}\left(\frac{k}{k'} + \frac{k'}{k}\right)\sinh(i\,k'\,w)\,, \\ m_{2,1} &= \frac{1}{2}\left(\frac{k}{k'} - \frac{k'}{k}\right)\sinh(i\,k'\,w) \end{aligned} \tag{3.63}$$

Man beachte, daß $k' \in i\,\mathbb{R}$ für $E \leq V_0$ und $k' \in \mathbb{R}$ für $E \geq V_0$ ist.

Zur Interpretation der Bedeutung dieses Ergebnisses für ein Streuexperiment betrachtet man nun rechts nur eine auslaufende Welle, d.h. $B = 0$. Die Amplitude der links einlaufenden Welle ist a, die Amplitude der links auslaufenden Welle b. Nach (3.61) und (3.62) gilt in diesem Fall $a = m_{1,1}\,A$, bzw. $A = t\,a$ mit dem Transmissionskoeffizienten

$$t = \frac{1}{m_{1,1}}\,. \tag{3.64}$$

Analog gilt für die Amplitude der links auslaufenden Welle

$$b = m_{2,1}\,A = m_{2,1}\,t\,a = r\,a\,, \tag{3.65}$$

woraus folgende Beziehung mit dem Reflektionskoeffizienten r folgt:

$$m_{2,1} = \frac{r}{t}\,. \tag{3.66}$$

Kennt man die Matrixelemente $m_{1,1}$ und $m_{2,1}$, so kann man also mit Hilfe von (3.64) und (3.66) die Werte von t und r bestimmen. Mit dem Ergebnis

(3.63) für den Potentialkasten findet man schließlich die *Transmissionswahrscheinlichkeit* $T = |t|^2$ zu

$$T^{-1} = |t|^{-2} = |m_{1,1}|^2 = 1 + \frac{1}{4} \sinh^2(i\,k'\,w) \left(2 - \left(\frac{k}{k'}\right)^2 - \left(\frac{k'}{k}\right)^2\right) . \quad (3.67)$$

Dieses Ergebnis gilt sowohl für $E \leq V_0$ wie auch für $E \geq V_0$, wobei auf den Faktor i in k' bzw. $i\,k'$ zu achten ist. Aufgrund von Wahrscheinlichkeitserhaltung gilt mit der Reflektionswahrscheinlichkeit R

$$T + R = 1 . \quad (3.68)$$

Im Zahlenbeispiel aus Aufgabe A3.2 gilt $k = k_0 = 64\,\pi/L$. Mit dem Wert von V_0 und der Beziehung (3.55) erhält man $k' = \sqrt{k_0^2 - 4 \cdot 10^4/L^2} \approx k_0/10$, wenn man eine ebene Welle mit der mittleren Wellenzahl k_0 des Wellenpakets betrachtet. Der Wert von k' ist reell, in diesem Fall ist also $E \geq V_0$, d.h. die mittlere Energie des einfallenden Wellenpaketes liegt geringfügig oberhalb der Potentialschwelle. Mit der Breite $w = L/64$ führt (3.67) zuerst auf $T^{-1} \approx 1 + 9.332/4 \approx 3.33$, d.h. eine Transmissionswahrscheinlichkeit $T \approx 0.3$. Aus (3.68) findet man dann die Reflektionswahrscheinlichkeit $R = 1 - T \approx 0.7$. Obwohl die Energie oberhalb der Potentialschwelle liegt, wird also trotzdem ein Großteil des einlaufenden Wellenpakets reflektiert.

Mit Hilfe der Transfermatrix-Formulierung kann man auch leicht ein allgemeines stückweise konstantes Potential beschreiben. Wir betrachten die Streuung an einem Potential, das aus N konstanten Abschnitten der Höhe V_j und Breite w_j besteht. Die Anschlußbedingungen für die Transmission durch ein solches Potential können wieder in der Form (3.61) geschrieben werden, wobei die Matrix M_V nun gegeben ist durch

$$M_V = \left(\prod_{j=1}^{N} M_\Delta(k_{j-1}, k_j)\; M_{w_j}(k_j)\right) M_\Delta(k_N, k_{N+1}) . \quad (3.69)$$

Diese Matrix kann wie oben interpretiert werden, d.h. die Zusammenhänge (3.64) und (3.66) gelten auch mit den Matrixelementen $m_{1,1}$ und $m_{2,1}$ der Matrix (3.69). Die Berechnung der Streuung an einem eindimensionalen Stufenpotential reduziert sich also auf die Berechnung eines Produktes explizit bekannter 2×2 Matrizen (siehe (3.57) und (3.60)). Eine solche Berechnung kann leicht mit Hilfe eines Computers durchgeführt werden. Da ein ganz allgemeines Potential durch ein stückweise konstantes Potential genähert werden kann, kann man auf die Weise die Transmissionswahrscheinlichkeit $T(E)$ für eine einfallende ebene Welle der Energie $E = \frac{\hbar^2 k_0^2}{2\,m}$ numerisch bestimmen.

Für eine analytische Diskussion allgemeiner Potentiale sind Näherungen nützlich. Wir wollen deswegen eine spezielle Näherung diskutieren (vgl. auch Kapitel 4.3.3 von [26] für die folgende Diskussion). Ziel dieser Näherung ist die Beschreibung des Tunnelns durch eine Potentialbarriere, d.h. die quantenmechanische Ausbreitung durch klassisch verbotene Bereiche. Dazu betrachten wir Energien E, die klein gegen das Potential V_0 sind, wir machen also die Annahme

$$i\,k'\,w = \frac{w}{\hbar}\,\sqrt{2\,m\,(V_0 - E)} \gg 1\,. \tag{3.70}$$

Mit dieser Annahme dominiert dann eine Exponentialfunktion in

$$\sinh^2\,(i\,k'\,w) \approx \frac{1}{4}\,\mathrm{e}^{2\,i\,k'\,w} \gg 1\,. \tag{3.71}$$

Die inverse Transmissionswahrscheinlichkeit (3.67) kann nun wie folgt genähert werden:

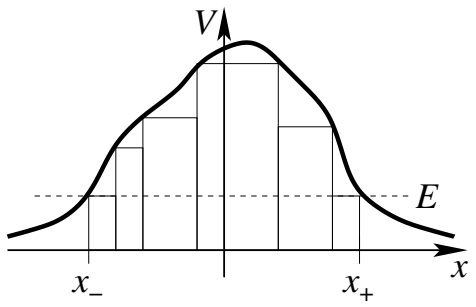
$$T(E)^{-1} \approx \frac{1}{16}\,\mathrm{e}^{2\,i\,k'\,w}\,\left(2 + \left(\frac{k}{i\,k'}\right)^2 + \left(\frac{i\,k'}{k}\right)^2\right) = \frac{1}{16}\,\mathrm{e}^{2\,i\,k'\,w}\,\frac{\left(k^2 + (i\,k')^2\right)^2}{(i\,k'\,k)^2}\,. \tag{3.72}$$

Durch Einsetzen von (3.70) finden wir für die Transmissionswahrscheinlichkeit weiter

$$\begin{aligned} T(E) &\approx \frac{16\,E\,(V_0 - E)}{V_0^2}\,\exp\left(-\frac{2}{\hbar}\,\sqrt{2\,m\,(V_0 - E)}\,w\right) \\ &= \exp\left(-\frac{2}{\hbar}\,\sqrt{2\,m\,(V_0 - E)}\,w + \ln\left(\frac{16\,E\,(V_0 - E)}{V_0^2}\right)\right) \\ &\approx \exp\left(-\frac{2}{\hbar}\,\sqrt{2\,m\,(V_0 - E)}\,w\right)\,, \end{aligned} \tag{3.73}$$

wobei im letzten Schritt der Logarithmus im Exponenten gegenüber dem deutlich größeren ersten Term vernachlässigt wurde.

Ein allgemeines Potential nähert man nun N durch konstante Stücke:



Ferner nimmt man an, daß die Gesamttransmissionswahrscheinlichkeit das Produkt der Transmissionswahrscheinlichkeiten der einzelnen Stücke ist. Dies ähnelt ein wenig den Produkten der Transfermatrizen in (3.69), allerdings vernachlässigt man nun alle Intereferenzterme. Mit dieser Näherung und (3.73) findet man

$$\begin{aligned} T(E) &\approx T_1(E)\,T_2(E)\cdots T_N(E) \\ &\approx \prod_{j=1}^{N} \exp\left(-\frac{2}{\hbar}\sqrt{2\,m\,(V_j-E)}\,w_j\right) \\ &= \exp\left(-\frac{2}{\hbar}\sum_{j=1}^{N}\sqrt{2\,m\,(V_j-E)}\,w_j\right)\,. \end{aligned} \tag{3.74}$$

Führt man schließlich den Kontinuumsgrenzfall $w_j \to 0$ aus, so gelangt man zur *Wenzel-Kramers-Brillouin-(WKB)-Näherung* für die Transmissionswahrscheinlichkeit

$$T(E) \approx \exp\left(-\frac{2}{\hbar}\int_{x_-}^{x_+} \mathrm{d}x\,\sqrt{2\,m\,(V(x)-E)}\right)\,. \tag{3.75}$$

Die Integrationsgrenzen $x_\pm$ sind hierbei die Umkehrpunkte der klassischen Bahnen, d.h. die Stellen, an denen $V(x_\pm) = E$ gilt. Man integriert in (3.75) also über den klassisch verbotenen Bereich mit $E \leq V(x)$. Das Ergebnis (3.75) wurde hier auf einem elementaren Weg hergeleitet. Allerdings liegt ein allgemeineres Näherungsverfahren zugrunde, nämlich eine semiklassische Näherung.

An (3.75) liest man qualitativ ab, daß die beim Tunneleffekt die Transmissionswahrscheinlichkeit $T(E)$ mit zunehmender Energie aus zwei Gründen zunimmt: Erstens wird $V(x) - E$ kleiner, zweitens wird der Integrationsbereich $x_+ - x_-$ kleiner. Umgekehrt nimmt $T(E)$ ab mit zunehmender Teilchenmasse m. Die Näherung (3.75) kann z.B. die Energieabhähngigkeit der Zerfallsprodukte beim α-Zerfall von Atomkernen erfolgreich beschreiben (siehe u.a. Kapitel 4.3.4 von [26]).

3.2.1 Aufgabe

A3.3 Als ganz grobe Näherung an ein Kernpotential betrachten wir ein abgeschnittenes Coulomb-Potential

$$V(x) = \begin{cases} 0 & \text{für } x < L/2, \\ \frac{V_0\, L}{x} & \text{für } x > L/2. \end{cases} \tag{3.76}$$

a. Simulieren Sie wie in A3.2 die Streuung eines Gaußschen Wellenpaketes (3.37) Wellenpaketes an diesem Potential ! Wählen Sie die gleichen Parameter wie in A3.2: $\Delta x = L/512$, $x_0 = L/5$, $\sigma = L/32$, $k_0 = 64\,\pi/L$! Betrachten Sie als Stärken des Potentials $V_0\, m\, L^2/\hbar^2 = 1.5 \cdot 10^4$, $2 \cdot 10^4$ und $3 \cdot 10^4$! Berechnen Sie sowohl die Wahrscheinlichkeit $P_l(t) = \int_0^{L/2} \mathrm{d}x\, P(x,t)$ das Teilchen links zu finden als auch die Wahrscheinlichkeit $P_r(t) = \int_{L/2}^{L} \mathrm{d}x\, P(x,t)$ das Teilchen rechts zu finden ! Schätzen Sie die Transmissionswahrscheinlichkeit T für alle drei Werte von V_0 wie in A3.2 durch eine geeignete Wahl von t !

b. Bestimmen Sie die Umkehrpunkte $x_\pm$ der klassischen Bahnen für die drei Werte $V_0\, m\, L^2/\hbar^2 = 1.5 \cdot 10^4$, $2 \cdot 10^4$ und $3 \cdot 10^4$! Schätzen Sie dann die Transmissionswahrscheinlichkeit $T(E)$ mit Hilfe der WKB-Näherung (3.75) ! Vergleichen Sie dieses Ergebnis quantitativ und qualitativ mit dem aus Aufgabenteil a !

c.** Nehmen Sie an, daß $V(x)$ auf Stücken der Länge $w_j = \Delta x$ konstant ist ! Berechnen Sie dann die Transmissionswahrscheinlichkeit $T = |m_{1,1}|^{-2}$ der Anordnung, indem Sie das Produkt (3.69) der Transfermatrizen ausrechnen ! Vergleichen Sie dieses Ergebnis quantitativ mit denen aus Aufgabenteil a und b !
Hinweis: Hier bietet sich der Einsatz komplexer Arithmetik an. Einige grundlegende Bemerkungen dazu finden Sie in Anhang B.

3.3 Operator-Splitting

Im folgenden soll ein weiteres Verfahren vorgestellt werden, das konzeptionell anders arbeitet und somit ggfs. günstigere Merkmale aufweist als die bisher diskutierten. Insbesondere ist die zeitversetzte Definition (3.25) von Realteil und Imaginärteil im Verfahren nach Visscher (siehe Kapitel 3.1.1) störend, will man z.B. einer Wellenfunktion eine bestimmte Energie zuordnen.

Ausgangspunkt ist die *Campbell-Baker-Hausdorff-Formel* für die Exponentialfunktion der Summe von zwei Operatoren A, B:

$$\mathrm{e}^{A+B} = \mathrm{e}^{A}\,\mathrm{e}^{B}\,\mathrm{e}^{-[A,B]/2}\,. \tag{3.77}$$

Man beachte den zusätzlichen Faktor, der für nicht-kommutierende Größen auftritt $[A,B] \neq 0$. In Kombination mit

$$\mathrm{e}^{A+B} = \left(\mathrm{e}^{\frac{1}{n}(A+B)}\right)^{n} \tag{3.78}$$

(n z.B. eine positive ganze Zahl) lassen sich im allgemeinen Algorithmen entwicklen, die e^{A+B} nähern. Dazu beachte man, daß mit (3.77)

$$\mathrm{e}^{\frac{1}{n}(A+B)} = \mathrm{e}^{A/n}\,\mathrm{e}^{B/n}\,\mathrm{e}^{-[A,B]/(2\,n^2)} = \mathrm{e}^{A/n}\,\mathrm{e}^{B/n} + \mathcal{O}\left(n^{-2}\right)\,, \tag{3.79}$$

wobei im zweiten Schritt die Exponentialfunktion in niedrigster Ordnung zu $\mathrm{e}^{-[A,B]/(2\,n^2)} = \mathbb{1} - \frac{1}{2\,n^2}\,[A,B] + \ldots$ genähert wurde. Im allgemeinen sucht man nun für einen gegebenen Operator C, bei dem die Berechnung von e^{C} schwierig ist, eine Aufteilung $C = A + B$, so daß e^{A} und e^{B} (möglichst einfach) berechenbar sind. Gelingt dies, so erhält man mit (3.78) und (3.79) für hinreichend großes n eine Näherung für e^{C} zu einer gewünschten Genauigkeit.

