Simulationstechniken

Andreas Greiner, Lars Pastewka

24. Januar 2022

Institut für Mikrosystemtechnik
Albert-Ludwigs-Universität Freiburg

Dank an Anna Hoppe, Johannes Hörmann, Indre Jödicke, Antoine Sanner und die Teilnehmer der Vorlesung “Simulationstechniken” an der Universität Freiburg im Wintersemester 2020/21 für Kommentare, Anmerkungen und Überarbeitungen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Kontext: Die *Simulation* beschäftigt sich mit der numerischen (computergestützten) Lösung von *Modellen*. In diesem einleitenden Kapitel gehen wir auf Modellbildung ein und stellen unterschiedliche Klassen von Modellen vor. Diese Modelle werden mathematisch üblicherweise mit Hilfe von *Differentialgleichungen* beschrieben, d.h. die Simulation ist oft (aber nicht immer) die numerische Lösung von gewöhnlichen oder partiellen Differentialgleichungen. In dieser Lehrveranstaltung werden wir vornehmlich die Lösung von partiellen Differentialgleichungen mit Hilfe der *Methode der finiten Elemente* besprechen.

1.1 Modelle

Modelle sind üblicherweise für bestimmte Längenskalen angemessen. So kann z.B. ein Modell, welches explizit Atome beschreibt, auf Längenskalen $\sim$ nm angemessen sein; ein "makroskopisches" System mit Abmessungen $\sim$ mm würden wir mit einem solchen Modell aber nicht beschreiben wollen. Wir müssen uns daher darüber klar werden, welches der Phänomene der Ingenieurswissenschaften die Anwendung welcher physikalischer Modelle und welcher mathematischer Methoden verlangt.

Anmerkung: "Wollen" ist hier das falsche Wort. Zum einen ist es auf Grund begrenzter Computerressourcen nicht möglich, zum anderen versteckt sich in einem solchen Modell unter Umständen die essentielle Frage, die wir beantworten wollen, wie die legendäre Nadel im Heuhaufen.

Abbildung 1.1 zeigt in der vertikalen Anordnung von *Längenskalen* und

deren Zuordnung zu verschiedenen Beschreibungsebenen. Auf der kürzesten Längenskala ist meist eine quantenmechanische Beschreibung notwendig. Dies bedeutet, wenn wir die Phänomene in Å auflösen wollen, befinden wir uns auf der Beschreibungsebene der Quantenmechanik und alle zugrundeliegenden Modelle sind von quantenmechanischer Natur. D.h. wir haben es hier im nichtrelativistischen Fall mit der Schrödingergleichung zu tun. Diese ist in verschiedenen Methoden implementiert, wie z.B. der *Dichtefunktionaltheorie*, einer Vielteilchenbeschreibung des quantenmechanischen elektronischen Systems. Bei dieser Art der Vielteilchenbeschreibung handelt es sich, im Gegensatz zur *Molekulardynamik* als Methode auf einer größeren Längenskala, nicht um eine Beschreibung von Punktteilchen, sondern um gekoppelte Felder, was den Aufwand im Vergleich zu einer reinen Punktmechanik wesentlich erhöht. In der Punktmechanik haben wir es mit drei Orts- und drei Geschwindigkeitsvariablen für jedes der n wechselwirkenden Teilchen zu tun, während wir in einer quantenmechanischen Vielteilchenbeschreibung es mit einem Feld mit je drei n Ortsvariablen zu tun haben, nämlich $\Psi(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n; t)$.

Anmerkung:

- $1\,\mathring{A} = 10^{-10}\,\mathrm{m}$
- Atome in unserer Umwelt werden durch quantenmechanische Phänomene zusammengehalten. Modelle die auf quantenmechanischen Prinzipien fußen, heißen auch *ab-initio* ("von Anfang an") Modelle. Im Englischen werden solche Modell auch als "first principles" Modelle bezeichnet. Die fundamentale Gleichung, welche quantenmechanische Objekte beschreibt, ist die *Schrödingergleichung*. Diese selbst ist in der Tat bereits eine Näherung!
- Die Einteilchen-Schrödingergleichung lautet $i\hbar\frac{\partial}{\partial t}\Psi(\vec{r}, t) = \hat{H}\Psi(\vec{r}, t)$. Dies ist eine partielle Differentialgleichung für das orts- und zeitabhängige skalare Materiefeld $\Psi(\vec{r}, t)$, mit der Planckschen Konstanten $\hbar$ und dem Hamiltonoperator $\hat{H}$, der die Details des Modells enthält. Eine Bewegungsgleichung für viele wechselwirkende Teilchen, wie sie durch $\Psi(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n; t)$ beschrieben wird, ist unvergleichlich umfangreicher.
- "Semiklassisch" bedeutet, dass die Bewegung der Teilchen nach der klassischen Mechanik berechnet werden, die Wechselwirkungen der Teilchen untereinander aber aus quantenmechanischen Gesetzen abgeleitet sind. Dies ist natürlich eine Näherung, die es zu rechtfertigen gilt.

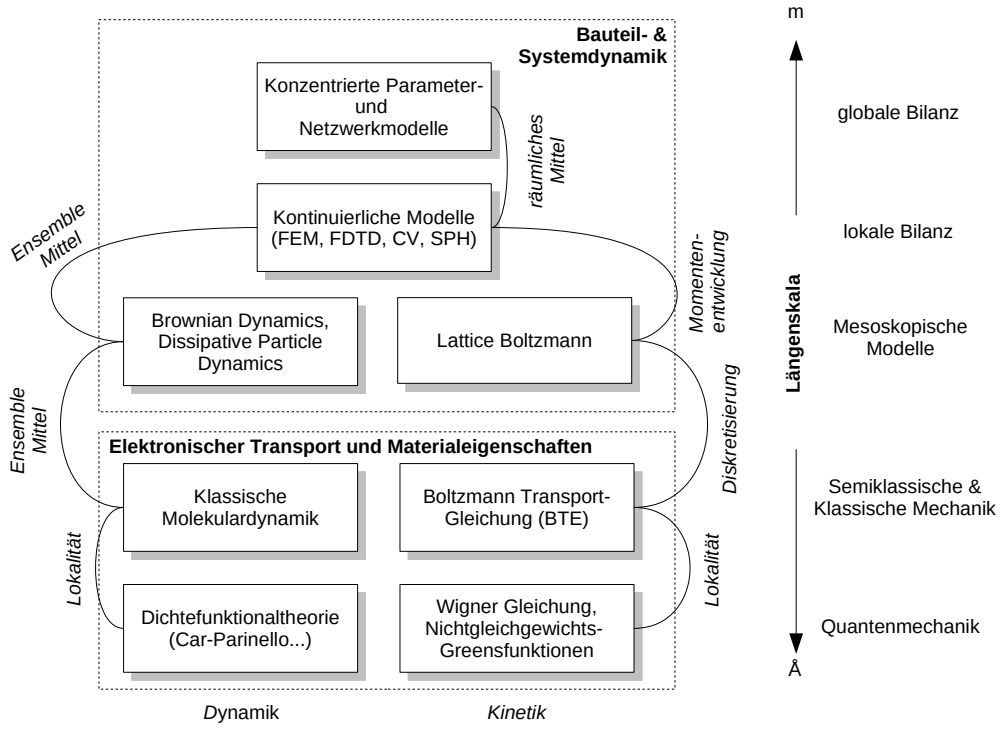

Abbildung 1.1: Die vertikale Anordnung der Kisten repräsentiert die Längenskale, welche auf der rechten Seite gezeigt ist. In den Kästen selbst stehen Simulationsmethode welche auf diesen Skalen Anwendung finden. In dieser Lehrveranstaltung beschäftigen wir uns mit der Diskretisierung von Feldern und wählen einen spezifischen Anwendungsfall, der in die *lokale Bilanz* hineinfällt.

- "Mesoskopisch" heißt, dass das Modell über eine innere Längenskala verfügt und/oder thermische Fluktuationen wichtig sind. Diese Modelle operieren meistens auf Längenskalen oberhalb der atomaren Skala ($\sim$ nm) aber unterhalb der Skalen unserer Wahrnehmung der Umwelt ($\sim$ mm).

- "Bilanz" heißt, dass der Kern der Beschreibung eine *Erhaltungsgröße* ist, die einfach gezählt werden kann. Erhalten sind z.B. Teilchenzahlen. Eine *Bilanzgleichung* oder Bilanzierung zählt dann einfach die Teilchen, die über ein gewisses Zeitinterval in ein Volumen hinein fließen, heraus fließen oder darin produziert werden. Weitere Erhaltungsgrößen, die man bilanzieren kann, sind der Impuls und die Energie. Die Bilanzgleichung wird auch *Kontinuitätsgleichung* genannt.

Auf der Ebene der semiklassischen und klassischen Mechanik, auch als kinetische Ebene bezeichnet, werden die Modelle entweder durch die Molekulardynamik beschrieben oder durch die Bewegungsgleichung der Einteilchen-Wahrscheinlichkeitsdichte im Phasenraum $f(\vec{r}, \vec{p})$ - mit den unabhängigen Variablen Ort $\vec{r}$ und Impuls $\vec{p}$. Im zweiten Fall haben wir eine Funktion $f(\vec{r}(t), \vec{p}(t), t)$ die von Ort, Impuls und der Zeit sowohl explizit, als auch implizit über $\vec{r}(t)$ und $\vec{p}(t)$ abhängt. Nehmen wir an, wir müssen $f(\vec{r}(t), \vec{p}(t), t)$ durch diskrete Stützstellen interpolieren. Dies sind bei einer geringen Auflösung von 10 Punkten pro Variabler schon bereits 10.000.000 Interpolationspunkte. Dies ist vielleicht handhabbar, die Auflösung ist aber nicht besonders gut. Und daher ist dieses Unterfangen eher unnütz. Wir wollen nicht verschweigen, dass es durchaus Methoden zur numerischen Lösung der beiden oben beschriebenen Probleme gibt, auf diese werden aber in dieser Veranstaltung nicht näher eingegangen.

1.2 Partikel

Grob können wir daher zwei Arten von Modellen unterscheiden: Modelle, die einzelne diskrete Elemente, beispielsweise Partikel (Atome, Moleküle, Körner, etc.), als zentrales Element haben und Modelle die kontinuierliche Felder (elektrostatisches Potential, Ionenkonzentrationen, mechanische Spannungen und Dehnungen) als zentrales Element haben. Im ersten Modelltyp werden Evolutionsgleichungen für diskrete Eigenschaften, welche auf den Partikeln definiert sind, wie z.B. deren Positionen $\vec{r}_i$ und Geschwindigkeiten $\vec{v}_i$, formuliert.

Um beispielsweise die Kinetik dieser Partikel zu beschreiben, könnten wir die Newtonschen Bewegungsgleichungen lösen. D.h. wir müssen für jedes der n Teilchen 6 gewöhnliche Differentialgleichungen, die noch untereinander gekoppelt sind, lösen, nämlich:

$$\dot{\vec{r}}_i(t) = \vec{v}_i(t) = \frac{\vec{p}_i(t)}{m_i} \tag{1.1}$$

Dies ist die Gleichung für die Bahnkurve des Teilchens i im Ortsraum. Da $\vec{r}$ ein Vektor ist, ist Gl. (1.1) ein System aus 3 gewöhnlichen Differentialgleichungen. Die Geschwindigkeit $\vec{v}_i$ des Teilchen i zum Zeitpunkt t unterliegt durch den Impuls $\vec{p}_i$ ebenfalls einem System von Differentialgleichungen:

$$\dot{\vec{p}}_i(t) = \sum_j \vec{F}_{ij}(t) \tag{1.2}$$

Gleichung (1.2) beschreibt die zeitliche Entwicklung des Impulses des Teilchens i. Gleichung (1.1) und (1.2) sind je $3 \times n$ gekoppelte gewöhnliche Differentialgleichungen. Wollen wir z.B. die Bewegung sämtlicher Moleküle in einem Liter Wasser durch eine Simulation beschreiben, so ist dies ob der großen Zahl an Gleichungen unmöglich und wir müssen übergehen zu einer Beschreibung mit Hilfe von Bilanzgleichungen und Feldern.

Die Newtonschen Bewegungsgleichungen (1.1) und (1.2) sind von ihrer Natur her *physikalische Grundprinzipien*. Sie gelten so für Atome oder Planeten. Die Natur der Kraft selbst, $\vec{F}_{ij}$ in den Gleichungen oben, basiert natürlich auf physikalisch beschreibbaren Effekten, ist aber nicht notwendigerweise ein Naturprinzip. Als einfaches Beispiel sei die Lennard-Jones-Wechselwirkung genannt, für welche die Wechselwirkungsenergie

$$V_{ij} = 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{1.3}$$

und damit die Kraft

$$\vec{F}_{ij} = -4\varepsilon \left[12 \left(\frac{\sigma^{12}}{r_{ij}^{13}} \right) - 6 \left(\frac{\sigma^{6}}{r_{ij}^{7}} \right) \right] \hat{r}_{ij} \tag{1.4}$$

gilt, wobei r_{ij} der Abstand zwischen den Partikeln (hier Atomen oder Molekülen) i und j ist. Der Term $\propto r^{-13}$ beschreibt die Abstoßung der Atome auf Grund des Paulischen Ausschließungsprinzips und der Term $\propto r^{-7}$ beschreibt die Anziehung der Atome auf Grund der Londonschen Dispersionswechselwirkung. Beide Wechselwirkungen fußen auf physikalischen Grundprinzipien,

aber die Formulierung Gl. (1.4) reduziert diese komplexen Phänomene auf ein einfaches konstituierendes Gesetz. Solche Gesetze werden oft *Konstitutivgesetze* genannt. Die numerische Lösung der Newtonschen Bewegungsgleichungen für Atome wird als *Molekulardynamiksimulation* bezeichnet.

Anmerkung: Der Begriff Konstitutivgesetz taucht öfter im Rahmen von Feldtheorien auf. Für das Lennard-Jones-Potential ist dieser Begriff eher ungewöhnlich, dieses Gesetzes ist aber trotzdem durchaus einer konstitutiven Natur.

Ein weiteres Beispiel von Modellen mit diskreten Elemente sind Netzwerkmodelle für elektrische Schaltkreise. Hierbei verknüpft ein Element eine Potentialdifferenz (Energiedifferenz) mit einem fließenden Strom, beispielweise beschreibt

$$i = u/R \tag{1.5}$$

den Strom i, der durch einen Widerstand R fließt, über den die Spannung u abfällt. Hinzu kommen die Kirchhoffschen Regeln für Strom und Spannung. Solche Modelle werden im englischen oft als "lumped-element models" bezeichnet. Gleichung (1.5) hat natürlich auch die Qualität eines *Konstitutivgesetzes*, da komplexe elektronische Prozesse hinter dem einzelnen Parameter R stehen. Die Kirchhoffschen Regeln hingegen haben die Qualität einer *Bilanzgleichung*. In Abb. 1.1 werden diese Modelle daher mit dem Begriff *globale Bilanz* bezeichnet. "Lumped-element models" führen auch zu Systemen gewöhnlicher Differentialgleichungen, die oft numerisch durch explizite Zeitpropagation gelöst werden. Bekannte Vertreter dieser Gattung Simulationssoftware sind z.B. *SPICE* oder *MATLAB Simulink*.

Diese globale Bilanzebene ist geprägt durch das Desinteresse an lokaler Auflösung. Wir interessieren uns nicht für Dichten, sondern nur für Gesamtmassen, nicht für Stromdichten sondern nur für Ströme. Am besten lässt sich dies an dem o.g. Widerstand verdeutlichen, dessen Kontakte auf unterschiedlichen Potentialen liegen. Dies hat im einfachsten Fall einen Stromfluss zur Folge. Wir fragen uns nicht, wie der Strom in dem Widerstand verteilt ist. Wir fragen nicht einmal, ob der Widerstand homogen oder inhomogen ist, wir nehmen ihn als Gesamtwert, als schwarzen Kasten, dem wir einen Parameterwert zuordnen. Diese Herangehensweise wurden schon ausführlich in der Elektrotechnik und der Systemtheorie besprochen.

1.3 Felder

Wenn wir nun aber feststellen, dass der schwarze Kasten mit einem Parameter nur ungenügend beschrieben ist, dann fangen wir an, ihn zu ersetzen durch einen komplexeren Ersatzschaltkreis mit Details, die den inneren Zustand des Bauteils auflösen. Dies kann man wiederum soweit treiben, dass zum Schluss ein Kontinuum entsteht - wir sind auf der *lokalen Bilanzebene* angekommen. Dazu wiederum brauchen wir Parameter, wie z.B. die Leitfähigkeit, die Viskosität oder die Diffusivität, die ihrerseits als Modelle nicht aus der kontinuierlichen Beschreibung der lokalen Bilanz ableitbar sind. Man braucht beispielsweise Experimente oder *ab-initio* Simulationen, um diese Größen zu ermitteln.

Lokale Bilanz bedeutet, dass wir an jedem Raumpunkt dem System eine Dichte, Konzentration, Temperatur oder ähnliche Größe zuordnen können. Damit sind aber die zeitlichen Veränderungen der Ortsfreiheitsgrade - also die Impulse, respektive die Geschwindigkeiten - in ihrer Verteilung durch eine *lokale, thermodynamische Gleichgewichtsbedingung* festgelegt. (Die Impulse genügen im thermodynamischen Gleichgewicht einer Maxwell-Boltzmann Verteilung.) Dieses lokale Gleichgewicht bedeutet nicht, dass wir keine Dynamik mehr haben. Aber wenn wir an einen Schwarm von Gas- oder Flüssigkeitsteilchen denken, dann folgen eben deren individuelle Geschwindkeiten einer Gleichgewichtsverteilungsfunktion, deren Mittel folgt aber der Bilanzgleichung. Die Dynamik läuft also gemittelt über eine riesige Zahl dieser Teilchen hinweg ab. Lokale Bilanz bedeutet auch nicht, dass an unterschiedlichen Orten nicht unterschiedliche Temperaturen oder Dichten vorliegen können. Die Unterschiede in diesen Parametern sind dann die treibenden Kräfte der Dynamik - Temparaturgradient, Dichtegradient, etc.

Solche Modelle fallen in den Bereich der Feldtheorien, und deren mathematische Beschreibung erfolgt über *partielle* Differentialgleichungen. (Dies steht im Gegensatz zu den gewöhnlichen Differentialgleichungen der diskreten Modelle.) Eine Feldtheorie, die auf Bilanzierung von Masse, Impuls oder Energie basiert, benötigt immer Konstitutivgesetze für die Beschreibung des Materialverhaltens. Diese Konstitutivgesetze enthalten *Transportparameter* wie die Viskosität oder Diffusionskonstante. Es gibt auch Feldtheorien, die den Charakter eines physikalischen Grundprinzips haben. Dies ist beispielsweise die o.g. Schrödingergleichung oder aber auch die Maxwell-Gleichungen der Elektrodynamik. In dieser Lehrveranstaltung wollen wir uns auf solche kontinuierliche Systeme konzentrieren, die als Feldtheorie mit Hilfe von partiellen Differentialgleichungen formuliert werden.

1.4 Welches Modell ist das richtige?

Wohlgemerkt, wir haben keine der Beschreibungsebenen auf verschiedenen Längenskalen mit irgendeiner Wertung versehen. Nur weil sie Quantenmechanik heisst und den Einen oder die Andere ob ihrer Komplexität in Ehrfurcht erstarren lässt, bietet sie nicht notwendigerweise die Lösung. Ganz im Gegenteil, es kann sogar hinderlich sein, zu viel Detail aufösen zu wollen und wir müssen uns ständig fragen, wieviel Detail in der Simulation notwendig ist. Mehr noch, fragen wir uns stets bevor wir eine Simulation angehen: "Ist eine Simulation dieser Komplexität wirklich notwendig, oder kann ich das Problem vereinfachen?" Die Simulation sollte als Hilfsmittel gesehen werden und nicht als Selbstzweck, frei nach dem amerikanischen Mathematiker Richard Wesley Hamming (*1915, †1998): *The purpose of computing is insight, not numbers.*

Kapitel 2

Gleichungstypen

Kontext: Die meisten Phänomene denen wir in den Ingenieurwissenschaften begegnen, werden sehr gut durch Differentialgleichungen beschrieben. Wir erinnern uns an die diskreten Netzwerkmodelle aus Elektrotechnik und Systemtheorie. Sie werden z.B. durch ein System linearer gewöhnlicher Differentialgleichungen (engl. "ordinary differential equations") mit der Zeit als unabängiger Veränderlicher beschrieben. Wir erinnern uns auch an den Diffusionsprozess, wie z.B. den Wärmetransport in einem Bauteil auf einem Kühlkörper, das einer Wärmequelle ausgesetzt ist. Dieses Phänomen wird am Besten mit einer partiellen Differentialgleichung (engl. "partial differential equation") beschrieben. In diesem Kapitel beschäftigen wir uns mit einer abstrakten Klassifikation von Differentialgleichungen. Der Diffusionsprozess wird in mehr Detail im nächsten Kapitel wiederholt.

2.1 Gewöhnliche Differentialgleichungen

Wir erinnern uns an die Klassifizierung (Eigenschaften) der *gewöhnlichen* Differentialgleichungen (GDGLs) und erkennen die verschiedenen Typen von Differentialgleichungen. Bei all diesen Differentialgleichungen sind wir immer an einer Lösung für einen bestimmten Anfangswert (oder Randwert) interessiert, also z.B. $x(t = 0) = x_0$ etc. Dieser Anfangswert ist immer Teil der Definition der Differentialgleichung.

2.1.1 Linear und nichtlinear

Eine lineare Differentialgleichung ist beispielsweise

$$m\ddot{x}(t) + c\dot{x}(t) + kx = f(t) \tag{2.1}$$

die den gedämpften und getriebenen harmonischen Oszillator beschreibt, während

$$\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} + \mu(x^2 - 1)\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0 \tag{2.2}$$

eine nichtlineare Bewegungsgleichung für x ist. Sie beschreibt den so genannten Van-der-Pol Oszillator. Die Nichtlinearität ist hier dadurch zu erkennen, dass x^2 die Ableitung $\mathrm{d}x/\,\mathrm{d}t$ multipliziert.

Anmerkung: Die Ableitung erster oder höherer Ordnung ist eine lineare Operation, da

$$\frac{\mathrm{d}^n}{\mathrm{d}x^n}\lambda f(x) = \lambda\frac{\mathrm{d}^n}{\mathrm{d}x^n}f(x) \tag{2.3}$$

für eine Konstante λ und

$$\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left[f(x) + g(x)\right] = \frac{\mathrm{d}^n}{\mathrm{d}x^n}f(x) + \frac{\mathrm{d}^n}{\mathrm{d}x^n}g(x). \tag{2.4}$$

Zeitliche Ableitungen werden mit einem Punkt angezeigt,

$$\dot{x}(t) = \frac{\mathrm{d}}{\mathrm{d}t}x(t). \tag{2.5}$$

Für Funktionen einer Variable wird die Ableitung oft mit einem Strich angezeigt,

$$f'(x) = \frac{\mathrm{d}}{\mathrm{d}x}f(x). \tag{2.6}$$

Für Funktionen mehrere Variablen ist das nicht mehr möglich. Hier werden wir daher immer explizit den Differentialoperator verwenden.

2.1.2 Ordnung

Die Ordnung einer Differentialgleichung ist gegeben durch die höchste Ableitung die in der Gleichung auftaucht. So sind Gl. (2.1) und Gl. (2.2) Beispiele für Differentialgleichungen zweiter Ordnung.

2.1.3 Systeme

Ein System von Differentialgleichungen 1. Ordnung bilden z.B. die Gleichungen

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x(m - ny), \tag{2.7}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -y(\gamma - \delta x), \tag{2.8}$$

die bekannten Räuber-Beute-Gleichungen oder auch Lotka-Volterra-Gleichungen. Gleichungen (2.7) und (2.8) sind weiterhin nichtlinear.

Differentialgleichungen höherer Ordnung können in ein System von Gleichungen 1. Ordnung umgeschrieben werden. Im Beispiel des gedämpften harmonische Oszillators,

$$\ddot{x}(t) + c\dot{x}(t) + kx = f(t), \tag{2.9}$$

ersetzen wir $\dot{x} = y$ und erhalten dadurch zwei Gleichungen erster Ordnung anstatt der ursprünglichen Gleichung zweiter Ordnung, nämlich

$$\dot{x} = y \tag{2.10}$$

$$m\dot{y} = -cy - kx + f(t) \tag{2.11}$$

2.2 Partielle Differentialgleichungen

Partielle Differentialgleichungen (PDGLs) sind Differentialgleichungen mit mehr als einer unabhängigen Variablen. Als Beispiel stellen wir uns ein zeitabhängiges Wärmetransportproblem in einer Dimension vor. Dieses wird mit einer Diffusionsgleichung für die lokale Temperatur des Systems dargestellt. Die Temperatur wird daher als Funktion zweier unabhängiger Variablen, der Zeit t und der räumlichen Position x, dargestellt: $T(x, t)$. Die Zeitentwicklung der Temperatur ist gegeben durch

$$\frac{\partial T(x,t)}{\partial t} = \kappa \frac{\partial^2 T(x,t)}{\partial x^2}, \tag{2.12}$$

wobei κ den Wärmeleitungskoeffizienten bezeichnet. Diese Gleichung wurde von Joseph Fourier (*1768, †1830) entwickelt, dem wir im Laufe dieser Veranstaltung wieder begegnen werden.

Anmerkung: In Gl. (2.12) bezeichnet $\partial/\partial t$ die *partielle Ableitung*. Dies ist die Ableitung nach einem der Argumente (hier t), also die Variation der Funktion, wenn alle anderen Argumente konstant gehalten werden. Bei GDGLs tauchen im Gegensatz zu PDGLs nur Ableitung nach einer Variable (üblicherweise der Zeit t) auf, die dann mit dem Differentialoperator $\mathrm{d}/\,\mathrm{d}t$ bezeichnet werden.

2.2.1 Erste Ordnung

Quasilineare PDGLs erster Ordnung, also Gleichungen der Form

$$P(x,t;u)\frac{\partial u(x,t)}{\partial x} + Q(x,t;u)\frac{\partial u(x,t)}{\partial t} = R(x,t;u), \tag{2.13}$$

für eine (unbekannte) Funktion $u(x,t)$ und der Anfangsbedingung $u(x,t=0) = u_0(x)$ können systematisch auf ein System gekoppelter GDGLs erster Ordnung zurückgeführt werden. Diese wichtige Eigenschaft wollen wir untersuchen.

Anmerkung: In Gl. (2.13) wurde zur Illustration eine Darstellung mit zwei Variablen x und t gewählt. Allgemein können wir schreiben:

$$\sum_i P_i(\{x_i\};u)\frac{\partial u(\{x_i\})}{\partial x_i} = R(\{x_i\};u) \tag{2.14}$$

Hier wurde als Notation $u(\{x_i\}) = u(x_0, x_1, x_2, \ldots)$ genutzt, also die geschweiften Klammern bezeichnen alle Freiheitsgrade x_i.

Gleichung (2.13) können wir auf ein System von GDGLs transformieren. Dies wird die Methode der Charakteristiken genannt. Wir können dann die Formalismen (analytisch oder numerisch) zur Lösung von Systemen von GDGLs anwenden, die wir in der Vorlesung "Differentialgleichungen" kennengelernt haben.

Wir gehen folgendermaßen vor:

1. Zunächst parametrisieren wir die unabhängigen Veränderlichen in Gl. (2.13) mit einem Parameter s gemäß $x(s)$ und $t(s)$.

2. Wir bilden dann die *totale Ableitung* von $u(x(s),t(s))$ nach s

$$\frac{\mathrm{d}u(x(s),t(s))}{\mathrm{d}s} = \frac{\partial u(x(s),t(s))}{\partial x}\frac{\mathrm{d}x(s)}{\mathrm{d}s} + \frac{\partial u(x(s),t(s))}{\partial t}\frac{\mathrm{d}t(s)}{\mathrm{d}s}. \tag{2.15}$$

3. Durch den Vergleich der Koeffizienten der totalen Ableitung (2.15) mit der PDGL (2.13) sieht man, dass diese DGL genau denn gelöst wird, wenn

$$\frac{dx(s)}{ds} = P(x,t,u), \tag{2.16}$$

$$\frac{dt(s)}{ds} = Q(x,t,u) \quad \text{und} \tag{2.17}$$

$$\frac{du(s)}{ds} = R(u(s)). \tag{2.18}$$

erfüllt ist. Dies beschreibt die Lösung entlang bestimmter Kurven in der (x,t)-Ebene.

Wir haben damit die PDGL in einen Satz gekoppelter GDGLs erster Ordnung, Gl. (2.21)-(2.23) umgewandelt.

Beispiel: Die Transportgleichung

$$\frac{\partial u(x,t)}{\partial t} + c\frac{\partial u(x,t)}{\partial x} = 0 \tag{2.19}$$

mit der Anfangsbedingung $u(x, t = 0) = u_0(x)$ soll gelöst werden. Wir gehen nach obigem Rezept vor:

1. Wir parameterisieren die Variablen x und t mit Hilfe einer neuen Variable s, also $x(s)$ und $t(s)$. Wir suchen nun nach einem Ausdruck, mit dem wir $x(s)$ und $t(s)$ bestimmen können.

2. Wir stellen nun die Frage, wie sich die Funktion $u(x(s), t(s))$ verhält. Diese Funktion beschreibt die Änderung eines Anfangswertes $u(x(0), t(0))$ mit der Variable s. Die totale Ableitung wird zu

$$\frac{\mathrm{d}u(x(s),t(s))}{\mathrm{d}s} = \frac{\partial u}{\partial t}\frac{\mathrm{d}t(s)}{\mathrm{d}s} + \frac{\partial u}{\partial x}\frac{\mathrm{d}x(s)}{\mathrm{d}s}. \tag{2.20}$$

3. Die totale Ableitung ist genau dann identisch zu der partiellen Differentialgleichung, die wir lösen wollen, wenn

$$\frac{\mathrm{d}x(s)}{\mathrm{d}s} = c \quad \text{und} \tag{2.21}$$

$$\frac{\mathrm{d}t(s)}{\mathrm{d}s} = 1. \tag{2.22}$$

In diesem Fall gilt

$$\frac{\mathrm{d}u(s)}{\mathrm{d}s} = 0. \tag{2.23}$$

4. Die allgemeinen Lösungen für die drei gewöhnlichen Differentialgleichungen (2.21)-(2.23) sind gegeben durch

$$x(s) = cs + \text{const.}, \tag{2.24}$$
$$t(s) = s + \text{const.} \quad \text{und} \tag{2.25}$$
$$u(s) = \text{const.} \tag{2.26}$$

5. Mit den Anfangsbedingungen $t(0) = 0$, $x(0) = \xi$ und $u(x, t = 0) = f(\xi)$ erhält man $t = s$, $x = ct + \xi$ und $u = f(\xi) = f(x - ct)$,

Die Anfangsbedingung $f(\xi)$ wird mit der Geschwindigkeit c in die positive x-Richtung transportiert. Die Lösung für u bleibt konstant, da die Ableitung von u Null ist, also behält u den durch die Anfangsbedingung gegebenen Wert. Das Feld $u(x, 0)$ wird also mit einer konstanten Geschwindigkeit c verschoben: $u(x, t) = u(x - ct, 0)$.

2.2.2 Zweite Ordnung

Beispiele von PDGLs zweiter Ordnung sind die...

- ...Wellengleichung:
$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0 \tag{2.27}$$
- ...Diffusionsgleichung (mit der wir uns hier näher beschäfigen werden):
$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0 \tag{2.28}$$
- ...Laplacegleichung (die wir auch näher kennen lernen werden):
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.29}$$

Die zweite Ordnung bezieht sich hier auf die zweite Ableitung. Diese Beispiel sind für zwei Variablen formuliert, aber diese Differentialgleichungen können auch für mehr Freiheitsgrade aufgeschrieben werden.

Für zwei Variablen lautet die allgemeine Form linearer PDGLs zweiter Ordnung,

$$a(x, y)\frac{\partial^2 u}{\partial x^2} + b(x, y)\frac{\partial^2 u}{\partial x \partial y} + c(x, y)\frac{\partial^2 u}{\partial y^2} = F\left(x, y; u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right), \tag{2.30}$$

wobei F selbst natürlich auch linear in den Argumenten sein muss, wenn die gesamte Gleichung linear sein soll. Wir nehmen nun eine Klassifizierung von PDGLs 2. Ordnung vor, stellen aber vorweg, dass diese Klassifizierung nicht erschöpfend ist und dass sie nur punktweise gilt. Letzteres heißt, dass die PDGL an unterschiedlichen Raumpunkten in eine andere Klassifizierung fallen kann.