Dieses Konzept soll im folgenden auf den Zeitentwicklungsoperator der Quantenmechanik für ein Intervall Δt angewendet werden:

$$U(\Delta t) = \mathrm{e}^{-i\,\mathcal{H}\,\Delta t/\hbar}\,. \tag{3.80}$$

In diesem Fall liegt die Aufteilung

$$\mathcal{H} = \mathcal{H}_0 + V = \frac{\vec{p}^{\,2}}{2\,m} + V(\vec{r}) \tag{3.81}$$

nahe. Die Näherung (3.79) führt dann mit der Ersetzung $1/n \to \Delta t$ auf

$$U(\Delta t) = \mathrm{e}^{-i\,\mathcal{H}_0\,\Delta t/\hbar}\,\mathrm{e}^{-i\,V\,\Delta t/\hbar} + \mathcal{O}\left(\Delta t^2\right) = \widetilde{U}(\Delta t) + \mathcal{O}\left(\Delta t^2\right)\,. \tag{3.82}$$

Man erhählt also folgende Näherung für den Zeitentwicklungsoperator

$$\widetilde{U}(\Delta t) = \mathrm{e}^{-i\,\mathcal{H}_0\,\Delta t/\hbar}\,\mathrm{e}^{-i\,V\,\Delta t/\hbar}\,. \tag{3.83}$$

An diesem Ergebnis lassen sich bereits wesentliche Eigenschaften des resultierenden Verfahrens ablesen:

⊕ Laut (3.83) ist $\widetilde{U}(\Delta t)$ das Produkt zweier unitärer Operatoren, also manifest unitär. Dies gilt unabhängig von der Wahl von Δt. Man beachte, daß unter den bisher diskutierten Verfahren nur das implizite Euler-Verfahren für beliebige Δt unitär war.

⊕ Wie man (3.82) entnimmt, ist die Näherung zweiter Ordnung in Δt. Die Eigenschaft teilt es mit den bisher diskutierten stabilen Verfahren, d.h. dem Leapfrog-, dem impliziten Euler- und dem Verfahren nach Visscher[11].

⊕ Der Fehler wird nach (3.79) durch $\langle\Psi\,|\,\Psi]\rangle - \langle\Psi\,|\,\mathrm{e}^{-[H_0,V]}\,|\,\Psi\rangle \approx \langle\Psi\,|\,[H_0,V]\,|\,\Psi\rangle$ kontrolliert. Im Fall $[H_0,V]=0$ ist $\widetilde{U}(\Delta t)=U(\Delta t)$, d.h. die Zeitentwicklung wird exakt beschrieben. Dies gilt insbesondere für ein freies Teilchen ($V=0$). Man beachte, daß bei allen anderen bisher diskutierten Verfahren selbst in diesem einfachen Fall Abweichungen zweiter Ordnung in Δt auftreten ! Im allgemeinen ist der Vorfaktor des Korrekturterms zweiter Ordnung mit der Näherung (3.83) evtl. kleiner, so daß für gewünschte Genauigekit ein größeres Δt verwendet werden kann als bei den anderen Verfahren.

⊕ Die kinetische Energie $\mathcal{H}_0$ ist diagonal im Impulsraum. Somit kann in dieser Darstellung erstens $\mathrm{e}^{-i\,\mathcal{H}_0\,\Delta t/\hbar} = \mathrm{e}^{-i\,E(k)\,\Delta t/\hbar}$ leicht berechnet werden. Zweitens kann man nun den Kontinuumsausdruck $E(k)=\frac{\hbar^2\,k^2}{2\,m}$ einsetzen und so die Ordnung der Ortsraumdiskretisierung gegenüber dem Gitter-Ausdruck zweiter Ordnung (3.44) erheblich verbessern (die Ordnung hängt nun von der Gesamtzahl N der räumlichen Stützstellen ab). Bei hinreichend glattem Potential $V(\vec{r})$ kann dann die Zahl der Stützstellen verringert werden. Trotz des im folgenden diskutierten zusätzlichen Rechenaufwands zur Berechnug der Exponentialfunktionen kann das Verfahren bei geeigneten Problemen letzten Endes CPU-Zeit-effizienter sein als konkurrierende Verfahren.

- Das Potential V ist diagonal im Ortsraum, $\mathcal{H}_0$ im Impulsraum. Diese beiden Darstellungen der Wellenfunktion $|\,\Psi(t_n)\rangle$ können mit Hilfe einer *Fourier-Transformation* ineinander überführt werden. Es ergibt sich folgendes Verfahren für die Berechnung der Zeitentwicklung:

$$|\,\Psi(t_n)\rangle \quad \stackrel{\text{Ortsraum}}{\longrightarrow} \quad \mathrm{e}^{-i\,V\,\Delta t/\hbar}\;|\,\Psi(t_n)\rangle$$

[11] Obwohl letzteres Verfahren in Kapitel 3.1.1 ausführlicher diskutiert wurde, haben wir die Ordnung dieses Verfahrens im Gegensatz zu den anderen Verfahren nicht explizit hergeleitet. Somit sei für diese Aussage auf die Literatur verwiesen [24].

$$
\begin{aligned}
&\xrightarrow{\text{Fourier}} && \mathrm{e}^{-i\,V\,\widehat{\Delta t/\hbar}}\;\widehat{|\,\Psi(t_n)\rangle} \\
&\xrightarrow{\text{Impulsraum}} && \mathrm{e}^{-i\,E(k)\,\Delta t/\hbar}\,\mathrm{e}^{-i\,V\,\widehat{\Delta t/\hbar}}\;\widehat{|\,\Psi(t_n)\rangle} \;=\; \widehat{|\,\Psi(t_{n+1})\rangle} \\
&\xrightarrow{\text{inverse Fourier}} && |\,\Psi(t_{n+1})\rangle\;. \qquad (3.84)
\end{aligned}
$$

Eine etwas ausführlichere Diskussion dieses Verfahren findet man z.B. in Kapitel 2 von [27], sowie eine Zusammenfassung in Kapitel VI von [22]. Insbesondere wird dieses Verfahren auch für die Berechnung der Zeitentwicklung der eindimensionalen Schrödingergleichung (3.1) in der Xtoys-Sammlung von Michael Creutz verwendet [28].

Um (3.84) praktisch einzusetzen wird ein effizientes Verfahren zur Berechnung der Fourier-Transformation benötigt. Dies ist Gegenstand des folgenden Unterkapitels.

3.3.1 Schnelle Fourier-Transformation

Zuerst erinnern wir kurz an die diskrete Fourier-Transformation (für eine etwas ausführlichere Dikussion vgl. z.B. Kapitel 12.1 von [5]). Gegeben seien Funktionswerte f_r an N äquidistanten Punkten $r = 0,\, 1,\, \ldots,\, N-1$. Dann definieren wir die komplexe diskrete Fourier-Transformation über

$$
\hat{f}_s = \frac{1}{\sqrt{N}} \sum_{r=0}^{N-1} \mathrm{e}^{-2\,\pi\,i\,r\,s/N}\, f_r\,, \qquad (3.85)
$$

wobei s ebenfalls die ganzzahligen Werte von 0 bis $N-1$ annimmt. Die inverse Transformation ist gegeben durch

$$
f_r = \frac{1}{\sqrt{N}} \sum_{s=0}^{N-1} \mathrm{e}^{2\,\pi\,i\,r\,s/N}\, \hat{f}_s\,. \qquad (3.86)
$$

Für die Vorfaktoren gibt es unterschiedliche Konventionen in der Literatur. In (3.85) und (3.86) wurden sie so gewählt, daß man unitäre Abbildungen erhält.

Nach (3.85) und (3.86) sind für die Berechnung der Transformation jeweils N Elemente zu berechnen, die ihrerseits aus N Summanden bestehen. In dieser Darstellung sind somit N^2 Operationen durchzuführen. Hierbei handelt es sich um einen erheblichen Aufwand, der z.B. für die Berechnung der Zeitentwicklung über einen Zeitschritt Δt unerwünscht ist.

Im Spezialfall, daß $N = 2^m$ eine 2er-Potenz ist, bietet die „Schnelle Fourier-Transformation" (*Fast Fourier Transformation* (FFT)) eine Alternative, die mit $N \log_2 N$ Operationen auskommt, also deutlich schneller ist (siehe z.B. Kapitel 12.2 von [5]). Wir führen zuerst ein wenig Notation ein, und zwar die fundamentale Nte Einheitswurzel

$$W_N = \mathrm{e}^{2\,\pi\,i/N} \tag{3.87}$$

sowie eine etwas anders normierte Form der (transformierten) Daten

$$F_s = \frac{1}{\sqrt{N}}\,\hat{f}_s\,. \tag{3.88}$$

Dann kann man (3.86) zuerst mit diesen Größen ausdrücken und anschließend die Summe in die geraden bzw. ungeraden Terme zerlegen:

$$\begin{aligned} f_r &= \sum_{s=0}^{N-1} W_N^{r\,s}\,F_s \\ &= \sum_{s=0}^{N/2-1} W_N^{r\,(2\,s)}\,F_{2\,s} + \sum_{s=0}^{N/2-1} W_N^{r\,(2\,s+1)}\,F_{2\,s+1} \\ &= \sum_{s=0}^{N/2-1} W_{N/2}^{r\,s}\,F_{2\,s} + W_N^r \sum_{s=0}^{N/2-1} W_{N/2}^{r\,s}\,F_{2\,s+1} \\ &= F_r^e + W_N^r\,F_r^o\,. \end{aligned} \tag{3.89}$$

Neben der Zerlegung wurde $W_N^2 = W_{N/2}$ verwendet. In der vorletzten Zeile von (3.89) erkennt man dann die Fourier-Transformation der geraden $F_r^e = \sum_{s=0}^{N/2-1} W_{N/2}^{r\,s}\,F_{2\,s}$ bzw. ungeraden $F_r^o = \sum_{s=0}^{N/2-1} W_{N/2}^{r\,s}\,F_{2\,s+1}$ Hälften der Daten. Man beachte, daß in (3.89) $0 \le r \le N$ zu wählen ist, die Teil-Transformationen F_r^e und F_r^o jedoch Periode $N/2$ besitzen.

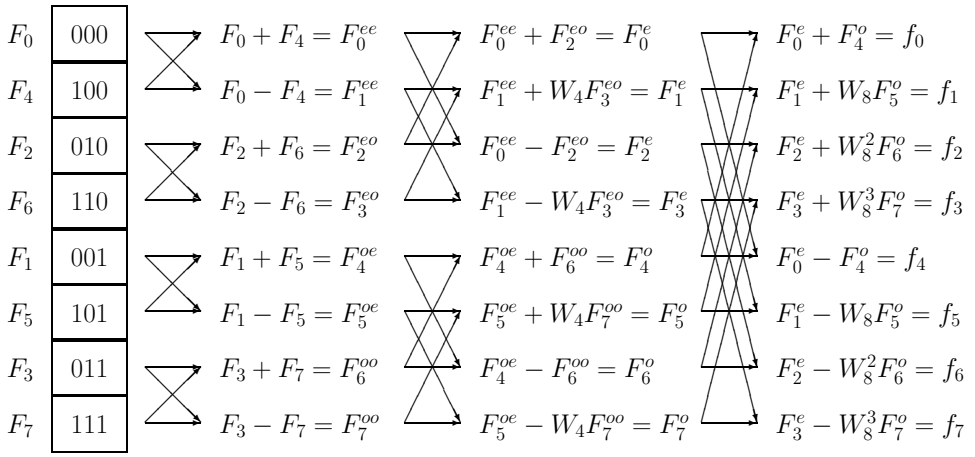
Die Zerlegung (3.89) kann nun rekursiv verwendet werden, und zwar solange bis die Unterfolgen nur noch aus einem Element bestehen. Für $N = 4 = 2^2$ findet man z.B.

$$\begin{aligned} f_r &= F_r^e + W_4^r\,F_r^o \\ &= F_r^{ee} + W_2^r\,F_r^{eo} + W_4^r\;(F_r^{oe} + W_2^r\,F_r^{oo}) \\ &= F_0 + (-1)^r\,F_2 + i^r\;(F_1 + (-1)^r\,F_3)\;. \end{aligned} \tag{3.90}$$

Hierbei wurde $W_2 = \mathrm{e}^{2\,\pi\,i/2} = -1$ und $W_4 = \mathrm{e}^{2\,\pi\,i/4} = i$ eingesetzt, sowie $F_r^{ee} = F_0$, $F_r^{eo} = F_2$, $F_r^{oe} = F_1$ und $F_r^{oo} = F_3$ identifiziert.

Im allgemeinen gelangt man nach $m = \log_2 N$ Schritten zu einem einzelnen Element F_t der Ursprungsdaten, wobei noch das richtige t zu identifizieren ist. Wir schreiben $F_t = F_r^{a_1\, a_2 \ldots a_m}$ mit $a_j = 0$ für e („even" – die gerade Unterfolge) bzw. $a_j = 1$ für o („odd" – die ungerade Unterfolge). Faßt man die Folge $(a_1\, a_2 \ldots a_m)$ nun als Binärdarstellung auf, so gilt $t = (a_m\, a_{m-1} \ldots a_1)$. t erhält man somit einfach durch Bitumkehr ! Dies ist insofern einsichtig, da man zuerst nach gerade bzw. ungerade fragt (a_1 ist also das niedrigste Bit von t) usw. bis schließlich nach oberer bzw. unterer Hälfte klassifiziert wird (d.h. a_m ist das höchste Bit von t).