Wir nehmen zunächst an, dass $F = 0$ und a, b, c konstant seien. Dann erhalten wir:

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} = 0. \tag{2.31}$$

Wir schreiben diese Gleichung um als die quadratische Form

$$\begin{pmatrix} \partial/\partial x \\ \partial/\partial y \end{pmatrix} \cdot \begin{pmatrix} a & b/2 \\ b/2 & c \end{pmatrix} \cdot \begin{pmatrix} \partial/\partial x \\ \partial/\partial y \end{pmatrix} u = \nabla \cdot \underline{C} \cdot \nabla u = 0 \tag{2.32}$$

Die Koeffizientenmatrix $\underline{C}$ können wir nun diagonalisieren. Dies für zu

$$\underline{C} = \underline{U} \cdot \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \cdot \underline{U}^T, \tag{2.33}$$

wobei $\underline{U}$ auf Grund der Symmetrie von $\underline{C}$ unitär ist, $\underline{U}^T \cdot \underline{U} = \underline{1}$. Die geometrische Interpretation der Operation $\underline{U}$ ist eine Rotation. Wir führen nun transformierte Koordinaten x' und y' ein, so dass

$$\nabla = \underline{U} \cdot \nabla' \tag{2.34}$$

mit $\nabla' = (\partial/\partial x', \partial/\partial y')$. Mit anderen Worten, die Transformationsmatrix ist gegeben als

$$\underline{U} = \begin{pmatrix} \partial x'/\partial x & \partial y'/\partial x \\ \partial x'/\partial y & \partial y'/\partial y \end{pmatrix}. \tag{2.35}$$

Gleichung (2.31) wird zu

$$\lambda_1 \frac{\partial^2 u}{\partial x'^2} + \lambda_2 \frac{\partial^2 u}{\partial y'^2} = 0. \tag{2.36}$$

Wir haben die Koeffizienten der Differentialgleichung diagonalisiert. Für eine beliebige zweifach differenzierbare Funktion $f(z)$, ist

$$u(x', y') = f\left(\sqrt{\lambda_2}x' + i\sqrt{\lambda_1}y'\right) \tag{2.37}$$

eine Lösung von Gl. (2.36).

Wir unterscheiden nun drei Fälle:

- Der Fall $\det \underline{C} = \lambda_1 \lambda_2 = ac - b^4/4 = 0$ mit $b \neq 0$ und $a \neq 0$ führt zu einer parabolischen PDGL. Diese PDGL heißt parabolisch, weil die quadratische Form Gl. (2.32) bzw. (2.33) eine Parabel beschreibt. (Dies ist natürlich eine Analogie. Man muss die Differentialoperatoren durch Koordinaten ersetzen damit diese funktioniert.) Ohne Beschränkung der Allgemeinheit sei $\lambda_2 = 0$. Dann bekommen wir

$$\frac{\partial^2 u}{\partial x'^2} = 0. \tag{2.38}$$

  Dies ist die kanonische Form einer parabolischen PDGL.

- Der Fall $\det \underline{C} = \lambda_1 \lambda_2 = ac - b^2/4 > 0$ führt zu einer elliptischen PDGL. Diese PDGL heißt elliptisch, weil die quadratische Form Gl. (2.32) bzw. (2.33) für eine konstante rechte Seite eine Ellipse beschreibt. (Für $\lambda_1 = \lambda_2$ ist es ein Kreis.) Wir formen nun die Gleichung für den elliptischen Fall auf eine standardisierte Form um und führen die skalierten Koordinaten $x' = \sqrt{\lambda_1} x''$ und $y' = \sqrt{\lambda_2} y''$ ein. Dann wird aus Gl. (2.36) die kanonische elliptische PDGL

$$\frac{\partial^2 u}{\partial x''^2} + \frac{\partial^2 u}{\partial y''^2} = 0. \tag{2.39}$$

  Die kanonische elliptische PDGL ist daher die Laplace-Gleichung, Gl. (2.39) (hier im Zweidimensionalen). Lösungen der Laplace-Gleichung heißen *harmonische Funktionen.*

- Der Fall $\det \underline{C} = \lambda_1 \lambda_2 = ac - b^2/4 < 0$ ergibt die so genannte hyperbolische PDGL. Diese PDGL heißt hyperbolisch, weil die quadratische Form Gl. (2.32) bzw. (2.33) für eine konstante rechte Seite eine Hyperbel beschreibt. Ohne Beschränkung der Allgemeinheit fordern wir nun $\lambda_1 > 0$ und $\lambda_2 < 0$. Dann können wir wieder skalierte Koordinaten $x' = \sqrt{\lambda_1} x''$ und $y' = \sqrt{-\lambda_2} y''$ einführen, so dass

$$\frac{\partial^2 u}{\partial x''^2} - \frac{\partial^2 u}{\partial y''^2} = \begin{pmatrix} \partial u/\partial x'' \\ \partial u/\partial y'' \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \partial u/\partial x'' \\ \partial u/\partial y'' \end{pmatrix} = 0. \tag{2.40}$$

  Wir können nun durch eine weitere Koordinatentransformation, nämlich eine Rotation um 45°, die Koeffizientenmatrix in Gl. (2.40) auf eine Form bringen, in der die Diagonalelemente 0 und die Nebendiagonalelemente 1 sind. Dies ergibt die Differentialgleichung

$$\frac{\partial^2 u}{\partial x''' \partial y'''} = 0, \tag{2.41}$$

wobei x''' und y''' die entsprechend rotierten Koordinaten sind. Diese Gleichung ist die kanonische Form einer hyperbolischen PDGL und äquivalent zu Gl. (2.31) in den neuen Variablen x''' und y'''.

Für höherdimensionale Probleme müssen wir uns die Eigenwerte der Koeffizientenmatrix $\underline{C}$ anschauen. Die PDGL heißt *parabolisch*, wenn es einen Eigenwert gibt der verschwindet, aber alle anderen Eigenwerte entweder größer oder kleiner als Null sind. Die PDGL heißt *elliptisch*, wenn alle Eigenwerte entweder größer Null oder kleiner Null sind. Die PDGL heißt *hyperbolisch*, wenn es genau einen negativen Eigenwert gibt und alle anderen positiv sind oder es genau einen positiven Eigenwert gibt und alle anderen negativ sind. Es ist klar, dass für PDGLs mit mehr als zwei Variablen, diese drei Klassen von PDGLs nicht erschöpfend sind und es Koeffizientenmatrizen gibt, die aus diesem Klassifizierungschema fallen. Für Probleme mit genau zwei Variablen führt diese Klassifzierung zu den Bedingungen für die Determinanten der Koeffizientenmatrix die oben genannt wurden.

Diese drei Typen linearer PDEs 2. Ordnung lassen sich für manche Problemstellungen auch analytisch lösen. Wir geben im Folgenden ein Beispiel hierzu.

Beispiel: Wir lösen die eindimensionale Wellengleichung.

$$\frac{\partial^2 u}{\partial x^2} - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = 0 \tag{2.42}$$

durch Separation der Variablen. Dafür machen wir den Ansatz $u(x,t) = X(x)T(t)$, was zu

$$\frac{1}{X}\frac{\partial^2 X}{\partial x^2} = \frac{1}{c^2}\frac{1}{T}\frac{\partial^2 T}{\partial t^2} \tag{2.43}$$

führt. In Gl. (2.43) hängt die linke Seite nur von der Variablen x ab, während die rechte Seite nur von t abhängt. Für beliebige x und t kann diese Gleichung nur erfüllt werden, wenn beide Seiten gleich einer Konstanten sind und wir erhalten somit

$$\frac{1}{X}\frac{\partial^2 X}{\partial x^2} = -k^2 = \frac{1}{c^2}\frac{1}{T}\frac{\partial^2 T}{\partial t^2}\,. \tag{2.44}$$

Dies ergibt die folgenden zwei Gleichungen

$$\frac{\partial^2 X}{\partial x^2} + k^2 X = 0$$

mit der Lösung $X(x) = e^{\pm ikx}$ und

$$\frac{\partial^2 T}{\partial t^2} + \omega^2 T = 0$$

mit der Lösung $T(t) = e^{\pm i\omega t}$, wobei wir $\omega^2 = c^2 k^2$ gesetzt haben. Dieses Beispiel braucht zur Ergänzung Anfangsbedingungen, damit wir eine Lösung finden können.

Kapitel 3

Transporttheorie

Kontext: Dieses Lernmodul führt die Grundlagen des Modellproblems ein, welches wir im Laufe der Lehrveranstaltung mit numerischen Methoden lösen werden. Hier geht es zunächst um die Grundlagen der Transporttheorie.

3.1 Diffusion und Drift

Diffusiver Transport ist einfach zugänglich über das Bild des "Random Walk", einer zufälligen stochastischen Bewegung von Teilchen. Solche zufälligen Bewegungsprozesse wurden zuerst von dem Botaniker *Robert Brown* (1773-1858) beschrieben und tragen den Namen *Brownsche Bewegung* oder *Brownsche Molekularbewegung*. Robert Brown wusste damals allerdings nicht von Molekülen und dachte zu seinen Lebzeiten, dass diese Bewegung auf aktive Prozesse (der "Lebenskraft" der Pollen) zurückzuführen sei. Heute wissen wir, dass diese Bewegung durch thermische Fluktuationen verursacht wird, also Moleküle die zufällig auf die Pollen treffen und diese in eine Richtung stoßen. Diese Erklärung benötigt die Existenz von Atomen und wurde erst 1905 von Albert Einstein (Einstein, 1905) hoffähig gemacht.

Brownsche Molekularbewegung führt zu diffusivem Transport. Abbildung 4.1 zeigt ein einfaches qualitatives Gedankenexperiment. Die Konfiguration in Abb. 4.1a zeigt eine Lokalisierung der "Pollen" in der linken Hälfte der gezeigten Domäne. Durch deren zufällige Bewegung (als Beispiel gezeigt an der roten Linie in Abb. 4.1a) werden einige der Pollen die gestrichelte Grenzlinie in die rechte Hälfte überschreiten und auch wieder zurück kommen. Nach einer gewissen Zeit lässt sich der Anfangszustand nicht mehr identifizieren und die Pollen verteilen sich in der gesamten Domäne (Abb. 4.1b). Die Konzentration ist nun konstant. Die Pollen bewegen sich zwar weiter, aber in

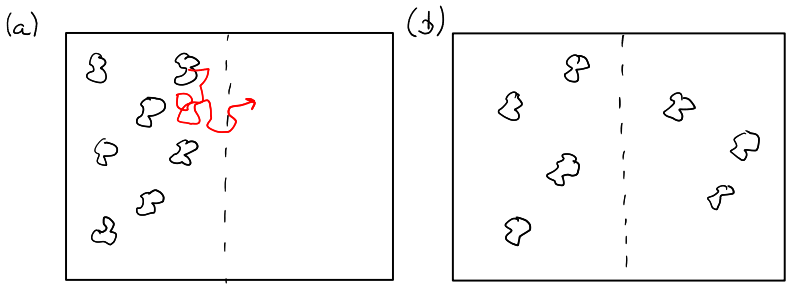

Abbildung 3.1: Illustration eines Diffusionsprozesses. Die “Pollen” in (a) bewegen sich zufällig in der gezeigten Domäne. Nach einer gewissen Zeit (b) ist der anfängliche Konzentrationsunterschied zwischen dem linken und rechten Teil der Domäne ausgeglichen.

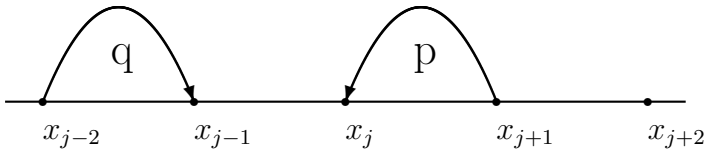

Abbildung 3.2: Zufallsbewegung in einer Dimension ist gegeben durch Übergangswahrscheinlichkeiten p (für eine Bewegung nach links) und q für eine Bewegung nach rechts.

Mittel bewegt sich die gleiche Zahl Pollen nach links wie nach rechts. Im Fall der in Abb. 4.1a gezeigt ist, ist diese Symmetrie gebrochen.

Dieses Gedankenexperiment kann einfach mathematisch formalisiert werden. Wir betrachten ein Teilchen, das eine Zufallsbewegung in einer Dimension vollführt. Wir beginnen mit einem Teilchen, das zufällig auf einer Geraden hin- und herspringt. Die Gerade liege entlang der x-Richtung. Das Teilchen kann nur zu vorher festgelegten Positionen auf der x-Achse springen, die wir mit x_j bezeichnen und die äquidistant verteilt seien, $x_j - x_{j-1} = \Delta x$ für $j \in \mathbb{Z}$ (siehe Abb. 3.2).

Ein Teilchen springe mit einer Wahrscheinlichkeit p nach links und mit einer Wahrscheinlichkeit q nach rechts. Darüberhinaus haben wir die Wahrscheinlichkeit überhaupt ein Teilchen zur Zeit t bei Position x zu finden. Diese ist auf dem 1D Gitter durch die Funktion $P(x,t)$ gegeben.

3.1.1 Diffusion

Wir betrachten zunächst den Fall $p = q = 1/2$, also dass die Wahrscheinlichkeiten für die Sprünge nach links und rechts identisch sind. Wir nehmen an,

das Teilchen springe von einem Platz zum benachbarten in einem diskreten, endlichen und konstanten Zeitschritt τ. Dann ist die Wahrscheinlichkeit ein Teilchen zur Zeit $t+\tau$ am Ort x zu finden, wenn zur Zeit t mit einer Wahrscheinlichkeit $P(x-\Delta x,t)$ ein Teilchen bei der Position $x-\Delta x$ zu finden war und mit der Wahrscheinlichkeit $P(x+\Delta x,t)$ eines bei $x+\Delta x$, gegeben durch

$$P(x,t+\tau)=\frac{1}{2}P(x+\Delta x,t)+\frac{1}{2}P(x-\Delta x,t). \tag{3.1}$$

Indem wir $P(x,t)$ auf beiden Seiten abziehen und durch τ teilen, erhalten wir die folgende äquivalente Form:

$$\frac{P(x,t+\tau)-P(x,t)}{\tau}=\frac{\Delta x^2}{2\tau}\frac{P(x+\Delta x,t)-2P(x,t)+P(x-\Delta x,t)}{\Delta x^2} \tag{3.2}$$

Wir können nun den Grenzübergang zum "Kontinuum" machen. Für $\tau\to 0$ und gleichzeitig $h\to 0$ unter der Bedingung, dass

$$\lim_{\Delta x\to 0,\tau\to 0}\frac{\Delta x^2}{2\tau}=D \tag{3.3}$$

erhält man

$$\frac{\partial P(x,t)}{\partial t}=D\frac{\partial^2 P(x,t)}{\partial x^2}. \tag{3.4}$$

Dies die wohlbekannte Diffusionsgleichung. In mehreren Dimensionen wird aus der zweiten Ableitung der Laplace-Operator ∇^2,

$$\frac{\partial P(x,t)}{\partial t}=D\nabla^2 P(x,t). \tag{3.5}$$

Diese Gleichung ist nur dann korrekt, wenn die Diffusionskonstante auch wirklich konstant ist und nicht räumlich variiert.

3.1.2 Drift

Wie sieht es aus, wenn die Wahrscheinlichkeiten für die Sprünge nach rechts oder links nicht gleich sind, $p\neq q$ (aber natürlich $p+q=1$)? Wir gehen nach wie vor von diskreten, gleichförmigen Zeitschritten und äquidistanten Stützstellen aus.

In diesem Fall haben wir

$$P(x,t+\tau)=pP(x+\Delta x,t)+qP(x-\Delta x,t) \tag{3.6}$$

und somit folgt

$$\frac{P(x,t+\tau)-P(x,t)}{\tau}=\frac{\Delta x^2}{\tau}\frac{pP(x+\Delta x,t)-P(x,t)+qP(x-\Delta x,t)}{\Delta x^2}. \tag{3.7}$$

Wir schreiben

$$p = \frac{1}{2} - \varepsilon \quad \text{und} \quad q = \frac{1}{2} + \varepsilon \quad \text{mit} \quad 0 \leq |\varepsilon| \leq \frac{1}{2} \quad \text{oder} \quad 2\varepsilon = q - p, \tag{3.8}$$

wobei ε nun angibt, um wieviel wahrscheinlicher ein Sprung nach *rechts* als nach links ist. Ein positives ε heißt also, dass die Partikel sich im Mittel nach rechts bewegen werden – dies ist die Driftbewegung. Wir können nun Gl. (3.7) mit Hilfe von ε als

$$\begin{aligned} \frac{P(x,t+\tau) - P(x,t)}{\tau} =& \frac{\Delta x^2}{2\tau} \frac{P(x+\Delta x,t) - 2P(x,t) + P(x-\Delta x,t)}{\Delta x^2} \\ & - \frac{2\varepsilon \Delta x}{\tau} \frac{P(x+\Delta x,t) - P(x-\Delta x,t)}{2\Delta x} \end{aligned} \tag{3.9}$$

ausdrücken. In den Grenzfällen $\tau \to 0$ und $\Delta x \to 0$ fordern wir

$$\lim_{\Delta x \to 0, \tau \to 0} \frac{\Delta x^2}{2\tau} = D \quad \text{und} \quad \lim_{\Delta x \to 0, \tau \to 0} \frac{2\varepsilon \Delta x}{\tau} = v \tag{3.10}$$

und erhalten somit die Drift-Diffusions-Gleichung

$$\frac{\partial P(x,t)}{\partial t} = \left(D \frac{\partial^2}{\partial x^2} - v \frac{\partial}{\partial x} \right) P(x,t). \tag{3.11}$$

Hier beschreibt der erste Summand auf der rechten Seite wieder den Diffusionsprozess. Der zweite Summand ist ein Driftprozess und v eine konstante *Drift*geschwindigkeit. (Aus Gl. (3.10) und (3.11) wird ersichtlich, dass die Einheit von v genau einer Geschwindigkeit entspricht.) Es ist die Geschwindigkeit mit der sich das Teilchen (im Mittel) entlang der x-Achse bewegen. Die Lösung dieser Gleichung wurde bereits im vorhergehenden Kapitel besprochen. Aus der Lösung des vorhergehenden Kapitels wird auch klar, dass die Driftbewegung für positive v in Richtung der positiven x-Achse ist. Dies ist konsistent mit der obigen Definition von ε.

Anmerkung: Die Bewegung unseres Teilchen wurde mit Hilfe einer Aufenthalts*wahrscheinlichkeit* P modelliert. Im thermodynamischen Limes, also für ganz viele Teilchen (üblicherweise in Größenordnung der Avogadroschen Zahl N_A), wird aus dieser Wahrscheinlich die Dichte ρ oder die Konzentration c. Man kann also einfach in den oben genannten Gleichungen die Wahrscheinlichkeit P durch eine Teilchenkonzentration c ersetzen. Der Grund hierfür ist, dass wir die Teilchenkonzentration als *Ensemble*-Mittel schreiben können,

$$c(x,t) = \langle 1 \rangle (x,t), \tag{3.12}$$

wobei der Mittelwert

$$\langle f(x)\rangle(x,t) = f(x)P(x,t). \tag{3.13}$$

3.2 Kontinuität

Die Gleichungen (3.5) und (3.11) vermischen zwei Konzepte, die wir hier jetzt getrennt behandeln wollen: Die Erhaltung der Anzahl der Teilchen (Kontinuität) und der Prozess, welcher zu einem Teilchenstrom führt (Diffusion oder Drift). Die Teilchenzahl ist einfach deshalb erhalten, weil wir keine Atome aus dem Nichts erzeugen oder in das Nichts vernichten können. Wir wissen also, wenn wir eine gewissen Anzahl Teilchen N_{tot} in unserem Gesamtsystem haben, dass diese Anzahl

$$N_{\text{tot}} = \int \mathrm{d}^3 r\, c(\vec{r}) \tag{3.14}$$

sich nicht über die Zeit ändern kann: $\mathrm{d}N_{\text{tot}}/\,\mathrm{d}t = 0$.

Für einen kleinen Ausschnitt mit Volumen V aus diesem Gesamtvolumen kann sich die Teilchenzahl ändern, weil diese über die Wände des Probevolumens fließen können (siehe Abb. 3.3). Die Änderung dieser Teilchenzahl ist zum einen gegeben durch

$$\dot{N} = \frac{\partial}{\partial t}\int_V \mathrm{d}^3 r\, c(\vec{r},t) = \int_V \mathrm{d}^3 r\, \frac{\partial c}{\partial t}. \tag{3.15}$$

Die Änderung $\dot{N}$ muss aber auch durch die Anzahl der Partikel, die über die Seitenwände abfließen, gegeben sein. Für einen Würfel (Abb. 3.3) mit sechs Wänden gilt

$$\begin{aligned}\dot{N} = &- j_{\text{rechts}}A_{\text{rechts}} - j_{\text{links}}A_{\text{links}} \\ &- j_{\text{oben}}A_{\text{oben}} - j_{\text{unten}}A_{\text{unten}} \\ &- j_{\text{vorne}}A_{\text{vorne}} - j_{\text{hinten}}A_{\text{hinten}}\end{aligned} \tag{3.16}$$

wenn die Wände klein genug sind, so dass j nahezu konstant über A ist. (Die Stromdichte j hat die Einheit Anzahl Partikel/Zeit/Fläche.)

Hier bezeichnet der skalare Strom j den Strom, der aus der Fläche heraus fließt. Für eine allgemeine vektorielle Stromdichte $\vec{j}$, welche die Stärke und Richtung des Teilchenstroms angibt, ist $j_i = \vec{j}_i \cdot \hat{n}_i$ wobei $\hat{n}_i$ der Normalenvektor

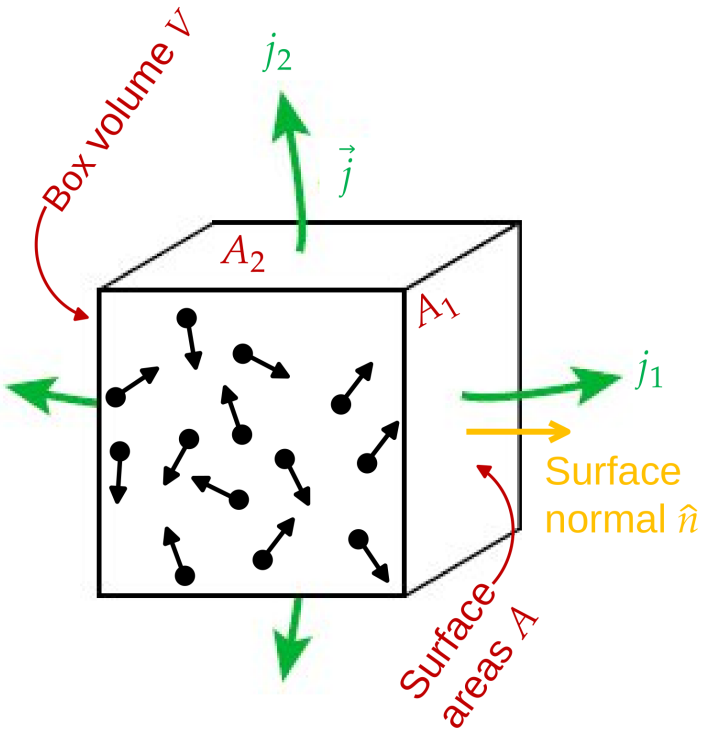

Abbildung 3.3: Teilchen können das Volumen V nur durch die Seitenwände verlassen. Die Änderung der Teilchenzahl N über ein Zeitintervall τ ist daher durch die Anzahl der Teilchen gegeben, die durch die Wände fließen. Hierzu brauchen wir die Teilchenströme j. Die Anzahl der Teilchen, welche durch eine Oberfläche fließen ist dann gegeben durch $j\,A\tau$, wobei A die Fläche der Seitenwand ist.

auf die Wand i ist. Der Strom durch die Wand ist also nur die Komponente von $\vec{j}$, die parallel zur Oberflächennormale steht. Mit diesem Argument können wir die Änderung der Teilchenzahl allgemein als

$$\dot{N} = -\int_{\partial V} \mathrm{d}^2 r\, \vec{j}(\vec{r}) \cdot \hat{n}(\vec{r}) \tag{3.17}$$

ausdrücken, wobei ∂V die Oberfläche des Volumens V bezeichnet. In dieser Gleichung ist explizit angezeigt, dass selbstverständlich sowohl der Fluss $\vec{j}$ als auch die Oberflächennormale $\hat{n}$ von der Position $\vec{r}$ auf der Oberfläche abhängen.

Alternativ können wir auch die Änderung der Teilchenzahl Gl. (3.16) folgendermaßen gruppieren:

$$\begin{aligned} \dot{N} = &-(j_{\text{rechts}} + j_{\text{links}}) A_{\text{rechts/links}} \\ &-(j_{\text{oben}} + j_{\text{unten}}) A_{\text{oben/unten}} \\ &-(j_{\text{vorne}} + j_{\text{hinten}}) A_{\text{vorne/hinten}} \end{aligned} \tag{3.18}$$

Hierbei haben wir die Tatsache genutzt, dass $A_{\text{rechts}} = A_{\text{links}} \equiv A_{\text{rechts/links}}$. Nun ist aber

$$\begin{aligned} j_{\text{rechts}} &= \hat{x} \cdot \vec{j}(x + \Delta x/2, y, z) = j_x(x + \Delta x/2, y, z) \quad \text{und} \\ j_{\text{links}} &= -\hat{x} \cdot \vec{j}(x - \Delta x/2, y, z) = -j_x(x - \Delta x/2, y, z) \end{aligned} \tag{3.19}$$

da $\hat{n} = \hat{x}$ für die rechte Wand aber $\hat{n} = -\hat{x}$ für die linke Wand. Hierbei ist $\hat{x}$ der Normalenvektor entlang der x-Achse des Koordinatensystems. Es dreht sich also zwischen der rechten und linken Fläche das Vorzeichen der Oberflächennormale um. Das gleiche gilt für die Wände oben/unten und vorne/hinten. Wir können diese Gleichung weiterhin umschreiben als

$$\begin{aligned} \dot{N} = &-\frac{j_x(x + \Delta x/2, y, z) - j_x(x - \Delta x/2, y, z)}{\Delta x} V \\ &-\frac{j_y(x, y + \Delta y/2, z) - j_y(x, y - \Delta y/2, z)}{\Delta y} V \\ &-\frac{j_z(x, y, z + \Delta z/2) - j_z(x, y, z - \Delta z/2)}{\Delta z} V, \end{aligned} \tag{3.20}$$

da $V = A_{\text{rechts/links}} \Delta x = A_{\text{oben/unten}} \Delta y = A_{\text{vorne/hinten}} \Delta z$. Die Faktoren vor dem Volumen V in Gl. (3.20) sind nun aber genau der Differenzenquotienten der Flüsse j_i, jeweils in die x-, y- und z-Richtung. Für kleine Volumina (und kleine Δx, etc.) wird dies zu

$$\dot{N} = -\int_V \mathrm{d}^3 r\, \nabla \cdot \vec{j}(\vec{r}). \tag{3.21}$$

Wir habe hier gerade heuristisch den Gaussschen Satz (engl. “Divergence Theorem” - siehe auch Gl. (**??**)) hergeleitet, um Gl. (3.17) als Volumenintegral auszudrücken.

Anmerkung: Der Gausssche Satz ist ein wichtiges Ergebnis der Vektoranalysis. Er wandelt ein Integral über ein Volumen V in ein Integral über die Oberfläche ∂V dieses Volumens um. Für ein Vektorfeld $\vec{f}(\vec{r})$ gilt:

$$\int_V \mathrm{d}^3 r \, \nabla \cdot \vec{f}(\vec{r}) = \int_{\partial V} \mathrm{d}^2 r \, \vec{f}(\vec{r}) \cdot \hat{n}(\vec{r}) \tag{3.22}$$

Hier ist $\hat{n}(\vec{r})$ der Normalenvektor, welcher auf dem Rand ∂V des Volumens V nach außen zeigt.

Gleichung (3.15) und (3.21) zusammen ergeben

$$\int_V \mathrm{d}^3 r \, \left\{ \frac{\partial c}{\partial t} + \nabla \cdot \vec{j} \right\} = 0. \tag{3.23}$$

Da dies für jedes beliebige Volumen V gilt, muss auch

$$\frac{\partial c}{\partial t} + \nabla \cdot \vec{j} = 0 \tag{3.24}$$

erfüllt sein. Diese Gleichung trägt den Namen *Kontinuitätsgleichung*. Sie beschreibt die Erhaltung der Teilchenzahl bzw. der Masse des Systems.

Anmerkung: In der hier dargestellten Herleitung haben wir implizit bereits die *starke* Formulierung und eine *schwache* Formulierung (engl. “weak formulation” einer Differentialgleichung kennengelernt. Gleichung (3.24) ist die starke Formulierung der Kontinuitätsgleichung. Diese verlangt, dass die Differentialgleichung für jeden räumlichen Punkt $\vec{r}$ erfüllt ist. Die entsprechende schwache Formulierung ist Gl. (3.23). Hier wird nur verlangt, dass die Gleichung in einer Art Mittelwert, hier als Integral über ein Probevolumen V, erfüllt ist. Innerhalb des Volumens muss die starke Form nicht erfüllt sein, aber das Integral über diese Abweichungen (die wir später als “Residuum” bezeichnen werden) muss verschwinden. Die schwache Formulierung ist für endliche Probevolumina V damit eine Näherung. In der Methode der finiten Elemente löst man eine schwache Gleichung für eine gewissen (approximative) Ansatzfunktion exakt. Die schwache Formulierung wird daher im Verlauf dieser Veranstaltung wichtig werden.

Wir können weiterhin noch verlangen, dass innerhalb unseres Probevolumens "Teilchen" produziert werden. In der aktuellen Interpretation der Gleichung wären dies z.B. chemische Reaktionen, die einen Teilchentyp in einen anderen umwandeln. Eine identische Gleichung gilt für den Wärmetransport. Hier wäre ein Quellterm die Produktion von Wärme, z.B. durch ein Heizelement. Gegeben ein Quellenstrom Q (mit Einheit Anzahl Partikel/Zeit/Volumen), kann die Kontinuitätsgleichung auf

$$\frac{\partial c}{\partial t} + \nabla \cdot \vec{j} = Q \tag{3.25}$$

erweitert werden. Die Kontinuitätsgleichung mit Quellterm wird auch manchmal als *Bilanzgleichung* bezeichnet.

Anmerkung: Gleichung (3.25) beschreibt die zeitliche Veränderung der Konzentration c. Eine verwandte Frage ist die nach der Lösung dieser Gleichung nach sehr langer Zeit - wenn sich ein dynamisches Gleichgewicht eingestellt hat. Dieses Gleichgewicht ist dadurch gekennzeichnet, dass $\partial c/\partial t = 0$. Die Gleichung

$$\nabla \cdot \vec{j} = Q \tag{3.26}$$

ist die *stationäre* Variante der Kontinuitätsgleichung.

3.2.1 Drift

Kommen wir zurück zu Transportprozessen, zunächst zu Drift. Wenn sich alle Teilchen in unserem Probevolumen in mit der Geschwindigkeit $\vec{v}$ bewegen, dann führt das zu einem Teilchenstrom

$$\vec{j}_{\text{Drift}} = c\vec{v}. \tag{3.27}$$

Eingesetzt in die Kontinuitätsgleichung (3.24) ergibt dies den Drift-Beitrag zur Drift-Diffusions-Gleichung (3.11).

3.2.2 Diffusion

Aus unserem obigen Gedankenexperiment wird klar, dass der Diffusionstrom immer in Richtung der niedrigen Konzentration, also in entgegengesetzte Richtung des Gradienten ∇c der Konzentration, gehen muss. Der entsprechende Strom ist gegeben durch

$$\vec{j}_{\text{Diffusion}} = -D\nabla c. \tag{3.28}$$

Eingesetzt in die Kontinuitätsgleichung (3.24) ergibt dies die Diffusionsgleichung (3.5).

Die gesamte Drift-Diffusionsgleichung hat daher die Form

$$\frac{\partial c}{\partial t} + \nabla \cdot \{-D\nabla c + c\vec{v}\} = 0. \tag{3.29}$$

Im Gegensatz zu Gleichungen (3.5) und (3.11) gilt diese Gleichung auch wenn die Diffusionskonstante D oder Drift-Geschwindigkeit $\vec{v}$ räumlich variiert.

Anmerkung: Wir haben hier die Transporttheorie im Sinne einer Teilchenkonzentration c eingeführt. Die Kontinuitätsgleichung beschreibt jedoch allgemein die *Erhaltung* einer bestimmten Größe, in unserem Fall der Teilchenzahl (oder äquivalent der Masse). Andere physikalisch erhaltene Größen sind der Impuls und die Energie. Die Kontinuitätsgleichung für den Impuls führt zur Navier-Stokes Gleichung. Die Kontinuitätsgleichung für die Energie führt zur Wärmeleitungsgleichung. Für das Beispiel dieser Veranstaltung ist nur die Erhaltung der Masse relevant.

Kapitel 4

Ladungstransport

Kontext: In diesem Lernmodul führen wir die spezifischen Gleichungen ein, welche Ladungstransport beschreiben. Ähnliche Gleichungen finden sich für Ladungstransport in Halbleitern und in Elektrolyten. Insbesondere sollten ähnliche Gleichungen bereits in der Vorlesung "Halbleiterphysik" aufgetaucht sein. Wir werden die Gleichungen hier im Kontext der Elektrochemie entwickeln. Ziel des Kapitels ist die Einführung der Poisson-Nernst-Planck Gleichung, die wir im Rest der Veranstaltung numerisch lösen werden.