Der gesamte Sachverhalt ist nochmals für $N = 8$, d.h. $m = \log_2 8 = 3$ in dem folgenden Diagramm dargestellt:



Nach der Herleitung zerteilt man von rechts beginnend die Berechnung der Summanden in f_r unter Verwendung von (3.89) iterativ in jeweils zwei Unterhälften. Schließlich gelangt man zu den Ursprungsdaten F_s, die durch Bit-Umkehr identifiziert werden. Bei der Berechnung wollen wir jedoch umgekehrt vorgehen. Wir führen zuerst die Bit-Umkehr aus, und fassen dann jeweils paarweise Summanden zusammen.

Damit ergibt sich der folgende FFT Algoritmus:

1. Vertausche $F_r \leftrightarrow F_t$ für t Bit-Umkehr von r. In diesem Schritt kann auch der Normierungsfaktor in $F_r = \hat{f}_r/\sqrt{N}$ implementiert werden.

2. Schleife $m = 1$, $n = 2^m \leq N$, $m \to m + 1$

   Berechne W_n nach (3.87)[12].

[12] An dieser Stelle läßt sich auch leicht das unterschiedliche Vorzeichen aus (3.85) bzw.

Schleife $k = 0$, $k < n/2$, $k \to k + 1$

Schleife $l = k$, $l < N$, $l \to l + n$

Berechne gleichzeitig[13]

$$F_l + W_n^k \, F_{l+n/2} \to F_l$$
$$F_l - W_n^k \, F_{l+n/2} \to F_{l+n/2}$$

W_n^k sollte durch iterative Multiplikation berechnen werden: $W_n^0 = 1$, $W_n^{k+1} = W_n \, W_n^k$.

Beachtet man die korrekte Reihenfolge der (gleichzeitigen) Ersetzungen, kann das gleiche Array für die Eingangsdaten f_r (oder $\hat{f}_s$), die Zwischenergebnisse F_l sowie die Ausgangsdaten $\hat{f}_s$ (oder f_r) verwendet werden.

Die FFT kann zwar in Realteil und Imaginärteil aufgeteilt und dann reell implementiert werden. Es liegt jedoch nahe, komplexe Arithmetik zu verwenden. Entsprechende Programmierhinweise finden Sie in Angang B.

Wir fassen zusammen: Durch geschickte Wiederverwendung von Zwischenergenissen in der Berechnung reduziert die FFT den Aufwand der Berechnung der N Summen (3.85) oder (3.86) von N^2 auf $N \log_2 N$ Operationen, vorausgesetzt N ist eine Zweierpotenz. In vielen Anwendungen (speziell auch der Zeitentwicklung in der Quantenmechanik) ist es kein Problem, die Zahl der Stützstellen als Zweierpotenz zu wählen.

Bei der Anwendung der FFT auf ein physikalisches Problem sind die Stützstellen, die in diesem Kapitel mit Abstand eins angenommen wurden, noch umzurechnen. Insbesondere gilt für die k-Punkte auf einem Intervall der Länge L

$$k_s = \frac{2\,\pi}{L}\, s\,. \tag{3.91}$$

Im Impulsraum können die Eigenwerte des Hamiltonoperators eines freien Teilchens (3.11) nun leicht angegeben werden:

$$\mathcal{H}_0 \, \hat{\Psi}_s = E_s^{(0)} \, \hat{\Psi}_s\,. \tag{3.92}$$

Allerdings ist bei der Umrechnung des Bereiches $0 \leq s < N$ aus der FFT noch zu beachten, daß der Wertebereich symmetrisch um das Minimum bei

(3.86) berücksichtigen, so daß die gleiche Routine sowohl zur Berechnung der Fourier-Transformation als auch ihrer Inversen verwendet werden kann.

[13]Man beachte, daß $W_n^{l+n/2} = -W_n^l$ ist, sowie $W_n^l = W_n^k$, da $l - k$ ein Vielfaches von n ist.

$k = 0$, d.h. $s = 0$ gewählt werden sollte. Man wählt also

$$E_s^{(0)} = \frac{\hbar^2 k_s^2}{2m} = \begin{cases} \frac{\hbar^2 (2\pi s)^2}{2mL^2} & \text{für } s \le N/2, \\ \frac{\hbar^2 (2\pi (N-s))^2}{2mL^2} & \text{für } s \ge N/2. \end{cases} \tag{3.93}$$

3.3.2 Aufgabe

A3.4 a. Implementieren Sie analog Aufgabe A3.2.a das Operator-Splitting-Verfahren (3.84) für einen eindimensionalen Hamilton-Operator mit allgemeinem Potential $V(x)$ auf einem Intervall $[0, L]$ der Länge L mit periodischen Randbedingungen ! Verwenden Sie für die kinetische Energie dabei die quadratische Form (3.93) !
Hinweis: Die in Anhang B gezeigten FFT-Routinen in C++, C99 und Java stehen auf der Kurs-Homepage zum Herunterladen zur Verfügung.

b. Wiederholen Sie die Simulation aus A3.3.a mit dem Operator-Splitting-Verfahrenbis zur Zeit $\hbar t/m = 1/200$! Verwenden Sie $\Delta x = L/256$, $\hbar \Delta t/m = 5 \cdot 10^{-6}$ und ansonsten die gleichen Parameter wie in A3.3.a ($x_0 = L/5$, $\sigma = L/32$, $k_0 = 64\pi/L$, $V_0 m L^2/\hbar^2 = 1.5 \cdot 10^4$, $2 \cdot 10^4$ bzw. $3 \cdot 10^4$) ! Vergleichen Sie Laufzeit und Ergebnisse der beiden Programme !

3.3.3 Variationsprinzip

Eine Motivation für die Einführung des Operator-Splitting-Verfahrens war der Wunsch, Erwartungswerte zu berechnen. Insbesondere kann die Energie E eines Zustandes $|\Psi\rangle$ als Erwartungswert des Hamilton-Operators bestimmt werden

$$E = \langle \mathcal{H} \rangle = \frac{\langle \Psi | \ \mathcal{H} \ | \Psi \rangle}{\langle \Psi \mid \Psi \rangle} \,. \tag{3.94}$$

In diesem Zusammenhang ist es nützlich, sich an das Variationsprinzip zu erinnern. Dazu betrachten wir einen vollständigen Satz von Eigenfunktionen (3.2) und ordnen die Energie so, daß $E_0 \le E_1 \le E_2 \le \ldots$. Insbesondere machen wir die physikalisch sinnvolle Forderung, daß der Hamilton-Operator nach unten beschränkt ist, so daß ein Grundzustand mit Energie E_0 existiert. Für eine Eigenfunktion $|\Psi_j\rangle$ gilt nun

$$E_j = \frac{\langle \Psi_j | \ \mathcal{H} \ | \Psi_j \rangle}{\langle \Psi_j \mid \Psi_j \rangle} \,. \tag{3.95}$$

Eine allgemeine Wellenfunktion $|\,\Psi\rangle$ kann nach Eigenfunktionen entwickelt werden

$$|\,\Psi\rangle = \sum_j c_j \;|\,\Psi_j\rangle\;. \tag{3.96}$$

Zur Vereinfachung der folgenden Diskussion wählen wir die Eigenbasis orthonormal $\langle\Psi_j\,|\,\Psi_j\rangle = \delta_{i,j}$. Nun setzen wir (3.96) in (3.94) ein und finden

$$E = \langle\mathcal{H}\rangle = \frac{\langle\Psi\,|\;\mathcal{H}\,|\,\Psi\rangle}{\langle\Psi\,|\,\Psi\rangle} = \frac{\sum\limits_{k,j} c_k^\star\, c_j \langle\Psi_k\,|\;\mathcal{H}\,|\,\Psi_j\rangle}{\sum\limits_{j=0}^{\infty} |c_j|^2} = \frac{\sum\limits_{j=0}^{\infty} |c_j|^2\; E_j}{\sum\limits_{j=0}^{\infty} |c_j|^2} \geq E_0\,. \tag{3.97}$$

Hierbei wurde ferner $\mathcal{H}\;|\;\Psi_j\rangle = E_j\;|\;\Psi_j\rangle$ sowie die Orthonormalität der Eigenfunktionen verwendet. Schließlich wurde $E_j \geq E_0$ verwendet.

Setzt man $|\,\Psi_0\rangle$ in (3.94) ein, so erhält man die Grundzustandsenergie E_0. Andererseits ist dies nach (3.97) auch der kleinste Wert, der für ein beliebiges $|\,\Psi\rangle$ angenommen wird. Es folgt somit das *Variationsprinzip*:

$$E_0 = \underset{\text{alle } |\Psi\rangle}{\text{Minimum}} \frac{\langle\Psi\,|\;\mathcal{H}\,|\,\Psi\rangle}{\langle\Psi\,|\,\Psi\rangle}\,. \tag{3.98}$$

Dieses Ergebnis ist aus konzeptioneller Sicht wichtig, denn es bedeutet, daß jede nach (3.94) bestimmte Energie oberhalb der Grundzustandsenergie liegt, bzw. im besten Fall gleich ihr ist.

Die Minimierung in (3.98) kann normalerweise nicht praktisch durchgeführt werden. Man kann jedoch hierauf ein Näherungsverfahren für den Grundzustand aufbauen, das sogenannte *Variationsverfahren*. Dazu betrachtet man Kandidaten für Grundzustandswellenfunktionen $|\,\Psi(\alpha_1,\ldots,\alpha_N)\rangle\,$, die von Parametern $\alpha_1,\;\ldots,\;\alpha_N$ abhängen. Die optimale Näherung für die Grundzustandsenergie E_0 bzw. die Grundzustandswellenfunktion ergibt sich dann im Sinne von (3.98) als

$$\underset{(\alpha_1,\ldots,\alpha_N)}{\text{Minimum}} \frac{\langle\Psi(\alpha_1,\ldots,\alpha_N)|\;\mathcal{H}\,|\Psi(\alpha_1,\ldots,\alpha_N)\rangle}{\langle\Psi(\alpha_1,\ldots,\alpha_N)|\;\Psi(\alpha_1,\ldots,\alpha_N)\rangle}\,. \tag{3.99}$$

Der Grundzustand des Helium-Atoms (0.1) ist ein typisches Anwendungsbeispiel aus der Quantenmechanik. Man setzt wasserstoffartige Grundzustandswellenfunktionen für beide Elektronen an, betrachtet die Kernladungszahl Z in dieser Wellenfunktion jedoch als einen Parameter, der so gewählt wird, daß man einen möglichst kleinen Wert für die Energie E erhält. Analog kann man auch das Wasserstoff-Molekül (0.3) behandeln. Man konstruiert

aus Wasserstoff-Grundzustandswellenfunktionen bzgl. der beiden Kernorte unter Berücksichtigung des Spin-Anteils einen Ansatz für das Molekül. Der Abstand R der beiden Kerne ist hierbei der Parameter, den es zu optimieren gilt. Bei antisymmetrischem Spin-Anteil findet man tatsächlich ein Minimum, d.h. ein gebundenes Molekül.

3.3.4 Doppelmulden-Potential

Als weiteres Beispiel für den Tunneleffekt wollen wir im folgenden das Doppelmulden-Potential untersuchen. Die Diskussion folgt Kapitel 13.5 und 13.6 von [4]. Gegeben sei der eindimensionale Hamilton-Operator

$$\mathcal{H} = \frac{p^2}{2\,m} + V(x) \tag{3.100}$$

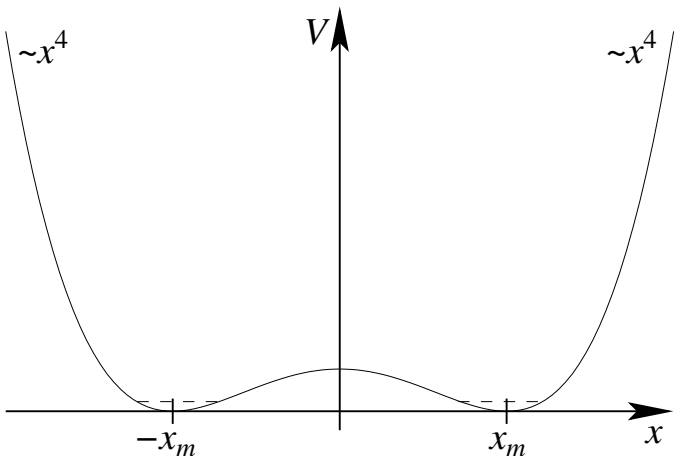
mit

$$V(x) = m\,\omega^2\,\frac{(x-x_m)^2(x+x_m)^2}{8\,x_m^2}\,. \tag{3.101}$$

Der Vorfaktor ist hierbei so gewählt, daß $V''(\pm x_m) = m\,\omega^2$ ist. In der Nähe von $\pm x_m$ hat man somit näherungsweise jeweils einen harmonischen Oszillator der gewohnten Form

$$V(x) \approx \frac{1}{2}\,m\,\omega^2\;(x \pm x_m)^2 \tag{3.102}$$

für $|x \pm x_m| \ll x_m$. Das Doppelmulden-Potential (3.101) ist im folgenden skizziert:



Die beiden Minima bei $\pm x_m$ sind durch eine Barriere der Höhe $V(0) = m\,\omega^2\,x_m^2/8$ getrennt. Für $x \to \pm\infty$ geht das Potential in den anharmonischen Oszillator $V(x) \sim x^4$ über.

Bei $\pm x_m$ findet man näherungsweise harmonische Oszillatoren der Frequenz ω. Als erste Näherung kann man bei einer hinreichend hohen Barriere, d.h. hinreichend großem Abstand x_m, die Niveaus dieser beiden harmonischen Oszillatoren betrachten. Die Grundzustandswellenfunktion eines harmonischen Oszillators hat die Form

$$\Psi_{x_z}(x) = C\,\mathrm{e}^{-\alpha(x-x_z)^2/2}\,. \tag{3.103}$$

Im Prinzip könnte man α und x_z in dieser Wellenfunktion für das Potential (3.101) optimieren. Dies ist jedoch mühselig, so daß wir näherungsweise die Grundzustandswellenfunktionen der harmonischen Oszillatoren (3.102) bei $\pm x_m$ verwenden:

$$x_z = \pm x_m\,, \qquad \alpha = \frac{m\,\omega}{\hbar}\,. \tag{3.104}$$

Demenstprechend finden wir für die Energie näherungsweise die Grundzustandsenergie eines harmonischen Oszillators:

$$E \approx \frac{1}{2}\,\hbar\,\omega\,. \tag{3.105}$$

In einer Simulation kann die Energie der Wellenfunktion (3.103) als Erwartungswert (3.94) des Hamilton-Operators gemessen werden.