4.1 Elektrostatik

Wir wiederholen hier die Grundlagen der Elektrostatik. Eine Punktladung q am Ort $\vec{r}_0$ erzeugt ein elektrostatisches Potential der Form

$$\Phi(\vec{r}) = \frac{1}{4\pi\varepsilon}\frac{q}{|\vec{r}-\vec{r}_0|}, \tag{4.1}$$

wobei $\varepsilon = \varepsilon_0\varepsilon_r$ die Permittivität ist. Im Vakuum ist $\varepsilon_r = 1$. Wir werden hier ausschließlich (wässrige) Elektrolyte behandeln, also Ionen die in Wasser gelöst sind. Für Wasser ist $\varepsilon_r \approx 80$. Gleichung (**??**) ist die spezifische Lösung der Poisson-Gleichung,

$$\nabla^2\Phi(\vec{r}) = -\frac{\rho(\vec{r})}{\varepsilon} \tag{4.2}$$

für eine Punktladung $\rho(\vec{r}) = q\delta(\vec{r}-\vec{r}_0)$.

Die Poisson-Gleichung hat die gleiche Form wie die (stationäre) Diffusionsgleichung aus Kapitel 3. Wir können auch diese wieder in zwei Gleichungen aufsplitten. Zunächst ist das elektrische Feld $\vec{E}$ gegeben durch

$$\vec{E} = -\nabla\Phi, \tag{4.3}$$

den (negativen) Gradienten des Potentials. (Im Sinne der Analogie zur Diffusionsgleichung ist das Feld eine Art Stromdichte.) Die "Kontinuitätsgleichung" für das Feld ist gegeben durch

$$\nabla \cdot \vec{E} = \frac{\rho}{\varepsilon}. \tag{4.4}$$

Zusammen ergeben diese Gleichungen die Poisson-Gleichung.

Wir werden die Poisson-Gleichung benötigen, um das elektrostatische Potential (und damit auch das elektrische Feld) innerhalb eines Elektrolyten zu berechnen. Innerhalb des Elektrolyten haben wir üblicherweise eine positiv und eine negativ geladenen Spezies, mit entsprechenden Konzentrationen $c_+(\vec{r})$ und $c_-(\vec{r})$. Die entsprechende Ladungsdichte ist dann proportional zur Differenz dieser Konzentrationen, $\rho(\vec{r}) = |e|(c_+(\vec{r}) - c_-(\vec{r}))$.

4.2 Drift im elektrischen Feld

Die Ladungen in unserem Elektrolyten erzeugen nicht nur ein elektrischen Feld, sie reagieren auch auf dieses. Die Kraft $\vec{f}$, welche auf ein Teilchen mit Ladung q wirkt, ist gegeben durch

$$\vec{f}_{\mathrm{E}} = q\vec{E}. \tag{4.5}$$

Positiv geladene Teilchen bewegen sich in Richtung des elektrischen Feldes, negativ geladene Teilchen entgegen dieser Richtung.

Es wirkt also auf Grund des elektrischen Feldes eine *Kraft* auf unsere Ionen. Diese Kraft alleine würde zu einer *Beschleunigung* der Ionen führen, also einer kontinuierlichen Zunahme der Geschwindigkeit. Da sich das Ion in einem Medium (Lösungsmittel, z.B. Wasser) bewegt erfährt es einen Strömungswiderstand (siehe Abb. 4.1). Im Fall laminaren Flusses um ein sphärisches Teilchen mit Radius R wird dieser nur durch innere Reibung innerhalb des Fluids hervorgerufen. Die resultieren Kraft wirkt entgegen der Bewegungsrichtung und wird durch das Stokessche Gesetz,

$$\vec{f}_{\mathrm{Stokes}} = -6\pi\eta R\vec{v} = -\vec{v}/\Lambda, \tag{4.6}$$

mit $\Lambda = (6\pi\eta R)^{-1}$ beschrieben. Hier ist η die Viskosität der Flüssigkeit. Die Größe Λ nennt sich die *Mobilität*. Im Gleichgewicht $\vec{f}_{\mathrm{E}} + \vec{f}_{\mathrm{Stokes}} = 0$ ergibt sich die Driftgeschwindigkeit

$$\vec{v} = q\Lambda\vec{E}. \tag{4.7}$$

Diese Driftgeschwindigkeit ergibt zusammen mit $\vec{j} = c\vec{v}$ den durch das elektrische Feld hervorgerufenen Driftstrom,

$$\vec{j} = q\Lambda c\vec{E} = \sigma\vec{E} \tag{4.8}$$

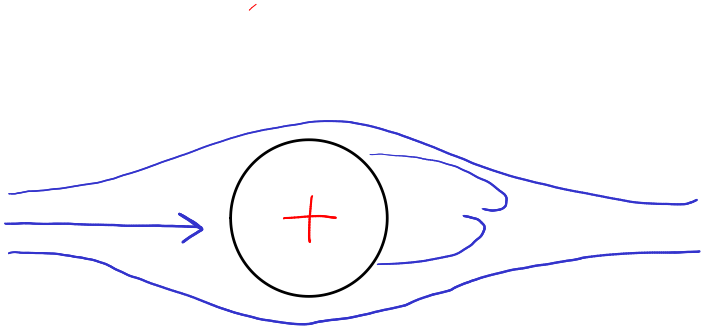

Abbildung 4.1: Ein Teilchen (z.B. ein Ion), welches sich in einer Flüsiggkeit bewegt, erfährt einen Strömungswiderstand. Bei geringen Geschwindigkeiten wird dieser durch innere Reibung innerhalb des umströmenden Fluids hervorgerufen.

mit $\sigma = q\Lambda c$. Die Größe σ wird auch Leitfähigkeit genannt. Ein äquivalentes Gesetz gilt beispielsweise für die Elektronenleitung in Metallen.

4.3 Nernst-Planck-Gleichung

Ein Diffusionsstrom in Kombination mit Drift im elektrischen Feld ergibt die *Nernst-Planck-Gleichung*. Die Stromdichte für Ionenspezies α ist gegeben durch

$$\vec{j}_\alpha = -D_\alpha \nabla c_\alpha + q_\alpha \Lambda_\alpha c_\alpha \vec{E}, \tag{4.9}$$

wobei wir explizit durch den Index α darauf hingewiesen haben, dass die Transportparameter (D, Λ), die Ladung q und die Konzentration c von der ionischen Spezies abhängen. Mit Hilfe der Einstein-Smoluchowski-Beziehung, $D = \Lambda k_B T$ kann die Mobilität Λ mit der Diffusionskonstante D ausgedrückt werden. Dies führt zu der üblichen Form der Nernst-Planck-Gleichung,

$$\vec{j}_\alpha = -D_\alpha \left(\nabla c_\alpha + \frac{q_\alpha}{k_B T} c_\alpha \nabla \Phi \right), \tag{4.10}$$

in der wir das elektrische Feld als $\vec{E} = -\nabla\Phi$ ausgedrückt haben.

4.4 Poisson-Nernst-Planck-Gleichungen

Wir kombinieren nun das Nernst-Plancksche Transportproblem mit der Lösung der Poisson-Gleichung um das elektrostatische Potential Φ zu bestimmen.

Hierzu müssen hier zwei Ionenspecies betrachten, eine positiv (Ladung q_+) und eine negativ (Ladung q_-) geladene. Damit müssen wir neben dem Potential Φ zwei Konzentrationen c_+ und c_- bestimmen.

Das gekoppelte Gleichungssystem, welches die Transportprozesse in unserer Elektrolytlösung beschreibt, sieht daher folgendermaßen aus:

$$\frac{\partial}{\partial t} c_+ + \nabla \cdot \vec{j}_+ = 0 \qquad \text{(Erhaltung der positiven Spezies)} \tag{4.11}$$

$$\vec{j}_+ = -D_+ \left(\nabla c_+ + \frac{q_+}{k_B T} c_+ \nabla \Phi \right) \qquad \text{(Transport der positiven Spezies)} \tag{4.12}$$

$$\frac{\partial}{\partial t} c_- + \nabla \cdot \vec{j}_- = 0 \qquad \text{(Erhaltung der negative Spezies)} \tag{4.13}$$

$$\vec{j}_- = -D_- \left(\nabla c_- + \frac{q_-}{k_B T} c_- \nabla \Phi \right) \qquad \text{(Transport der negative Spezies)} \tag{4.14}$$

$$\nabla^2 \Phi = -\frac{q_+ c_+ + q_- c_-}{\varepsilon} \qquad \text{(Elektrostatisches Potential)} \tag{4.15}$$

Wir werden dieses gekoppelte Differentialgleichungssystem im Rahmen dieses Kurses mit Hilfe der Methode der finiten Elemente lösen. Man nennt diese Gleichungen die *Poisson-Nernst-Planck Gleichungen*.

Anmerkung: Ein Satz von Gleichungen identisch zu Gl. (4.11) bis (4.15) beschreibt den Transport von Ladungsträgern in Halbleitern. Die positiven Ladungsträger sind dann Löcher und die negativen Elektronen. Das was hier als "chemisches Potential" bezeichnet wird, heißt dort teilweise das Quasiferminiveau. Diese Art des Ladungsträgertransports wurde bereits in der Vorlesung "Halbleiterphysik" besprochen.

Neben der transienten Lösung des Problems, also der Zeitpropagation der beiden Konzentrationen c_+ und c_-, ist auch die stationäre Lösung interessant. Für die stationäre Lösung verschwindet die Zeitabhängigkeit, also $\partial c_{+/-}/\partial t = 0$, in diesen Gleichungen. Wir werden hier sowohl die transiente als auch die stationäre Lösung dieses und ähnlicher Gleichungssystems betrachten.

4.5 Poisson-Boltzmann-Gleichung

Die Nernst-Planck-Gleichung kann durch die Einführung eines *chemischen Potentials* vereinfacht werden. Das chemische Potential integriert den Effekt der Diffusion in ein effektives Potential

$$\mu_\alpha(\vec{r}) = q_\alpha \Phi(\vec{r}) + k_B T \ln c_\alpha(\vec{r}). \tag{4.16}$$

Der Term $q_\alpha \Phi$ ist hier die potentielle Energie eines Ions mit Ladung q_α in einem elektrischen Feld. Der Term $k_B T \ln c_\alpha$ ist die freie Energie eines idealen Gases mit Dichte c_α. Wir können hier die Ionen als ideales Gas beschreiben, weil diese (in unserem Modell) nur über das elektrostatische Potential wechselwirken. Die Stromdichte wird dann proportional zum Gradienten des chemischen Potentials μ,

$$\vec{j}_\alpha = -\frac{D_\alpha}{k_B T} c_\alpha \nabla \mu = -\Lambda_\alpha c_\alpha \nabla \mu. \tag{4.17}$$

Durch Einsetzen von Gl. (4.16) in Gl. (4.17) erhält man Gl. (4.10).

Gleichung (4.17) sagt uns, dass kein Strom fließt, wenn das chemische Potential μ räumlich konstant ist. Dies ist genau dann der Fall, wenn

$$c_\alpha(\vec{r}) = c_0 \exp\left(-\frac{q_\alpha \Phi(\vec{r})}{k_B T}\right) \tag{4.18}$$

mit einer Konstanten c_0. Diese Gleichung in Verbindung mit der Poisson-Gleichung zur Bestimmung von Φ heißt auch die *Poisson-Boltzmann-Gleichung*.

4.6 Anwendungsbeispiel

Im folgende Video diskutieren wir die Anwendung der Poisson-Nernst-Planck-Gleichung für die Modellierung von Ladungstransport in Superkondensatoren mit porösen Elektroden.

Kapitel 5

Numerische Lösungsstrategien

Kontext: Wir legen jetzt das Transportproblem für eine Weile zur Seite und wollen uns der *numerischen* Lösung von Differentialgleichungen widmen. Dieses Kapitel zeigt die Grundzüge der numerischen Analyse von Differentialgleichungen und führt ein paar wichtige Konzepte ein, insbesondere die Reihenentwicklung und das Residuum. Die Darstellung hier folgt Kapitel 1 aus Boyd (2000).

5.1 Reihenentwicklung

In abstrakter Schreibweise suchen wir nach unbekannten Funktionen $u(x, y, z, ...)$ die einen Satz von Differentialgleichungen

$$\mathcal{L}u(x, y, z, \ldots) = f(x, y, z, \ldots) \tag{5.1}$$

erfüllen. Hierbei ist $\mathcal{L}$ ein (nicht zwingend linearer) Operator, der die Differential- (oder Integral-)operationen enthält. Zur (numerischen) Lösung der Differentialgleichung führen wir nun ein wichtiges Konzept ein: Wir nähern die Funktion u durch eine *Reihenentwicklung* an. Wir schreiben

$$u_N(x, y, z, \ldots) = \sum_{n=0}^{N} a_n \varphi_n(x, y, z, \ldots) \tag{5.2}$$

wobei die φ_n als "Basisfunktionen" bezeichnet werden. Wir werden die Eigenschaften dieser Basisfunktionen in mehr Details im nächsten Kapitel diskutieren.

Die Differentialgleichung können wir nun schreiben als,

$$\mathcal{L}u_N(x, y, z, \ldots) = f(x, y, z, \ldots). \tag{5.3}$$

Durch diese Darstellung wird erreicht, dass wir nun die Frage nach der unbekannten Funktion u durch die Frage nach den unbekannte Koeffizienten a_n ersetzt haben. Den Differentialoperator $\mathcal{L}$ müssen wir nur auf die (bekannten) Basisfunktionen φ_n wirken lassen, und dies können wir analytisch berechnen.

Was verbleibt ist die Bestimmung der Koeffizienten a_n. Diese Koeffizienten sind Zahlen, und diese Zahlen können von einem Computer berechnet werden. Gleichung (5.2) ist selbstverständlich eine Näherung. Für gewisse Basisfunktionen kann gezeigt werden, dass diese "vollständig" sind und damit bestimmte Klassen von Funktionen exakt abbilden können. Dies stimmt aber nur unter der Bedingung, dass die Reihe Gl. (5.2) bis $N \to \infty$ geführt wird. Für alle praktischen Anwendungsfälle (so wie Implementierungen in Computercode), muss diese Reihenentwicklung jedoch abgebrochen werden. Eine "gute" Reihenentwicklung approximiert die exakte Lösung bereits bei niedrigem N mit kleinem Fehler. Wir müssten bei dieser Aussage natürlich noch spezifizieren, wie wir Fehler quantifizieren möchten. Numerisch suchen wir dann genau nach den Koeffizienten a_n, die den Fehler minimieren.

Die Wahl guter Basisfunktion ist nichttrivial. Wir werden hier hauptsächlich "finite Elemente" als Basisfunktionen nutzen und andere Arten kurz ansprechen. Bevor wir tiefer in dieses Thema einsteigen, brauchen wir noch weitere Konzepte für das Verständnis der numerischen Analyse.

5.2 Residuum

Ein wichtiges Konzept ist das des *Residuums*. Unser Ziel ist es, Gl. (5.1) zu lösen. Für die exakte Lösung wäre $\mathcal{L}u - f \equiv 0$. Da wir aber nur eine Näherungslösung konstruieren können, wird diese Bedingung nicht exakt erfüllt sein. Wir definieren das Residuum als genau diese Abweichung von der exakten Lösung, nämlich

$$R(x, y, z, \ldots; a_0, a_1, \ldots, a_N) = \mathcal{L}u_N(x, y, z, \ldots) - f(x, y, z, \ldots). \tag{5.4}$$

Das Residuum ist damit eine Art Maß für den Fehler den wir machen. Die Strategie zur numerischen Lösung der Differentialgleichung Gl. (5.1), ist es nun, die Koeffizienten a_n so zu bestimmen, dass das Residuum Gl. (5.4) minimal wird. Wir haben damit die Lösung der Differentialgleichung auf ein Optimierungsproblem abgebildet. Die unterschiedlichen numerischen Verfahren, die wir in den nächsten Kapiteln diskutieren werden, entscheiden sich hier hauptsächlich in der spezifischen Optimierungsstrategie.

Anmerkung: Numerische Verfahren für die *Optimierung* sind ein zentraler Kern der numerischen Lösung von Differentialgleichungen und damit der Simulationstechniken. Es gibt unzählige Optimierungsverfahren, die in unterschiedlichen Situationen besser oder schlechter funktionieren. Wir werden hier zunächst solche Optimierer als "Black Box" behandeln. Zum Ende der Lehrveranstaltung werden wir zur Frage der Optimierung zurückkehren und einige bekannte Optimierungsverfahren diskutieren. Der Begriff *Minimierungsverfahren* wird oft synonym zu Optimierungsverfahren verwendet. Eine gute Übersicht über Optimierungsverfahren bietet das Buch von Nocedal and Wright (2006).

5.3 Ein erstes Beispiel

Wir wollen nun diese abstrakten Ideen an einem Beispiel konkretisieren und ein paar wichtige Begriffe einführen. Wir schauen uns das eindimensionale Randwertproblem,

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} - (x^6 + 3x^2)u = 0, \tag{5.5}$$

mit den Randbedingungen $u(-1) = u(1) = 1$ an. (D.h. $x \in [-1, 1]$ ist die Domäne auf der wir die Lösung suchen.) Der abstrakte Differentialoperator $\mathcal{L}$ nimmt in diesem Fall die konkrete Form

$$\mathcal{L} = \frac{\mathrm{d}^2}{\mathrm{d}x^2} - (x^6 + 3x^2) \tag{5.6}$$

an. Die exakte Lösung dieses Problems ist gegeben durch

$$u(x) = \exp\left[(x^4 - 1)/4\right]. \tag{5.7}$$

Wir raten nun eine Näherungslösung als Reihenentwicklung für diese Gleichung. Diese Näherungslösung sollte bereits die Randbedingungen erfüllen. Die Gleichung

$$u_2(x) = 1 + (1 - x^2)(a_0 + a_1 x + a_2 x^2) \tag{5.8}$$

ist so konstruiert, dass die Randbedingungen erfüllt sind. Wir können diese als

$$u_2(x) = 1 + a_0(1 - x^2) + a_1 x(1 - x^2) + a_2 x^2(1 - x^2) \tag{5.9}$$

umschreiben, um die Basisfunktionen $\varphi_i(x)$ zu exponieren. Hier $\varphi_0(x) = 1 - x^2$, $\varphi_1(x) = x(1 - x^2)$ und $\varphi_2(x) = x^2(1 - x^2)$. Da diese Basisfunktionen auf der

gesamten Domäne $[-1, 1]$ ungleich Null sind, heißt diese Basis eine *spektrale* Basis. (Mathematisch: Der Träger der Funktion entspricht der Domäne.)

Im nächsten Schritt müssen wir das Residuum

$$R(x; a_0, a_1, a_2) = \frac{\mathrm{d}^2 u_2}{\mathrm{d}x^2} - (x^6 + 3x^2)u_2 \tag{5.10}$$

minimieren. Hierfür wählen wir eine Strategie, die als *Kollokation* bezeichnet wird: Wir verlangen, dass an drei ausgewählten Punkten das Residuum exakt verschwindet:

$$R(x_i; a_0, a_1, a_2) = 0 \quad \text{für} \quad x_0 = -1/2, x_1 = 0 \text{ und } x_2 = 1/2. \tag{5.11}$$

Anmerkung: Das Verschwinden des Residuums bei x_i bedeutet nicht, dass auch $u_2(x_i) \equiv u(x_i)$, also dass bei x_i unsere approximative Lösung der exakten Lösung entspricht. Wir sind immer noch auf einen begrenzten Satz von Funktionen, nämlich die Funktionen die durch Gl. (5.8) erfasst werden, beschränkt.

Aus der Kollokationsbedingung bekommen wir nun ein lineares Gleichungssystem mit drei Unbekannten:

$$R(x_0; a_0, a_1, a_2) \equiv -\frac{659}{256}a_0 + \frac{1683}{512}a_1 - \frac{1171}{1024}a_2 - \frac{49}{64} = 0 \tag{5.12}$$

$$R(x_1; a_0, a_1, a_2) \equiv -2(a_0 - a_2) = 0 \tag{5.13}$$

$$R(x_2; a_0, a_1, a_2) \equiv -\frac{659}{256}a_0 - \frac{1683}{512}a_1 - \frac{1171}{1024}a_2 - \frac{49}{64} = 0 \tag{5.14}$$

$$\tag{5.15}$$

Die Lösung dieser Gleichungen ergibt

$$a_0 = -\frac{784}{3807}, \quad a_1 = 0 \quad \text{und} \quad a_2 = a_0. \tag{5.16}$$

Abbildung 5.1 zeigt die "numerische" Lösung $u_2(x)$ im Vergleich mit der exakten Lösung $u(x)$.

In dem hier dargestellten numerischen Beispiel können sowohl die Basisfunktionen als auch die Strategie für die Minimierung des Residuums variiert werden. Im Laufe dieser Lehrveranstaltung werden wir als Basisfunktionen die finiten Elemente etablieren und als Minimierungsstrategie die Galerkin-Methode nutzen. Hierzu müssen wir zunächst Eigenschaften möglicher Basisfunktionen diskutieren.

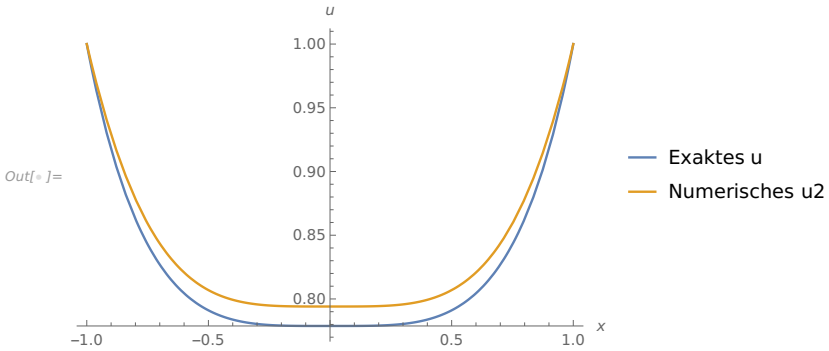

Abbildung 5.1: Analytische Lösung $u(x)$ und "numerische" approximative Lösung $u_2(x)$ der GDGL (5.5).

Anmerkung: Das hier dargestellte Beispiel ist ein einfacher Fall einer *Diskretisierung*. Wir sind von einer kontinuierlichen Funktion auf die diskreten Koeffizienten a_0, a_1, a_2 übergegangen.

Kapitel 6

Funktionenräume

Kontext: Bevor wir tiefer in die numerische Lösung von partiellen Differentialgleichungen einsteigen, müssen wir hier ein leicht abstraktes Konzept einführen: Das Konzept der *Funktionenräume*, bzw. konkreter des *Hilbertraums*. Funktionenräume sind nützlich, weil sie die Reihenentwicklung formalisieren und durch das Konzept der Basisfunktionen einen einfachen Zugang zu den Koeffizienten einer Reihenentwicklung liefern.

6.1 Vektoren

Zur Einführung erinnern wir an die üblichen kartesischen Vektoren. Einen Vektor $\vec{a} = (a_0, a_1, a_2)$ können wir als Linearkombination aus Basisvektoren $\hat{e}_0$, $\hat{e}_1$ und $\hat{e}_2$,

$$\vec{a} = a_0\hat{e}_0 + a_1\hat{e}_1 + a_2\hat{e}_2, \tag{6.1}$$

schreiben. Die Einheitsvektoren $\hat{e}_0$, $\hat{e}_1$ und $\hat{e}_2$ sind natürlich die Vektoren, welche das kartesische Koordinatensystem aufspannen. (In vorherigen Kapiteln wurde auch die Notation $\hat{x} \equiv \hat{e}_0$, $\hat{y} \equiv \hat{e}_1$ und $\hat{z} \equiv \hat{e}_2$ genutzt.) Die Zahlen a_0, a_1 und a_2 sind die Komponenten oder *Koordinaten* des Vektors, aber auch die Koeffizienten der Einheitsvektoren. In diesem Sinne sind sie identisch zu den Entwicklungskoeffizienten der Reihenentwicklung, mit dem Unterschied, dass die $\hat{e}_i$ orthogonal sind, also

$$\hat{e}_i \cdot \hat{e}_j = \delta_{ij} \tag{6.2}$$

wobei δ_{ij} das Kronecker-δ ist. Zwei kartesische Vektoren $\vec{a}$ und $\vec{b}$ sind orthogonal, wenn das Skalarprodukt zwischen ihnen verschwindet:

$$\vec{a} \cdot \vec{b} = \sum_i a_i b_i = 0 \tag{6.3}$$

Mit Hilfe der Basisvektoren und dem Skalarprodukt können wir direkt die Komponenten erhalten: $a_i = \vec{a} \cdot \hat{e}_i$. Dies ist eine direkte Konsequenz der Orthogonalität der Basisvektoren $\hat{e}_i$.

6.2 Funktionen

Im vorhergegangen Abschnitt steht die Behauptung, dass die Basisfunktionen aus Kapitel 5 nicht orthogonal sind. Hierzu brauchen wir eine Idee für Orthogonalität von Funktionen. Mit einer Definition eines Skalarprodukts zwischen zwei *Funktionen* könnten wir dann Orthogonalität als Verschwinden dieses Skalarprodukts definieren.

Wir führen nun Skalarprodukt auf Funktionen(räume) ein, das diese Eigenschaften auch erfüllt. Gegeben zwei Funktionen $g(x)$ und $f(x)$ auf dem Interval $x \in [a, b]$, *definieren* wir das Skalarprodukt als

$$(f, g) = \int_a^b \mathrm{d}x\, f^*(x) g(x), \tag{6.4}$$

wobei $f^*(x)$ die komplex-konjugierte von $f(x)$ ist. Dieses inneres Produkt oder Skalarprodukt ist eine Abbildung mit den Eigenschaften

- Positiv definit: $(f, f) \geq 0$ und $(f, f) = 0 \Leftrightarrow f = 0$
- Sesquilinear: $(\alpha f + \beta g, h) = \alpha^*(f, h) + \beta^*(g, h)$
- Hermitesch: $(f, g) = (g, f)^*$

Die Skalarprodukte Gl. (6.3) und (6.4) erfüllen beide diese Eigenschaften.

Anmerkung: Das Skalarprodukt zwischen zwei Funktionen wird oft allgemeiner mit einer Gewichtsfunktion $w(x)$ definiert,

$$(f, g) = \int_a^b \mathrm{d}x\, f^*(x) g(x) w(x). \tag{6.5}$$

Die Frage nach Orthogonalität zwischen Funktionen kann damit nur respektive einer bestimmten Definition des Skalarprodukts geklärt werden. So sind z.B. die Tschebyschow-Polynome respektive der Gewichtsfunktion $w(x) = (1-x^2)^{-1/2}$ orthogonal. Innerhalb dieser Lehrveranstaltung werden wir nur den Fall $w(x) = 1$ benötigen.

6.3 Basisfunktionen

Kommen wir nun zurück zur Reihenentwicklung,

$$f_N(x) = \sum_{n=0}^{N} a_n \varphi_n(x). \tag{6.6}$$

Die Funktionen $\varphi_i(x)$ heißen *Basisfunktionen*. Eine notwendige Eigenschaft der Basisfunktionen ist deren lineare Unabhängigkeit. Die Funktionen sind linear unabhängig, wenn keine der Basisfunktionen selbst als Linearkombination, also in der Form der Reiheentwicklung Gl. (6.6), der anderen Basisfunktionen geschrieben werden kann. D.h. es muss erfüllt sein, dass

$$\sum_{n=0}^{N} a_n \varphi_n(x) = 0 \tag{6.7}$$

dann und nur dann wenn alle $a_n = 0$. Linear unabhängige Elemente bilden eine Basis.

Diese Basis heißt vollständig, wenn alle relevanten Funktionen (= Elemente des zu Grunde liegenden Vektorraums) sich durch die Reihenentwicklung (6.6) abbilden lassen. (Beweise der Vollständigkeit von Basisfunktionen sind komplex und außerhalb des Fokus dieser Lehrveranstaltung.) Die Koeffizienten a_n heißen Koordinaten oder Koeffizienten. Die Anzahl der Basisfunktionen bzw. der Koordinaten N nennt man die *Dimension* des Vektorraums.

Anmerkung: Ein *Vektorraum* ist eine Menge, auf der die Operationen der Addition und Skalarmultiplikation mit den üblichen Eigenschaften, wie der Existenz von neutralen und inversen Elementen und Assoziativ-, Kommutativ- und Distributivgesetzen, definiert sind. Ist dieser Raum auf Funktionen definiert, dann spricht man auch von einem *Funktionenraum*. Existiert zusätzlich ein Skalarprodukt wie Gl. (6.4), dann spricht man von einem *Hilbertraum*.

Besonders nützliche Basisfunktionen sind orthogonal. Mit Hilfe des Skalarprodukts können wir nun Orthogonalität für diese Funktionen definieren. Zwei Funktionen f und g sind orthogonal wenn das Skalarprodukt verschwindet, $(f, g) = 0$. Ein Satz gegenseitig orthogonaler Basisfunktionen erfüllt

$$(\varphi_n, \varphi_m) = \nu_n \delta_{nm}, \tag{6.8}$$

wobei δ_{nm} das Kronecker-δ ist. Für $\nu_n \equiv (\varphi_n, \varphi_n) = 1$ heißt die Basis *orthonormal*.

Die Orthogonalität ist nützlich, weil sie uns einen Weg aufzeigt, mit dem wir die Koeffizienten der Reihenentwicklung (6.6) bekommen können:

$$(\varphi_n, f_N) = \sum_{i=0}^{N} a_i(\varphi_n, \varphi_i) = \sum_{i=0}^{N} a_i \nu_i \delta_{ni} = a_n \nu_n \tag{6.9}$$

bzw.

$$a_n = \frac{(\varphi_n, f_N)}{(\varphi_n, \varphi_n)}. \tag{6.10}$$

D.h. die Koordinaten sind gegeben durch die Projektion (das Skalarprodukt) der Funktion auf die Basisvektoren. Wir erinnern uns daran, dass auch für kartesische Vektoren gilt: $a_n = \vec{a} \cdot \hat{e}_n$. (Der Normierungsfaktor entfällt hier, weil $\hat{e}_n \cdot \hat{e}_n = 1$.) Die Koordinaten, die durch Gl. (6.10) gegeben sind, sind in genau dem gleichen Kontext zu sehen.

6.3.1 Fourier-Basis

Ein berühmter und wichtiger Satz von Basisfunktionen ist die *Fourier-Basis*,

$$\varphi_n(x) = \exp\left(i q_n x\right), \tag{6.11}$$

auf dem Interval $x \in [0, L]$ mit $q_n = 2\pi n/L$ und $n \in \mathbb{Z}$. Die Fourier-Basis ist periodisch auf diesem Interval und in Abb. 6.1 gezeigt. Es kann einfach gezeigt werden, dass

$$(\varphi_n, \varphi_m) = L\delta_{nm}, \tag{6.12}$$

also dass die Fourier-Basis orthogonal ist. Die Koeffizienten a_n der Fourier-Reihe,

$$f_N(x) = \sum_{n=-N}^{N} a_n \varphi_n(x), \tag{6.13}$$

können damit für eine Reihe $f_N(x)$ direkt als

$$a_n = \frac{1}{L}(\varphi_n, f_N) = \frac{1}{L}\int_0^L \mathrm{d}x\, f_N(x) \exp\left(-i q_n x\right) \tag{6.14}$$

bestimmt werden. Dies ist die bekannte Formel für die Koeffizienten der Fourier-Reihe. Man beachte, dass die Summe in Gl. (6.13) von $-N$ bis N läuft und man $2N + 1$ Koeffizienten erhält.

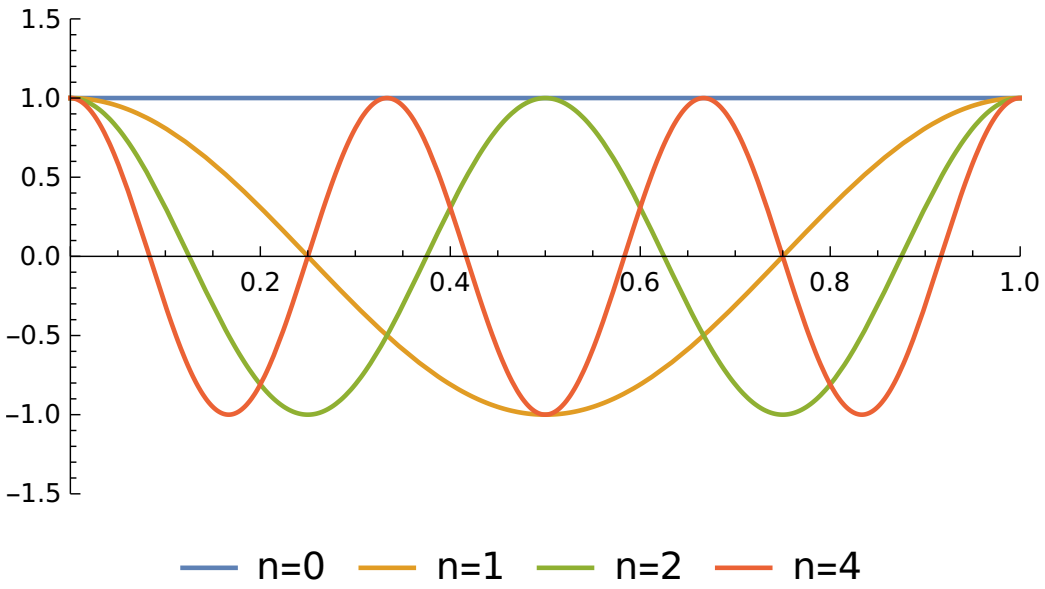

Abbildung 6.1: Realteil der Fourier-Basisfunktionen, Gl. (6.11), für $n = 1, 2, 3, 4$. Die Basisfunktionen höherer Ordnung oszillieren mit einer kleineren Periode und repräsentieren höhere Frequenzen.

Anmerkung: Konzeptuell beschreibt die Fourier-Basis unterschiedliche Frequenzkomponenten, während die Basis der im nächsten Abschnitt beschriebenen finiten Elemente unterschiedliche Raumbereiche beschreibt.

6.3.2 Finite Elemente

Wir werden hier hauptsächlich mit der Basis der finiten Elemente arbeiten. Im Gegensatz zur Fourier-Basis, die auf der gesamten Domäne nur an isolierten Punkten gleich Null wird, ist die Finite-Elemente-Basis im Raum lokalisiert und für große Bereiche der Domäne Null. Sie zerlegt damit die Domäne in räumliche Abschnitte.

In ihrer einfachsten Form besteht die Basis aus lokalisierten abschnittsweise linearen Funktionen, der "Zelt"-Funktion,

$$\varphi_n(x) = \begin{cases} \frac{x - x_{n-1}}{x_n - x_{n-1}} & \text{für } x \in [x_{n-1}, x_n] \\ \frac{x_{n+1} - x}{x_{n+1} - x_n} & \text{für } x \in [x_n, x_{n+1}] \\ 0 & \text{sonst} \end{cases} \tag{6.15}$$

Hierbei sind die x_n die Stützstellen (auch Gitterpunkte oder Knoten - engl.

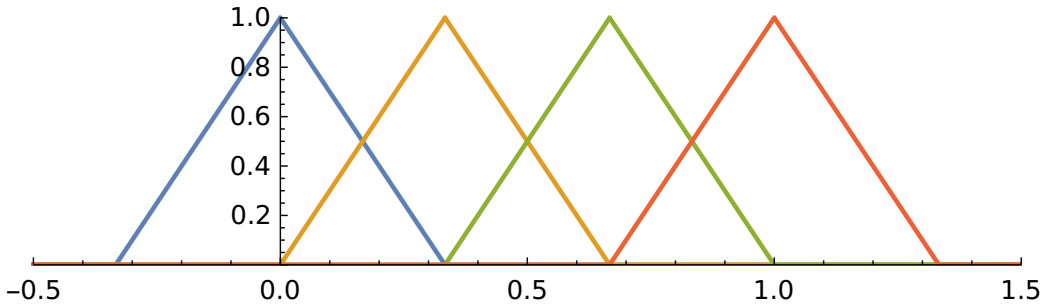

Abbildung 6.2: Die Basis der finiten Elemente in ihrer einfachsten, linearen Inkarnation. Jede Basisfunktion ist ein "Zelt", dass über ein gewisses Interval zwischen 0 und 1 und wieder zurück verläuft, siehe auch Gl. (6.15).

"node"), zwischen denen die Zelte aufgespannt sind. Die Funktionen sind so konstruiert, dass $\int_0^L \mathrm{d}x\, \varphi_n(x) = (x_{n+1} - x_{n-1})/2$. Diese Basis ist die einfachste Form der finite Elemente-Basis und in Abb. 6.2 gezeigt. Für höhere Genauigkeit werden auch Polynome höherer Ordnung eingesetzt.

Ein wichtiger Hinweis an dieser Stelle ist, dass die Basis der finiten Elemente *nicht* orthogonal ist. In unserem eindimensionalen Fall verschwindet das Skalarprodukt nicht für die nächsten Nachbarn. Dies ist deshalb der Fall, weil bei zwei Nachbarn jeweils eine steigende und eine fallende Flanke überlappt. Man erhält

$$M_{nn} \equiv (\varphi_n, \varphi_n) = \frac{1}{3}(x_{n+1} - x_{n-1}) \tag{6.16}$$

$$M_{n,n+1} \equiv (\varphi_n, \varphi_{n+1}) = \frac{1}{6}(x_{n+1} - x_n) \tag{6.17}$$

$$M_{nm} \equiv (\varphi_n, \varphi_m) = 0 \quad \text{für } |n-m| > 1 \tag{6.18}$$

für die Skalarprodukte.

Trotzdem können wir diese Relationen nutzen, um die Koeffizienten einer Reihenentwicklung,

$$f_N(x) = \sum_{n=0}^{N} a_n \varphi_n(x) \tag{6.19}$$

zu berechnen. Man erhält

$$\begin{aligned}(\varphi_n, f_N(x)) &= a_{n-1}(\varphi_n, \varphi_{n-1}) + a_n(\varphi_n, \varphi_n) + a_{n+1}(\varphi_n, \varphi_{n+1}) \\ &= M_{n,n-1}a_{n-1} + M_{nn}a_n + M_{n,n+1}a_{n+1}.\end{aligned} \tag{6.20}$$

Dies können wir als

$$(\varphi_n, f_N(x)) = [\underline{M} \cdot \vec{a}]_n \tag{6.21}$$

schreiben, wobei $[\vec{v}]_n = v_n$ die $n$te Komponente des Vektors, welcher durch die beiden eckigen Klammer $[\cdot]_n$ eingeschlossen wird, bezeichnet. Die Matrix $\underline{M}$ ist *dünnbesetzt* (engl. "sparse"). Für eine orthogonale Basis, wie beispielsweise die Fourier-Basis in Abschnitt 6.3.1, ist diese Matrix diagonal. Für eine Basis mit identischen Abständen $x_{n+1} - x_n = 1$ der Stützstellen x_n hat die Matrix die folgende Form

$$\underline{M} = \begin{pmatrix} 2/3 & 1/6 & 0 & 0 & 0 & 0 & \cdots \\ 1/6 & 2/3 & 1/6 & 0 & 0 & 0 & \cdots \\ 0 & 1/6 & 2/3 & 1/6 & 0 & 0 & \cdots \\ 0 & 0 & 1/6 & 2/3 & 1/6 & 0 & \cdots \\ 0 & 0 & 0 & 1/6 & 2/3 & 1/6 & \cdots \\ 0 & 0 & 0 & 0 & 1/6 & 2/3 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} . \tag{6.22}$$

Um die Koeffizienten a_n zu finden, muss also ein (dünnbesetztes) lineares Gleichungssystem gelöst werden. Wir werden $\underline{M}$ später unter dem Namen *Massematrix* wieder treffen.

Anmerkung: Basissätze, die nur an individuellen Punkten von Null verschieden sind, nennt man *spektrale* Basissätze. Insbesondere ist die Fourier-Basis ein spektraler Basissatz für periodische Funktionen. Grundsätzlich bilden die *orthogonalen Polynome* wichtige spektrale Basissätze die auch in der Numerik Anwendung finden. So sind beispielsweise die Tschebyschow-Polynome gute Basissätze für auf abgeschlossenen Intervallen definierte nicht-periodische Funktionen. Die Basis der finiten Elemente ist keine spektrale Basis.

Kapitel 7

Approximation und Interpolation

Kontext: Wir wenden nun die Idee der Basisfunktionen an, um Funktionen zu approximieren. Hierfür kommen wir zu dem Konzept des Residuums zurück. Ziel der Funktionsapproximation ist es, dass die approximierte Funktion das Residuum minimiert. Aufbauend auf diesen Ideen besprechen wir dann im nächsten Kapitel die Approximation von Differentialgleichungen.

7.1 Residuum

Im vorherigen Abschnitt haben wir beschrieben, wie mit Basisfunktionen eine Reihenentwicklung aufgebaut werden kann. Eine typische Reihenentwicklung enthält eine endliche Zahl an Elementen $N + 1$ und hat die Form

$$f_N(x) = \sum_{n=0}^{N} a_n \varphi_n(x), \tag{7.1}$$

wobei die $\varphi_n(x)$ die im vorherigen Kapitel eingeführten Basisfunktionen sind.

Wir wollen uns nun der Frage nähern, wie wir ein beliebige Funktion $f(x)$ über eine solche Basisfunktionsentwicklung annähern können. Hierzu definieren wir das Residuum

$$R(x) = f_N(x) - f(x), \tag{7.2}$$

welches an jedem Punkt x verschwindet wenn $f_N(x) \equiv f(x)$. Für eine Approximation wollen wir dieses Residuum "minimieren". (Mit minimieren ist

hier gemeint, es möglichst nah an Null zu bringen.) Wir suchen also die Koeffizienten a_n der Reihenentwicklung, welche die Funktion $f(x)$ im Sinne einer Minimierung des Residuums approximiert.

An dieser Stelle sei noch bemerkt, dass die Basisfunktionen auf dem gleichen Raum für die Zielfunktion $f(x)$ definiert sein müssen. Für die Approximation einer periodischen Funktion $f(x)$ sollte auch ein periodischer Basissatz verwandt werden.

7.2 Kollokation

Als erste Minimierungsstrategie wird hier die *Kollokation* eingeführt. In diese Methode wird verlangt, dass das Residuum an ausgewählten Kollokationspunkten y_n verschwindet,

$$R(y_n) = 0 \quad \text{bzw.} \quad f_N(y_n) = f(y_n). \tag{7.3}$$

Die Anzahl der Kollokationspunkte muss hier der Anzahl der Koeffizienten in der Reihenentwicklung entsprechen. Die Wahl der idealen Kollokationspunkte y_n selbst ist nicht-trivial, und wir werden hier nur spezifische Fälle besprechen.

Als erstes Beispiel diskutieren wir hier eine Entwicklung mit N finiten Elementen. Als Kollokationspunkte wählen wir die Stützstellen der Basis, $y_n = x_n$. An diesen Stützstellen ist nur eine der Basisfunktionen ungleich Null, $\varphi_n(y_n) = 1$ und $\varphi_n(y_k) = 0$ falls $n \neq k$. Damit führt die Bedingung

$$R(y_n) = 0 \tag{7.4}$$

trivial zu

$$a_n = f(y_n). \tag{7.5}$$

Die Koeffizienten a_n sind also durch den Funktionswert der zu approximierenden Funktion am Kollokationspunkt gegeben. Die Approximation ist damit eine stückweise lineare Funktion zwischen den Funktionswerten von $f(x)$.

Als zweites Beispiel diskutieren wir hier eine Fourier-Reihe mit entsprechenden $2N + 1$ Fourier-Basisfunktionen,

$$\varphi_n(x) = \exp\left(iq_n x\right). \tag{7.6}$$

Im Rahmen einer Kollokationsmethode, verlangen wir, dass das Residuum auf $2N + 1$ äquidistanten Punkten verschwindet, $R(y_n) = 0$ mit

$$y_n = nL/(2N + 1). \tag{7.7}$$

Die Bedingung dafür lautet dann

$$\sum_{k=-N}^{N} a_k \exp\left(iq_k y_n\right) = \sum_{k=-N}^{N} a_k \exp\left(i2\pi \frac{kn}{2N+1}\right) = f(y_n). \tag{7.8}$$

Gleichungen (7.7) können nun nach a_k aufgelöst werden. Wir nutzen dazu, dass für äquidistanter Kollokationspunkte die Fourier-Matrix $W_{kn} = \exp(i2\pi kn/(2N+1))$ (bis auf einen Faktor) unitär ist, d.h. ihr Inverses ist durch die Adjungierte gegeben: $\sum_n W_{kn} W^*_{nl} = (2N+1)\delta_{kl}$. Wir können also Gl. (7.7) mit W^*_{nl} multiplizieren und über n summieren. Dies ergibt

$$\sum_{n=-N}^{N} \sum_{k=-N}^{N} a_k \exp\left[i2\pi \frac{(k-l)n}{2N+1}\right] = \sum_{k=-N}^{N} N a_k \delta_{kl} = N a_l \tag{7.9}$$

wobei

$$\sum_{n=-N}^{N} \exp\left[i2\pi \frac{(k-l)n}{2N+1}\right] = (2N+1)\delta_{kl} \tag{7.10}$$

genutzt wurde. Damit können die Koeffizienten als

$$a_l = \frac{1}{2N+1} \sum_{n=-N}^{N} f\left(\frac{nL}{2N+1}\right) \exp\left(-i2\pi \frac{ln}{2N+1}\right), \tag{7.11}$$

bestimmt werden. Dies ist die *diskrete Fourier-Transformation* der auf den Kollokationspunkten diskretisierten Funktion $f(y_n)$.

Als einfaches Beispiel zeigen wir hier Approximation der Beispielfunktion $f(x) = \sin(2\pi x)^3 + \cos(6\pi(x^2 - 1/2))$ mit Hilfe der Fourier-Basis und der finiten Elemente. Abbildung 7.2 zeigt diese Approximation für $2N+1 = 5$ und $2N+1 = 11$ Basisfunktionen mit äquidistanten Kollokationspunkten.

Die Abbildung zeigt, dass alle Approximationen, wie von der Kollokationsbedingung verlangt, exakt durch die Kollokationspunkte laufen. Zwischen den Kollokationspunkten *interpolieren* die beiden Ansätze unterschiedlich. Die finiten Elementen führen zu einer linearen Interpolation zwischen den Punkten. Die Fourier-Basis ist komplizierter. Der Kurvenverlauf zwischen den Kollokationspunkten wird *Fourier-Interpolation* genannt.

7.3 Gewichtete Residuen

Wir möchten nun die Kollokationsmethode verallgemeinern. Hierzu führen wir das Konzept der *Testfunktion* ein. Anstelle zu verlangen, dass das Residuum an individuellen Punkten verschwindet, verlangen wir, dass das Skalarprodukt

$$(v, R) = 0 \tag{7.12}$$

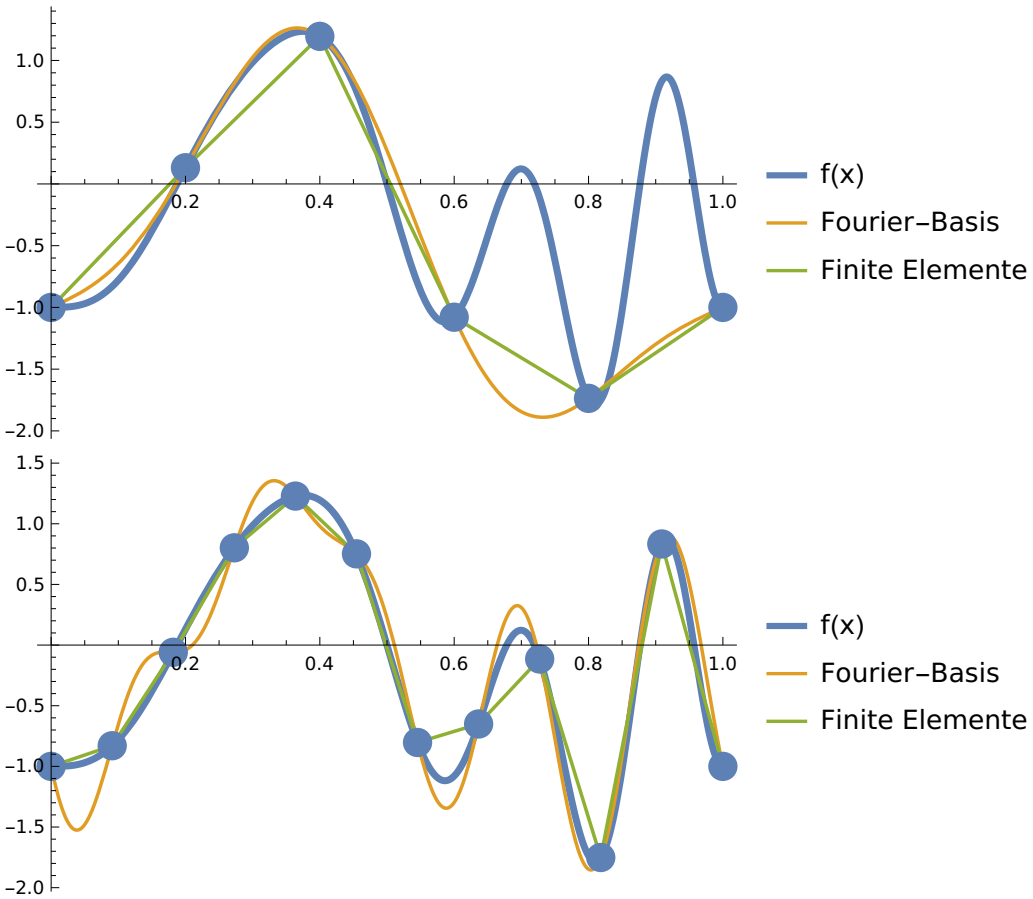

Abbildung 7.1: Approximation der auf dem Interval $[0, 1]$ periodischen Funktion $f(x) = \sin(2\pi x)^3 + \cos(6\pi(x^2 - 1/2))$ mit einer Fourier-Basis und finiten Elementen. Es wurde jeweils 5 (oben) und 11 (unten) Basisfunktionen genutzt. Die Koeffizienten wurden mit der Kollokationsmethode bestimmt. Die runden Punkte zeigen die Kollokationspunkte. Beide Approximationen laufen exakt durch diese Kollokationspunkte. (Der rechte Kollokationspunkt ist auf Grund der Periodizität identisch zum linken.) Die Approximation mit $N = 5$ Basisfunktionen kann die beiden rechten Oszillationen der Zielfunktion $f(x)$ in beiden Fällen nicht abbilden.

mit einer Funktion $v(x)$ verschwindet. Wenn Gl. (7.11) für jede beliebigen Testfunktion $v(x)$ verschwindet, dann ist die "schwache" Formulierung Gl. (7.11) identisch zur starken Formulierung $R(x) = 0$. Gleichung (7.11) heißt "schwache" Formulierung, weil die Bedingung nur im integralen Sinne erfüllt ist. Insbesondere wird in Kapitel 9 gezeigt, dass diese schwache Formulierung zu einer schwachen *Lösung* (engl. "weak solution") führt, die die ursprüngliche (starke) PDGL nicht in jedem Punkt erfüllen kann. Die Bedingung (7.11) wird oft unter dem Begriff der *gewichteten Residuen* subsumiert.

Ein spezieller Satz an Testfunktion führt direkt zur Kollokationsmethode. Wir wählen den Satz von N Testfunktionen

$$v_n(x) = \delta(x - y_n) \tag{7.13}$$

wobei $\delta(x)$ die Diracsche δ-Funktion ist und y_n die Kollokationspunkte. Die Bedingung $(v_n, R) = 0$ für alle $n \in [0, N-1]$ führt direkt zur Kollokationsbedingung $R(y_x) = 0$.

Anmerkung: Die Diracsche δ-Funktion sollte aus Vorlesungen zur Signalverarbeitung bekannt sein. Die wichtigste Eigenschaft dieser Funktion ist die Filtereigenschaft,

$$\int_{-\infty}^{\infty} \mathrm{d}x\, f(x)\delta(x - x_0) = f(x_0), \tag{7.14}$$

also das Integral über das Produkt der δ-Funktion ergibt den Funktionswert, bei dem das Argument der δ-Funktion verschwindet. Hieraus folgen alle weiteren Eingenschaften, z.B.

$$\int \mathrm{d}x\, \delta(x) = \Theta(x), \tag{7.15}$$

wobei $\theta(x)$ die (Heaviside-)Stufenfunktion ist.

7.4 Galerkin-Methode

Die Galerkin-Methode basiert auf der Idee, als Testfunktionen die Basisfunktionen φ_n der Reihenentwicklung zu verwenden. Dies führt zu den N Bedingungen

$$(\varphi_n, R) = 0, \tag{7.16}$$

bzw.

$$(\varphi_n, f_N) = (\varphi_n, f). \tag{7.17}$$

Für einen orthogonalen Satz von Basisfunktionen erhält man direkt

$$a_n = \frac{(\varphi_n, f)}{(\varphi_n, \varphi_n)}. \tag{7.18}$$

Dieser Ansatz wurde bereits in Abschnitt 6.3 diskutiert.

Für einen nicht-orthogonalen Basissatz, z.B. der Basis der finiten Elemente, erhält man ein lineares Gleichungssystem,

$$\sum_{m=0}^{N} (\varphi_n, \varphi_m) a_m = (\varphi_n, f), \tag{7.19}$$

wobei die Matrix $A_{nm} = (\varphi_n, \varphi_m)$ für die finiten Elemente dünnbesetzt ist.

Wir wollen nun wieder zu unserer Beispielfunktion $f(x) = \sin(2\pi x)^3 + \cos(6\pi(x^2 - 1/2))$ zurückkommen. Abbildung 7.2 zeigt die Approximation dieser Funktion mit Fourier und finite Elemente Basissätzen und der Galerkin-Methode. Es gibt keine Kollokationspunkte und die Approximation mit Hilfe der finiten Elemente stimmt auch nicht an den Stützstellen exakt mit der zu approximierenden Funktion überein. Die Funktion wird nur im integralen Sinne approximiert.

Anmerkung: Die Galerkin-Bedingung (siehe auch Gl. (7.15))

$$(\varphi_n, R) = 0, \tag{7.20}$$

bedeutet, dass das Residuum *orthogonal* zu allen Basisfunktionen ist. Anders ausgedrückt, im Residuum können nur noch Beiträge zur Funktion vorkommen, die nicht mit dem gegeben Basissatz abgebildet werden können. Das heißt aber auch, dass wir durch Erweiterung des Basissatzes unsere Lösung systematisch verbessern können.

7.5 Minimales Fehlerquadrat

Ein alternativer Ansatz zur Approximation ist es, das Fehlerquadrat des Residuums, (R, R), zu minimieren. Für eine allgemeine Reihenentwicklung mit N Basisfunktionen erhält man

$$\begin{aligned}(R, R) &= (f, f) + (f_N, f_N) - (f_N, f) - (f, f_N) \\ &= (f, f) + \sum_{n=0}^{N} \sum_{m=0}^{N} a_n^* a_m (\varphi_n, \varphi_m) - \sum_{n=0}^{N} a_n^* (\varphi_n, f) - \sum_{n=0}^{N} a_n (f, \varphi_n).\end{aligned} \tag{7.21}$$

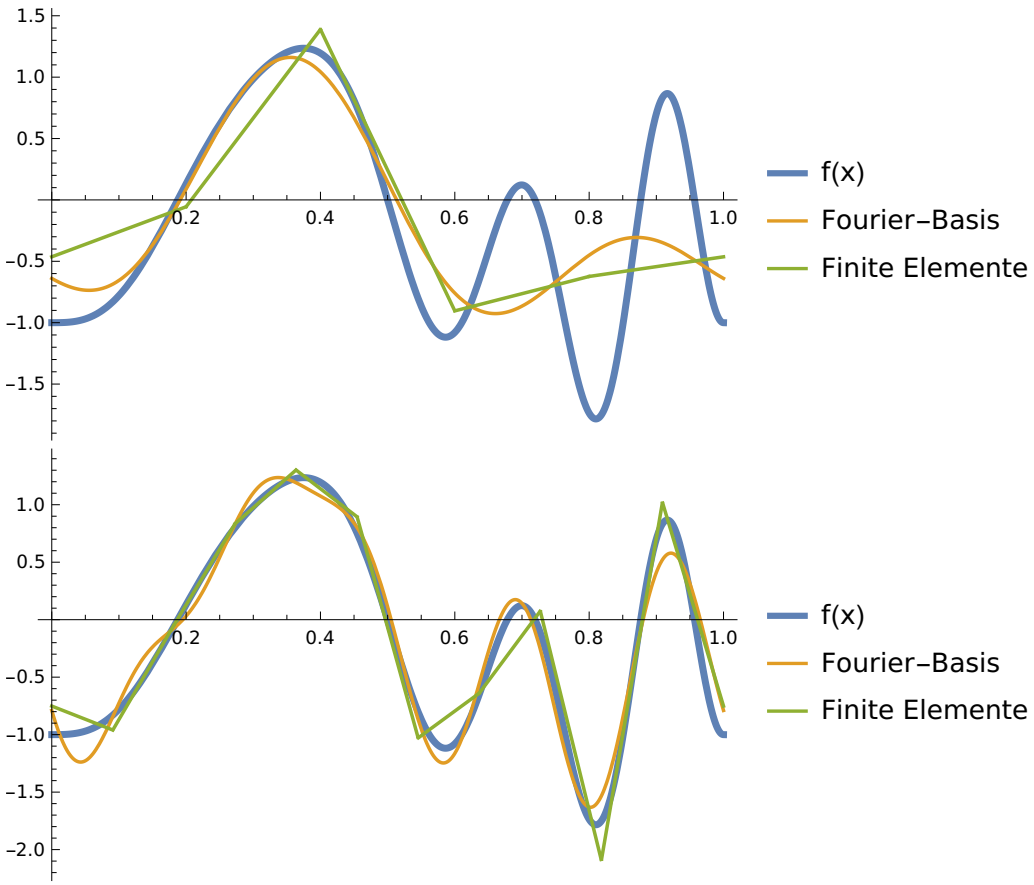

Abbildung 7.2: Approximation der auf dem Interval $[0, 1]$ periodischen Funktion $f(x) = \sin(2\pi x)^3 + \cos(6\pi(x^2 - 1/2))$ mit einer Fourier-Basis und finiten Elementen. Es wurde jeweils 5 (oben) und 11 (unten) Basisfunktionen genutzt. Die Koeffizienten wurde mit Hilfe der Galerkinmethode bestimmt. Die Approximation mit 5 Basisfunktionen kann die beiden rechten Oszillationen der Zielfunktion $f(x)$ in beiden Fällen nicht abbilden.

Diese Fehlerquadrat ist dann minimiert, wenn

$$\frac{\partial(R,R)}{\partial a_k} = \sum_{n=0}^{N} a_n^*(\varphi_n, \varphi_k) - (f, \varphi_k) = 0 \tag{7.22}$$

und

$$\frac{\partial(R,R)}{\partial a_k^*} = \sum_{n=0}^{N} a_n(\varphi_k, \varphi_n) - (\varphi_k, f) = 0. \tag{7.23}$$

Dieser Ausdruck ist identisch zu Gl. (7.18) der Galerkin-Methode.

Kapitel 8

Spektrale Lösungsansätze

Kontext: Wir entwickeln nun die in den vorherigen Kapiteln skizzierten Ideen für die Lösung partieller Differentialgleichungen weiter. In diesem Kapitel beschäftigen wir uns mit der Fourier-Basis. Dies führt zu einem Lösungsverfahren, das üblicherweise unter dem Namen *Integraltransformation* subsumiert wird. Zwar können spektrale Methoden auch für numerische Lösungen eingesetzt werden, hier besprechen wir aber nur die Fourier-Transformation als analytisches Lösungsverfahren. Neben der Anwendung dieser Verfahren, erweitern wir hier die Lösungsansätze auch auf mehrdimensionale Räume.

8.1 Differentialoperatoren

Für die Lösung von Differentialgleichungen nutzen wir nun exakt die gleichen Methoden, die wir im vorhergehenden Kapitel entwickelt haben: Minimierung des Residuums mit Hilfe der Galerkin-Methode. Unser Residuum hat nun die allgemeine Form,

$$R(x, y, z, \ldots; a_0, a_1, \ldots, a_N) = \mathcal{L}u_N(x, y, z, \ldots) - f(x, y, z, \ldots), \tag{8.1}$$

wobei die unbekannte Funktion u_N hier als eine Reihenentwicklung in eine bestimme Basis $\varphi_n(x, y, z)$ dargestellt ist. In der Galerkin-Methode verlangt man

$$(\varphi_n, R) = 0 \tag{8.2}$$

für jedes n.

Wir diskutieren zunächst die Fourier-Basis für periodische Funktionen auf $x \in [0, L]$ in einer Dimension,

$$\varphi_n(x) = \exp(iq_n x) \tag{8.3}$$

mit $q_n = 2\pi n/L$. Der Operator $\mathcal{L}$ kann beliebige Differentialoperationen enthalten, die auf die Basisfunktionen wirken, beispielsweise

$$\frac{d}{dx}\varphi_n(x) = iq_n\varphi_n(x) \tag{8.4}$$

$$\frac{d^2}{dx^2}\varphi_n(x) = -q_n^2\varphi_n(x). \tag{8.5}$$

D.h. die Ableitungen der (Fourier-)Basisfunktionen ergeben die *gleiche* Basisfunktion und einen algebraischen Faktor. Man sagt auch, die Basisfunktionen *diagonalisieren* den Differentialoperator. (Dies wird für die finiten Elemente, die im nächsten Kapitel besprochen werden, anders sein.)

Diese Eigenschaft ist besonders nützlich, weil zumindest für lineare Differentialgleichungen damit das Residuum wieder eine triviale Reihenentwicklung wird und wir auf Grund der Orthogonalität der Basis die Koeffizienten leicht bestimmen können.

8.2 Poisson-Gleichung in einer Dimension

Als Demonstrator für diese Verhalten nutzen wir die (eindimensionale) Poisson-Gleichung,

$$\nabla^2\Phi \equiv \frac{\mathrm{d}^2\,\Phi}{\mathrm{d}x^2} = -\frac{\rho}{\varepsilon}. \tag{8.6}$$

Hier ist ρ eine Ladungsdichte und Φ das elektrostatische Potential. Das Residuum ist daher

$$R(x) = \frac{\mathrm{d}^2\,\Phi}{\mathrm{d}x^2} + \frac{\rho}{\varepsilon}, \tag{8.7}$$

und die Lösung von Gl. (8.6) ist gegeben durch $R(x) = 0$.

Formal schreiben wir nun das Potential als die Reihenentwicklung

$$\Phi(x) \approx \Phi_N(x) = \sum_{n=-N}^{N} a_n\varphi_n(x), \tag{8.8}$$

wobei wir die Summationsgrenzen im folgenden nicht weiter explizit angeben werden. Wir entwickeln auch die rechte Seite der Gl. (8.6) in eine Reihe mit den gleichen Basisfunktionen,

$$\rho_N(x) = \sum_{n=-N}^{N} b_n\varphi_n(x). \tag{8.9}$$

Eingesetzt in Gl. (8.7) erhalten wir

$$R_N(x) = -\sum_n a_n q_n^2 \varphi_n(x) + \frac{1}{\varepsilon} \sum_n b_n \varphi_n(x). \tag{8.10}$$

Wir multiplizieren dies nun von links mit den Basisfunktionen, (φ_k, R_N) (Galerkinmethode) und erhalten auf Grund der Orthogonalität der Basisfunktionen die Gleichungen

$$(\varphi_k, R_N) = -L q_k^2 a_k + L b_k/\varepsilon. \tag{8.11}$$

(Der Faktor L erscheint, weil die Basisfunktionen nicht normalisiert sind.) Die Bedingung $(\varphi_k, R_N) = 0$ führt zu $a_k = b_k/(q_k^2\varepsilon)$. Die approximative Lösung der Poisson-Gleichung ist damit gegeben durch

$$\Phi_N(x) = \sum_n \frac{b_n}{q_n^2 \varepsilon} \varphi_n(x). \tag{8.12}$$

Dies ist die Fourier-Reihe der Lösung.

8.3 Übergang zur Fourier-Transformation

Die Fourier-Basis Gl. (**??**) ist auf einem finiten Gebiet der Länge L periodisch. Wenn wir die Länge L gegen unendlich gehen lassen, bekommen wir eine Formulierung für nicht-periodische Funktionen. Dies führt direkt zur *Fourier-Transformation*.

Wir schreiben die Reihenentwicklung als

$$\Phi_N(x) = \sum_{n=-N}^{N} a_n \varphi_n(x) = \sum_{n=-N}^{N} a_n \exp(i q_n x) = \sum_{n=-N}^{N} \frac{\Delta q}{2\pi} \tilde{\Phi}(q_n) \exp(i q_n x) \tag{8.13}$$

mit $\Delta q = q_{n+1} - q_n = 2\pi/L$ und umskalierten Koeffizienten $\tilde{\Phi}(q_n) = L a_n$. Hier wurde auf der rechten Seite von Gl. (8.13) lediglich der Faktor $1 = L\Delta q/2\pi$ eingefügt. Dies hilft nun, den Limes $L \to \infty$ und $N \to \infty$ zu bilden. In diesem Fall wird $\Delta q \to dq$ und die Summe zum Integral. Man erhält

$$\Phi(x) = \int_{-\infty}^{\infty} \frac{\mathrm{d}q}{2\pi} \tilde{\Phi}(q) \exp(iqx), \tag{8.14}$$

die Fourier-Rück*transformation*.

Die (Hin-)Transformation erhält man über ein ähnliches Argument. Wir wissen nun, dass

$$\tilde{\Phi}(q_n) = La_n = L\frac{(\varphi_n, \Phi_N)}{(\varphi_n, \varphi_n)} = (\varphi_n, \Phi_N) = \int_0^L \mathrm{d}x\, \Phi_N(x) \exp\left(-iq_n x\right). \quad (8.15)$$

Im Grenzfall $L \to \infty$ und $N \to \infty$ wird dies zu

$$\tilde{\Phi}(q) = \int_{-\infty}^{\infty} \mathrm{d}x\, \Phi(x) \exp\left(-iqx\right), \quad (8.16)$$

der Fourier-Transformation. Die Fourier-Transformation ist nützlich, um analytische Lösungen für partielle Differentialgleichungen auf unendlichen Gebieten zu erhalten.

Anmerkung: Eine Tilde $\tilde{f}(q)$ bezeichnet die Fourier-Transformierte einer Funktion $f(x)$. Die Fourier-Transformierte ist eine Funktion des Wellenvektors q. Im Gegensatz dazu erhalten wir bei der Fourier-Reihe abzählbare Koeffizienten a_n. Der Grund hierfür ist die Periodizität des betrachteten Gebiets.