Wählt man als Anfangswellenfunktion $\Psi(x, t = 0) = \Psi_{\pm x_m}(x)$, d.h. setzt man ein Teilchen bei $t = 0$ in eines der beiden Minima, so ist dies keine Eigenfunktion. Klassisch würde ein Teilchen natürlich im Minimum ('Grundzustand') bei $\pm x_m$ liegen bleiben, quantenmechanisch kann es jedoch in das Nachbarminimum tunneln.

Eine verbesserte Näherung für die Eigenfunktionen ist folgende Kombination der Wellenfunktionen $\Psi_{\pm x_m}$:

$$\Psi_0 \approx \Psi_{x_m} + \Psi_{-x_m}\,, \qquad \Psi_1 \approx \Psi_{x_m} - \Psi_{-x_m}\,. \tag{3.106}$$

Die Berechnung der Energie dieser Wellenfunktionen für das Potential (3.101) ist nicht ganz trivial. Jedoch folgt aus dem Knotensatz, daß der Grundzustand symmetrisch ist und der erste angeregte Zustand antisymmetrisch, d.h. wir identifizieren Ψ_0 als Grundzustand.

Einen Ausgangszustand, bei dem sich das Teilchen links befindet, können wir also als

$$\Psi(x, t = 0) = \Psi_{-x_m}(x) \approx \frac{1}{2}\,\left(\Psi_0(x) - \Psi_1(x)\right) = \varphi(x, t = 0) \tag{3.107}$$

schreiben. Hierbei haben wir die Kombination der exakten Eigenfunktionen Ψ_0 und Ψ_1 als φ bezeichnet. Für deren Zeitentwicklung gilt:

$$\begin{aligned}\varphi(x,t) &= \frac{1}{2}\left(\mathrm{e}^{-i\,E_0\,t/\hbar}\,\Psi_0(x) - \mathrm{e}^{-i\,E_1\,t/\hbar}\,\Psi_1(x)\right) \\ &= \mathrm{e}^{-i\,E_0\,t/\hbar}\,\frac{1}{2}\left(\Psi_0(x) - \mathrm{e}^{-i\,(E_1-E_0)\,t/\hbar}\,\Psi_1(x)\right)\,. \qquad (3.108)\end{aligned}$$

Man liest ab, daß $\varphi(x,t) \propto \Psi_{-x_m}(x)$ für $t = n\,\tau$, bzw. $\varphi(x,t) \propto \Psi_{x_m}(x)$ für $t = \left(n + \frac{1}{2}\right)\,\tau$ mit ganzzahligen n und

$$\tau = \frac{2\,\pi\,\hbar}{E_1 - E_0}\,. \qquad (3.109)$$

Dementsprechend findet man Oszillationen z.B. in dem Erwartungswert des Ortes

$$\langle x\rangle(t) \approx -x_m\,\cos\left(\frac{2\,\pi\,t}{\tau}\right)\,. \qquad (3.110)$$

In einer Simulation kann man nun z.B. $\langle x\rangle(t)$ messen, aus (3.110) die Periode τ bestimmen und schließlich aus (3.109) die Tunnel-Aufspaltung $E_1 - E_0$ ablesen. Beimischung von höheren angeregten Wellenfunktionen in $\Psi(x, t = 0)$ führt zu höherfrequenten Beiträgen in den Oszillationen von $\langle x\rangle(t)$. Die kleinste Oszillationsfrequenz $\propto 1/\tau$ ist dennoch im allgemeinen gut abzulesen, so daß $E_1 - E_0$ ziemlich genau bestimmt werden kann.

Bisher haben wir gezeigt, wie Energie E und die Tunnel-Aufspaltung $E_1 - E_0$ aus einer zeitabhängigen Simulation bestimmt werden können. Zur Bestimmung der Grundzustandsenergie könnten wir nun im Sinne von (3.107) $E \approx \frac{1}{2}\,(E_0 + E_1)$ schätzen. Dies ist jedoch nicht besonders genau, so daß eine direkte Bestimmung von E_0 wünschenswert ist. Dies läßt sich beim Operator-Splitting-Verfahren mit einem kleinen Trick bewerkstelligen, der auch in dem Programm aus der Xtoys-Sammlung [28] zum Einsatz kommt. Die Idee ist, den Beitrag einer Eigenfunktion Ψ_j mit einem Faktor $\mathrm{e}^{-E_j\,\eta\,t/\hbar}$ zu dämpfen. Normiert man $\Psi(x,t)$ jeweils neu, so bleibt dann für große t nur Ψ_0 übrig. Bei der Implementierung ersetzt man entsprechend in der Zeitentwicklung

$$\mathrm{e}^{-i\,\mathcal{H}\,t/\hbar} \quad \rightarrow \quad \mathrm{e}^{-i\,\mathcal{H}\,(t - i\,\eta)/\hbar}\,. \qquad (3.111)$$

Man verwendet also den gewohnten Algorithmus für die Zeitentwicklung, wählt den Zeitschritt jedoch komplex $\Delta t \in \mathbb{C}$ mit $\Im\mathrm{m}(\Delta t) < 0$. Normiert man weiterhin $\Psi(x,t)$ nach jedem Zeitschritt neu, so konvergiert $\Psi(x,t) \rightarrow \Psi_0(x)$. Somit kann schließlich die Grundzustandsenergie als Erwartungswert des Hamilton-Operators bestimmt werden $\langle\mathcal{H}\rangle(t) \rightarrow E_0$.

3.3.5 Aufgabe

A3.5 Wir wollen das Tunneln im Doppelmulden-Potential (3.101) simulieren. Dazu wählen wir 'natürliche Einheiten' $\hbar = m = \omega = 1$.

a. Passen Sie Ihr Programm aus Aufgabe A3.4 für das Operator-Splitting-Verfahren (3.84) auf das Doppelmulden-Potential an ! Wählen Sie dabei ein Intervall $[-4\,x_m, 4\,x_m]$ mit periodischen Randbedingungen, d.h. $\Psi(x + L, t) = \Psi(x, t)$ mit $L = 8\,x_m$! Wählen Sie $1024 = 2^{10}$ Stützstellen, d.h. $\Delta x = L/1024$, sowie $\Delta t = 1/20$! Wir betrachten nun den Fall $x_m = 3$. Bei $t = 0$ sitze das Teilchen im linken Potentialminimum, d.h. $\Psi(x, t = 0)$ ist gegeben durch (3.103) mit $x_z = -x_m$, $\alpha = 1$. Führen Sie eine Simulation bis zur Zeit $t = 2000$ durch ! Betrachten Sie die Aufenthaltswahrscheinlichkeit als Funktion der Zeit t und berechnen Sie den Erwartungswert des Ortes $\langle x \rangle(t)$! Extrahieren Sie aus $\langle x \rangle(t)$ die größte Oszillationsperiode τ und schätzen Sie mit (3.109) die Aufspaltung $E_1 - E_0$!

b. Berechnen Sie die Energie $E(t) = \langle \mathcal{H} \rangle$! Da der Hamilton-Operator nicht explizit zeitabhängig ist, sollte $E(t) = E$ zeitunabhängig sein. Testen Sie Ihr Ergebnis auf diese Eigenschaft und diskutieren Sie mögliche Abweichungen !
Hinweis: Da die kinetische Energie gemäß (3.93) im Impulsraum definiert ist, benötigen Sie bei der Energieberechnung eine FFT.

c. Schalten Sie nun eine Dämpfung ein, indem Sie $\Delta t = 1/20 - i/40$ wählen und die Simulation wiederholen ! Schätzen Sie die Grundzustandsenergie E_0 über den Erwartungswert des Hamilton-Operators $\langle \mathcal{H} \rangle$ im späten Bereich dieser Simulation !
Hinweis: Um Bereichsüberläufe zu vermeiden, sollte die Wellenfunktion nach jedem Zeitschritt Δt neu normiert werden.

3.4 Hartman-Effekt

Motiviert von der Frage, wie schnell Schichtstruktur-Bauelemente arbeiten können, hat Hartman 1962 [29] die Frage untersucht, wieviel Zeit beim Tunneln durch klassisch verbotene Bereiche vergeht. Konkret hat Hartman die Zeitentwicklung eines auf die Rechteck-Barriere (3.53) einfallenden Gaußschen Wellenpaketes durchgerechnet[14]. Dabei machte er die überraschende Beobachtung, daß bei einer dicken Barriere das Maximum des transmittierten Wellenpakets *schneller* ankommt, als das Maximum des Wellenpakets ohne Barriere.

Eine erste Erklärung findet sich bereits in [29]: Aufgrund der Energieabhängigkeit der Transmissionswahrscheinlichkeit werden höhere k stärker transmittiert als kleinere. Dementsprechend ist der mittlere Impuls des transmittierten Wellenpakets höher als des einfallenden, so daß sich das transmittierte Wellenpaket schneller bewegt.

Die zentrale Frage ist, ob ein Detektor jenseits der Barriere früher anspricht, als er es ohne Barriere tun würde. Um dies ein wenig genauer zu diskutieren, bezeichnen wir die normierten Wellenfunktionen ohne Barriere als $\Psi^{(0)}(x,t)$ und die mit Barriere als $\Psi^{(B)}(x,t)$. Wir können nun die Aufenthaltswahrscheinlichkeit rechts der Barriere definieren:

$$P_r^{(\cdot)}(t) = \int\limits_{L/2+a}^{\infty} \mathrm{d}x \, \left|\Psi^{(\cdot)}(x,t)\right|^2 . \tag{3.112}$$

Auch unter den Bedingungen des Hartman-Effekts gilt $P_r^{(B)}(t) \leq P_r^{(0)}(t)$, d.h. ein Detektor mißt zu keinem Zeitpunkt eine erhöhte Teilchendichte. Dies ist insofern plausibel, da der Hartman-Effekt auf dem Herausfiltern der Teilchen mit niedriger Geschwindigkeit basiert. Eine solche Messung setzt aber Wissen über die genaue Anzahl der auf die Barriere einfallenden Teilchen voraus. Beschränkt man sich auf Messungen, bei denen ein Teilchen rechts beobachtet wird, muß man anders normieren – z.B.

$$\mathcal{P}_r^{(\cdot)}(t) = \frac{P_r^{(\cdot)}(t)}{\lim\limits_{t\to\infty} P_r^{(\cdot)}(t)} . \tag{3.113}$$

Es ist nun plausibel, daß für bestimmte Zeiten eine durch die Barriere erhöhte Meßwahrscheinlichkeit $\mathcal{P}_r^{(B)}(t) > \mathcal{P}_r^{(0)}(t)$ auftreten kann.

[14] Nebenbei findet man in [29] eine grafische Darstellung der Transmissionsamplitude einer einfallenden ebenen Welle (vgl. (3.67)).

Dennoch bleibt die Frage offen, wie lange sich ein jenseits der Barriere beobachtetes Teilchen unter ihr aufgehalten hat. Hierbei handelt es sich um eine Grundlagenfrage der Quantenmechanik, mit der sich nach Hartman zahlreiche Arbeiten auch namhafter Autoren beschäftigt haben. Häufig wurden Simulationen zur Rechtfertigung des eigenen Standpunkts herangezogen.

Wir wollen an dieser Stelle kurz einige Überlegungen aus [30] zur Definition von Transmissions- und Reflektionszeiten zusammenfassen. Die Wahrscheinlichkeit, ein Teilchen zur Zeit t im Intervall $[a, b]$ zu finden ist gegeben durch

$$\int_a^b \mathrm{d}x \; |\langle x| \; \Psi(t)\rangle \,|^2 = \langle D\rangle \tag{3.114}$$

mit dem Projektor

$$D = D(a,b) = \int_a^b \mathrm{d}x \;\; |x\rangle \, \langle x| \;\; . \tag{3.115}$$

Die *Aufenthaltszeit* in dem Intervall $[a, b]$ ergibt sich als Integral

$$\tau_D(a,b) = \int_{-\infty}^{\infty} \mathrm{d}t \, \langle \Psi(t)| \;\; D(a,b) \; |\Psi(t)\rangle \; . \tag{3.116}$$

Die Ausdehnung der unteren Integrationsgrenze zu $t = -\infty$ ist für den Fall ausschließlich positiver Impulse in der Anfangswellenfunktion gerechtfertigt. Die Definition (3.116) entspricht (3.112).

Zum Problem führt erst die Einschränkung auf transmittierte Teilchen. Dies wird formalisiert, indem man den Projektor auf irgendwann in der Zukunft transmittierte Teilchen einführt

$$P = \int_0^{\infty} \mathrm{d}p \;\; |p^{(-)}\rangle \, \langle p^{(-)}| \;\; . \tag{3.117}$$

Bei $|p^{(\pm)}\rangle$ handelt es sich um Streuzustände. Diese entsprechen asymptotisch ebenen Wellen, sind Eigenzustände mit der entsprechenden Energie

$$\mathcal{H} \;\; |p^{(\pm)}\rangle \; = \frac{p^2}{2\,m} \;\; |p^{(\pm)}\rangle \; , \tag{3.118}$$

und können wie folgt normiert werden:

$$\langle p^{\pm}| \; p'^{\pm}\rangle \; = \langle p| \; p'\rangle \; = \delta(p - p') \, . \tag{3.119}$$

Während es sich bei $|p^{(+)}\rangle$ um die üblichen Streulösungen für eine von links einfallende ebene Welle handelt (vgl. Kapitel 3.2), enthält $|p^{(-)}\rangle$ auch einen von rechts einlaufenden Anteil. Mit (3.117) gilt für die Transmissionswahrscheinlichkeit

$$T = \langle P \rangle \,. \tag{3.120}$$

Man kann sich nun z.B. die folgenden Experimente vorstellen:

1. Man stellt zuerst mit einer Messung fest, ob das Teilchen irgendwann in der Zukunft transmittiert wird und prüft dann, ob sich das Teilchen im Intervall $[a, b]$ befindet. Dies wird durch die Kombination $D\,P$ beschrieben.