8.4 Poisson-Gleichung in mehreren Dimensionen

Ähnlich wie wir eine approximierte Lösung für eine Differentialgleichung mit Hilfe einer Reihenentwicklung konstruiert haben, können wir nun den Ansatz Gl. (8.14) nutzen, um analytische Lösungen zu erhalten. In diesem Abschnitt wird dies mit Hilfe der Poisson-Gleichung in drei Dimensionen demonstriert.

In drei Dimensionen lautet die Poisson-Gleichung

$$\nabla^2\Phi \equiv \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial z^2} = -\frac{\rho}{\varepsilon}. \quad (8.17)$$

Im Gegensatz zu Gl. (8.6) taucht hier nun die partielle Ableitung ∂ auf, weil $\Phi(x, y, z)$ nun von drei Variablen (den kartesischen Koordinaten) abhängt.

Die Verallgemeinerung der Fourier-Basis und damit auch der Fourier-Transformation auf drei Dimensionen ist trivial. Man erhält eine Basis, in dem man Basisfunktionen in die kartesischen Richtungen (x, y und z) multipliziert. Üblicherweise braucht man nun drei Indices für die Koeffizienten, die jeweils

die Basis in x, y und z bezeichnen. Man erhält als Reihenentwicklung

$$\begin{aligned}\Phi_{NMO}(x,y,z) &= \sum_{n=-N}^{N}\sum_{m=-M}^{M}\sum_{o=-O}^{O} a_{nmo}\varphi_n(x)\varphi_m(y)\varphi_o(z) \\ &\equiv \sum_{n=-N}^{N}\sum_{m=-M}^{M}\sum_{o=-O}^{O} a_{nmo}\varphi_{nmo}(x,y,z)\end{aligned} \tag{8.18}$$

mit (möglicherweise unterschiedlicher) Entwicklungsordnung N, M und O. Der Basissatz ist hier gegeben durch die Menge der Funktionen $\varphi_{nmo}(x,y,z) = \varphi_n(x)\varphi_m(y)\varphi_o(z)$. Orthogonalität dieses Basissatzes geht trivialerweise aus der Orthogonalität der eindimensionalen Basisfunktionen $\varphi_n(x)$ hervor. Die Verallgemeinerung der Fourier-Transformation folgt hieraus direkt. Die Fourier-Rücktransformation schreibt sich als

$$\Phi(x,y,z) = \int_{-\infty}^{\infty} \frac{\mathrm{d}^3 q}{(2\pi)^3}\, \tilde{\Phi}(q_x,q_y,q_z) \exp\left(iq_x x + iq_y y + iq_z z\right), \tag{8.19}$$

wobei die Fouriertransformierte $\tilde{\Phi}$ jetzt natürlich von drei Wellenvektoren q_x, q_y und q_z abhängt. Der Differentialoperator $\mathrm{d}^3 q = \mathrm{d}q_x\,\mathrm{d}q_y\,\mathrm{d}q_z$ ist eine Kurznotation für die dreidimensionale Integration.

Wir können nun Gl. (8.19) in die PDGL Gl. (8.17) einsetzen und erhalten

$$R(\vec{r}) = \int_{-\infty}^{\infty} \frac{\mathrm{d}^3 q}{(2\pi)^3} \left[\left(-q_x^2 - q_y^2 - q_z^2\right)\tilde{\Phi}(\vec{q}) + \frac{\tilde{\rho}(\vec{q})}{\varepsilon}\right] \exp\left(i\vec{q}\cdot\vec{r}\right) = 0 \tag{8.20}$$

mit $\vec{r} = (x,y,z)$ und $\vec{q} = (q_x,q_y,q_z)$. Diese Gleichung muss für jedes x,y,z erfüllt sein und damit muss das Argument der Integration verschwinden, also

$$-q^2\tilde{\Phi}(\vec{q}) + \frac{\tilde{\rho}(\vec{q})}{\varepsilon} = 0. \tag{8.21}$$

Anmerkung: Ein alternatives Argument erhält man, wenn man die Fourier-Transformation von $R(x,y,z)$ hinschreibt:

$$R(q_x',q_y',q_z') = \int \mathrm{d}^3 r\, R(\vec{r}) \exp\left(-iq_x x - iq_y y - iq_z z\right). \tag{8.22}$$

Diese enthält Terme der Form

$$\int_{-\infty}^{\infty} \mathrm{d}x\, \exp\left(i(q_x - q_x')x\right) = 2\pi\delta(q_x - q_x'), \tag{8.23}$$

welche Ausdruck der Orthogonalität der Basisfunktionen sind. Da die Basisfunktionen nun mit einem Kontinuierlichen q_x (anstelle eines diskreten n) "parameterisiert" sind, erhält man eine Diracsche δ-Funktion anstelle des Kronecker-δ in der Orthogonalitätsrelation.

Gleichung (8.21) kann einfach analytisch gelöst werden. Man erhält

$$\tilde{\Phi}(\vec{q}) = \frac{\tilde{\rho}(\vec{q})}{\varepsilon q^2} \tag{8.24}$$

mit $q = |\vec{q}|$. Dies ist äquivalent zur Lösung Gl. (8.12) für die Poisson-Gleichung auf einem periodischen Gebiet. Die Schwierigkeit besteht nun da drin, für ein gegebenes $\rho(x, y, z)$ die Hin- und Rücktransformation auszuwerten.

Beispiel: Als Beispiel betrachten wir nun die Lösung für eine Punktladung Q am Ursprung,

$$\rho(x, y, z) = Q\delta(x)\delta(y)\delta(z). \tag{8.25}$$

Die Fourier-Transformierte der Ladungsdichte ρ erhält man aus Gl. (8.16),

$$\tilde{\rho}(q_x, q_y, q_z) = Q. \tag{8.26}$$

D.h. die Fourier-Transformierte des elektrostatischen Potentials ist gegeben durch (siehe Gl. (8.24))

$$\tilde{\Phi}(\vec{q}) = \frac{Q}{\varepsilon q^2}, \tag{8.27}$$

und damit lautet die Darstellung im Realraum

$$\begin{aligned} \Phi(\vec{r}) =& \int_{-\infty}^{\infty} \frac{\mathrm{d}^3 q}{(2\pi)^3} \frac{Q}{\varepsilon q^2} \exp\left(i\vec{q}\cdot\vec{r}\right) \\ =& \frac{Q}{(2\pi)^3\varepsilon} \int_0^{\infty} \mathrm{d}q \int_0^{2\pi} \mathrm{d}\phi \int_{-1}^{1} \mathrm{d}(\cos\theta)\, \exp\left(iqr\cos\theta\right) \end{aligned} \tag{8.28}$$

wobei $\mathrm{d}^3 q = q^2\, \mathrm{d}q\, \mathrm{d}\phi\, \mathrm{d}(\cos\theta)$ mit Azimutwinkel ϕ und Elevationswinkel θ, genutzt wurde (siehe auch Abb. 8.1). Wir verlangen hier (ohne Beschränkung der Allgemeinheit), dass $\vec{r}$ in Richtung Zenit zeigt.

Man erhält

$$\begin{aligned} \Phi(\vec{r}) =& \frac{Q}{(2\pi)^2\varepsilon} \int_0^{\infty} \mathrm{d}q \int_{-1}^{1} \mathrm{d}(\cos\theta)\, \exp\left(iqr\cos\theta\right) \\ =& \frac{Q}{(2\pi)^2\varepsilon} \int_0^{\infty} \mathrm{d}q\, \frac{\exp(iqr) - \exp(-iqr)}{iqr} \\ =& \frac{Q}{(2\pi)^2\varepsilon} \int_{-\infty}^{\infty} \mathrm{d}q\, \frac{\sin qr}{qr} \\ =& \frac{Q}{4\pi\varepsilon r}, \end{aligned} \tag{8.29}$$

wobei $\int \mathrm{d}x\, \sin x/x = \pi$ genutzt wurde. Dies ist die bekannte Lösung für das elektrostatische Potential einer Punktladung. Man nennt sie auch

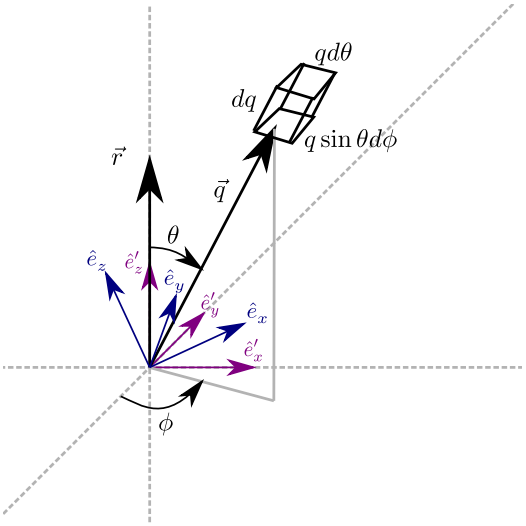

Abbildung 8.1: Volumenelement für die Integration in Kugelkoordination

die Fundamentallösung oder *Greensche Funktion* der (dreidimensionalen) Poisson-Gleichung.

Kapitel 9

Finite Elemente in einer Dimension

Kontext: In diesem Kapitel werden einige Eigenschaften der finiten Elemente anhand eindimensionaler Beispiele diskutiert. Insbesondere wird gezeigt, wie mit linearen Elementen PDGLs zweiter Ordnung diskretisiert werden und wie Randbedingungen in den finiten Elementen eingebaut werden können.

9.1 Differenzierbarkeit der Basisfunktionen

Der einfachste Fall einer Basis der finiten Elemente in einer Dimension wurde bereits in den vorherigen Kapiteln eingeführt. Die Basisfunktionen sind "Zelt-" oder "Hutfunktionen" die jeweils auf einem Knoten maximal (= 1) sind und dann zu den beiden benachbarten Knoten linear abfallen. Im Gegensatz zu der Fourier-Basis ist diese Basis nicht (im klassischen Sinne noch nicht einmal) differenzierbar. Dies heißt eigentlich, dass wir diese Basis nicht als Reihenentwicklung für die Ansatzfunktion zur Lösung einer PDGL zweiter (bzw. sogar erster) Ordnung verwenden können, in denen Ableitungen auftauchen. Wir zeigen hier nun dass im Rahmen einer *schwachen* Formulierung eine schwache Lösung erhalten werden kann, die diese Anforderungen an die Differenzierbarkeit nicht erfüllen muss. Damit können diese Basisfunktionen doch für PDGLs zweiter Ordnung verwendet werden.

Wir betrachten als Beispiel hier weiterhin die eindimensionale Poisson-Gleichung mit dem Residuum

$$R(x) = \frac{\mathrm{d}^2 \Phi}{\mathrm{d}x^2} + \frac{\rho(x)}{\varepsilon}. \tag{9.1}$$

In der Methode der gewichteten Residuen wird verlangt, dass das Skalarprodukt des Residuums mit einer Testfunktion $v(x)$ verschwindet, $(v, R) = 0$. In dieser integralen Formulierung kann man nun die Regeln der partiellen Integration nutzen, um eine Ableitung auf die Testfunktion zu übertragen. Man bekommt

$$(v(x), R(x)) = \int_a^b \mathrm{d}x\, v(x) \left(\frac{\mathrm{d}^2 \Phi}{\mathrm{d}x^2} + \frac{\rho(x)}{\varepsilon} \right) \tag{9.2}$$

$$= \int_a^b \mathrm{d}x \left[\frac{\mathrm{d}}{\mathrm{d}x} \left(v(x) \frac{\mathrm{d}\Phi}{\mathrm{d}x} \right) - \frac{\mathrm{d}v}{\mathrm{d}x} \frac{\mathrm{d}\Phi}{\mathrm{d}x} \right] + \int_a^b \mathrm{d}x\, v(x) \frac{\rho(x)}{\varepsilon} \tag{9.3}$$

$$= v(x) \frac{\mathrm{d}\Phi}{\mathrm{d}x} \bigg|_a^b - \int_a^b \mathrm{d}x\, \frac{\mathrm{d}v}{\mathrm{d}x} \frac{\mathrm{d}\Phi}{\mathrm{d}x} + \int_a^b \mathrm{d}x\, v(x) \frac{\rho(x)}{\varepsilon}. \tag{9.4}$$

Gleichung (9.4) enthält nun *keine* zweite Ableitung mehr. D.h. sowohl die Testfunktion $v(x)$ als auch die Lösung $\Phi(x)$ müssen für alle x lediglich einmal differenzierbar sein. Wir können daher lineare Basisfunktion nutzen, um eine PDGL zweiter Ordnung zu diskretisieren. Im Folgenden werden wir das Simulationsgebiet als Ω bezeichnen. In dem eindimensionalen Fall der hier diskutiert wird ist $\Omega = [a, b]$.

Anmerkung: Die linearen Basisfunktionen sind im klassischen Sinne noch nicht einmal einfach differenzierbar, weil sich der linksseitige und rechtsseitige Differenzenquotient an den Knicken der Funktion unterscheidet. Es existiert aber die sogenannte schwache Ableitung $f'(x)$ der Funktion $f(x)$, wenn

$$\int_\Omega \mathrm{d}x\, v(x) f'(x) = -\int_\Omega \mathrm{d}x\, v'(x) f(x) \tag{9.5}$$

für beliebige (stark differenzierbare) Testfunktionen $v(x)$. Die schwache Ableitung ist nicht eindeutig; so kann man an den Knicken der Zeltfunktionen beispielsweise 0 (oder jeden beliebigen anderen Wert) als schwache Ableitung annehmen. Dies funktioniert, weil in dem Integral Gl. (9.5) dieser einzelne Punkt keinen Beitrag hat. Genauso wie die schwache Ableitung nicht eindeutig ist, ist die schwache Lösung einer Differentialgleichung nicht eindeutig.

In diesem Lernmaterial werden wir nicht systematisch zwischen starker und schwacher Ableitung unterscheiden sondern implizit annehmen, dass wir mit der schwachen Ableitung operieren. Wir führen oft intuitive Rechenoperationen mit schwachen Ableitungen durch, wie z.B. eine

Stufenfunktion als Ableitung der finiten Zelte zu nehmen und eine Dirac-δ-Funktion als Ableitung der Stufenfunktion. Dieser mathematisch unpräzise Umgang mit Ableitungen ist in den Ingenieurswissenschaften und der Physik verbreitet und führt meist zu richtigen Ergebnissen. Funktionenräume schwach differenzierbarer Funktionen heißen *Sobolev-Räume*. Da man für die schwache Differenzierbarkeit immer ein Skalarprodukt braucht (siehe Gl. (9.5)), sind Sobolev-Räume auch immer Hilbert-Räume.

Wir haben damit unser Verständnis der Lösung einer PDGL erweitert. Während für die starke Lösung $R(x) \equiv 0$, Gl. (9.1), die Funktion $\Phi(x)$ zweimal stark differenzierbar sein muss, muss diese Funktion für die schwache Lösung $(v, R) \equiv 0$, Gl. (9.4), lediglich einmal schwach differenzierbar sein. Gleichung (9.4) muss hier natürlich für alle einmal diffenzierbaren Testfunktionen $v(x)$ erfüllt sein.

Anmerkung: In der Kollokationsmethode wird die "starke" Anforderung der zweimaligen Differenzierbarkeit nicht aufgehoben. Das sieht man z.B. daran, dass der Dirac-δ als Testfunktion für die Kollokationsmethode noch nicht einmal im schwachen Sinne differenzierbar ist. D.h. die Testfunktionen der Kollokationsmethode sind nicht in der Menge der Testfunktionen, für welche $(v, R) \equiv 0$ im Sinne der schwachen Lösung verlangt wird. Dies ist der Grund, warum die gewichteten Residuen und hier spezifisch die Galerkin-Methode eine so weite Verbreitung gefunden hat.

9.2 Galerkin-Methode

Im Rahmen der schwachen Formulierung Gl. (9.4), wird nun die Galerkin-Methode angesetzt. Wir schreiben wieder

$$\Phi(x) \approx \Phi_N(x) = \sum_{n=0}^{N} a_n \varphi_n(x) \tag{9.6}$$

als (endliche) Reihenentwicklung mit unseren finiten Elementen, den Zeltfunktionen $\varphi_n(x)$. Wir betrachten zunächst den Fall, in dem die Testfunktion auf dem Rand des Gebietes Ω, also bei $x = a$ und $x = b$, verschwindet; damit verschwindet der erste Term in Gl. (9.4). Dieser Term verschwindet z.B. bei periodischen Gebieten. (Dieser Term wird aber wieder wichtig, wenn wir über Randbedingungen für nicht-periodische Gebiete sprechen.)

Die Galerkin-Bedingung wird damit zu

$$(\varphi_k, R_N) = -\sum_n a_n \int_\Omega \mathrm{d}x \, \frac{\mathrm{d}\varphi_k}{\mathrm{d}x} \frac{\mathrm{d}\varphi_n}{\mathrm{d}x} + \frac{1}{\varepsilon} \int_\Omega \mathrm{d}x \, \varphi_k(x)\rho(x) = 0 \tag{9.7}$$

mit unbekanntem a_n. Die Integrale in Gl. (9.7) sind lediglich Zahlen; die Gleichung ist damit ein System gekoppelter linearer Gleichungen. Mit

$$K_{kn} = \int_\Omega \mathrm{d}x \, \frac{\mathrm{d}\varphi_k}{\mathrm{d}x} \frac{\mathrm{d}\varphi_n}{\mathrm{d}x} \quad \text{und} \quad f_k = \frac{1}{\varepsilon} \int_\Omega \mathrm{d}x \, \varphi_k(x)\rho(x) \tag{9.8}$$

erhält man damit

$$\sum_n K_{kn} a_n = f_k, \tag{9.9}$$

oder in dyadischer (Matrix-Vektor) Schreibweise

$$\underline{K} \cdot \vec{a} = \vec{f}. \tag{9.10}$$

Damit ist die Differentialgleichung in eine algebraische Gleichung überführt, die numerisch gelöst werden kann. Die Matrix $\underline{K}$ heißt Systemmatrix oder Steifigkeitsmatrix. (Letzter Begriff kommt aus der Anwendung der finiten Elemente im Rahmen der Strukturmechanik.) Der Term $\vec{f}$ wird oft als "rechte Seite" (engl. "right hand side", oft als "rhs" abgekürzt) oder Lastvektor (wiederum aus der Strukturmechanik) bezeichnet.

Anmerkung: In der Strukturmechanik, die sich mit der Verformung von Festkörpern beschäftigt, ist $\underline{K}$ so etwas wie eine Federkonstante, $\vec{f}$ eine Kraft und $\vec{a}$ die *Verschiebungen* (engl. "displacements") der Knoten, also der Abstand zum unverformten Zustand. Damit ist Gl. (9.10) so etwas wie die Verallgemeinerung eines Federgesetzes, des Hookschen Gesetzes. Aus diesem Grund werden traditionell die Symbole $\underline{K}$ und $\vec{f}$ genutzt. Die Strukturmechanik ist komplizierter als die meisten anderen numerischen Anwendungsfälle, weil sich das Gebiet, auf dem die konstituierenden Gleichungen diskretisiert wurden, durch die Verformung selbst verändert. Dies führt automatisch zu nicht-linearen Gleichungen, sogenannten geometrischen Nichtlinearitäten. In diesem Fall hängt dann $\underline{K}$ selbst von $\vec{a}$ ab.

Wir können nun die Elemente der Matrix $\underline{K}$ direkt ausrechnen. Für die Basis der finiten Elemente folgt

$$\frac{\mathrm{d}\varphi_n(x)}{\mathrm{d}x} = \begin{cases} \frac{1}{x_n - x_{n-1}} & \text{für } x \in [x_{n-1}, x_n] \\ -\frac{1}{x_{n+1} - x_n} & \text{für } x \in [x_n, x_{n+1}] \\ 0 & \text{sonst} \end{cases} \tag{9.11}$$

und damit

$$K_{nn} = \frac{1}{x_n - x_{n-1}} + \frac{1}{x_{n+1} - x_n} \tag{9.12}$$

$$K_{n,n+1} = -\frac{1}{x_{n+1} - x_n} \tag{9.13}$$

und $K_{kn} = 0$ für $|n-k| > 1$. Die Matrix $\underline{K}$ ist also dünnbesetzt, symmetrisch und nahezu tridiagonal.

Beispiel: Für äquidistante Knoten mit Abstand $\Delta x = x_{n+1} - x_n$ auf einer periodischen Gebiet erhält man für beispielsweise 6 Knoten ($N = 5$):

$$\underline{K} = \frac{1}{\Delta x} \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & -1 & 2 \end{pmatrix} \tag{9.14}$$

Man beachte, dass die -1 in der rechten oberen und linken unteren Ecke (K_{0N} und K_{N0}) auf Grund der Periodizität erscheinen. Die Matrix ist daher nicht rein tridiagonal. Die rechte Seite f_k hängt von der spezifischen Wahl des Quellterms, also der Ladungsdichte $\rho(x)$ ab und sieht folgendermaßen aus:

$$\vec{f} = \begin{pmatrix} (\varphi_0(x), \rho(x))/\varepsilon \\ (\varphi_1(x), \rho(x))/\varepsilon \\ (\varphi_2(x), \rho(x))/\varepsilon \\ (\varphi_3(x), \rho(x))/\varepsilon \\ (\varphi_4(x), \rho(x))/\varepsilon \\ (\varphi_5(x), \rho(x))/\varepsilon \end{pmatrix} \tag{9.15}$$

Im Verlauf dieses Lernmaterials ist bislang vollkommen unter den Tisch gefallen, dass man für die Lösung von DGLs auch immer *Randbedingungen* braucht. Selbst dieser periodische Fall ist so nicht vollständig. In der Fourier-Darstellung repräsentiert $n = 0$ (mit Wellenvektor $q_0 = 0$) den Mittelwert der Fourier-Reihe. Die Lösung der Poisson-Gleichung spezifiziert diesen Mittelwert nicht und dieser muss damit als zusätzliche Bedingung angegeben werden.

Im Rahmen der Diskretisierung mit Hilfe der finite Elemente äußert sich dies darin, dass die Systemmatrix $\underline{K}$, so wie sie z.B. in Gl. (9.14)

aufgeschrieben ist, nicht *regulär* (auch *invertierbar* oder *nichtsingulär*) ist. Dies sieht man z.B. daran, dass die Determinante von $\underline{K}$ verschwindet. Anders ausgedrückt, die linearen Gleichungen sind nicht linear unabhängig. In dem hier diskutierten Fall ist der *Rang* der Matrix, also die Anzahl der linear unabhängigen Zeilen, genau eins kleiner als deren Dimension. Wir können daher eine dieser Gleichungen (bzw. Zeilen) entfernen und dafür die entsprechende Mittelwertbedingung einführen. Für den periodischen Fall lautet diese

$$\int_\Omega \mathrm{d}x\, \Phi_N(x) = \sum_n a_n \int_\Omega \mathrm{d}x\, \varphi_n(x) = \sum_n a_n \Delta x = 0. \tag{9.16}$$

Die reguläre Systemmatrix sieht dann folgendermaßen aus,

$$\underline{K} = \frac{1}{\Delta x} \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{9.17}$$

wobei nun auch $f_N = 0$ gelten muss. Die rechte Seite wird also zu

$$\vec{f} = \begin{pmatrix} (\varphi_0(x), \rho(x))/\varepsilon \\ (\varphi_1(x), \rho(x))/\varepsilon \\ (\varphi_2(x), \rho(x))/\varepsilon \\ (\varphi_3(x), \rho(x))/\varepsilon \\ (\varphi_4(x), \rho(x))/\varepsilon \\ 0 \end{pmatrix}. \tag{9.18}$$

Die letzte Zeile entspricht der Mittelwertbedingung Gl. (9.16), man hätte aber jede beliebige Zeile durch diese Bedingung ersetzen können.

9.3 Randbedingungen

Die Mittelwertbedingung des vorhergehenden Beispiels ist ein Spezialfall einer Randbedingung. Meistens werden Probleme auf endlichen Gebieten Ω behandelt, in denen entweder der Funktionswert $\Phi(x)$ oder die Ableitung $\mathrm{d}\Phi/\,\mathrm{d}x$ auf dem Rand $\partial\Omega$ des Gebiets vorgegeben sind. (In unserem eindimensionalen

Fall, $\partial\Omega = \{a, b\}$.) Der erste Fall nennt sich eine Dirichlet-Randbedingung, der zweite Fall heißt Neumann-Randbedingung. Für Differentialgleichungen höherer Ordnung als zwei könnten natürlich auch noch höhere Ableitungen auf dem Rand auftreten. Auch sind Kombinationen aus Dirichlet- und Neumann-Randbedingungen möglich. Wir werden hier lediglich reine Dirichlet- und reine Neumann-Randbedingungen diskutieren.

9.3.1 Dirichlet-Randbedingungen

Im dem hier diskutierten eindimensionalen Fall lauten die Dirichlet-Randbedingungen

$$\Phi(a) \approx \Phi_N(a) = \Phi_a \quad \text{und} \quad \Phi(b) \approx \Phi_N(b) = \Phi_b. \tag{9.19}$$

Diese Bedingungen führen direkt zu $a_0 = \Phi_a$ und $a_N = \Phi_b$. D.h. die Dirichlet-Bedingungen fixieren direkt die entsprechenden Koeffizienten der Reihenentwicklung. Man beachte, dass hier implizit die Bedingungen $(\varphi_0, R) = 0$ und $(\varphi_N, R) = 0$ aus dem Gleichungssystem herausgenommen wurden. Die Basisfunktionen $\varphi_0(x)$ und $\varphi_N(x)$ tauchen aber selbstverständlich noch in der Reihenentwicklung $\Phi_N(x)$ auf.

Beispiel: In unserem Beispiel wird die Systemmatrix mit Dirichlet-Randbedingungen dann zu

$$\underline{K} = \frac{1}{\Delta x} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{9.20}$$

und $f_0 = \Phi_a/\Delta x$ und $f_N = \Phi_b/\Delta x$:

$$\vec{f} = \begin{pmatrix} \Phi_a/\Delta x \\ (\varphi_1(x), \rho(x))/\varepsilon \\ (\varphi_2(x), \rho(x))/\varepsilon \\ (\varphi_3(x), \rho(x))/\varepsilon \\ (\varphi_4(x), \rho(x))/\varepsilon \\ \Phi_b/\Delta x \end{pmatrix} \tag{9.21}$$

Diese Matrix $\underline{K}$ ist regulär.

Das Δx taucht auf der rechten Seite f_0 und f_N nur auf, weil es als Vorfaktor in der Systemmatrix steht. In einer Implementierung würde man Δx komplett auf die rechte Seite schieben und es verschwindet damit komplett aus der ersten und letzten Zeile.

9.3.2 Neumann-Randbedingungen

Die Neumann-Randbedingungen lauten

$$\left.\frac{\mathrm{d}\Phi}{\mathrm{d}x}\right|_a \approx \left.\frac{\mathrm{d}\Phi_N}{\mathrm{d}x}\right|_a = \Phi'_a \quad \text{und} \quad \left.\frac{\mathrm{d}\Phi}{\mathrm{d}x}\right|_b \approx \left.\frac{\mathrm{d}\Phi_N}{\mathrm{d}x}\right|_b = \Phi'_b. \tag{9.22}$$

Um diese Bedingungen in unsere algebraische Gleichung aufzunehmen, dürfen wir nun nicht mehr den ersten Term aus Gl. (9.4) vernachlässigen. Für die Dirichlet-Randbedingungen spielt dieser Term keine Rolle, da die Basisfunktionen $\varphi_1(x)$ bis $\varphi_{N-1}(x)$ auf dem Rand $x = a, b$ verschwinden. Die Basisfunktionen $\varphi_0(x)$ und $\varphi_N(x)$ tun dies nicht, tauchen aber im Dirichlet-Fall auch nicht mehr in der Menge unserer Testfunktionen auf.

Wir müssen nun aber bestimmen, wie wir die Basisfunktionen $\varphi_0(x)$ und $\varphi_N(x)$ interpretieren, die nun über den Rand des Gebiets hinausragen. Eine natürliche Interpretation ist es, nur die Hälfte des "Zeltes" zu betrachten, die in dem Gebiet verbleiben.

Der Galerkin-Ansatz führt damit zu

$$(\varphi_k, R_N) = \varphi_k(x)\left.\frac{\mathrm{d}\Phi}{\mathrm{d}x}\right|_a^b - \sum_n a_n \int_\Omega \mathrm{d}x \, \frac{\mathrm{d}\varphi_k}{\mathrm{d}x}\frac{\mathrm{d}\varphi_n}{\mathrm{d}x} + \frac{1}{\varepsilon}(\varphi_k(x), \rho(x)) = 0. \tag{9.23}$$

Der zusätzliche Term auf der linken Seite spielt nur bei $k = 0$ und $k = N$ eine Rolle. Man erhält

$$(\varphi_0, R_N) = -\Phi'_a - \sum_n K_{0n} a_n + \frac{1}{\varepsilon}(\varphi_0(x), \rho(x)) \tag{9.24}$$

mit

$$K_{00} = \frac{1}{x_1 - x_0} \quad \text{und} \quad K_{01} = -\frac{1}{x_1 - x_0} \tag{9.25}$$

sowie alle weiteren $K_{0n} = 0$. Wir können dieses wieder (siehe Gl. (9.9)) mit

$$f_0 = \frac{1}{\varepsilon}(\varphi_0(x), \rho(x)) - \Phi'_a \tag{9.26}$$

schreiben. Ein entsprechender Satz von Gleichungen gilt für den rechten Rand mit $k = N$.

Beispiel: In unserem Beispiel wird die Sytemmatrix für zwei Neumann-Randbedingungen dann zu

$$\underline{K} = \frac{1}{\Delta x} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}. \tag{9.27}$$

Man beachte, dass diese Systemmatrix sich leicht von der für Dirichlet-Randbedingungen, Gl. (9.27), unterscheidet. Diese Matrix ist nicht regulär und damit ist das Problem mit zwei Neumann-Randbedingungen unbestimmt. Der Grund hierfür ist der gleiche wie im periodischen Fall: Die Neumann-Randbedingungen legen den Absolutwert (Mittelwert) des Potentials Φ nicht fest. Man braucht also entweder eine Dirichlet-Randbedingung (links oder rechts) oder wieder die Fixierung des Mittelwertes.

Die rechte Seite wird zu:

$$\vec{f} = \begin{pmatrix} (\varphi_0(x), \rho(x))/\varepsilon - \Phi'_a \\ (\varphi_1(x), \rho(x))/\varepsilon \\ (\varphi_2(x), \rho(x))/\varepsilon \\ (\varphi_3(x), \rho(x))/\varepsilon \\ (\varphi_4(x), \rho(x))/\varepsilon \\ (\varphi_5(x), \rho(x))/\varepsilon + \Phi'_b \end{pmatrix}. \tag{9.28}$$

Zu beachten ist, dass für die Auswertung von (φ_0, ρ) und (φ_5, ρ) nur über die Hälfte des entsprechenden Zeltes integriert werden muss.

9.4 Formfunktionen

Die Methode der finiten Elemente wird oft nicht mit Hilfe der Basisfunktionen sondern mit Hilfe der sogenannten *Formfunktionen* (engl. "shape functions") formuliert. Der Grund hierfür ist, dass die Formfunktionen einen intuitiveren Zugang zu der Interpolationsvorschrift, die hinter den finiten Elementen steckt, liefert und damit eine einfachere Erweiterung auf mehrere Dimensionen ermöglicht. In dem eindimensionalen Fall wirken die Formfunktionen komplizierter als der oben skizzierte Ansatz, für die Formulierung der Methode

der finiten Elemente in mehrere Dimensionen wird dies aber der einfachere Zugang sein.

Wir müssen zunächst darüber reden, was ein *Element* ist. Die Zeltfunktionen führen zu einer linearen Interpolation zwischen den Knoten. Die Gebiete zwischen den Knoten heißen *Elemente*. In einer Dimension sind dies eindimensionale Intervalle, in höheren Dimensionen können die Elemente komplexe Formen annehmen. Eine lineare Basisfunktion, wie sie hier eingeführt wurde, ist auf den Knoten zentriert und auf zwei Elementen ungleich Null.

Anstelle die Interpolation im Sinne der Basisfunktionen zu definieren, können wir auch verlangen, dass innerhalb eines Elements die Funktionswerte auf den Knoten in gegebener Form, hier zunächst linear, interpoliert werden. Für das n-te Element zwischen den Knoten n und $n+1$, also $x \in \Omega^{(n)} = [x_n, x_{n+1}]$, tragen nur die Basisfunktionen φ_n und φ_{n+1} bei, da alle weiteren Basisfunktionen auf diesem Interval verschwinden. Hier bezeichnet $\Omega^{(n)}$ das Gebiet des n-ten Elements. Damit hat die Funktion $\Phi_N(x)$ in diesem Element den Verlauf

$$\begin{aligned}\phi^{(n)}(x) =& a_n\varphi_n(x) + a_{n+1}\varphi_{n+1}(x)\\ =& a_n\frac{x_{n+1}-x}{x_{n+1}-x_n} + a_{n+1}\frac{x-x_n}{x_{n+1}-x_n}\\ =& a_n N_0^{(n)}(\xi^{(n)}(x)) + a_{n+1}N_1^{(n)}(\xi^{(n)}(x)),\end{aligned} \tag{9.29}$$

mit $\xi^{(n)}(x) = (x-x_n)/(x_{n+1}-x_n)$ und $x \in \Omega^{(n)}$. Hier und im Folgenden bezeichnen hochgestellte Indices $x^{(n)}$ Elemente und tiefgestellte Indices x_n Knoten. Die Funktionen

$$N_0^{(n)}(\xi) = 1-\xi \quad \text{und} \quad N_1^{(n)}(\xi) = \xi \tag{9.30}$$

mit $\xi \in [0,1]$ heißen *Formfunktionen* (auch *Ansatzfunktionen*) und $\xi^{(n)}(x)$ ist eine Reskalierungsfunktion, die am linken Rand des n-ten Elements $\xi^{(n)} = 0$ und am rechten Rand des Elements $\xi^{(n)} = 1$ wird. Hiermit wird die Größe des Elements von der Interpolationsvorschrift (der "Form") entkoppelt.

Anmerkung: Man beachte, dass es genau ein Element weniger als Knoten gibt, die wir bislang auch mit dem Index n bezeichnet haben. In dem eindimensionalen Fall ist der Zusammenhang zwischen globalen Knotenindice und Elementindice trivial, in höheren Dimensionen wird die Buchhaltung der Indices kompliziert. Die Kombination aus lokalem Index des Knoten innerhalb eines Elements (hier 0 für den linken und 1 für den rechten Knoten) und Elementindex ergibt den globalen Knotenindex (hier n).

Die Interpolationsvorschrift hängt vom Index n des Elements ab, da die Elemente unterschiedliche Größen haben können. Es gibt für jeden Knoten des Elements eine Formfunktion, die hier mit links, $N_0^{(n)}$, und rechts, $N_1^{(n)}$ bezeichnet sind. Die Menge der Formfunktionen eines Elements bestimmen den *Elementtyp*. Gleichungen (9.30) definieren ein lineares Element. Im Prinzip können die Elementtypen für jedes Element unterschiedlich sein, für den Basissatz den wir bislang verwendet haben, sind allerdings die linke und rechte Formfunktion (und damit der Elementyp) für alle Elemente identisch.

Anmerkung: Die Basisfunktionen $\varphi_n(x)$ sind global definiert, d.h. sie leben auf dem gesamten Simulationsgebiet Ω (verschwinden aber in Abschnitten daraus). Die Formfunktion $N_i^{(n)}(x)$ ist nur auf dem einzelnen Element n definiert, d.h. sie leben auf $\Omega^{(n)}$. Hier bezeichnet i den Knoten innerhalb des Elements. Wir können aber natürlich die Basisfunktionen mit Hilfe der Formfunktionen ausdrücken, indem man die Funktionen sammelt, welche den entsprechenden Entwicklungskoeffizienten a_n als Vorfaktor tragen. Man erhält in dem eindimensionalen Fall

$$\varphi_n(x) = N_0^{(n)}(x) + N_1^{(n-1)}(x) \tag{9.31}$$

mit der kompakten Schreibweise $N_I^{(n)}(x) = N_I^{(n)}(\xi^{(n)}(x))$. Gleichung (9.31) ist so zu interpretieren, dass die Formfunktionen verschwinden wenn das Argument nicht im Element liegt, also für $x \notin \Omega^{(n)}$. In zwei oder drei Dimensionen ist es üblicherweise einfacher mit den Formfunktionen als mit den Basisfunktionen zu arbeiten. Umgekehrt sind die Formfunktionen letztendlich der Teil der Basisfunktionen, der auf den Elementen lebt, siehe auch Gl. (9.29).

Formfunktionen sind nützlich, weil man mit ihrer Hilfe die approximierte Lösung als Summe über Elemente schreiben kann, also im eindimensionalen Fall

$$\Phi_N(x) = \sum_{n=0}^{N-1} \phi^{(n)}(x) = \sum_{n=0}^{N-1} \left(a_n N_0^{(n)}(x) + a_{n+1} N_1^{(n)}(x) \right). \tag{9.32}$$

Für eine allgemeine PDGL, $R = \mathcal{L}u_N - f$, wird die Galerkin-Bedingung zu

$$(\varphi_k, R) = (N_0^{(k)} + N_1^{(k-1)}, R) = 0 \tag{9.33}$$

mit

$$(N_I^{(k)}, R) = \sum_{n=0}^{N-1} \left(a_n (N_I^{(k)}, \mathcal{L} N_0^{(n)}) + a_{n+1} (N_I^{(k)}, \mathcal{L} N_1^{(n)}) \right) - (N_I^{(k)}, f) \quad (9.34)$$

$$= a_k (N_I^{(k)}, \mathcal{L} N_0^{(k)}) + a_{k+1} (N_I^{(k)}, \mathcal{L} N_1^{(k)}) - (N_I^{(k)}, f) \quad (9.35)$$

wobei die Summe in Gl. (9.34) verschwindet, weil die Formfunktionen auf unterschiedlichen Elementen trivialerweise orthogonal sind. Hier und im Folgenden bezeichnen Großbuchstaben I lokale Knotenindices, während kleine Buchstaben i globale Knotenindices bezeichnen.

Motiviert durch diese Gleichung, definieren wir eine *Elementmatrix* (oder *Elementsteifigkeitsmatrix*) für Element n als

$$K_{IJ}^{(n)} = (N_I^{(n)}, \mathcal{L} N_J^{(n)}), \quad (9.36)$$

sowie

$$f_I^{(n)} = (N_I^{(n)}, f). \quad (9.37)$$

In dieser Schreibweise erhalten wir

$$(N_I^{(k)}, R) = \sum_{J=0}^{1} K_{IJ}^{(k)} a_{k+J} - f_I^{(k)} \quad (9.38)$$

wobei J über die Knoten innerhalb des Elements k läuft und damit (in einer Dimension) $k + J$ der globale Knotenindex des entsprechenden linken oder rechten Knotens ist. Die Galerkin-Bedingung Gl. (9.33) wird damit zu

$$\begin{aligned} (\varphi_k, R) =& (N_0^{(k)}, R) + (N_1^{(k-1)}, R) \\ =& \sum_{I=0}^{1} \left(\sum_{J=0}^{1} K_{IJ}^{(k-I)} a_{k-I+J} - f_I^{(k-I)} \right) = 0. \end{aligned} \quad (9.39)$$

Diese Bedingung entspricht einer Zeile der Systemmatrix und der rechten Seite. Zeile k der Systemmatrix, die Knoten k entspricht, hat damit einen Beitrag von Element k (dessen linker Knoten der Knoten k ist) und Element $k - 1$ (dessen rechter Knoten der Knoten k ist). Das Zusammenbauen der Systemmatrix wird im Englischen oft als “assembly” bezeichnet.

Beispiel: Wir formulieren das Beispielproblem (Poisson-Gleichung) hier noch einmal im Rahmen der Formfunktionen. Dies ist ein Prozess, welcher drei Schritte benötigt:

1. *Elementmatrix* und rechte Seite: Da alle Elemente identisch sind, sind auch die einzelnen Elementmatrizen identisch. Zunächst wenden wir wieder den Trick mit der partiellen Integration an, um die Differenzierbarkeitsbedingung zu reduzieren. Man erhält den Ausdruck

$$K_{IJ}^{(n)} = \left(N_I^{(n)}, \frac{\mathrm{d}^2 N_J^{(n)}}{\mathrm{d}x^2} \right) = - \left(\frac{\mathrm{d}N_I^{(n)}}{\mathrm{d}x}, \frac{\mathrm{d}N_J^{(n)}}{\mathrm{d}x} \right), \tag{9.40}$$

der nur noch erste Ableitungen der Formfunktionen enthält. Diese Ableitungen sind Konstanten, da die Formfunktionen linear sind, Gl. (9.30):

$$\frac{\mathrm{d}N_0^{(n)}}{\mathrm{d}x} = -\frac{1}{\Delta x} \tag{9.41}$$

$$\frac{\mathrm{d}N_1^{(n)}}{\mathrm{d}x} = \frac{1}{\Delta x} \tag{9.42}$$

Damit erhält man die Elementsteifigkeitsmatrix

$$\underline{K}^{(n)} = \frac{1}{\Delta x} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{9.43}$$

die für alle Elemente identisch ist. Die rechte Seite für die Elemente wird zu

$$\underline{f}^{(n)} = \begin{pmatrix} (N_0^{(n)}, \rho)/\varepsilon \\ (N_1^{(n)}, \rho)/\varepsilon \end{pmatrix}. \tag{9.44}$$

Diese rechte Seite kann für die unterschiedlichen Elemente anders sein, wenn ρ räumlich variiert.

2. Zusammenbauen der *Systemmatrix* und der rechten Seite (engl. "assembly"): Die Regeln für das Zusammenbauen der Systemmatrix ergeben sich aus der Galerkin-Bedingung, Gl. (9.39). Hierzu muss die 2×2 Elementmatrix auf die 6×6 Systemmatrix aufgespannt und aufsummiert werden: Zeilen und Spalten der Elementmatrix entsprechen Elementknoten und diese müssen jetzt auf die entsprechenden globalen Knoten in der Systemmatrix abgebildet werden.

In unserem eindimensionalen Fall ist dies trivial. Man erhält

$$\underline{K}\Delta x = \underbrace{\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\text{Element } n=0} + \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\text{Element } n=1} + \cdots \tag{9.45}$$

und damit

$$\underline{K} = \frac{1}{\Delta x} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}. \tag{9.46}$$

Der Vorteil der Formulierung mit Formfunktionen ist daher, dass man die Elementmatrix für einen Elementtyp einmal ausrechnen muss, und dann daraus einfach die Systemmatrix zusammenbauen kann. Der Zusammenbau der rechten Seite folgt analog, ist aber einfacher da es sich um einen Vektor handelt.

3. Randbedingungen: Wir haben im Rahmen der Formfunktionen noch nicht über Randbedingungen gesprochen. Hierbei müssen die entsprechenden Zeilen der Systemmatrix und der rechten Seite durch die entsprechende Randbedingung ersetzt werden. Die Matrix $\underline{K}$ aus Gl. (9.46) ist ohne die entsprechenden Randbedingungen nicht regulär.

Kapitel 10

Finite Elemente in mehreren Dimensionen

Kontext: Wir verallgemeinern nun die Ergebnisse des vorhergehenden Kapitels auf mehrere Dimensionen. Dies hat mehrere technische Hürden: Für die partielle Integration müssen wir nun Ergebnisse der Vektoranalysis, insbesondere den Gaussschen Satz bzw. die Greenschen Formeln nutzen. Die Diskretisierung erfolgt in Form von Elementen, üblicherweise Dreiecke oder Tetraeder. Durch diese komplexere Geometrie der Elemente wird eine saubere Buchhaltung der Indices, also die Unterscheidung von globalen Knoten, Elementknoten und Elementen wichtig.

10.1 Differenzierbarkeit

Zur Illustration, wie die Anforderung an die Differenzierbarkeit in höherdimensionalen Problemen reduziert werden kann, und wir damit wieder lineare Basisfunktionen verwenden können, wird hier weiter die Poisson-Gleichung betrachtet.
In D-Dimensionen (mit üblicherweise $D = 2$ oder $D = 3$) führt die Poisson-Gleichung zu dem Residuum

$$R(\vec{r}) = \nabla^2\Phi + \frac{\rho(\vec{r})}{\varepsilon} = \nabla \cdot (\nabla\Phi) + \frac{\rho(\vec{r})}{\varepsilon} \tag{10.1}$$

wobei $\vec{r} = (x, y, z, \ldots)$ nun ein D-dimensionaler Vektor ist, der die Position bezeichnet, und ∇^2 der Laplace-Operator in D-Dimensionen. Das Potential $\Phi(\vec{r})$ hängt nun selbstverständlich auch von der räumlichen Position $\vec{r}$ ab. Die rechte Seite von Gl. (10.1) ist explizit so geschrieben, dass man die

Verknüpfung aus Divergenz und Gradient erkennt, die den Laplace-Operator ergibt.

Wir schreiben nun das gewichtete Residuum auf. Das Skalarprodukt mit einer Testfunktion $v(\vec{r})$ ergibt

$$(v(\vec{r}), R(\vec{r})) = \int_\Omega \mathrm{d}^D r\, v(\vec{r}) \left(\nabla^2 \Phi + \frac{\rho(\vec{r})}{\varepsilon} \right) \tag{10.2}$$

$$= \int_{\partial\Omega} \mathrm{d}^{D-1} r\, v(\vec{r}) \left(\nabla\Phi \cdot \hat{n}(\vec{r}) \right) - \int_\Omega \mathrm{d}^D r\, \nabla v \cdot \nabla\Phi + \int_\Omega \mathrm{d}^D r\, \frac{v(\vec{r})\rho(\vec{r})}{\varepsilon}, \tag{10.3}$$

wobei nun die erste Greensche Formel verwandt wurde, um das Flächen- bzw. Volumenintegral über Ω umzuschreiben und dabei den Gradienten auf die Testfunktion zu überführen. Die Greensche Formel nimmt hier die Rolle ein, die in dem eindimensionalen Fall die partielle Integration hatte. $\hat{n}(\vec{r})$ ist der Normalenvektor, welcher auf dem Rand $\partial\Omega$ der Domäne nach außen zeigt (siehe Abb. 10.1. Der erste Term auf der rechten Seite von Gl. (10.3) wird wieder wichtig werden, wenn wir Neumann-Randbedingungen auf den Rand $\partial\Omega$ der Domäne vorgeben möchten. Im Dirichlet-Fall verschwindet dieser Term.

Anmerkung: Die *Greenschen Formeln* sind ein weiteres wichtiges Ergebnis der Vektoranalysis. Sie gehen aus dem Gausschen Satz hervor. Der Gausssche Satz lautet

$$\int_\Omega \mathrm{d}^D r\, \nabla \cdot \vec{f}(\vec{r}) = \int_{\partial\Omega} \mathrm{d}^{D-1} r\, \vec{f}(\vec{r}) \cdot \hat{n}(\vec{r}) \tag{10.4}$$

wobei $\partial\Omega$ den $D-1$ dimensionalen Rand des D-dimensionalen Integrationsgebiets Ω bezeichnet. Weiterhin ist $\hat{n}(\vec{r})$ der Normalenvektor, der senkrecht auf den Rand steht und aus dem Integrationsgebiet hinaus zeigt (siehe auch Abb. 10.1). Wir wenden den Gausschen Satz jetzt auf ein Vektorfeld $\vec{f}(\vec{r}) = \phi(\vec{r})\vec{v}(\vec{r})$ an, wobei $\phi(\vec{r})$ ein skalares Feld und $\vec{v}(\vec{r})$ wiederum ein Vektorfeld ist. Man erhält

$$\int_\Omega \mathrm{d}^D r\, \nabla \cdot (\phi\vec{v}) = \int_{\partial\Omega} \mathrm{d}^{D-1} r\, (\phi\vec{v}) \cdot \hat{n}. \tag{10.5}$$

Auf Grund der Kettenregel der Ableitung gilt

$$\nabla \cdot (\phi\vec{v}) = (\nabla\phi) \cdot \vec{v} + \phi\, (\nabla \cdot \vec{v})\,. \tag{10.6}$$

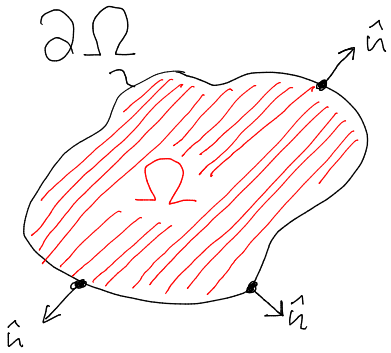

Abbildung 10.1: Der Rand $\partial\Omega$ begrenzt das Integrations- bzw. Simulationsgebiet Ω. Der Normalenvektor $\hat{n}$ ist auf dem Rand $\partial\Omega$ definiert und zeigt dort senkrecht zum Rand nach außen. In dieser Skizze ist der zweidimensionale Fall dargestellt. Im dreidimensionalen Fall ist Ω ein Volumen und $\partial\Omega$ die Fläche, welche das Volumen begrenzt. Auch in diesem Fall kann man einen Normalenvektor $\hat{n}$ auf diese begrenzende Fläche definieren.

(Das Gl. (10.6) gilt sieht man am einfachsten, wenn man sie komponentenweise hinschreibt.) Dies führt zu

$$\int_\Omega \mathrm{d}^D r\ (\nabla\phi\cdot\vec{v}+\phi\nabla\cdot\vec{v}) = \int_{\partial\Omega} \mathrm{d}^{D-1} r\ (\phi\vec{v})\cdot\hat{n}. \tag{10.7}$$

Mit $\vec{v}(\vec{r}) = \nabla\psi$ erhält man die übliche Darstellung der *ersten Greenschen Formel*,

$$\int_\Omega \mathrm{d}^D r\ \left(\nabla\phi\cdot\nabla\psi+\phi(\vec{r})\nabla^2\psi\right) = \int_{\partial\Omega} \mathrm{d}^{D-1} r\ (\phi\nabla\psi)\cdot\hat{n}. \tag{10.8}$$

In Gl. (10.3) ist nun die Anforderung an die Differenzierbarkeit reduziert. Der Laplace-Operator, also die zweite Ableitung, taucht nicht mehr in dieser Gleichung auf. Wir müssen lediglich Gradienten der Testfunktion $v(\vec{r})$ und des Potentials $\Phi(\vec{r})$ berechnen können.

10.2 Gitter

Wir müssen nun geeignete Basisfunktionen wählen, um mit Hilfe der Galerkin-Bedingungen ein lineares Gleichungssystem zu erhalten. Hierbei ist es nützlich,

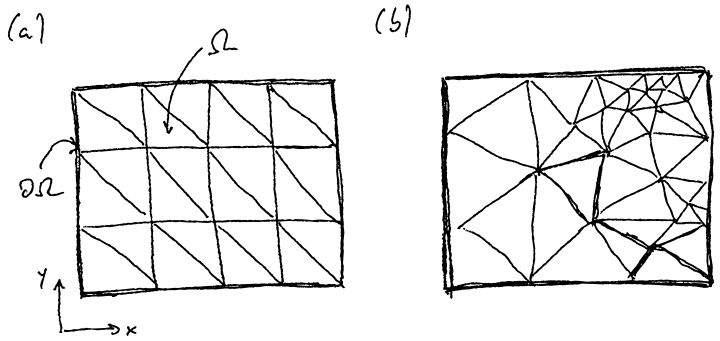

Abbildung 10.2: Triangulierung eines rechteckigen Gebiets Ω in ein (a) strukturiertes Gitter und ein (b) unstrukturiertes Gitter.

die Formulierung im Sinne von Formfunktionen und nicht von Basisfunktionen vorzunehmen. Die weitere Entwicklung der Theorie in diesem Kapitel wird für zweidimensionale Probleme ($D = 2$) vorgenommen.

10.2.1 Triangulierung

Bevor wir über diese Details der Basisfunktionen reden können, müssen wir die Zerlegung des Simulationsgebiet in Elemente diskutieren. In zwei Dimensionen sind diese Elemente üblicherweise (aber nicht zwingend) Dreiecke, d.h. man nimmt eine *Triangulierung* des Gebiets vor. Abbildung 10.2 zeigt eine solche Triangulierung für ein rechteckiges (2-dimensionales) Gebiet. Diese Zerlegung nennt man auch das *Gitter* (oder *Netz*, engl. "grid" oder "mesh"), die einzelnen Dreiecke *Elemente* (engl. "elements") und die Ecken der Dreiecke *Knoten* (engl "nodes"). Der Prozess der Zerlegung des Gebiets wird *Vernetzung* genannt. Wir werden hier ausschließlich mit strukturierten Gittern, wie in Abb. 10.2a dargestellt, arbeiten. Viele Simulationspakete unterstützen auch unstrukturierte Gitter wie in Abb. 10.2b gezeigt.

Auch für diese Art der Zerlegung kann man die Zielfunktion $\Phi(\vec{r})$ durch eine Summe approximieren. Wir schreiben

$$\Phi(x, y) \approx \Phi_N(x, y) = \sum_n a_n \varphi_n(x, y), \tag{10.9}$$

wobei n ein eindeutiger Knotenindex ist. D.h. die Freiheitsgrade bzw. Expansionskoeffizienten a_n leben auf den Knoten (mit den Positionen $\vec{r}_n$) des Simulationsgebietes, und für eine Finite-Elemente Basis wird wieder gelten $\Phi(\vec{r}_n) = a_n$, d.h. a_n ist der Funktionswert auf dem entsprechenden Knoten. Die Formfunktion ist dann eine Vorschrift, wie zwischen den Knoten (also über die Dreiecke) dieser Funktionswert interpoliert wird.

Anmerkung: In drei Dimensionen erfolgt die Zerlegung des Raumes üblicherweise in Tetraeder. Die Vernetzung eines solchen dreidimensionalen Gebiets ist höchst nicht-trivial. Alle kommerziellen Finite-Elemente Pakete haben Vernetzer eingebaut, die diesen Prozess übernehmen oder zumindest unterstützen. Eine freie Softwarelösung für die Vernetzung von komplexen Geometrien ist GMSH (`https://gmsh.info/`).

10.2.2 Strukturierung

Die Nutzung eines strukturierten Gitters vereinfacht die Zuweisung eines Knotenindices n bzw. eines Elementindices (n) zu entsprechenden räumlichen Positionen. Abbildung 10.3 zeigt eine solche strukturierte Zerlegung in $M_x \times M_y$ (mit $M_x = L_x/\Delta x = 3$ und $M_y = L_y/\Delta y = 3$) Kästen mit jeweils zwei Elementen. Das Gitter beinhaltet $N_x \times N_y$ (mit $N_x = M_x + 1 = 4$ und $N_y = M_y + 1 = 4$) Knoten.

In diesem strukturierten Gitter können wir von *Koordinaten* der Knoten auf deren *globalen* Index n_K schließen. Da die Freiheitsgrade auf den Knoten leben, identifiziert der globale Knotenindex n_K später eine Spalte oder Zeile aus der Systemmatrix. Sei i, j die (ganzzahlige) Koordinate des Knotens, dann ist

$$n_\mathrm{K} = i + N_x j \tag{10.10}$$

der entsprechende globale Knotenindex. Hierbei laufen die Knotenkoordinaten bei Null los, d.h. $i \in \{0, 1, \ldots, N_x - 1\}$ und $j \in \{0, 1, \ldots, N_y - 1\}$. Genauso können wir von Elementkoordinaten k, l, m auf den Elementindex n_E schließen,

$$n_\mathrm{E} = k + 2(l + M_x m), \tag{10.11}$$

wobei nun l, m mit $l \in \{0, 1, \ldots, M_x - 1\}$ und $m \in \{0, 1, \ldots, M_y - 1\}$ die Koordinate des Kastens ist und $k \in \{0, 1\}$ das Element innerhalb eines Kastens indiziert. Der Faktor 2 taucht in Gl. (10.11) auf, weil es zwei Elemente pro Kasten gibt.

Anmerkung: Gleichungen (10.10) und (10.11) sind die vermutlich einfachste Abbildung von Koordinaten auf einen linearen, konsekutiven Index. Andere Möglichkeiten, die auch in der Numerik eingesetzt werden, ergeben sich durch die raumfüllenden Kurven, wie z.B. die Hilbert- oder Peano-Kurve. Raumfüllende Kurven haben teilweise vorteilhafte Eigenschaften, wie z.B. dass Koordinaten die nah beieinander liegen auch Indices bekommen, die nah bei einander liegen. Dies führt zu einer kompakteren Struktur

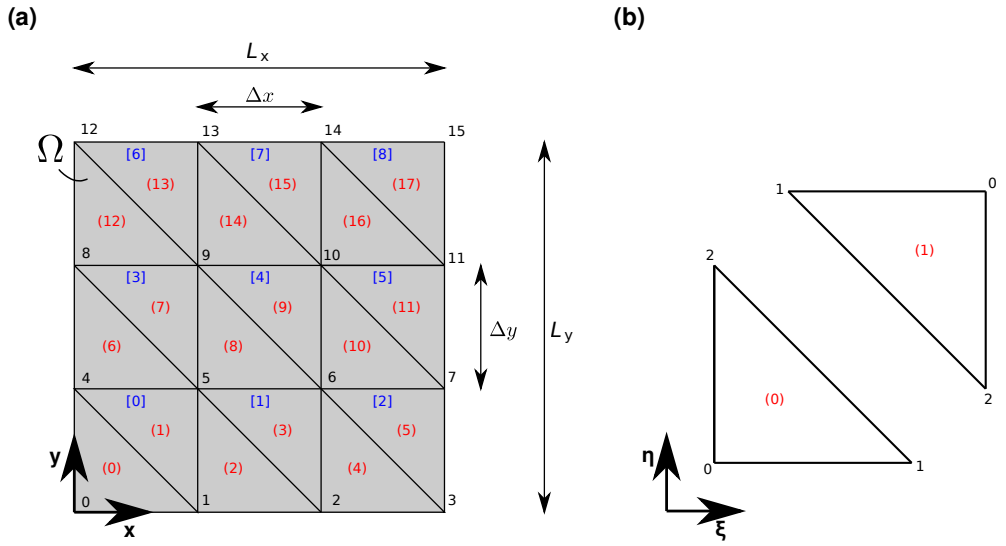

Abbildung 10.3: Zerlegung eines rechteckigen Gebiets in ein strukturiertes Gitter. (a) Die strukturierte Zerlegung erfolgt zuerst in kleinere Kästen, deren eindeutiger *globaler* Index in eckigen Klammern [·] und blau gezeigt ist. Diese Kästen werden dann in zwei Dreiecke, die Elemente, geteilt. Der eindeutige *globale* Elementindex ist in runden Klammern (·) und rot gezeigt. Weiterhin sind die Knoten mit ihrem einem eindeutigen *globalen* Index (schwarz) bezeichnet. (b) Die Kästen werden in zwei Dreicke mit *lokalen* Elementindices (0) und (1) zerlegt. Innerhalb eines Elements, werden die Knoten mit einem entsprechenden *lokalen* Knotenindex identifiziert.

der dünnbesetzten Systemmatrix und kann Vorteile in der Laufzeit der Algorithmen mit sich bringen. Der Grund für solche Laufzeitvorteile ist eng mit der Hardware verknüpft, z.B. wie die Hardware Speicherzugriff organisiert und *Caches* nutzt. Eine Optimierung von Algorithmen für spezifische Hardwarearchitekturen ist höchst nicht-trivial und benötigt detailliertes Wissen über die Rechnerarchitektur.

10.3 Formfunktionen

Unsere Formfunktionen leben auf den einzelnen Dreiecken der Triangulierung und müssen auf den jeweiligen Knoten entweder 1 sein oder verschwinden. Wir drücken die Formfunktionen hier mit Hilfe der skalierten Koordinaten $\xi = (x - x_0)/\Delta x$ und $\eta = (y - y_0)/\Delta y$ aus, wobei x_0 und y_0 der Ursprung des jeweiligen Kastens ist. Damit ist in der linken unteren Ecke des Kastens $\xi = 0$ und $\eta = 0$ und in der rechten obere Ecke $\xi = 1$ und $\eta = 1$.

Die Formfunktionen für das Element mit lokalen Elementindex (0) (siehe Abb. 10.3b) lauten

$$N_0^{(0)}(\xi, \eta) = 1 - \xi - \eta \tag{10.12}$$

$$N_1^{(0)}(\xi, \eta) = \xi \tag{10.13}$$

$$N_2^{(0)}(\xi, \eta) = \eta, \tag{10.14}$$

wobei der Index i in $N_i^{(0)}$ den lokalen Knotenindex bezeichnet, bei dem die Formfunktion 1 wird. Diese Formfunktionen sind in Abb. 10.4 gezeigt. Auf Element (1) lauten die Formfunktionen

$$N_0^{(1)}(\xi, \eta) = \xi + \eta - 1 \tag{10.15}$$

$$N_1^{(1)}(\xi, \eta) = 1 - \xi \tag{10.16}$$

$$N_2^{(1)}(\xi, \eta) = 1 - \eta. \tag{10.17}$$

Diese Formfunktionen erfüllen die Eigenschaft $N_0^{(n)} + N_1^{(n)} + N_2^{(n)} = 1$, die sich *Zerlegung der Eins* (engl. "partition of unity") nennt.

Wir brauchen im folgenden die Ableitungen der Formfunktionen respektive der Positionen x und y. Im allgemeinen Fall erhält man

$$\frac{\partial N_i^{(n)}}{\partial x} = \frac{\partial N_i^{(n)}}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial N_i^{(n)}}{\partial \eta}\frac{\partial \eta}{\partial x} \tag{10.18}$$

$$\frac{\partial N_i^{(n)}}{\partial y} = \frac{\partial N_i^{(n)}}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial N_i^{(n)}}{\partial \eta}\frac{\partial \eta}{\partial y}, \tag{10.19}$$

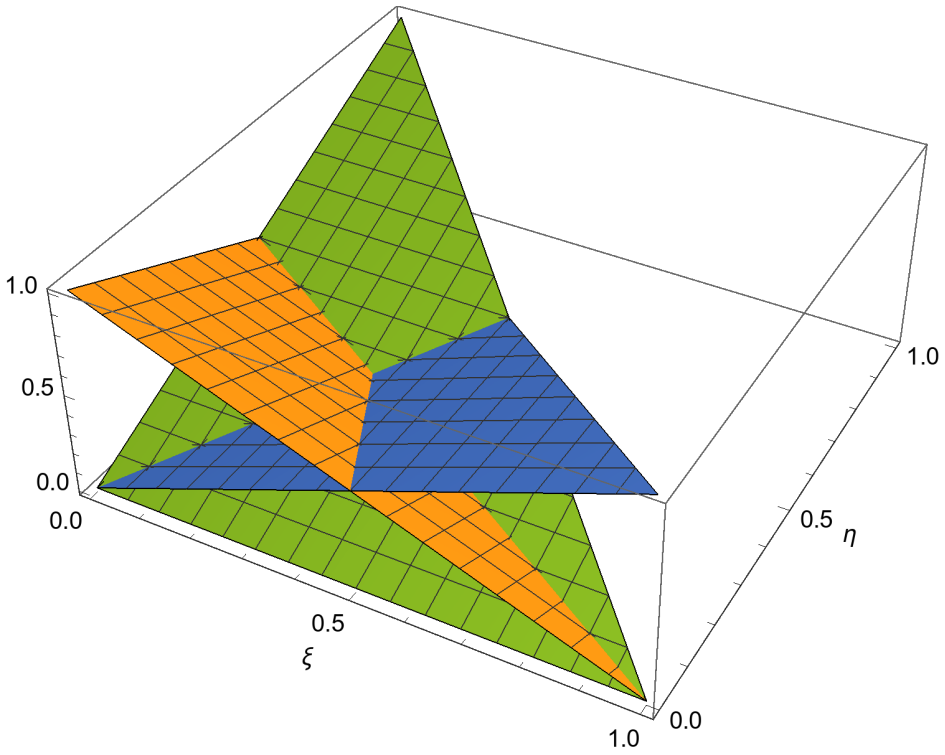

Abbildung 10.4: Formfunktionen für lineare Dreieckelemente in zwei Dimensionen. Jeweils eine der Formfunktionen ist an einem der Knoten 1. An den anderen beiden Knoten fallen die Formfunktionen auf 0 ab.

oder in kompakter Matrix-Vektor Schreibweise

$$\nabla_{x,y} N_i^{(n)} \cdot \underline{J} = \nabla_{\xi,\eta} N_i^{(n)}, \tag{10.20}$$

wobei $\nabla_{x,y}$ den Gradienten respektive der angezeigten Koordinaten meint. Die Matrix

$$\underline{J} = \begin{pmatrix} \partial x/\partial \xi & \partial x/\partial \eta \\ \partial y/\partial \xi & \partial y/\partial \eta \end{pmatrix} = \begin{pmatrix} \Delta x & 0 \\ 0 & \Delta y \end{pmatrix} \tag{10.21}$$

nennt sich *Jacobi-Matrix*. In unserem Beispiel ergibt sich die explizite Matrix auf der rechten Seite von Gl. (10.21), die unabhängig vom Kasten ist, den wir betrachten. Für komplexere Gitter (z.B. Abb. 10.2b) beschreibt die Jacobi-Matrix die Form der Dreiecke und damit die Struktur des Gitters. Die Darstellung mit Hilfe der reskalierten Koordinaten ξ und η, und damit Gl. (10.20) als Gradienten, entkoppelt die Interpolationsvorschrift Gl. (10.13)-(10.17) von der Struktur des Gitters und ist daher besonders für unstrukturierte Gitter nützlich.

Für unser Gitter finden wir also

$$\frac{\partial N_0^{(0)}}{\partial x} = -1/\Delta x, \quad \frac{\partial N_0^{(0)}}{\partial y} = -1/\Delta y, \tag{10.22}$$

$$\frac{\partial N_1^{(0)}}{\partial x} = 1/\Delta x, \quad \frac{\partial N_1^{(0)}}{\partial y} = 0, \tag{10.23}$$

$$\frac{\partial N_2^{(0)}}{\partial x} = 0, \quad \frac{\partial N_2^{(0)}}{\partial y} = 1/\Delta y, \tag{10.24}$$

$$\frac{\partial N_0^{(1)}}{\partial x} = 1/\Delta x, \quad \frac{\partial N_0^{(1)}}{\partial y} = 1/\Delta y, \tag{10.25}$$

$$\frac{\partial N_1^{(1)}}{\partial x} = -1/\Delta x, \quad \frac{\partial N_1^{(1)}}{\partial y} = 0, \tag{10.26}$$

$$\frac{\partial N_2^{(1)}}{\partial x} = 0, \quad \frac{\partial N_2^{(1)}}{\partial y} = -1/\Delta y \tag{10.27}$$

für die Ableitungen der Formfunktionen. Da wir lediglich lineare Elemente verwandt haben, sind diese Ableitungen alle Konstanten.