2. Man stellt zuerst mit einer Ortsbestimmung fest, ob sich das Teilchen im Intervall $[a, b]$ befindet und prüft später, ob es auch transmittiert wird. Die führt auf die Kombination $P\,D$.

Man beachte, daß $P\,D \neq D\,P$ ist, während $(D\,P)^\dagger = P\,D$ gilt. Die beiden genannten Kombination sind also nicht nur inäquivalent, sondern jeweils nichthermitesch, nach den Axiomen der Quantenmechanik somit nicht meßbar. Die Observablen können nun auf verschiedene Weise symmetrisiert werden, z.B. sind $P\,D\,P$ und $D\,P\,D$ hermitesch – für weitere Details vergleiche [30].

Im Rahmen einer Simulation sind D und P als Projektionen zu implementieren. $\langle \Psi(t) \mid D\,P\,D \mid \Psi(t) \rangle$ kann z.B. berechnet werden, indem man zuerst $\mid \Psi(t) \rangle$ auf das Intervall $[a, b]$ projeziert und anschließend die Entwicklung von $D \mid \Psi(t) \rangle$ bis zu einem geeigneten Zeitpunkt weiter verfolgt, um schließlich den transmittierten Anteil zu bestimmen. Die Berechnung von $\langle \Psi(t) \mid P\,D\,P \mid \Psi(t) \rangle$ ist noch etwas schwieriger. Man muß zuerst die Zeitentwicklung von $|\Psi(t)\rangle$ bis zu einem geeigneten Zeitpunkt durchführen, um dann auf den transmittierten Anteil (d.h. den Teil rechts von der Barriere) zu projezieren. Das Ergebnis ist nun durch Umkehr der Zeitentwicklung bis zu t zurückzuverfolgen. Schließlich kann der im Intervall $[a, b]$ befindliche Anteil von $P \mid \Psi(t) \rangle$ gemessen werden. In beiden Fällen ist zu beachten, daß die erste Messung die Wellenfunktion verändert.

3.4.1 Aufgabe

A3.6* Wir wollen abschließend den Hartman-Effekt beim Tunneln durch die Rechteck-Barriere (vgl. A3.2) simulieren.

a. Verwenden Sie das Operator-Splitting-Verfahren (3.84) und wählen Sie als Parameter $\Delta x = L/2048$ sowie als Anfangsbedingungen ein Gaußsches Wellenpaket (3.37) mit $x_0 = L/5$, $\sigma = L/16$, $k_0 = 64\,\pi/L$! Wir verwenden ferner eine Potential-Barriere (3.53) mit der Breite $a = L/32$ und der Höhe $V_0 = 6 \cdot 10^4\,\frac{\hbar^2}{m\,L^2}$. Simulieren Sie die Zeitentwicklung des Wellenpaketes bis zur Zeit $\hbar\,t/m = 1/400$ mit $\hbar\,\Delta t/m = 10^{-7}$!
Achtung: Wird die Simulation nicht mit ausreichender Genauigkeit durchgeführt (hinreichend kleines Δt, etc.), dominieren Näherungs-Artefakte das Ergebnis.

b. Vergleichen Sie für geeignete Zeiten t die Position des Maximums der Aufenthaltswahrscheinlichkeit rechts von der Barriere zwischen der Simulation mit und ohne Barriere !
Hinweis: Die kleine Transmissionswahrscheinlichkeit T führt zu sehr kleinen Zahlen. Bei einer grafischen Auftragung ist somit eine logarithmische Skalierung der Aufenthaltswahrscheinlichkeit nützlich.

c. Berechnen und vergleichen Sie die Aufenthaltswahrscheinlichkeiten (3.112) für $x \geq L/2+a$ mit Barriere $P_r^{(B)}(t)$ und ohne Barriere $P_r^{(0)}(t)$! Wir führen nun die Normierung in (3.113) durch, indem wir durch $P_r^{(\cdot)}(t_s)$ für ein geeignetes t_s teilen. Wie sieht der Vergleich von dem so geschätzten $\mathcal{P}_r^{(B)}(t)$ mit $\mathcal{P}_r^{(0)}(t)$ aus ?

d.** Berechnen Sie für $D = D(L/2, L/2 + a)$

$$\begin{aligned} \tau_T^{DPD} &= \frac{1}{T} \int \mathrm{d}t\, \langle \Psi(t) |\ D\,P\,D\ | \Psi(t) \rangle\ , \\ \tau_T^{PDP} &= \frac{1}{T} \int \mathrm{d}t\, \langle \Psi(t) |\ P\,D\,P\ | \Psi(t) \rangle \end{aligned} \tag{3.121}$$

mit Hilfe einer Projektion der Wellenfunktion $|\,\Psi(t_s)\rangle$ zu einem geeigneten Zeitpunkt t_s der Simulation auf die Intervalle $[L/2, L/2+a]$ bzw. $[L/2+a, L]$!
Hinweis: Im ersten Fall sind viele Simulationen durchzuführen, über deren Ergebnisse Sie anschließend numerisch integrieren; im zweiten Fall kann das Integral aus einer Simulation bestimmt werden.

4 Ausblick

Am Ende dieses Kurses ist eine Bilanz angebracht. Wie eingangs angekündigt, mußte eine Themenauswahl getroffen werden. In Kapitel 0.3 haben wir das z.B. aus der Elektrodynamik bekannte Potentialproblem diskutiert und im Kapitel 0.5 das Gauß- sowie das Conjugate Gradient Verfahren zum Lösen linearer Gleichungssysteme kennengelernt. In Kapitel 1 wurden explizite und implizite Lösungsverfahren für die Zeitentwicklung (nicht-linearer) partieller Differentialgleichung vorgestellt, die wir dann in Kapitel 2 auf ein einfaches Modell für das Tsunami-Phänomen angewandt haben. In Kapitel 3 haben wir uns schließlich mit der zeitabhängigen Schrödingergleichung beschäftigt und dabei verschiedene Lösungsverfahren kennengelernt.

Einige Aspekte wurden dabei stärker vertieft als in vergleichbaren Kursen, an anderen Stellen mußten dafür Lücken bleiben. Nahegelegen hätten u.a. quantenmechanische Mehr- und Vielteilchen-Probleme, insbesondere die in Kapitel 0.1 angerissenen einfachen Atome und Moleküle. Allerdings ist die Herausforderung hier primär die Implementierung das Hamilton-Operators, die vorgestellten Algorithmen können hingegen direkt auf die Zeitentwicklung angewendet werden. Eine natürliche Erweiterung wäre auch die Behandlung der stationären Schrödingergleichung, die nach Kapitel 3 in diskretisierter Form auf das Eigenwertproblem für hermitesche Matrizen führt. Hierbei würden den diskutierten Algorithmen analoge Verfahren zum Einsatz kommen. Zur direkten Bestimmung aller Eigenwerte eignet sich z.B. das Householder-Verfahren (siehe z.B. Kapitel 11.2 von [5]), das dem Gauß-Verfahren zur Lösung von Gleichungssystemen entspricht. Extremale Eigenwerte größerer hermitescher Matrizen können iterativ z.B. mit dem Lanczos-Verfahren berechnet werden, ganz ähnlich wie man mit dem Conjugate Gradient Verfahren iterativ zu einer Lösung eines linearen Gleichungssystems gelangt.

Mit der ,,II" im Kurstitel wurden gewöhnliche Differentialgleichungen und damit verbundene Anwendungen aus der klassischen Mechanik bewußt ausgeklammert. Im Vergleich mit anderen ,,Computational Physics"-Kursen [1, 4, 25] 'fehlt' vor allem der komplette Themenkomplex von Monte-Carlo-Simulationen, angefangen bei Zufallszahlen (siehe auch Kapitel 7 von [5]), über Anwendungen aus der klassischen statistischen Physik bis hin zu Quanten-Monte-Carlo-Verfahren. Das Weglassen dieses Themenkomplexes ist allerdings zumindest teilweise durch die Zielgruppe (ab 4. Semster) bedingt, im übrigen entspricht der Umfang dieses Stoffs einer eigenen Vorlesung.

Dank

Danken möchte ich Jean-Mathias Grießmeier für Literatur-Hinweise zu Tsunamis sowie Reinhard F. Werner für Diskussionen über Grundlagen der Quantenmechanik. Was wäre ein solcher Kurs aber ohne Teilnehmer und praktische Übungen ? Mein besonderer Dank gilt somit allen Kursteilnehmern, ohne deren Engagement dieses Skript nicht nur deutlich mehr Fehler hätte, sondern gar nicht erst zustande gekommen wäre. Ein Dankeschön auch an das Rechenzentrum der TU Braunschweig für die Bereitstellung eines Übungsraumes.

A Java-Schnellkurs

A.1 Grundlagen

Java ist C bzw. C++ sehr ähnlich, so daß Programme oft ohne großen Aufwand zwischen den beiden Sprachen konvertiert werden können. Allerdings gibt es im Detail auch einige Unterschiede, wie wir bereits in unserem ersten Beispiel sehen werden. Eine recht kompakte Einführung in Java ist [31], ausführliche Informationen findet man z.B. auch online in [32].

Ein erstes ganz einfaches Java-Programm soll `first.java` heißen:

```
public class first {
 public static void main(String[] args)
  {
   System.out.println("Ein allererstes Programm in Java");
  }
}
```

Auch in Java muß der Programm-Quelltext compiliert werden. Das Kommando für den Compiler-Aufruf heißt[15]:

```
javac first.java
```

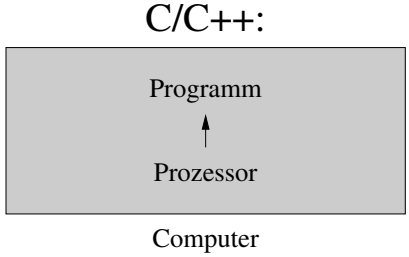
Nun haben wir eine Datei `first.class`. Allgemeiner erzeugt `javac` aus einer Datei 'Name`.java`' eine Datei 'Name`.class`'. Dieses Java-Programm kann man wie folgt ausführen:

```
java first
```

Nun sollte in der Konsole die folgende Ausgabe erscheinen:

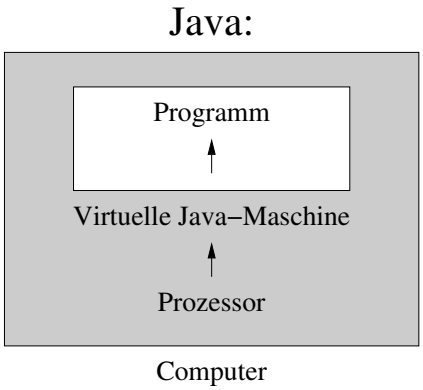
```
Ein allererstes Programm in Java
```

Java hat einen fundamentalen Unterschied zu C/C++ und anderen Compilern. So erzeugt C/C++ ein Programm (z.B. `first`), das von dem Prozessor des Computers direkt ausgeführt wird:



[15] Beim GNU-Compiler muß man '`javac`' durch '`gcj -C`' ersetzen.

Java hingegen erzeugt einen speziellen Bytecode (z.B. `first.class`), zu dessen Ausführung ein weiteres Programm, eine Virtuelle Java-Maschine benötigt wird. Das Kommando `java` ruft die Virtuelle Java-Maschine auf, die dann den Bytecode interpretiert und ausführt:



Diese Konzeption mit einer virtuellen Maschine und einem Bytecode hat verschiedene Vorteile, so z.B.:

1. **Portabilität**
   Ist er einmal auf einem Entwicklungsrechner erzeugt worden, kann der Bytecode auf jede beliebige Plattform kopiert werden (vorausgesetzt es gibt dort eine Virtuelle Java-Maschine) und verhält sich immer gleich. Es gibt für Java auch verschiedene Schnittstellen, die die gleiche Portabilität besitzen.
   So gibt es Pakete, die einen einfachen und Betriebssystem-unabhängigen Zugriff auf Grafik ermöglichen. Der einfache Zugriff auf Grafik ist auch der Hauptgrund, warum wir uns hier mit Java beschäftigen.

2. **Sicherheit**
   Der Java-Bytecode besitzt keinen direkten Zugriff auf den Computer, sondern läuft in einer Art Sandkasten. Die virtuelle Maschine kann dann so eingestellt werden, daß sie den Zugriff auf bestimmte Komponenten des Systems (wie z.B. Dateien) einschränkt oder ganz verbietet.

Diese beiden Punkte sind besonders im Zusammenhang mit Internet und WWW wichtig. Internet und WWW sind somit sicher auch ein wichtiger Grund für die Popularität der verschiedenen Java-Varianten.

Neben den erwähnten Vorteilen hat Java aber auch einen gravierenden Nachteil:

1. **Effizienz**
   Das Interpretieren des Bytecodes ist im Regelfall (deutlich) langsamer als die direkte Ausführung eines vergleichbaren Programmes von dem Prozessor. Auch ist Java nicht für die Ausnutzung aller System-Resourcen (z.B. Speicher) gedacht.

Rechen- oder Speicher-intensive Programme, d.h. Lösungen für numerisch anspruchsvolle Probleme, werden deswegen besser nicht in Java, sondern z.B. in C++ geschrieben.

Java ist wie C++ eine Objekt-orientierte Sprache. Die Syntax für Klassen und Objekte ist der von C++ sehr ähnlich. Insbesondere haben die Deklarationen '`public`', '`class`' und '`static`' die gleiche Bedeutung wie in C++.

Allgemein wird eine beliebige Klasse durch Hinzufügen einer Methode

```
public static void main(String[] args)
```

zu einer Java-Anwendung. `main()` ist als statische Methode deklariert, da zum Startzeitpunkt noch kein Objekt existiert.

Einige weitere Sprachelemente von Java illustriert das folgende kleine Programm

```
public class beispiel {                 // Die Datei MUSS beispiel.java heissen
 final static int N=21;                 // Diese Zahl kann vom Programm nicht
                                        // geaendert werden

 public static void main(String[] args) // Das Hauptprogramm MUSS so
  {                                     // deklariert werden
   System.out.println("sqrt(2*Pi/" + N + ") = \t"
                      + Math.sqrt(2*Math.PI/((double) N)) );
  }
 }
```

Java-spezifische Konstruktionen sind:

1. Eine Ausgabe auf die Konsole erfolgt mit

   ```
   System.out.println(Ausgabe);
   ```

   Dies gibt eine *Zeile* aus, erzeugt also automatisch nach der Ausgabe einen Zeilenvorschub ('`System.out.print(Ausgabe);`' erzeugt keinen Zeilenvorschub).