10.4 Galerkin-Methode

Wir können nun die Galerkin-Methode anwenden, um das lineare Gleichungssystem, welches die diskretisierte Differentialgleichung beschreibt, zu bestimmen. Wir unterscheiden hier wieder zwischen Elementmatrizen und der

Systemmatrix. Für die Elementmatrix schreiben wir den Beitrag der Formfunktion zur Galerkin-Bedingung auf. Man erhält

$$\begin{aligned}(N_I^{(n)}, R) &= \left(N_I^{(n)}, \nabla^2\Phi + \frac{\rho(\vec{r})}{\varepsilon}\right) \\ &= \int_{\partial\Omega} \mathrm{d}^2 r\, N_I^{(n)}(\vec{r}) \nabla\Phi \cdot \hat{n}(\vec{r}) - \sum_J a_J (\nabla N_I^{(n)}, \nabla N_J^{(n)}) + \frac{1}{\varepsilon}(N_I^{(n)}, \rho), \\ &= -\sum_J K_{IJ}^{(n)} a_J + f_I^{(n)},\end{aligned} \tag{10.28}$$

wobei $K_{IJ}^{(n)}$ nun die Elementmatrix ist und $f_I^{(n)}$ der Beitrag des Elements zur rechten Seite. Man erhält

$$K_{IJ}^{(n)} = (\nabla N_I^{(n)}, \nabla N_J^{(n)}). \tag{10.29}$$

wobei für zwei Vektorfelder $\vec{f}(\vec{r})$ und $\vec{g}(\vec{r})$ das Skalarprodukt als

$$(\vec{f}, \vec{g}) = \int_\Omega \mathrm{d}^3 r\, \vec{f}^*(\vec{r}) \cdot \vec{g}(\vec{r}), \tag{10.30}$$

also als kartesisches Skalarprodukt zwischen den beiden Funktionswerten, zu verstehen ist. Der Beitrag des Elements zur rechten Seite lautet

$$f_I^{(n)} = \frac{1}{\varepsilon}(N_I^{(n)}, \rho) + \int_{\partial\Omega} \mathrm{d}^2 r\, N_I^{(n)}(\vec{r}) \nabla\Phi \cdot \hat{n}(\vec{r}). \tag{10.31}$$

Beispiel: Wir berechnen nun die Elementmatrizen für die beiden Elemente unseres strukturierten Beispielgitters. So lautet z.B. die Komponente $I = 0$ und $J = 0$ des Elements (0),

$$\begin{aligned}K_{00}^{(0)} &= (\nabla N_0^{(0)}, \nabla N_0^{(0)}) \\ &= \int_{\Omega^{(0)}} \mathrm{d}^2 r \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) \\ &= \frac{\Delta x \Delta y}{2}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) \\ &= \frac{1}{2}\left(\frac{\Delta y}{\Delta x} + \frac{\Delta x}{\Delta y}\right)\end{aligned} \tag{10.32}$$

wobei der Faktor $\Delta x \Delta y/2$ die Fläche des Elements ist. Wir betrachten im folgenden den Spezialfall $\Delta x = \Delta y$, in dem man $K_{00}^{(0)} = 1$ erhält. (Im

eindimensionalen Fall blieb hier noch ein $1/\Delta x$ stehen. Die Einheiten von K unterscheiden sich damit im eindimensionalen und zweidimensionalen Fall!)

Die andere Skalarprodukte können ähnlich ausgerechnet werden. Man erhält

$$\underline{K}^{(0)} = \underline{K}^{(1)} = \begin{pmatrix} 1 & -1/2 & -1/2 \\ -1/2 & 1/2 & 0 \\ -1/2 & 0 & 1/2 \end{pmatrix} \tag{10.33}$$

für beide Elementmatrizen. Diese Matrizen sind identisch, weil die Nummerierung der lokalen Elementknoten so gewählt wurde, dass die beiden Dreiecke ineinander rotiert werden können (siehe Abb. 10.3b).

Aus diesen Elementmatrizen müssen wir nun die Systemmatrix zusammenbauen. Die lokalen Knotenindices sind in Abb. 10.3b gezeigt. Diese entsprechen den Spalten und Zeilen von Gl. (14.10). Sie müssen nun auf die globalen Knotenindices abgebildet werden und in der Systemmatrix summiert werden. So müssen wir Beispielsweise für Element (8) in Abb. 10.3a den lokalen Knoten 0 auf den globalen Knoten 5 abbilden, $0 \to 5$. D.h. die erste Zeile der Elementmatrix wird zur sechsten Zeile der Systemmatrix. (Es ist die sechste Zeile, und nicht die fünfte Zeile, da die Indizierung bei Null beginnt.) Weiterhin müssen wir $1 \to 6$ und $2 \to 9$ abbilden. Der Beitrag $\Delta\underline{K}^{(8)}$ von Element (8) zur 16×16 Systemmatrix ist daher

$$\Delta\underline{K}^{(8)} = \begin{pmatrix}
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&1&-\frac{1}{2}&\cdot&\cdot&-\frac{1}{2}&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&-\frac{1}{2}&\frac{1}{2}&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&-\frac{1}{2}&\cdot&\cdot&\cdot&\frac{1}{2}&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{pmatrix} \tag{10.34}$$

wobei Enträge mit Wert 0 zur besseren visuellen Darstellung als Punkt $(\cdot)$

gezeigt sind. Die gesamte Systemmatrix ergibt sich dann als Summe über alle Elemente, $\underline{K} = \sum_n \underline{K}^{(n)}$. Man erhält

$$\underline{K} = \begin{pmatrix}
1 & \frac{\bar{1}}{2} & . & . & \frac{\bar{1}}{2} & . & . & . & . & . & . & . & . & . & . & . \\
\frac{\bar{1}}{2} & 2 & \frac{\bar{1}}{2} & . & . & \bar{1} & . & . & . & . & . & . & . & . & . & . \\
. & \frac{\bar{1}}{2} & 2 & \frac{\bar{1}}{2} & . & . & \bar{1} & . & . & . & . & . & . & . & . & . \\
. & . & \frac{\bar{1}}{2} & 1 & . & . & . & \frac{\bar{1}}{2} & . & . & . & . & . & . & . & . \\
\frac{\bar{1}}{2} & . & . & . & 2 & \bar{1} & . & . & \frac{\bar{1}}{2} & . & . & . & . & . & . & . \\
. & \bar{1} & . & . & \bar{1} & 4 & \bar{1} & . & . & \bar{1} & . & . & . & . & . & . \\
. & . & \bar{1} & . & . & \bar{1} & 4 & \bar{1} & . & . & \bar{1} & . & . & . & . & . \\
. & . & . & \frac{\bar{1}}{2} & . & . & \bar{1} & 2 & . & . & . & \frac{\bar{1}}{2} & . & . & . & . \\
. & . & . & . & \frac{\bar{1}}{2} & . & . & . & 2 & \bar{1} & . & . & \frac{\bar{1}}{2} & . & . & . \\
. & . & . & . & . & \bar{1} & . & . & \bar{1} & 4 & \bar{1} & . & . & \bar{1} & . & . \\
. & . & . & . & . & . & \bar{1} & . & . & \bar{1} & 4 & \bar{1} & . & . & \bar{1} & . \\
. & . & . & . & . & . & . & \frac{\bar{1}}{2} & . & . & \bar{1} & 2 & . & . & . & \frac{\bar{1}}{2} \\
. & . & . & . & . & . & . & . & \frac{\bar{1}}{2} & . & . & . & 1 & \frac{\bar{1}}{2} & . & . \\
. & . & . & . & . & . & . & . & . & \bar{1} & . & . & \frac{\bar{1}}{2} & 2 & \frac{\bar{1}}{2} & . \\
. & . & . & . & . & . & . & . & . & . & \bar{1} & . & . & \frac{\bar{1}}{2} & 2 & \frac{\bar{1}}{2} \\
. & . & . & . & . & . & . & . & . & . & . & \frac{\bar{1}}{2} & . & . & \frac{\bar{1}}{2} & 1
\end{pmatrix} \tag{10.35}$$

wobei der Balken über der Zahl ein Minus anzeigt, also z.B. $\bar{1} = -1$. Diese 16×16 Matrix ist wiederum nicht regulär; ihr Rang ist 15. D.h. man braucht mindestens eine Dirichlet-Randbedingung, um zu einem lösbaren Problem zu kommen. Anhand dieses Beispiels wird auch vermutlich klar, dass die Systemmatrix nur schwer von Hand auszurechnen ist und wir einen Computer zu Hilfe ziehen müssen.

10.5 Randbedingungen

10.5.1 Dirichlet-Randbedingungen

Dirichlet-Randbedingungen funktionieren in höheren Dimensionen analog zum eindimensionalen Fall. Wir fixieren den Funktionswert auf einem Knoten. Man erhält $\Phi_N(\vec{r}_n) = a_n \equiv \Phi_n$, wobei Φ_n der gewählte Funktionswert ist, und ersetzt damit eine Galerkin-Bedingung durch diese Fixierung des Potential auf den Knoten.

10.5.2 Neumann-Randbedingungen

Wie im eindimensionalen Fall werden Neumann-Randbedingungen über den Oberflächenterm in Gl. (10.31) in die rechte Seite eingebaut. Diese Bedingungen werden daher auf den Seiten der Dreiecke definiert. Für eine konstanten Richtungsableitung $\Phi' = \nabla\Phi \cdot \hat{n}$ erhält man

$$f_I^{(n)} = \frac{1}{\varepsilon}(N_I^{(n)}, \rho) + \Phi' \int_{\partial\Omega} \mathrm{d}r\, N_I^{(n)}(\vec{r}). \tag{10.36}$$

Das Integral in Gl. (10.36) wird über die Seite des Dreiecks ausgeführt. Für die Seite zwischen den Knoten 2 und 3 (siehe Abb. 10.3), beispielsweise, sind die Formfunktionen $N_0^{(0)}$ und $N_1^{(0)}$ (Element (4)) ungleich Null. Man erhält

$$f_0^{(4)} = \frac{1}{\varepsilon}(N_0^{(0)}, \rho) + \Phi' \int_0^{\Delta x} \mathrm{d}x\, (1 - x/\Delta x) = \frac{1}{\varepsilon}(N_0^{(0)}, \rho) + \frac{1}{2}\Phi'\Delta x \tag{10.37}$$

$$f_1^{(4)} = \frac{1}{\varepsilon}(N_1^{(0)}, \rho) + \Phi' \int_0^{\Delta x} \mathrm{d}x\, x/\Delta x = \frac{1}{\varepsilon}(N_1^{(0)}, \rho) + \frac{1}{2}\Phi'\Delta x \tag{10.38}$$

für die rechte Seite. Die Randbedingungen, welche man ohne weitere Modifikation der rechten Seite erhält (siehe Gl. (10.35)), sind daher Neumann-Bedingungen mit verschwindender Ableitung, $\Phi' = 0$.

Kapitel 11

Datenstrukturen & Implementierung

Kontext: In diesem Kapitel wird Anhand unseres Beispiels, der Lösung der Poisson-Gleichung, gezeigt, wie nun ein einfacher Finite-Elemente-Löser in der Programmiersprache PYTHON implementiert werden kann. Wir nehmen hier an, dass die Ladungsdichte verschwindet. Wir berechnen also die räumliche Verteilung des elektrostatischen Potentials unter entsprechenden Randbedingungen. Dies werden wir nutzen, um die Kapazität eines Plattenkondensators auszurechnen.

11.1 Beispielproblem

Wir werden nun unser Beispielproblem aus den vorhergehenden Kapitel weiterentwickeln und die Kapazität eines Plattenkondensators berechnen. Hierfür nehmen wir eine verschwindende Ladungsdichte im Kondensator an, $\rho = 0$. Die Poisson-Gleichung wird dann zur *Laplace-Gleichung*,

$$\nabla^2 \Phi = 0. \tag{11.1}$$

Die Platten des Kondensators werden als metallisch angenommen, d.h. das Potential auf den Kondensatorplatten ist konstant (siehe Abb. 11.1a). Dies wird durch eine Dirichlet-Randbedingung abgebildet. Der Rest des Gebiets erhält eine Neumann-Randbedingung, in der die Ableitung auf der Oberfläche verschwindet.

Im Kontext der Poisson- bzw. Laplace-Gleichung haben Richtungsableitungen am Rand eine einfache Interpretation. Wir schauen uns ein kleines Volumenelement $\Delta\Omega$ am Rand des Gebiets an (siehe Abb. 11.1b). Integration

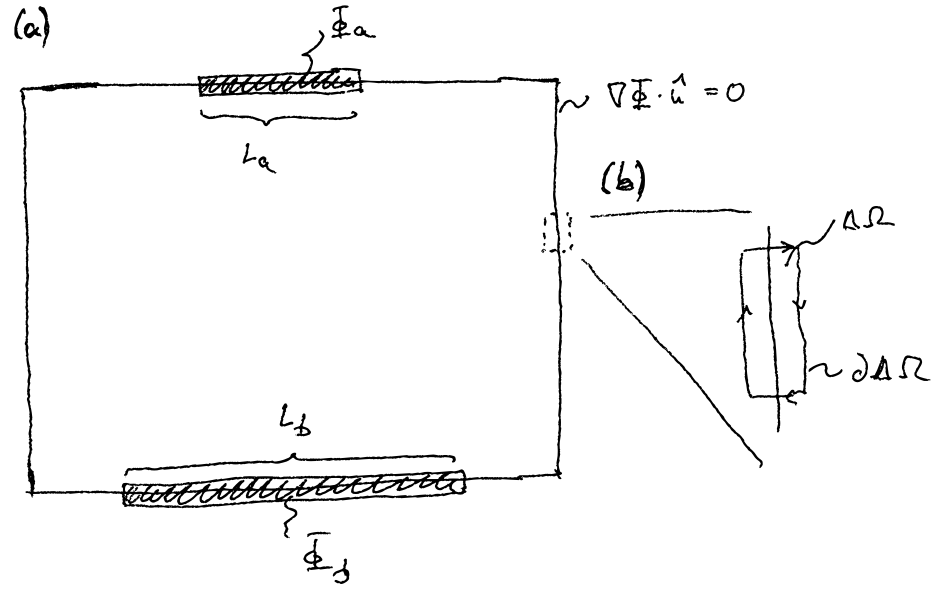

Abbildung 11.1: (a) Geometrie des in diesem Kapitel betrachteten Plattenkondensators. Auf den Elektroden ist das Potential Φ konstant. (b) Ausschnitt aus dem Randbereich mit Integrationsbereich für die Herleitung der Intepretation der Richtungsableitung am Rand.

der Poisson-Gleichung über dieses Gebiet ergibt,

$$\int_{\Delta\Omega} \mathrm{d}^3 r\, \nabla^2\Phi = \int_{\partial\Delta\Omega} \mathrm{d}^2 r\, \nabla\Phi\cdot\hat{n}(\vec{r}) = \frac{1}{\varepsilon}\int_{\Delta\Omega} \mathrm{d}^3 r\, \rho(\vec{r}), \tag{11.2}$$

wobei die Integration über $\partial\Delta\Omega$ entlang des in Abb. 11.1b gezeigten Pfades erfolgt. Nun nehmen wir an, dass die beiden Seiten des Pfades, die senkrecht auf den Rand des Gebiets stehen, vernachlässigbar gegenüber den beiden anderen Seiten sind. Weiterhin nehmen wir an, dass in dem Gebiet lediglich eine Oberflächenladung $\sigma(\vec{r})$ lebt. Damit erhält man

$$\int_{\partial\Delta\Omega} \mathrm{d}^2 r\, \nabla\Phi\cdot\hat{n}(\vec{r}) = \frac{1}{\varepsilon}\int_{\Delta A} \mathrm{d}^2 r\, \sigma(\vec{r}), \tag{11.3}$$

wobei ΔA die Fläche des Randes des Simulationsgebiets ist, welcher in $\Delta\Omega$ liegt. Nimmt man nun an, dass der Raum außerhalb des Simulationsgebietes feldfrei ist, also $\nabla\Phi = 0$, dann erhält man

$$\nabla\Phi\cdot\hat{n}(\vec{r}) = \frac{\sigma(\vec{r})}{\varepsilon}, \tag{11.4}$$

die Richtungsableitung ergibt also die Oberflächenladung am Rand.

Die Feldfreiheit außerhalb unserer Simulationsdomäne ist exakt nur an den Elektroden erfüllt. Diese sind metallisch und daher per Definition feldfrei. (Ein Feld innerhalb eines idealen Metalls führt sofort zu einer Umordnung von Ladungen, die dieses Feld dann kompensieren.) D.h. wir können Gl. (11.4) nutzen, um die auf den Kondensatorplatten induzierte Ladung zu berechnen. Zusammen mit dem durch die Dirichlet-Randbedingungen vorgegebenen Potential, kann dies zur Berechnung der Kapazität auf den Kondensatorplatten genutzt werden.

Auf der anderen Seite heißt die implizite Neumann-Randbedingung $\nabla\Phi \cdot \hat{n} = 0$, dass unsere Simulation unter Bedingungen durchgeführt wird, in denen der Rand ladungsfrei ist aber außerhalb der Simulationsdomäne das Feld verschwindet. Dies ist eine künstliche Bedingung, die Fehler verursachen kann. Man muss also sicherstellen, dass die Simulationsdomäne in Richtung parallel zu den Kondensatorplatten groß genug ist, um die Streufelder am Rand den Kondensators vernünftig zu erfassen.

11.2 Datenstrukturen

Die zentrale Datenstruktur für numerische Anwendungen ist der multidimensionale Array, `numpy.ndarray`, der `numpy`-Bibliothek. Multidimensionale Arrays halten Speicher für eine gewisse Menge an Einträgen vor. (Diese Einträge werden auch *Elemente* genannt, aber um diese nicht mit den finite Elementen durcheinander zu werden, werden sie in diesem Dokument durchgehend Einträge genannt.) Ein mit Nullen gefülltes Array der Länge 10 (also 10 Einträge) erhält man durch

```
import numpy as np
a = np.zeros(10)
```

Diese Array hat die *Dimension* 1. Der mehrdimensionale Charakter der Arrays äußert sich darin, dass die Arrays implizit eine Abbildung von mehreren Koordinaten auf einen lineare Index implementieren. Einen zweidimensionalen Array bekommt man z.B. durch

```
b = np.zeros([2, 5])
```

Man kann nun jeweils auf die Array-Einträge zugreifen, z.B.

```
a[6]
b[1, 1]
```

Beide Befehle greifen auf Eintrag 6 des zu Grunde liegenden Speicherbereichs zu. Ein natürlicher Einsatz der multidimensionalen Arrays ist die Repräsentation von Vektoren (1-dimensionale Arrays) oder Martrizen (2-dimensionale Arrays).

Anmerkung: Die Aussage, dass

```
a[6]
b[1, 1]
```

in dem o.g. Beispiel auf den gleichen Eintrag des zu Grunde liegenden Speicherbereichs zugreift, hängt von der Speicherreihenfolge (engl. "storage order") ab. Sie gilt nur dann, wenn der letzte Index kompakt im Speicher steht. Diese Speicherreihenfolge nennt sich "row major", weil in einem zweidimensionalen Array, also einer Matrix, die Zeilen (engl. "row") kompakt im Speicher stehen. Ist der erste Index der kompakte spricht man von "column major". In `numpy` heißt "row major" auch "C-continguous" und "column major" heißt "F-contiguous". Dies kommt daher, dass in der Programmiersprache `C` die Speicherreihenfolge "row major" und in der Programmiersprache `Fortran` die Speicherreihenfolge "column major" ist. Arrays in `numpy` sind standardmäßig "column major", andere Speicherreihenfolgen werden aber unterstützt und kommen auch vor.

11.3 Initialisierung

Die Beispielimplementierungen folgen einfache Regeln für lesbaren Computercode. Dieser sollte immer so geschrieben sein, dass eine dritte Person diesen lesen und wiederverwenden kann. Wir werden daher...

1. ...ausschließlich englische Sprache verwenden.
2. ...Variablennamen ausschreiben und keine Symbole als Variablennamen verwenden (also z.B. `potential` und nicht das ausgeschriebene Symbol `phi` als Name).
3. ...Array-Variablen mit einem Suffix versehen, der den Typ der Indices anzeigt (z.B. `potential_xy` um anzuzeigen, dass es zwei Indices gibt die den Positionen x und y entsprechen).
4. ...den Code mit Kommentarblöcken und `Python` Docstrings dokumentieren. Wir empfehlen den numpydoc-Standard für Docstrings.

In dieser Implementierung verwenden wir explizite Schleifen, um die Lesbarkeit des Codes zu verbessern. Der Code kann durch Verwenden von NUMPY-Operationen noch vektorisiert werden.

Zunächst müssen wir den Code initialisieren und festlegen, wieviele Gitterpunkte wir verwenden wollen. Wir definieren die Variablen

```
# Grid size, number of nodes
nb_nodes = 32, 32
Nx, Ny = nb_nodes
```

Wir legen nun auch noch fest, über welchen Bereich sich die beiden Elektroden des Kondensators erstrecken sollen:

```
# Top capacitor plate
top_left = Nx//4
top_right = 3*Nx//4-1
top_potential = 1

# Bottom capacitor plate
bottom_left = Nx//4
bottom_right = 3*Nx//4-1
bottom_potential = -1
```

Der Bereich wird hier mit Knotenindices angegeben. Weiterhin benötigen wir noch die Elementmatrix die wir in einem `numpy.ndarray` speichern:

```
# Element matrix, index l indicates element-local node
element_matrix_ll = np.array([[1, -1/2, -1/2],
                              [-1/2, 1/2, 0],
                              [-1/2, 0, 1/2]])
```

Der Suffix `_ll` bezeichnet hier, dass es zwei Indices gibt (der Array ist zweidimensional), die beide einen lokalen Elementknoten bezeichnen. Wir initialisieren weiterhin die Systemmatrix und die rechte Seite, zunächst mit Nullen:

```
# System matrix, index g indicates global node
system_matrix_gg = np.zeros([Nx*Ny, Nx*Ny])

# Right hand side
rhs_g = np.zeros(Nx*Ny)
```

Der Suffix `_g` bezeichnet hier den Index des globalen Knoten. Die Variable `rhs_g` enthält den Vektor $\vec{f}$ und benötigt daher nur einen Index. Die Variable `system_matrix_gg` beinhaltet die Systemmatrix $\underline{K}$ und braucht daher zwei globale Knotenindices.

11.4 Systemmatrix

Kern des Simulationsprogramms ist der Aufbau der Systemmatrix. In diesem Abschnitt wird dies durch explizite Schleifen realisiert. Im nächsten Abschnitt wird gezeigt, wie dies mit speziellen `numpy`-Befehlen kompakter (und effizienter), aber weniger transparent gestaltet werden kann.

Zunächst definieren wir eine Funktion, die aus Knotenkoordinaten den globalen Knotenindex macht:

```
def node_index(i, j, nb_nodes):
    """
    Turn node coordinates (i, j) into their global node index
    .

    Parameters
    ----------
    i : int
        x-coordinate (integer) of the node
    j : int
        y-coordinate (integer) of the node
    nb_nodes : tuple of ints
        Number of nodes in the Cartesian directions

    Returns
    -------
    g : int
        Global node index
    """
    Nx, Ny = nb_nodes
    return i + Nx*j
```

Dies nutzen wir in einer weiteren Hilfsfunktion, die die Elementmatrix zu der Systemmatrix addiert. Hierzu muss zunächst die Elementmatrix auf die Systemmatrix aufgespannt werden. Die Funktion sieht folgendermaßen aus:

```
def add_element_matrix(system_matrix_gg, element_matrix_ll,
                       global_node_indices):
    """
    Add element matrix to global system matrix.

    Parameters
    ----------
    system_matrix_gg : array_like
        N x N system matrix where N is the number of global
        nodes. This matrix will be modified by this function.
    element_matrix_ll : array_like
        n x n element matrix where n is the number of local
        nodes
    global_node_indices : list of int
        List of length n that contains the global node
        indices for the local node index that corresponds to
        the list position.
    """
    assert element_matrix_ll.shape == \
        (len(global_node_indices), len(global_node_indices))
    for i in range(len(global_node_indices)):
```

```
        for j in range(len(global_node_indices)):
            system_matrix_gg[global_node_indices[i],
                             global_node_indices[j]] += \
                element_matrix_ll[i, j]
```

Die `assert`-Anweisung ist hier ein Wächter, der darauf achtet, dass die lokale Elementmatrix und der Array `global_node_indices` die gleiche Länge haben. Die beiden `for`-Schleifen laufen dann über alle Einträge der Elementmatrix. Der Ausdruck `global_node_indices[i]` liefert dann den globalen Knotenindex, der zu dem lokalen Knotenindex der Elementmatrix gehört. Der Zusammenbau der Systemmatrix erfolgt dann über einen Aufruf dieser Hilfsmethode pro Element:

```
def assemble_system_matrix(element_matrix_ll, nb_nodes):
    """
    Assemble system matrix from the element matrix

    Parameters
    ----------
    element_matrix_ll : array_like
        3 x 3 element matrix
    nb_nodes : tuple of ints
        Number of nodes in the Cartesian directions

    Returns
    -------
    system_matrix_gg : numpy.ndarray
        System matrix
    """

    Nx, Ny = nb_nodes
    Mx, My = Nx-1, Ny-1 # number of boxes

    # System matrix
    system_matrix_gg = np.zeros([Nx*Ny, Nx*Ny])

    # Construct system matrix
    for l in range(Mx):
        for m in range(My):
            # Element (0)
            n0 = node_index(l, m, nb_nodes)
            n1 = node_index(l+1, m, nb_nodes)
            n2 = node_index(l, m+1, nb_nodes)
            add_element_matrix(system_matrix_gg,
                               element_matrix_ll,
                               [n0, n1, n2])

            # Element (1)
            n0 = node_index(l+1, m+1, nb_nodes)
```

```
            n1 = node_index(l, m+1, nb_nodes)
            n2 = node_index(l+1, m, nb_nodes)
            add_element_matrix(system_matrix_gg,
                               element_matrix_ll,
                               [n0, n1, n2])

    return system_matrix_gg
```

Hier laufen die beiden `for`-Schleifen über die einzelnen Kästen. Die Schleife über die beiden Elemente pro Kasten ist explizit als zwei Aufrufe zu `add_element_matrix` geschrieben. Die Variablen `n0`, `n1` und `n2` enthalten die globalen Knotenindices, die die Ecken des jeweiligen Elements beschreiben.

Die nun aufgebaute Systemmatrix hat (implizit) Neumann-Randbedingungen mit $\nabla\Phi \cdot \hat{n}(\vec{r}) = 0$ auf dem Rand. Wir müssen nun noch die Dirichlet-Bedingungen für die Elektroden hinzufügen. Hierzu ersetzen wir Zeilen der Systemmatrix und die entsprechenden Einträge des Lastvektors:

```
def capacitor_bc(system_matrix_gg, rhs_g,
                 top_left, top_right, top_potential,
                 bottom_left, bottom_right, bottom_potential,
                 nb_nodes):
    """
    Set boundary conditions for the parallel plate capacitor.

    Parameters
    ----------
    system_matrix_gg : numpy.ndarray
        System matrix. The system matrix is modified by a
   call
        to this function
    rhs_g : numpy.ndarray
        Right-hand side vector. The right-hand side vector is
        modified by a call to this function.
    top_left : int
        Leftmost node of the top electrode
    top_right : int
        Rightmost node of the top electrode
    top_potential : float
        Electrostatic potential of the top electrode
    bottom_left : int
        Leftmost node of the bottom electrode
    bottom_right : int
        Rightmost node of the bottom electrode
    bottom_potential : float
        Electrostatic potential of the bottom electrode
    nb_nodes : tuple of ints
        Number of nodes in the Cartesian directions
    """
    Nx, Ny = nb_nodes
```

```
    # Dirichlet boundary conditions for top plate
    for i in range(top_left, top_right+1):
        n = node_index(i, Ny-1, nb_nodes)
        mat_g = np.zeros(Nx*Ny)
        mat_g[n] = 1
        system_matrix_gg[n] = mat_g
        rhs_g[n] = top_potential

    # Dirichlet boundary conditions for bottom plate
    for i in range(bottom_left, bottom_right+1):
        n = node_index(i, 0, nb_nodes)
        mat_g = np.zeros(Nx*Ny)
        mat_g[n] = 1
        system_matrix_gg[n] = mat_g
        rhs_g[n] = bottom_potential
```

Der gesamte Simulationscode enthält nun Aufrufe dieser Funktionen, gefolgt von der numerischen Lösung des linearen Gleichungssystems:

```
# Construct system matrix
system_matrix_gg = assemble_system_matrix(element_matrix_ll,
                                          nb_nodes)

# Boundary conditions
capacitor_bc(system_matrix_gg, rhs_g,
             top_left, top_right, top_potential,
             bottom_left, bottom_right, bottom_potential,
             nb_nodes)

# Solve system of linear equations
potential_g = np.linalg.solve(system_matrix_gg, rhs_g)
```

Die Variable `potential_g` enthält nun die Werte des elektrostatischen Potentials auf den Knoten.

11.5 Visualisierung

Das Ergebnis der Rechnung kann mit Hilfe der `matplotlib`-Bibliothek visualisiert werden. Die Funktion `matplotlib.pyplot.tripcolor` kann Daten auf einem triangulierten 2D-Gitter darstellen. Der folgende Codeblock visualisiert das Ergebnis der Simulation mit Hilfe dieser Funktion.

```
import matplotlib.pyplot as plt
import matplotlib.tri

def make_grid(nb_nodes):
    """
    Make an array that contains all elements of the grid. The
```

```
    elements are described by the global node indices of
    their corners. The order of the corners is in order of
    the local node index.

    They are sorted in geometric positive order and the first
    is the node with the right angle corner at the bottom
    left. Elements within the same box are consecutive.

    This is the first element per box:

        2
        | \
        |  \
    dy  |   \
        |    \
        0 --- 1

          dx

    This is the second element per box:

            dx
         1 ---0
          \   |
           \  |  dy
            \ |
             \|
              2

    Parameters
    ----------
    nb_nodes : tuple of ints
        Number of nodes in the Cartesian directions

    Returns
    -------
    triangles_el : numpy.ndarray
        Array containing the global node indices of the
        element corners. The first index (suffix _e)
        identifies the element number and the second index
        (suffix _l) the local node index of that element.
    """
    Nx, Ny = nb_nodes
    # These are the node position on a subsection of the grid
    # that excludes the rightmost and topmost nodes. The
    # suffix _G indicates this subgrid.
    y_G, x_G = np.mgrid[:Ny-1, :Nx-1]
    x_G.shape = (-1,)
    y_G.shape = (-1,)
```

```
56
57      # List of triangles
58      lower_triangles = np.vstack(
59          (node_index(x_G, y_G, nb_nodes),
60           node_index(x_G+1, y_G, nb_nodes),
61           node_index(x_G, y_G+1, nb_nodes)))
62      upper_triangles = np.vstack(
63          (node_index(x_G+1, y_G+1, nb_nodes),
64           node_index(x_G, y_G+1, nb_nodes),
65           node_index(x_G+1, y_G, nb_nodes)))
66      # Suffix _e indicates global element index
67      return np.vstack(
68          (lower_triangles, upper_triangles)).T.reshape(-1, 3)
69
70  def plot_results(values_g, nb_nodes, mesh_style=None,
71                   ax=None):
72      """
73      Plot results of a finite-element calculation on a
74      two-dimensional structured grid using matplotlib.
75
76      Parameters
77      ----------
78      nb_nodes : tuple of ints
79          Number of nodes in the Cartesian directions
80      values_g : array_like
81          Expansion coefficients (values of the field) on the
82          global nodes
83      mesh_style : str, optional
84          Will show the underlying finite-element mesh with
85          the given style if set, e.g. 'ko-' to see edges
86          and mark nodes by points
87          (Default: None)
88      ax : matplotlib.Axes, optional
89          Axes object for plotting
90          (Default: None)
91
92      Returns
93      -------
94      trim : matplotlib.collections.Trimesh
95          Result of tripcolor
96      """
97      Nx, Ny = nb_nodes
98
99      # These are the node positions on the full global grid.
100     y_g, x_g = np.mgrid[:Ny, :Nx]
101     x_g.shape = (-1,)
102     y_g.shape = (-1,)
103
104     # Gouraud shading linearly interpolates the color between
```

```
105     # the nodes
106     if ax is None:
107         ax = plt
108     triangulation = matplotlib.tri.Triangulation(
109         x_g, y_g, make_grid(nb_nodes))
110     c = ax.tripcolor(triangulation, values_g,
111                      shading='gouraud')
112     if mesh_style is not None:
113         ax.triplot(triangulation, mesh_style)
114     return c
115 
116 plt.subplot(111, aspect=1)
117 plot_results(potential_g, nb_nodes, show_mesh=True)
118 plt.xlabel(r'$x$-position ($\Delta x$)')
119 plt.ylabel(r'$y$-position ($\Delta y$)')
120 plt.colorbar().set_label(r'Potential $\Phi$ (V)')
121 plt.tight_layout()
122 plt.show()
```

Die Funktion `make_grid` erzeugt hier eine Liste der globalen Knotenindices pro Element. Der erste Index ist der Index des Elements (Suffix `e`), der zweite Index ist der lokale Knotenindex innerhalb des Elements (Suffix `l`). Die Knoten des jeweiligen Elements sind gegen den Uhrzeigersinn nummeriert. Für die Visualisierung wird "Gouraud"-Schattierung genutzt. Diese Art der Färbung interpoliert den Wert der Knoten linear auf den Dreiecken und entspricht exakt der Interpolationsvorschrift unserer Formfunktionen. Wir können damit die volle interpolierte Funktion $\Phi_N(\vec{r})$ darstellen.