2. Verschiedene Ausgabe-Elemente werden mit '`+`' aneinander gehängt.

3. Die explizite Umwandlung eines Datentyps erfolgt mit '`('Typ')`' – im Beispiel macht '`((double) N)`' aus der ganzen Zahl `N` eine Fließkommazahl (allerdings ist eine solche explizite Umwandlung in diesem Fall nicht erforderlich). Diese Konstruktion unterscheidet sich etwas von der C++-Variante, ist aber mit mit der C-Konvention identisch, kann also auch in C++ verwendet werden.

4. Mathematische Funktionen gehören zur Klasse `Math` und werden z.B. mit '`Math.sqrt()`', '`Math.cos()`', '`Math.sin()`' und '`Math.exp()`' aufgerufen.

5. Die Klasse `Math` enthält auch die Konstante π als '`Math.PI`'.

6. Bei der Deklaration übernimmt der Modifizierer '`final`' in etwa die Rolle, die '`const`' in C++ spielt (in Java gibt es letzeres nicht).

Die Bedeutung von '`String[] args`' illustriert folgendes Programm

```
public class argumente {            // Die Datei MUSS argumente.java
                                    // heissen
 public static void
        main(String[] args)         // args ist ein Array von Zeichenketten
  {
   for(int i=0; i<args.length; i++) // args.length enthaelt Anzahl
      System.out.println((i+1) + ". Argument: "
                         + args[i]);  // i-tes Argument ausgeben
  }
}
```

Ein Aufruf nach dem Compilieren z.B. mit

```
java argumente 1 Zwei C
```

sollte selbsterklärend sein.

A.2 Arrays

Zur Illustration von Besonderenheiten beim Umgang mit Arrays in Java diene folgendes kleine Programm `array.java`, das die exakte Lösung (0.14) von A0.2 auf einem Gitter in die Datei `PhiExakt.dat` schreibt:

```
// Ein kleines Beispielprogramm fuer Arrays und Ausgabe in Dateien
import java.io.*;

public class array {                    // Die Datei MUSS array.java heissen
final int N=41;                         // Die inneren Gitterpunkte
final int L=N+2;

// cosh ist in Java nicht implementiert, also machen wir es selber
public double cosh(double x)
 {
  return((Math.exp(x)+Math.exp(-x))/2);
 }

// Konvertiere einen Gitter-Punkt zu x-Koordinate
public double x_coord(int r)
 {
  return(Math.PI*(r+1)/((double) N+1)-Math.PI/2);
 }

// Konvertiere einen Gitter-Punkt zu y-Koordinate
public double y_coord(int s)
 {
  return(Math.PI*(s+1)/((double) N+1)-Math.PI/2);
 }

public array()                          // Das Programm steckt im Konstruktor
 throws IOException {                   // IO-Exceptions muessen gefangen werden
  {
   double[][] Phi = new double[N][N];

   // Initialisierung mit exakter Loesung fuer A0.2
   for(int r=0; r<Phi.length; r++)      // Array-Dimensionen koennen aus
    for(int s=0; s<Phi[r].length; s++)  // .length ausgelesen werden
     {
      double x=x_coord(r);
      double y=y_coord(s);
      Phi[r][s] = (Math.cos(x)*cosh(y)+cosh(x)*Math.cos(y))/cosh(Math.PI/2);
     }
```

```
   // Zur Illustration machen wir die Ausgabe ueber einen Klon von Phi
   double PhiKlon[][] = (double [][]) Phi.clone();

   // Ausgabe in Datei:
   // So koennen wir in PhiExakt.dat mit println() schreiben
   PrintWriter PhiDatei = new PrintWriter(new FileWriter("PhiExakt.dat"));

   for(int r=0; r<PhiKlon.length; r++)
    {
     for(int s=0; s<PhiKlon[r].length; s++)
      {
       double x=x_coord(r);
       double y=y_coord(s);
       PhiDatei.println(x + "\t" + y + "\t" + PhiKlon[r][s]);
      }
     PhiDatei.println();
    }

   PhiDatei.close();                    // Datei wieder schliessen
  }
 }

public static void main(String[] args)
 throws IOException {                   // IO-Exceptions muessen gefangen werde
  {
   array start = new array();           // Starte Programm ueber Konstruktor
  }
 }
}
```

Folgene Eigenschaften von Java-Arrays sind besonders hervorzuheben:

1. Ein Array wird über

   Datentyp`[]`...`[]` Name`;`

   deklariert. Die Anzahl der eckigen Klammern gibt die Anzahl der Indizes an.

2. Zur *Erzeugung* eines Arrays wird '`new`' verwendet:

   `new` Datentyp`[`n_1`]`...`[`n_r`]`

n_1 bis n_r legen hierbei die Größe fest und müssen ganzzahlige Ausdrücke sein. Deklaration und Erzeugung können zusammengefaßt werden (siehe obiges Beispiel).

3. Die Anzahl der Elemente eines Arrays wird gespeichert und kann mit 'Name`.length`' abgefragt werden. Diese Speicherung der Elementzahl hat zur Folge, daß:

   a. Dimensionen von Arrays bei Methoden nicht deklariert werden müssen (und auch nicht dürfen).

   b. Java die Einhaltung der Array-Grenzen zur Laufzeit testen kann und dies auch tut.

4. Mehrdimensionale Arrays werden als Arrays von Arrays behandelt. Folglich gibt 'Name`.length`' den Bereich des ersten Index an, 'Name`[Index].length`' den Bereich des zweiten Index, usw. (man beachte, daß die zweite Länge tatsächlich vom ersten 'Index' abhängen kann !).

5. Java implementiert aus Sicherheitsgründen keine Zeiger. Für Arrays kommt die '`clone()`'-Methode dem Kopieren von Zeigern am nächsten. Diese Methode klont (kopiert) die Elemente des Array-Objekts in ein neues. Bei mehrdimensionalen Arrays wird aber nur die erste Dimension kopiert, d.h. Unter-Arrays werden gemeinsam genutzt !

Im Übrigen (Indizierung von 0 bis $n-1$, Initialisierung bei Deklaration mit '`{` Werte `}`', ...) verhalten sich Java-Arrays wie C-Arrays.

Nebenbei zeigt `array.java` auch, wie man die Ausgabe in eine Datei schreiben kann:

1. Das Paket `java.io` implementiert u.a. eine Klasse `FileWriter`, die die Ausgabe in eine Datei erlaubt. Da diese Klasse eine '`IOException`' auswerfen kann, muß man entweder einen '`try`', '`catch`'-Block um die Anweisungen setzen, oder die Ausnahme mit '`throws`' weitergeben.

2. Die Klasse `PrintWriter` ermöglicht die Ausgabe verschiedener Datentypen als Text. Sie implementiert die Methoden '`print()`' und '`println()`', die wir bereits im Kontext der `System`-Klasse kennengelernt haben.

A.3 Grafik

Nun kommen wir endlich zur versprochenen Grafik mit Java, die vom Advanced Window Toolkit-Paket `java.awt` implementiert wird. Das folgende Programm `fenster.java` illustriert das Öffnen eines Fensters und einige elementare Zeichen-Methoden:

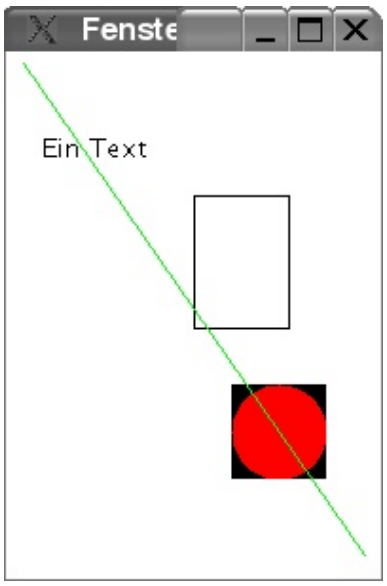
```
import java.awt.*;                    // Abstract Window Toolkit (awt)
                                      // einbinden
public class fenster extends Frame { // Die Datei "fenster.java"
                                      // erweitert die Klasse Frame
public static void main(String[] args)
 {
   fenster frame = new fenster();     // Ein Object "frame" dieser Klasse
 }                                    // erzeugen

public fenster()                      // Der Konstruktor dieser Klasse
 {
   super("Fenster");                  // Ein Objekt der Superklasse "Frame"
                                      // erzeugen
   setSize(200,300);                  // Groesse: 200*300 Pixel
   setLocation(50, 100);              // Position 50,100 auf Bildschirm
   setBackground(Color.white);        // Weissen Hintergrund und
   setForeground(Color.black);        // schwarzen Vordergrund setzen
   setVisible(true);                  // Dieses Fenster auch anzeigen
 }

public void paint(Graphics g)         // Diese Routine wird zum Neuzeichnen
 {                                    // des Fensterinhalts aufgerufen
  g.drawRect(100,100,50,70);          // Rechteck zeichnen
  g.setFont(new Font("SansSerif", Font.PLAIN, 14));  // Schrift waehlen
  g.drawString("Ein Text", 20, 80);   // Text ausgeben
  g.fillRect(120,200,50,50);          // Ausgefuelltes Rechteck zeichnen
  g.setColor(Color.red);              // Farbe Rot waehlen
  g.fillOval(120,200,50,50);          // Ausgefuellten Kreis zeichnen
  g.setColor(Color.green);            // Farbe Gruen waehlen
  g.drawLine(10,30,190,290);          // Linie von (10,30) nach (190,290)
 }                                    // zeichnen
}
```

Unter Linux mit KDE 3.1 erzeugt dieses Programm ein Fenster, das z.B. wie

folgt aussieht:



Das Aussehen des Rahmens hängt hier vom Betriebssystem bzw. Windowmanager ab, das Aussehen des Fenster-*Inhalts* wird allerdings von Java systemunabhängig definiert.

Da das Programm keine Funktionalität enthält, um das Fenster ordnungsgemäß zu schließen, muß es vom Benutzer abgebrochen werden, z.B. durch Eingabe von '`Strg-C`' in dem Fenster, aus dem das Programm gestartet wurde.

Weitere Bemerkungen zu `fenster.java`:

1. `import java.awt.*;`

   bindet alle Klassen ('`*`') des Abstract Window Toolkit ein.

2. Ein Fenster ist ein Objekt der Klasse '`Frame`'. Deswegen müssen wir diese Klasse erweitern.

3. Das Hauptprogramm `main` ruft lediglich den Konstruktor unserer neuen Klasse `fenster` auf. Letztere Methode öffnet dann tatsächlich ein Fenster unter Benutzung folgender Methoden:

   (a) Der Konstruktor der Klasse `Frame` kann mit einer Zeichenkette ('`String`') als Argument aufgerufen werden, die dann den Titel des Fensters definiert. Im Beispielprogramm handelt es sich um

einen Aufruf des Konstruktors der Superklasse. Deswegen kommt '`super()`' zum Einsatz.

(b) '`setSize(`b`,`h`)`' definiert die Breite b und Höhe h des Fensters.

(c) '`setLocation(`x`,`y`)`' plaziert die linke obere Ecke des Fensters an Position (x, y) (auf dem Bildschirm).

(d) '`setVisible(true)`' macht das Fenster sichtbar.
'`setVisible(false)`' würde es schließen.

(e) '`setBackground()`' und '`setForeground()`' setzen die Farbe für den Hinter- bzw. Vordergrund des Fensters.
Die Klasse `Color` enthält einige Konstanten (Attribute) für Farben – das Beispielprogramm verwendet '`white`', '`black`', '`red`' sowie '`green`'.
Der Konstruktor '`Color(`r`,`g`,`b`)`' der Klasse `Color` erlaubt die Auswahl beliebiger Farben. r, g und b sind die Rot-, Grün-, bzw. Blau-Anteile der neuen Farbe. Die ganzen Zahlen r, g und b laufen von 0 (dunkel) bis 255 (maximale Intensität). $r = g = b = 255$ liefert Weiß und $r = g = b = 0$ führt zu Schwarz.

4. Der Ursprung des Koordinatensystem $(0, 0)$ liegt links oben. Die y-Koordinaten werden nach *unten* positiv gezählt, x-Koordinaten sind wie gewohnt nach rechts positiv.

5. Um etwas in einem Fenster auszugeben, muß eine Methode `paint` definiert werden. `paint` wird immer dann automatisch aufgerufen, wenn die Grafik aktualisiert werden muß, z.B. weil das Fenster verdeckt war oder, weil das Fenster zum ersten Mal angezeigt wird.
Der Methode `paint` wird vom Virtuellen Java-Maschine ein Objekt der Klasse '`Graphics`' übergeben. Wir verwenden hier vor allem die folgenden Zeichen-Methoden aus dieser Klasse:

   (a) '`setColor()`' setzt die Farbe, die von den folgenden Zeichenoperationen verwendet wird.

   (b) '`drawLine(`x_1`,`y_1`,`x_2`,`y_2`)`' zeichnet eine Linie von (x_1, y_1) nach (x_2, y_2).

   (c) '`drawRect(`x`,`y`,`b`,`h`)`' zeichnet ein Rechteck der Breite $b + 1$ und Höhe $h + 1$ und der linken oberen Ecke (x, y).
   '`drawOval(`x`,`y`,`b`,`h`)`' zeichnet ein Oval innerhalb des durch die Argumente definierten Rechtecks. Ist $b = h$ (wie in unserem Beispiel), so entsteht ein Kreis.

Ersetzt man den Anfang '`draw`' des Namens dieser Methoden durch '`fill`', so erhält man ausgefüllte Varianten.

(d) '`Font(`Schriftart`,` Stil`,` Größe`)`' ist der Konstruktor der `Font`-Klasse (Font = Zeichensatz). Mögliche Schriftarten sind z.B. das hier verwendete `"SansSerif"`, `"Serif"`, `"Monospaced"` und `"Dialog"`.
Der Stil ist ein Attribut der Klasse '`Font`' und kann '`PLAIN`' (Normal – wie in diesem Beispiel), '`BOLD`' (Fett) und '`ITALIC`' (Kursiv) sein[16].
Die Größe ist die Schrifthöhe in Pixeln.

(e) '`setFont(`Zeichensatz`)`' wählt den Zeichensatz, der dann bei allen späteren Textausgaben verwenden wird. Das Argument dieser Methode muß ein `Font`-Objekt sein.

(f) '`drawString(`Text`,`x`,`y`)`' zeichnet den angegebenen Text, wobei (x, y) der linke Rand der Grundlinie ist.

6. **Achtung:** Ein Teil der Zeichenfläche eines Fensters wird von seinem Rahmen verwendet.

Als nächstes stellen wir ein Programm `animation.java` vor, das eine einfache Bewegung darstellt, nämlich einen Punkt entgegen den Uhrzeigersinn um einen Kreis kreisen läßt:

```
import java.awt.*;

public class animation extends Frame {
 static double x=90, y=0;              // Koordinaten des Punkts

public static void main(String[] args)
 {
   animation frame = new animation();  // Fenster mit Animation erzeugen
 }

public animation()
 {
   super("Ein Fenster mit Animation");
   setSize(250,250);
   setLocation(50, 100);
   setBackground(Color.white);
```

[16] '`BOLD`' und '`ITALIC`' können mittels Addition '`+`' kombiniert werden.

```
   setForeground(Color.black);
   setVisible(true);
                                        // Elegant ist es zwar nicht,
   while(true)                          // aber die Animation kann hierhin
    for(int phi=0; phi<360; phi++)      // Schleife ueber Winkel phi
     {
      x=90*Math.cos(phi*Math.PI/180);   // x- und y- Koordinate aktualisieren
      y=90*Math.sin(phi*Math.PI/180);   // Java kennt Pi !
      repaint((int) (118+x),            // Neu zeichnen -> Animation (!)
              (int) (118-y),            // Nur einen kleinen Bereich neu
              7, 7);                    // zeichnen, sonst flackert's arg
      warte(100);                       // 100 Millisekunden warten
     }
 }

 public void paint(Graphics g)
 {
  g.fillOval(100,100,40,40);            // Kreis in Mitte zeichen
  g.drawLine((int) (120+x),             // und einen Punkt daneben
             (int) (120-y),             // Da Java keinen speziellen
             (int) (120+x),             // Punkt-Zeichen-Befehl hat,
             (int) (120-y));            // verwenden wir drawLine()
 }

 public static void warte(long ms)      // Prozedur zum Warten von
 {                                      // ms Millisekunden
   try                                  // - ohne weitere Erklaerung
    { Thread.sleep(ms); }
   catch(InterruptedException e) {}
 }
}
```

Folgende Elemente dieses Programms sind neu bzw. bedürfen der Erklärung:

1. Die Animation kann in dem Konstruktor laufen[17]. In unserem Beispiel enthält dieser dann eine Endlosschleife, kehrt also nie zurück.

2. '`repaint()`' erzwingt die sofortige Aktualisierung einer Zeichnung. `repaint` ruft die vordefinierte Metthode `update` auf, die die Zeichen-

[17]Eigentlich wäre es besser, diese in einem speziellen `Thread` laufen zu lassen. Dieser Einstieg in den internen Prozeß-Ablauf würde aber hier zu weit führen.

fläche löscht und dann ihrerseits die Methode `paint` aufruft.
Um Flackern zu reduzieren, kann mit

`repaint(`x`,`y`,`b`,`h`)`

das Neuzeichnen auf das über die Argumente definierte Rechteck eingeschränkt werden.
Probieren Sie ruhig einmal aus, was passiert, wenn Sie im Programm `animation.java` erst die `warte`-Schleife herausnehmen, und in einem zweiten Schritt dann die Argumente von `repaint` löschen !

3. Die letzte Methode '`warte(`ms`)`' wartet die angegebene Anzahl von Millisekunden. Sie dient zum Steuern der Geschwindigkeit der Animation.
Hier wird die Methode '`sleep()`' der `Thread`-Klasse verwendet. Da diese nicht ohne Ausnahmebehandlung verwendet werden darf, kommen hier auch Ausnahmeklassen ('`try`', '`catch`') zum Einsatz.

Wir wollen nun dieses Unterkapitel mit einem letzten Beispielprogramm abschließen. Es handelt sich dabei um eine Variante des Programms `array.java` aus Kapitel A.2, das die Funktion Φ aus (0.14) nicht in eine Datei schreibt, sondern farbkodiert am Bildschirm darstellt. Damit nebenbei weitere Aspekte animierter Grafik illustriert werden können, dreht das Programm langsam die Helligkeit der Darstellung von ganz schwarz bis zum maximalen Kontrast auf. Das Programm heißt `einblend.java`:

```
import java.awt.*;                        // Abstract Window Toolkit (awt)
                                          // einbinden
public class einblend extends Frame {     // Die Datei "einblend.java"
                                          // erweitert die Klasse Frame
final int pixsize=4;                      // Groesse eines einzelnen Pixels
final int N=151;                          // Die inneren Gitterpunkte
final int L=N+2;

public double[][] Phi;                    // Das Feld
public double hell=0;                     // Helligkeit

// cosh ist nicht implementiert, also machen wir es selber
public double cosh(double x)
 {
  return((Math.exp(x)+Math.exp(-x))/2);
 }
```

```
// Konvertiere einen Gitter-Punkt zu x-Koordinate
public double x_coord(int r)
 {
  return(Math.PI*(r+1)/((double) N+1)-Math.PI/2);
 }

// Konvertiere einen Gitter-Punkt zu y-Koordinate
public double y_coord(int s)
 {
  return(Math.PI*(s+1)/((double) N+1)-Math.PI/2);
 }

public einblend()
 {
  super("Einblenden");
  setSize(pixsize*N+20,pixsize*N+40);
  setLocation(50, 100);
  setBackground(Color.black);
  setForeground(Color.white);
  setVisible(true);

  Phi = new double[N][N];
  // Initialisierung mit exakter Loesung fuer A0.2
  for(int r=0; r<Phi.length; r++)      // Array-Dimensionen koennen aus
   for(int s=0; s<Phi[r].length; s++)  // .length ausgelesen werden
    {
     double x=x_coord(r);
     double y=y_coord(s);
     Phi[r][s] = (Math.cos(x)*cosh(y)+cosh(x)*Math.cos(y))/cosh(Math.PI/2);
    }

  for(hell=0; hell<=0.991; hell+=0.01)    // Helligkeit langsam hochdrehen
   {
    repaint();                            // Neu zeichnen
    warte(100);                           // 0.1s warten
   }
  System.out.println("Fertig - Bitte Strg-C druecken");
 }

public static void main(String[] args)
```

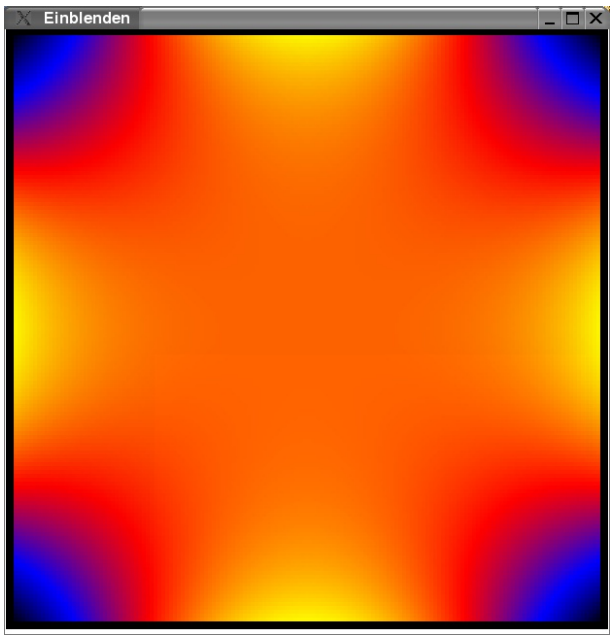
```
 {
  einblend start = new einblend();        // Starte Programm ueber Konstruktor
 }

public void paint(Graphics g)
 {
  for(int r=0; r<Phi.length; r++)
   for(int s=0; s<Phi[r].length; s++)
    {
     double blue = 3*255*Phi[r][s];       // Farbe berechnen
     if(blue > 255)                       // Blau-Anteil
       blue = 2*255-blue;
     if(blue < 0)
       blue = 0;
     double red = 255*(3*Phi[r][s]-1);    // Rot-Anteil
     if(red < 0)
       red = 0;
     if(red > 255)
       red = 255;
     double green = 255*(3*Phi[r][s]-2);  // Gruen-Anteil
     if(green < 0)
       green = 0;
     if(green > 255)
       green = 255;
     g.setColor(new Color((int) (red*hell),
                          (int) (green*hell), (int) (blue*hell) ) );
     g.fillRect(10+r*pixsize,30+s*pixsize,pixsize,pixsize);
    }
 }

/ Wir ueberschreiben die update-Methode um Flackern zu minimieren
public void update(Graphics g)         // Fenster-Inhalt aktualisieren
 {
  paint(g);                            // Zeichenfunktionen ausfuehren
 }

public static void warte(long ms)      // Prozedur zum Warten von
 {                                     // ms Millisekunden
   try { Thread.sleep(ms); } catch(InterruptedException e) {}
 }
}
```

Das Endergebnis sollte ungefähr wie folgt aussehen:



An dieser Stelle seien nur noch wenige Bemerkungen ergänzt:

1. Wie bereits erwähnt, ist '`update(`g`)`' die Methode, die immer zum Neuzeichnen des Fensterinhalts aufgerufen wird (z.B. durch Aufruf von `repaint`). Hier wird die vordefinierte Version dieser Methode überschrieben: Der Fenster-Inhalt wird nicht gelöscht, sondern wir rufen einfach nur `paint` zum Neuzeichnen auf. Auf diese Weise eliminieren wir Flackern.
   g ist ein `Graphics`-Objekt.

2. '`getGraphics()`' liefert den Graffikkontext (Datentyp '`Graphics`'). Mit diesem Grafikkontext können die oben beschriebenen Zeichen-Methoden in der dort angegebenen Weise ausgeführt werden.
   Man kann insbesondere auch den '`repaint()`'-Aufruf durch

   ```
   paint(getGraphics())
   ```

   ersetzen und auf das Überschreiben der `update`-Methode verzichten. So aktualisiert man den Fenster-Inhalt unter Umgehung der `update`-Methode sowie der dort vordefinierten Löschfunktion.

B Komplexe Zahlen in C++, C99 und Java

Hier soll kurz die Verwendung komplexer Zahlen illustriert werden, wobei als Beispiel eine Implementierung der FFT aus Kapitel 3.3.1 dient.

B.1 C++

C++ implementiert komplexe Zahlen in der Standard-Template Library. Zur Illustration diene die folgende Implementierung der FFT einschließlich eines Test-Teils:

```
// C++ Implementierung der Fast Fourier Transformation
#include <iostream>
#include <cmath>
#include <complex>      // Damit wir komplexe Zahlen kennen

using namespace std;    // Damit nicht ueberall "std::" vorstehen muss

////////// Erst einige Definitionen
const complex<double> I(0,1);   // Die imaginaere Einheit
const double Pi = 3.14159265358979323846264;

// *** Die eigentliche FFT
// f[]: Eingabe: zu transformierende Daten
//      Ausgabe: Ergebnis der Transformation
// N:   Anzahl der Datenpunkte (2er-Potenz !!!!)
// sign=-1: Hintransformation; sign=1: Ruecktransformation

void fft(complex<double> f[], int N, int sign) {
  int mask;
  // *** Teste, ob N 2er-Potenz ist
  for(mask=1; mask<N; mask <<= 1)       ;
  if(mask != N) {
    cerr << "N = " << N << " ist keine 2er-Potenz !\n";
    exit(13);
   }
  // *** Teile Daten durch sqrt(N)
  double isqrtN = 1/sqrt(double(N));
  for(int r=0; r<N; r++)
    f[r] *= isqrtN;
```

```
  // *** Bit-Umkehr
  for(int t=0, r=0; r<N; r++) {
     if(t > r) {        // Vertausche f[r] und f[t]
        complex<double> temp = f[r];
        f[r] = f[t];
        f[t] = temp;
      }
     mask = N;          // Bit-umgekehrtes Inkrement von t
     do {
        mask >>= 1;
        t ^= mask;
      } while((! (t & mask)) && mask);
   }
  // *** Danielson-Lanczos Teil
  int n, no2 = 1;
  for(int m=1; (n=(no2 << 1)) <= N; m++) {
     complex<double> W = exp(I*(sign*2*Pi/n));          // W_n
     complex<double> Wk = 1;
     for(int k=0; k<no2; k++) {
        for(int l=k; l<N; l+=n) {
           complex<double> temp = Wk*f[l+no2];
           f[l+no2] = f[l] - temp;
           f[l] += temp;
         }
        Wk *= W;        // Wk = W^k
      }
     no2 = n;
   }
 }

////////////////////////////// Test-Teil

const int N=8;
double Daten[8] = {1, 2, 3, 4, 3, 2, 1, Pi};

// Hauptprogramm

int main(int argc, char *argv[]) {
  complex<double> f[N];
```

```
  for(int r=0; r<N; r++)
    f[r] = Daten[r];
  fft(f, N, -1);                 // Hintransformation
  for(int r=0; r<N; r++)
    cout << "^f[" << r << "] = " << f[r] << endl;
  fft(f, N, 1);                  // Rueckransformation
  for(int r=0; r<N; r++)
    cout << "f[" << r << "] = " << f[r]
         << "\tOriginaldaten: " << Daten[r] << endl;
}
```

Erläuterungen:

1. `#include <complex>`
   definiert komplexe Zahlen als Klassen-Template.

2. Das Template erlaubt die Verwendung verschiedener Genauigkeiten. Im Beispiel kommen doppelt genaue komplexe Fließkommazahlen zum Einsatz, die über den Datentyp '`complex<double>`' definiert werden.

3. C++ implementiert die Grundrechenarten durch Überladen der entsprechenden Operatoren, so daß man bequem mit den Symbolen '`+`', '`-`', '`*`', und '`/`', arbeiten kann.

4. Die Ausgabefunktion wird erweitert, so daß komplexe Zahlen wie gewohnt in einen Ausgabe-Stream eingebaut werden können.

5. Von den meisten elementaren Funktionen existieren komplexe Varianten. Insbesondere verwendet das Beispiel die Exponentialfunktion '`exp(`c`)`' einer komplexen Zahl c.

6. Der Konstruktor der Klasse hat zwei Argumente, die den Realteil und den Imaginärteil der Zahl initialisiern. Im Beispiel definiert

   ```
   const complex<double> I(0,1);
   ```

   die imaginäre Einheit i als '`I`'.

7. '`real(`c`)`' und '`imag(`c`)`' extrahieren Real- bzw. Imaginärteil der komplexen Zahl c; '`conj(`c`)`' liefert die zu c komplex konjugierte Zahl (alle drei im Beispiel nicht verwendet).

B.2 C99

C-Compiler, die den C99-Standard implementieren, können mit komplexen Zahlen arbeiten. Unsere FFT einschließlich Test-Teil sieht in C99 wie folgt aus:

```
/* C99 Implementierung der Fast Fourier Transformation */
#include <stdio.h>
#include <math.h>
#include <complex.h>    /* Damit wir komplexe Zahlen kennen */

#define Pi 3.14159265358979323846264    /* Pi */

/* *** Die eigentliche FFT
 * f[]: Eingabe: zu transformierende Daten
 *      Ausgabe: Ergebnis der Transformation
 * N:   Anzahl der Datenpunkte (2er-Potenz !!!!)
 * sign=-1: Hintransformation; sign=1: Ruecktransformation */

void fft(complex f[], int N, int sign) {
  double isqrtN;
  complex temp, W, Wk;
  int r, t, mask, n, no2, m, k, l;
  /* *** Teste, ob N 2er-Potenz ist */
  for(mask=1; mask<N; mask <<= 1)       ;
  if(mask != N) {
    fprintf(stderr, "N = %d ist keine 2er-Potenz !\n", N);
    exit(13);
   }
  /* *** Teile Daten durch sqrt(N) */
  isqrtN = 1/sqrt(N);
  for(r=0; r<N; r++)
    f[r] *= isqrtN;
  /* *** Bit-Umkehr */
  for(t=0, r=0; r<N; r++) {
     if(t > r) {        /* Vertausche f[r] und f[t] */
        temp = f[r];
        f[r] = f[t];
        f[t] = temp;
      }
     mask = N;          /* Bit-umgekehrtes Inkrement von t */
```

```
    do {
       mask >>= 1;
       t ^= mask;
     } while((! (t & mask)) && mask);
  }
  /* *** Danielson-Lanczos Teil */
  no2 = 1;
  for(m=1; (n=(no2 << 1)) <= N; m++) {
     W = cexp(I*sign*2*Pi/n);   /* W_n (C99 kennt I) */
     Wk = 1;
     for(k=0; k<no2; k++) {
        for(l=k; l<N; l+=n) {
           complex temp = Wk*f[l+no2];
           f[l+no2] = f[l] - temp;
           f[l] += temp;
         }
        Wk *= W;        /* Wk = W^k */
      }
     no2 = n;
   }
 }

/* ************* Test-Teil */

const int N=8;
double Daten[8] = {1, 2, 3, 4, 3, 2, 1, Pi};

/* Hauptprogramm */

int main(int argc, char *argv[]) {
  int r;
  complex f[N];

  for(r=0; r<N; r++)
    f[r] = Daten[r];
  fft(f, N, -1);                /* Hintransformation */
  for(r=0; r<N; r++)
    printf("^f[%d] = %1.8g%s%1.8g*I\n", r, creal(f[r]),
       (cimag(f[r])>0)?"+":"", cimag(f[r]) );
  fft(f, N, 1);                 /* Rueckransformation */
```

```
  for(r=0; r<N; r++)
    printf("f[%d] = %1.8g%s%1.8g*I\tOriginaldaten: %1.8g\n",
        r, creal(f[r]), (cimag(f[r])>0)?"+":"", cimag(f[r]),
        Daten[r] );
}
```

Erläuterungen:

1. `#include <complex.h>`
   definiert einige Eigenschaften komplexer Zahlen. Insbesondere wird die imaginäre Einheit i als '`I`' definiert.

2. Der Datentyp '`complex`' ist äquivalent zu '`double complex`' und definiert eine doppelt genaue komplexe Fließkommazahl.

3. Da die komplexen Zahlen vom Compiler implementiert werden, kann man bequem mit den Symbolen '`+`', '`-`', '`*`', und '`/`' arbeiten.

4. Für komplexe Zahlen existieren keine speziellen Ausgabefunktionen. Deswegen müssen wir Real- bzw. Imaginärteil der komplexen Zahl c mit Hilfe von '`creal(`c`)`' bzw. '`cimag(`c`)`' extrahieren und gesondert ausgeben.

5. Von den meisten elementaren Funktionen existieren komplexe Varianten, die durch Voranstellung eines '`c`' gekennzeichnet werden. Im Beispiel kommt die komplexe Exponentialfunktion '`cexp(`c`)`' einer komplexen Zahl c zum Einsatz.

6. Weitere nützliche, im Beispiel jedoch nicht verwendete Funktionen für komplexe Zahlen c sind '`carg(`c`)`' und '`conj(`c`)`'.

B.3 Java

In Java werden komplexe Zahlen als Objekte implementiert. Wir verwenden die Klasse `Complex.java` [33]. Unsere FFT einschließlich Test-Teil sieht nun in Java wie folgt aus:

```
// Java Implementierung der Fast Fourier Transformation
// Braucht Complex.java von
//    http://www.cs.princeton.edu/introcs/97data/Complex.java

// *** Die eigentliche FFT
// f[]: Eingabe: zu transformierende Daten
```

```
//      Ausgabe: Ergebnis der Transformation
// sign=-1: Hintransformation; sign=1: Ruecktransformation

public class fft {
public static void FFT(Complex[] f, int sign) {
  int N=f.length;       // Java-Arrays kennen ihre Laenge
  int mask;
  // *** Teste, ob N 2er-Potenz ist
  for(mask=1; mask<N; mask <<= 1)       ;
  if(mask != N)
    throw new RuntimeException("N = " + " ist keine 2er-Potenz !");
  // *** Teile Daten durch sqrt(N)
  double isqrtN = 1/Math.sqrt(N);
  for(int r=0; r<N; r++)
    f[r] = f[r].times(isqrtN);
  // *** Bit-Umkehr
  for(int t=0, r=0; r<N; r++) {
     if(t > r) {        // Vertausche f[r] und f[t]
        Complex temp = f[r];
        f[r] = f[t];
        f[t] = temp;
      }
     mask = N;          // Bit-umgekehrtes Inkrement von t
     do {
        mask >>= 1;
        t ^= mask;
      } while(((t & mask) == 0) && (mask != 0));
   }
  // *** Danielson-Lanczos Teil
  int n, no2 = 1;
  for(int m=1; (n=(no2 << 1)) <= N; m++) {
     Complex W = new Complex(Math.cos(2*Math.PI/n),
                 sign*Math.sin(2*Math.PI/n));           // W_n
     Complex Wk = new Complex(1, 0);
     for(int k=0; k<no2; k++) {
        for(int l=k; l<N; l+=n) {
           Complex temp = Wk.times(f[l+no2]);
           f[l+no2] = f[l].minus(temp);
           f[l]     = f[l].plus(temp);
         }
```

```
        Wk = Wk.times(W);       // Wk = W^k
       }
      no2 = n;
    }
  }

  ////////////////////////////// Test-Teil

  // Hauptprogramm

  public static void main(String[] args) {
    double[] Daten = {1, 2, 3, 4, 3, 2, 1, Math.PI};
    int N=Daten.length;          // Java-Arrays kennen ihre Laenge
    Complex[] f = new Complex[N];

    for(int r=0; r<N; r++)
      f[r] = new Complex(Daten[r], 0);
    FFT(f, -1);                  // Hintransformation
    for(int r=0; r<N; r++)
      System.out.println("^f[" + r + "] = " + f[r]);
    FFT(f, 1);                   // Rueckransformation
    for(int r=0; r<N; r++)
      System.out.println("f[" + r + "] = " + f[r]
           + "\tOriginaldaten: " + Daten[r]);
  }
}
```

Erläuterungen:

1. Komplexe Zahlen und Arithmetik werden in `Complex.java` definiert [33]. Objekte der Klasse '`Complex`' sind doppelt genaue komplexe Fließkommazahlen.

2. Java verbietet aus Sicherheitsgründen bewußt das Überladen elementarer Operatoren. Da Java-Compiler auch keine komplexe Arithmetik implementieren, müssen die Grundrechenarten ein wenig umständlich über Methoden aufgerufen werden: 'a`.plus(`b`)`', 'a`.minus(`b`)`' und 'a`.times(`b`)`' erzeugen jeweils ein neues Objekt der Klasse '`Complex`', das den Wert von $a + b$, $a - b$ bzw. $a\,b$ enthält.

3. Die Methode '`toString()`' wird von `Complex.java` erweitert, so daß komplexe Zahlen bequem ausgegeben werden können.

4. Der Konstruktor der Klasse '`Complex`' hat zwei Argumente, die den Realteil und den Imaginärteil initialisieren. Im Beispiel wird z.B. $\mathrm{e}^{ix} = \cos(x) + i\,\sin(x)$ mit reellem x über

   `new Complex(Math.cos(`x`), Math.sin(`x`));`

   berechnet.

Literatur

[1] H. Gould, J. Tobochnik, *An Introduction to Computer Simulation Methods*, Addison-Wesley, Reading, Massachusetts (1996).

[2] N.V. Antonov, V.V. Nemoshkalenko, *Computational Methods in Solid State Physics*, Taylor & Francis, London (1999).

[3] K. Ohno, K. Esfarjani, Y. Kawazoe, *Computational Materials Science From Ab Initio to Monte Carlo Methods*, Springer Series in Solid-State Sciences, Band 129, Springer-Verlag, Berlin (1999).

[4] F. Knechtli, U. Wolff, B. Bunk, T. Korzec, *Computational Physics I*, Skriptum HU Berlin, WS 2004/05.

[5] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press (1992).

[6] A. Jennings, *Matrix Computation for Engineers and Scientists*, John Wiley, Chichester (1985).

[7] H.R. Schwarz, H. Rutishauser, E. Stiefel, *Numerik Symmetrischer Matrizen*, B.G. Teubner, Stuttgart (1968).

[8] J. Stoer, R. Burlisch, *Einführung in die Numerische Mathematik II*, Springer-Verlag, Berlin (1973).

[9] Ch. Großmann, H.-G. Roos, *Numerik partieller Differentialgleichungen*, B.G. Teubner, Stuttgart (1994).

[10] M. Bollhöfer, *Numerische Behandlung partieller Differentialgleichungen*, Skript TU Berlin, Sommersemester 2001.

[11] P.J. Roache, *Computational Fluid Dynamics*, Hermosa, Albuquerque (1982).

[12] J.H. Ferziger, M. Perić, *Computational Methods for Fluid Dynamics*, Springer-Verlag, Berlin (2002).

[13] C. Goto, Y. Ogawa, N. Shuto, F. Imamura, *IUGG/IOC TIME Project: Numerical Method of Tsunami Simulation with the Leap-Frog Scheme*, IOC Manual **35**, UNESCO (1997).

[14] G.B. Whitham, *Linear and Nonlinear Waves*, John Wiley, New York (1974).

[15] D. Chowdhury, L. Santen, A. Schadschneider, *Statistical Physics of Vehicular Traffic and Some Related Systems*, Phys. Rep. **329** (2000) 199-329. [cond-mat/0007053].

[16] A. Jüngel, *Dynamik eines Verkehrsflusses mit Ampelschaltung*, WWW-Seite an der Universität Mainz.

[17] R. Leis, *Initial Boundary Value Problems in Mathematical Physics*, B.G. Teubner, Stuttgart (1986).

[18] M. Holt, *Numerical Methods in Fluid Dynamics*, second edition, Springer-Verlag, Berlin (1984).

[19] US National Geophysical Data Center, *2-Minute Gridded Global Relief Data (ETOPO2)*.

[20] C.H. Su, C.S. Gardner, *Korteweg-de Vries Equation and Generalizations. III. Derivation of the Korteweg-de Vries Equation and Burgers Equation*, J. Math. Phys. **10** (1969) 536-539.

[21] J.-F. Gerbeau, B. Perthame, *Derivation of Viscous Saint-Venant System for Laminar Shallow Water; Numerical Validation*, INRIA Forschungsbericht Nr. 4084 (2000).

[22] C. Leforestier, R.H. Bisseling, C. Cerjan, M.D. Feit, R. Friesner, A. Guldberg, A. Hammerich, G. Jolicard, W. Karrlein, H.-D. Meyer, N. Lipkin, O. Roncero, R. Kosloff, *A Comparison of Different Propagation Schemes for the Time Dependent Schrödinger Equation*, J. Computational Phys. **94** (1991) 59-80.

[23] S.E. Koonin, *Computational Physics*, Benjamin/Cummings, Menlo Park, Kalifornien (1986).

[24] P.B. Visscher, *A Fast Explicit Algorithm for the Time-Dependent Schrödinger Equation*, Computers in Physics **5** (1991) 596-598.

[25] F. Knechtli, U. Wolff, B. Bunk, T. Korzec, *Computational Physics II*, Skriptum HU Berlin, SS 2004.

[26] W. Nolting, *Grundkurs: Theoretische Physik. 5. Quantenmechanik, Teil 1: Grundlagen*, Zimmermann-Neufang, Ulmen (1992).

[27] M.D. Feit, J.A. Fleck, Jr., A. Steiger, *Solution of the Schrödinger Equation by a Spectral Method*, J. Computational Phys. **47** (1982) 412-433.

[28] M. Creutz, *Xtoys*, WWW-Seite am Brookhaven National Laboratory.

[29] T.E. Hartman, *Tunneling of a Wave Packet*, J. Appl. Phys. **33** (1962) 3427-3433.

[30] S. Brouard, R. Sala, J.G. Muga, *Systematic Approach to Define and Classify Quantum Transmission and Reflection Times*, Phys. Rev. **A49** (1994) 4312-4325.

[31] D. Abts, *Grundkurs Java*, Vieweg, Braunschweig (1999).

[32] C. Ullenboom, *Java ist auch eine Insel*, 4. Auflage, Galileo-Computing, Bonn (2004).

[33] R. Sedgewick, K. Wayne, *Complex.java*, WWW-Seite an der Universität Princeton.