11.6 Kapazität eines Plattenkondensators

Mit Hilfe des hier entwickelten Codes kann nun das elektrostatische Potential innerhalb eines Plattenkondensators berechnet werden. Abbildung 11.2 zeigt das Ergebnis dieser Rechnung für unterschiedliche Auflösung der Simulation, also unterschiedliche Anzahl an Elementen. Durch Erhöhung der Auflösung kann die Simulation systematisch verbessert werden.

Für die Berechnung der Kapazität müssen wir nun noch die Ladung auf den Kondensatorplatten ermitteln. Die Gesamtladung Q_α auf der Elektrode α erhält man aus der Oberflächenladung, die durch Gl. (11.4) gegeben ist. Durch Integration über die Fläche der Kondensatorplatten A_α erhält man

$$Q_\alpha = \int_{A_\alpha} \mathrm{d}^2 r\, \sigma(\vec{r}) = \varepsilon \int_{A_\alpha} \mathrm{d}^2 r\, \nabla\Phi_N \cdot \hat{n}(\vec{r}). \tag{11.5}$$

Hier spielt nun die Permittivität ε eine wichtige Rolle für die Einheit der Ladung. Wir können nun wieder die Reihenentwicklung einsetzen. Zum Integral

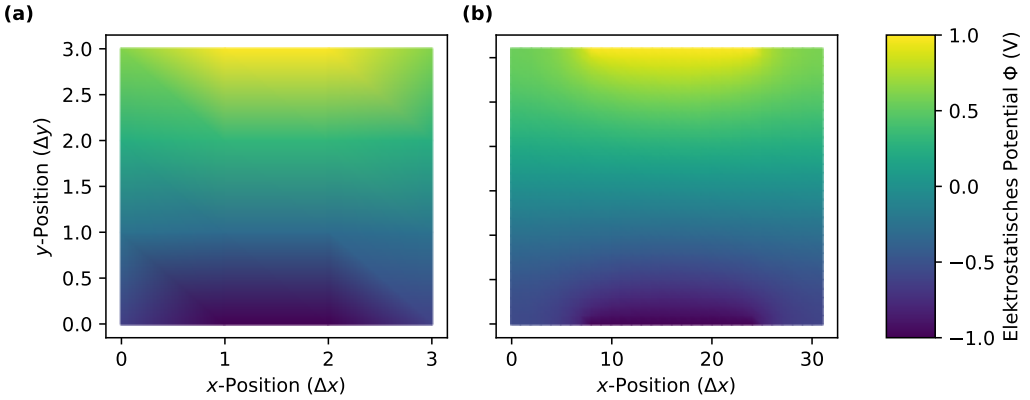

Abbildung 11.2: Elektrostatisches Potential innerhalb des Plattenkondensators, gerechnet mit (a) 4×4 Knoten (18 Elemente) und (b) 32×32 Knoten (1922 Elemente). In (a) sieht man in der farblichen Codierung den linearen Funktionsverlauf innerhalb der Elemente.

trägt nur Elementtyp (1) bei, und hier nur die Formfunktionen, bei denen die Ableitung in y-Richtung nicht verschwindet, da $\nabla\Phi_N \cdot \hat{n}(\vec{r}) = \pm\partial\Phi_N/\partial y$. Das Vorzeichnen ist bei oberer und unterer Kondensatorplatte umgedreht. Nicht-verschwindende Beiträge kommen von den Formfunktionen $N_0^{(1)}$ und $N_2^{(1)}$. Man erhält

$$\int_0^{\Delta x} \mathrm{d}x \frac{\partial N_0^{(1)}}{\partial y} = \int_0^{\Delta x} \mathrm{d}x \frac{1}{\Delta y} = \frac{\Delta x}{\Delta y} \tag{11.6}$$

$$\int_0^{\Delta x} \mathrm{d}x \frac{\partial N_2^{(1)}}{\partial y} = \int_0^{\Delta x} \mathrm{d}x \left(-\frac{1}{\Delta y}\right) = -\frac{\Delta x}{\Delta y} \tag{11.7}$$

und damit für $\Delta x = \Delta y$

$$Q^{(n)} = \varepsilon t(a_0 - a_2) \tag{11.8}$$

als Beitrag des Elements (n) zur Ladung auf der Elektrode. Hierbei bezeichnen die Indices der Koeffizienten a_0 und a_2 die jeweiligen lokalen Knotenindices. Die Größe t ist die Tiefe der Simulationsdomäne. Da wir das Problem hier in zwei Dimensionen betrachten sind alle Ladungen effektiv Linienladungen (pro Tiefe) und der Faktor t wird benötigt, um auf eine absolute Ladung zu kommen. Unser Plattenkondensator ist in die dritte Dimension unendlich lang. Der Faktor εt hat die Einheit Farad und ist damit eine Kapazität.

Die Kapazität des Kondensators ist nun gegeben als $C = Q_0/\Delta\Phi$, wobei Q_0 nun die Ladung auf einer Kondensatorplatte ist und $\Delta\Phi$ der (vorgegebene) Potentialunterschied. Die zweite Kondensatorplatte muss die Ladung $Q_1 = -Q_0$

tragen. Der Code für die Berechnung der Ladung auf den Kondensatorplatten sieht daher folgendermaßen aus:

```
def get_charge(potential_g, nb_nodes,
               top_left, top_right,
               bottom_left, bottom_right):
    """
    Compute charge on both capacitor plates.

    Parameters
    ----------
    potential_g : array_like
        Electrostatic potential
    nb_nodes : tuple of ints
        Number of nodes in the Cartesian directions
    top_left : int
        Leftmost node of the top electrode
    top_right : int
        Rightmost node of the top electrode
    bottom_left : int
        Leftmost node of the bottom electrode
    bottom_right : int
        Rightmost node of the bottom electrode

    Returns
    -------
    charge_top : float
        Charge (divided by permittivity and thickness) on top
        plate
    charge_bottom : float
        Charge (divided by permittivity and thickness) on
        bottom plate
    """
    Nx, Ny = nb_nodes
    charge_top = 0.0
    for i in range(top_left+1, top_right+1):
        charge_top += \
            potential_g[node_index(i, Ny-1, nb_nodes)] - \
            potential_g[node_index(i, Ny-2, nb_nodes)]

    charge_bottom = 0.0
    for i in range(bottom_left+1, bottom_right+1):
        charge_bottom += \
            potential_g[node_index(i, 0, nb_nodes)] - \
            potential_g[node_index(i, 1, nb_nodes)]

    return charge_top, charge_bottom
```

Natürlich wissen wir, wie die Kapazität eines Plattenkondensators aussieht.

Sie ist gegeben durch

$$C = \varepsilon \frac{A}{d}, \tag{11.9}$$

wobei $A = tL$ die Fläche der Kondensatorplatte ist und d der Abstand der Platten. (L ist die Länge der Platten, siehe Abb. 11.1a.) Dies können wir entdimensionalisiert als

$$\frac{C}{\varepsilon t} = \frac{L}{d} \tag{11.10}$$

schreiben. Die linke Seite erhalten wir direkt aus unserer Simulation. Das Ergebnis der Rechnung mit finiten Elementen ist in Abb. 11.3 im Vergleich mit diesem analytischen Ausdruck gezeigt. Man sieht, dass der analytische Ausdruck nur bei kleinen Aspektverhältnissen $d/L < 1$ gilt. Die Herleitung dieses Ausdrucks nimmt an, dass die Feldlinien überall parallel und senkrecht zu den Kondensatorplatten verlaufen. Für große Abstände der Kondensatorplatten ist dies nicht mehr der Fall und Streufelder am Rand der Platten fangen an, eine Rolle für die Kapazität zu spielen. Diese sind nicht von Gl. (11.9) erfasst, werden aber in der Simulation abgebildet.

Anmerkung: Die Systemmatrix der finite-Elemente Methode ist dünnbesetzt (engl. "sparse"). Für dünnbesetzte Matrizen gibt es spezielle Datenstrukturen, die den Umgang mit diesen Matrizen vereinfachen. Diese sind in dem Paket `scipy.sparse` implementiert. Wir können diese Routinen nutzen, um eine dünnbesetzte Systemmatrix zu konstruieren:

```
from scipy.sparse import coo_matrix

def assemble_system_matrix(element_matrix_ll, nb_nodes):
    """
    Assemble system matrix from the element matrix

    Parameters
    ----------
    element_matrix_ll : array_like
        3 x 3 element matrix
    nb_nodes : tuple of ints
        Number of nodes in the Cartesian directions

    Returns
    -------
    system_matrix_gg : numpy.ndarray
        System matrix
    """
    Nx, Ny = nb_nodes

```

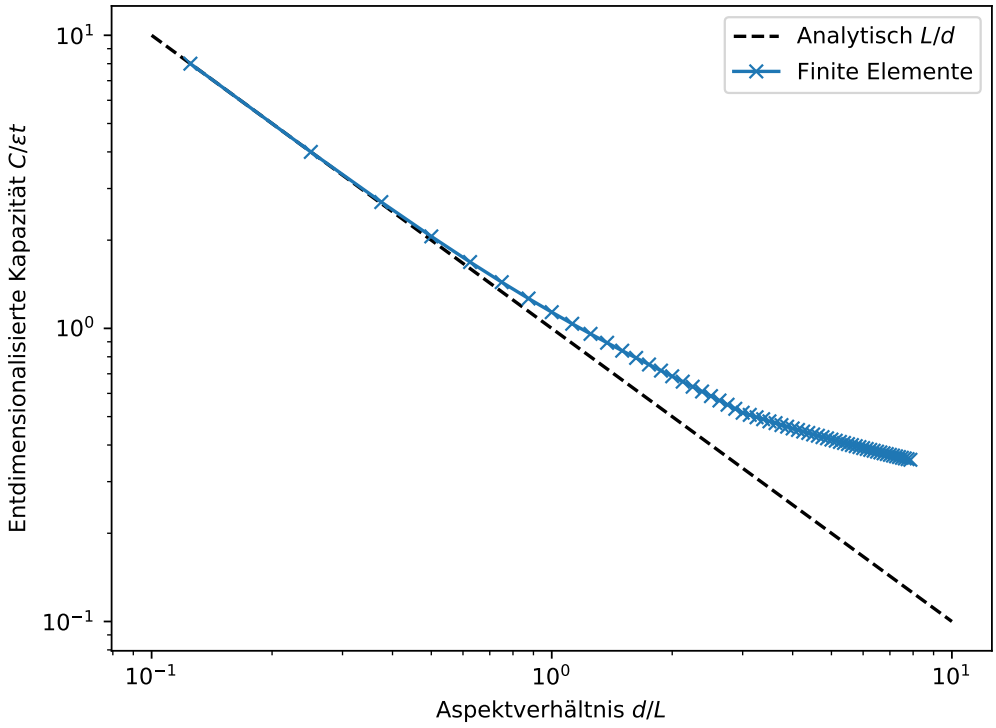

Abbildung 11.3: Kapazität C eines Plattenkondensators gegen den Abstand der Platten d. Beide Achsen sind entdimensionalisiert und zeigen Größen ohne Einheit. Die gestrichelte Linie ist die klassische Vorhersage für die Kapazität, die blaue Linie zeigt die Simulation. Hierbei wurde die Simulationsbox immer mindestens so Breit wie das dreifache der Plattenlänge L oder dem Abstand der Platten d gewählt. Die Plattenlänge L wurde mit 8 Elementen diskretisiert.

```
    # Compute grid
    grid_el = make_grid(nb_nodes)

    # Get number of elements
    nb_elements, nb_corners = grid_el.shape

    # Spread out grid and element matrix such that they can
    # be used as global node coordinates for the sparse
    # matrix
    grid1_ell = np.stack(
        [grid_el, grid_el, grid_el], axis=1)
    grid2_ell = np.stack(
        [grid_el, grid_el, grid_el], axis=2)
    element_matrix_ell = np.stack(
        [element_matrix_ll]*nb_elements, axis=0)

    # Construct sparse system matrix
    # `coo_matrix` will automatically sum duplicate entries
    system_matrix_gg = coo_matrix(
        (element_matrix_ell.reshape(-1),
         (grid1_ell.reshape(-1), grid2_ell.reshape(-1))),
        shape=(Nx*Ny, Nx*Ny))

    return system_matrix_gg.todense()
```

Diese Methode ersetzt die obige Implementation von `assemble_system_matrix`. Sie liefert (aus Gründen der Kompatibiltät zum Rest der hier gezeigten Implementierung) zum Schluss wieder eine dichtbesetzte (engl. “dense”) Matrix, man kann aber durchaus mit der dünnbesetzten Matrix weiterrechnen. Im Rahmen der Anwendung von `coo_matrix` werden hier die globalen Knotenindizes als “Koordinaten” der Matrixeinträge eingesetzt. Hierzu müssen sowohl die Knotenindizes als auch die Einträge der Elementmatrix auf die Größe der Systemmatrix vervielfältigt werden. Dies geschieht durch die `numpy.stack` Befehle in diesen Codefragment.

Kapitel 12

Nichtlineare Probleme

Kontext: Bislang haben wir nur lineare Probleme betrachtet. Als Beispiel für eine nichtlineare partielle Differentialgleichung ist uns allerdings bereits die Poisson-Boltzmann-Gleichung begegnet. Die Lösung nichtlinearer partieller Differentialgleichungen führt zu zwei zusätzlichen Schwierigkeitsgraden: Zum einen müssen wir nichtlineare Gleichungssysteme lösen können, zum anderen müssen wir Integrale über Funktionen mit Polynomordnung höher als Zwei in den Basisfunktion ausrechnen. Hierzu werden in diesem Kapitel das Newton-Raphson-Verfahren und Quadraturregeln eingeführt.

12.1 Newton-Raphson-Verfahren

Das *Newton-Raphson-Verfahren*, oder auch kurz nur *Newton-Verfahren*, ist ein Verfahren für die iterative Lösung einer nichtlinearen Gleichung. Zur Illustration beschreiben wir dieses hier zunächst für skalarwertige Funktionen und werden dann das Verfahren für vektorwertige Funktionen verallgemeinern.

Wir suchen die allgemeine Lösung für die Gleichung $f(x) = 0$ mit beliebiger Funktion $f(x)$. Die Idee des Newton-Verfahrens ist es nun, die Gleichung an einem Punkt zu linearisieren, also eine Taylorentwicklung bis zur ersten Ordnung hinzuschreiben, und dann dieses linearisierte System zu Lösen. Die um den Punkt x_i taylorentwickelte Gleichung

$$f(x) \approx f(x_i) + (x - x_i)f'(x_i) = 0 \tag{12.1}$$

hat die Lösung

$$x_{i+1} = x_i - f(x_i)/f'(x_i), \tag{12.2}$$

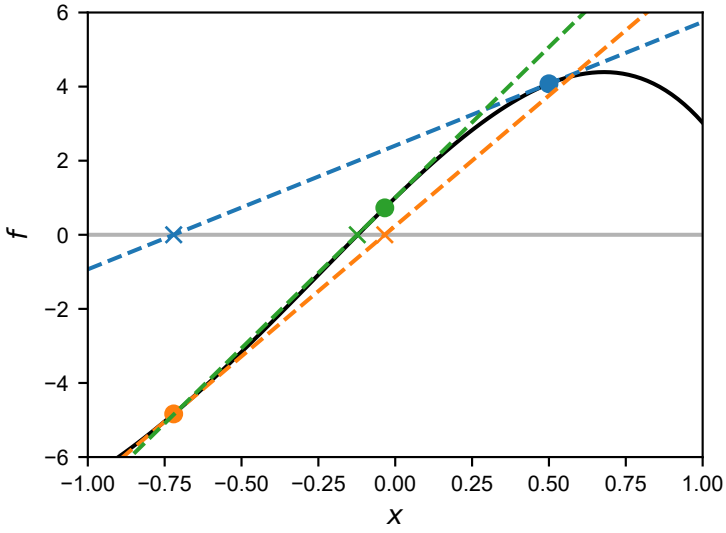

Abbildung 12.1: Illustration des Newton-Verfahrens zur Lösung der Gleichung $f(x) = 0$, hier für die Funktion $f(x) = 10\sin(x) - \exp 2x + 2$. Die gestrichelten Linien sind jeweils die linearisierte Form, Gl. (**??**). Das Verfahren started bei $x = 0.5$ (blauer Punkt) und liefert die Nullstelle der um diesen Punkt linearisierten Form (blaues Kreuz). Die zweite Iteration ist die orangene Linie, die dritte die grüne Linie. Hier sind nur die ersten drei Schritte dieser Newton-Iteration gezeigt, nach denen bereits eine gute Lösung der Nullstelle gefunden wurde.

wobei $f'(x) = \mathrm{d}f/\,\mathrm{d}x$ die erste Ableitung der Funktion f ist. Der Wert x_{i+1} ist nun (unter gewissen Bedingungen) näher an der Nullstelle x_0 (mit $f(x_0) = 0$) als der Wert x_i. Die Idee des Newton-Verfahrens ist es nun, eine Folge x_i von linearen Approximationen der Funktion f zu konstruieren, die auf die Nullstelle konvergiert. Wir nutzen also die Nullstelle der linearisierten Form der Gleichung als Startpunkt für die nächste Iteration. Ein Beispiel einer solchen Iteration ist in Abb. **??** gezeigt.

Die Diskretisierung unserer PDGLs führte uns auf ein lineares Gleichungssystem, welches wir Abstrakt als $\vec{f}(\vec{x}) = \vec{0}$ schreiben können. Wir werden in diesem Kapitel nichtlineare Gleichungssysteme dieser Form kennen lernen. Das Newton-Verfahren für die Lösung solcher gekoppelter nichtlinearer Gleichungen funktioniert analog zu dem skalaren Fall. Wir schreiben zunächst die Taylor-Entwicklung

$$\vec{f}(\vec{x}) \approx \vec{f}(\vec{x}_i) + \underline{K}(\vec{x}_i) \cdot (\vec{x} - \vec{x}_i) = 0 \tag{12.3}$$

mit der *Jacobi-Matrix*

$$K_{mn}(\vec{x}) = \frac{\mathrm{d}f_m(\vec{x})}{\mathrm{d}x_n}. \tag{12.4}$$

Im Kontext der finiten Element wird $\underline{K}(\vec{x}_i)$ auch häufig die *Tangentenmatrix* (engl. "tangent matrix" oder "tangent stiffness matrix") genannt. Das Newton-Verfahren lässt sich dann als

$$\vec{x}_{i+1} = \vec{x}_i - \underline{K}^{-1}(\vec{x}_i) \cdot \vec{f}(\vec{x}_i) \tag{12.5}$$

schreiben. In der numerischen Lösung von Gl. (**??**) wird der Schritt $\Delta\vec{x}_i = \vec{x}_{i+1} - \vec{x}_i$ meist über die Lösung des linearen Gleichungssystems $\underline{K}_i \cdot \Delta\vec{x}_i = -\vec{f}(\vec{x}_i)$ und nicht mit Hilfe einer expliziten Matrixinversion von $\underline{K}_i$ implementiert.

Für ein rein lineares Problem, wie wir sie ausschließlich in den vorhergehenden Kapiteln besprochen haben, ist die Tangentenmatrix $\underline{K}$ konstant und nimmt die Rolle der Systemmatrix ein. In diesem Fall konvergiert die Newton-Iteration in einem Schritt.

12.2 Numerische Integration

Alle Integrale aus den vorhergehenden Kapiteln, insbesondere die Integrale der Laplace- und Massematrizen, konnten vollständig analytisch gelöst werden. Dies ist bei nichtlinearen PDGLs in vielen Fällen nicht mehr möglich. Wir müssen daher über numerische, approximative Integration reden. Dies wird oft synonym auch *numerische Quadratur* genannt. Wir werden hier die Terme Integration und Quadratur austauschbar verwenden.

Wir betrachten zunächst eine Funktion $f(x)$ und möchten das Integral $\int_{-1}^{1} \mathrm{d}x\, f(x)$ über ein gewisses Gebiet $[-1, 1]$ auswerten. (Wie beschränken uns hier auf dieses Gebiet. Eine Integration über ein allgemeines Interval $[a, b]$ kann immer auf dieses Gebiet abgebildet werden.) Eine naheliegende Lösung wäre die Approximation des Integrals mit einer Summe von Rechtecken. Wir schreiben

$$\int_{-1}^{1} \mathrm{d}x\, f(x) \approx \sum_{n=0}^{N-1} w_n^Q f(x_n^Q), \tag{12.6}$$

wobei $\sum_n w_n = 2$. Der Quadraturpunkt x_n^Q muss im n-ten Interval liegen, $\sum_{i=0}^{n-2} w_i^Q - 1 < x_n^Q < \sum_{i=0}^{n-1} w_i^Q - 1$. Gleichung (**??**) ist eine *Quadraturregel* (engl. "quadrature rule"). Die Punkte x_n^Q heißen Quadraturpunkte (engl. "quadrature points") und die w_n^Q sind die Quadraturgewichte (engl. "quadrature weights"). Für die Rechteckregel sind diese Gewichte genau die Breite

der Rechtecke, andere Formen einer Quadraturregel werden im folgenden Besprochen.

Wir stellen nun die Frage, welche Wahl von x_n^Q und w_n^Q für eine gegebene Zahl and Quadraturpunkten N ideal wäre. Eine gute Wahl für $N = 1$ ist sicherlich $x_1^Q = 0$ und $w_1^Q = 2$. Diese Regel führt für lineare Funktionen zu der exakten Lösung. Verschieben wir den Quadraturpunkt x_1^Q an einen anderen Ort, so werden nur noch konstante Funktionen exakt approximiert.

Mit zwei Quadraturpunkten sollten wir daher ein Polynom dritter Ordnung exakt integrieren können. Wir können diese Punkte bestimmen, in dem wir genau die exakte Integration von Polynomen bis zu dritter Ordnung mit einer aus zwei Termen bestehenden Summe explizit verlangen:

$$\int_{-1}^{1} \mathrm{d}x\, 1 = 2 = w_1^Q + w_2^Q \tag{12.7}$$

$$\int_{-1}^{1} \mathrm{d}x\, x = 0 = w_1^Q x_1^Q + w_2^Q x_2^Q \tag{12.8}$$

$$\int_{-1}^{1} \mathrm{d}x\, x^2 = 2/3 = w_1^Q (x_1^Q)^2 + w_2^Q (x_2^Q)^2 \tag{12.9}$$

$$\int_{-1}^{1} \mathrm{d}x\, x^3 = 0 = w_1^Q (x_1^Q)^3 + w_2^Q (x_2^Q)^3 \tag{12.10}$$

Die Lösung dieser vier Gleichungen führt direkt zu $w_1^Q = w_2^Q = 1$, $x_1^Q = 1/\sqrt{3}$ und $x_2^Q = -1/\sqrt{3}$. Für drei Quadraturpunkte erhält man mit einer identischen Konstruktion $w_1^Q = w_3^Q = 5/9$, $w_2^Q = 8/9$, $x_1^Q = -\sqrt{3/5}$, $x_2^Q = 0$ und $x_3^Q = \sqrt{3/5}$. Diese Art der numerischen Integration nennt sich *Gauß-Quadratur*.

12.3 Poisson-Boltzmann-Gleichung

Wir diskutieren nun die numerische Lösung der nichtlinearen Poisson-Boltzmann-(PB-)Gleichung für zwei Spezies,

$$\begin{aligned} \nabla^2 \Phi &= -\frac{c_0}{\varepsilon} \left[q_+ \exp\left(-\frac{q_+ \Phi}{k_B T} \right) + q_- \exp\left(-\frac{q_- \Phi}{k_B T} \right) \right] \\ &= \frac{2\rho_0}{\varepsilon} \sinh\left(\frac{|e|\Phi}{k_B T} \right) \end{aligned} \tag{12.11}$$

wobei $\rho_0 = |e| c_0$ die Referenzladungsdichte und $q_+ = |e|$ and $q_- = -|e|$ die ionischen Ladungen sind. Da für kleine x gilt $\sinh x \approx x$, ist die linearisierte Variante der PB-Gleichung $\nabla^2 \Phi = \Phi/\lambda^2$ mit der Debye-Länge

$\lambda = \sqrt{\varepsilon k_B T/(2|e|\rho_0)}$. Wir können die Debye-Länge nutzen um die Gleichung zu

$$\tilde{\nabla}^2 \tilde{\Phi} = \sinh \tilde{\Phi} \tag{12.12}$$

mit dem entdimensionalisierten Potential $\tilde{\Phi} = \varepsilon\tilde{\Phi}/(2\rho_0\lambda^2)$ und der entdimensionalisierten Länge $\tilde{x} = x/\lambda$ (und $\mathrm{d}/\mathrm{d}\tilde{x} = \lambda\,\mathrm{d}/\mathrm{d}x$) umzuschreiben. In folgenden werden wir mit Gl. (**??**) arbeiten aber der Einfachheit halber die Tilde nicht weiter explizit schreiben.

Wir betrachten im Folgenden die eindimensionale Variante der PB-Gleichung in Interval $[0, L]$. Das Residuum ist

$$R(x) = \frac{\mathrm{d}^2\,\Phi}{\mathrm{d}x^2} - \sinh \Phi. \tag{12.13}$$

Multiplikation mit einer Testfunktion $v(x)$ liefert das gewichtete Residuum

$$\begin{aligned}(v, R) &= (v, \mathrm{d}^2\,\Phi/\,\mathrm{d}x^2) - (v, \sinh \Phi) \\ &= v\frac{\mathrm{d}\Phi}{\mathrm{d}x}\bigg|_0^L - (\mathrm{d}v/\,\mathrm{d}x, \mathrm{d}\Phi/\,\mathrm{d}x) - (v, \sinh \Phi).\end{aligned} \tag{12.14}$$

Der rechte Term ist nichtlinear in Φ und benötigt zur Lösung numerische Quadratur. Im Folgenden vernachlässigen wir weiterhin den Oberflächenterm.

Der nächste Schritt ist der Galerkin-Ansatz, $\Phi(x) \approx \Phi_N(x) = \sum_{n=0}^{N} a_n\varphi_n(x)$ und $v(x) = \varphi_k(x)$. Der Laplace-Operator wird zu

$$(\mathrm{d}\varphi_k/\,\mathrm{d}x, \mathrm{d}\Phi_N/\,\mathrm{d}x) = L_{kn}a_n \tag{12.15}$$

mit der bekannten (konstanten) Laplace-Matrix L_{kn}. (Die Laplace-Matrix ist konstant, da der Laplace-Operator linear ist.) Den nichtlinearen Term rechnen wir mit einer Quadraturregel aus, die wir *pro Element anwenden.* Man erhält für Element (k)

$$(N_I^{(k)}, \sinh \Phi_N) = \sum_i w_i^Q N_I^{(k)}(x_i^{(k)}) \sinh \Phi_N(x_i^{(k)}), \tag{12.16}$$

wobei $x_i^{(k)}$ der i-te Quadraturpunkt auf Element (k) ist.

Für die Lösung des Gleichungssystems $(\varphi_k, R_N) = 0$ mit dem Newton-Raphson-Verfahren brauchen wir die Tangentenmatrix $\underline{K}$. Es gilt,

$$K_{kn} = \frac{\partial(\varphi_k, R_N)}{\partial a_n} = -L_{kn} - \frac{\partial(N_1^{(k-1)}, \sinh \Phi_N)}{\partial a_n} - \frac{\partial(N_0^{(k)}, \sinh \Phi_N)}{\partial a_n} \tag{12.17}$$

mit

$$\frac{\partial(N_I^{(k)}, \sinh \Phi_N)}{\partial a_n} = \sum_i w_i^Q N_I^{(k)}(x_i^{(k)})\varphi_n(x_i^{(k)}) \cosh \Phi_N(x_i^{(k)}). \tag{12.18}$$

Kapitel 13

Festkörpermechanik

Kontext: Die Methode der finiten Elementen hat ihren Ursprung in der Festkörpermechanik (auch *Strukturmechanik*). So gut wie alle Simulationen in diesem Bereich werden weiterhin mit Hilfe diese Methode durchgeführt. In diesem Kapitel werden wir die Grundgleichungen des elastostatischen Gleichgewichts diskretisieren, die eine Form haben die dem in den vorherigen Kapiteln diskutierten Transportproblem ähnlich sind.

13.1 Elastostatisches Gleichgewicht

Die Grundgleichung der Festkörpermechanik beschreibt das elastostatische Gleichgewicht. Gegeben ein Tensorfeld $\underline{\sigma}(\vec{r})$, dass die *mechanische Spannung* im System beschreibt, lauten diese

$$\nabla \cdot \underline{\sigma} = \vec{0} \tag{13.1}$$

Wir berechnen nun die Divergenz eines *Tensors zweiter Ordnung* (bzw. einen Tensor*feldes* zweiter Ordnung) und müssen kurz klarstellen, was wir mit der Operation $\nabla\cdot = \text{div}$ meinen. In Komponentenschreibweise wird Gl. (15.1) zu

$$\partial_i \sigma_{ij} = 0, \tag{13.2}$$

wobei $\partial_i \equiv \partial/\partial r_i$ die Ableitung respektive der Komponente der Position r_i ist und wir hier die *Einsteinsche Summenkonvention* eingeführt haben. Diese Gleichgewichtsbedingung hat ihren Ursprung in der Erhaltung des Impulses; $\underline{\sigma}$ beschreibt drei unabhängige Impulsströme und Gl. (15.1) ist die stationäre Kontinuitätsgleichung, $\nabla \cdot \vec{j} = 0$, für diese drei Impulsströme die durch die Spaltenvektoren des Tensors $\underline{\sigma}$ gegeben sind.

Die mechanische Spannung beschreibt eine Kraft pro Fläche und wird der SI-Einheit *Pascal* gemessen. Wenn wir Gl. (15.1) über eine (beliebiges) Volumenelement $\omega \in \Omega$ integrieren, so erhalten wir

$$\int_{\omega} \mathrm{d}^3 r \, \nabla \cdot \underline{\sigma} = \int_{\partial\omega} \mathrm{d}^2 r \, \underline{\sigma} \cdot \hat{n} = 0 \tag{13.3}$$

nach Anwendung des Gaussschen Satzes, wobei $\partial\omega$ wieder die Oberfläche des Volumenelements ω ist. Der Ausdruck $\underline{\sigma} \cdot \hat{n}$ ist die Projektion der Spannung auf die Oberflächennormale $\hat{n}$ – die sogenannten Flächenlasten (engl. "traction" oder "surface traction"). Der Ausdruck $\mathrm{d}\vec{F} = \underline{\sigma} \cdot \hat{n} \, \mathrm{d}^2 r$ ist damit die (infinitesimale) Kraft $\mathrm{d}\vec{F}$ auf ein Oberflächenelement und das Integral damit die Summe über alle Oberflächenkräfte. Diese Summe muss verschwinden; dies ist ein Ausdruck für das *Kräftegleichgewicht* und damit statisches Gleichgewicht in dem Volumenelement. Weiterhin gilt noch *Gleichgewicht der Drehmomente* in allen Volumenelementen ω. Diese Bedingung führt dazu, dass der Spannungstensor $\underline{\sigma}$ symmetrisch ist, also $\underline{\sigma}^T = \underline{\sigma}$.

Anmerkung: In der Einsteinschen Summenkonvention lässt man Summationssymbole weg und impliziert Summation über wiederholte Indices. Im obigen Beispiel,

$$\partial_i \sigma_{ij} \equiv \sum_i \partial_i \sigma_{ij}. \tag{13.4}$$

Ein Skalarprodukt zwischen den Vektoren $\vec{a}$ und $\vec{b}$ wird in dieser Konvention geschrieben als

$$\vec{a} \cdot \vec{b} = a_i b_i. \tag{13.5}$$

Diese Indexschreibweise ist nützlich, weil sie eindeutig ist. In der dyadischen Notation müssen wir meistens dazu sagen, was wir mit einer Operation meinen. So ist z.B. bei der Divergenz eines Tensors nicht klar, ob die Divergenz auf den ersten oder den zweiten Index wirken soll. (Wir nutzen hier eine Konvention, in der die Divergenz auf den ersten Index wirkt.)

13.2 Hooksche Gesetz

Neben dem physikalischen Grundprinzip, dass die Erhaltung des Impulsstromes und damit das elastostatische Gleichgewicht beschreibt, brauchen wir noch ein Konstitutivgesetz, dass uns sagt wie der Spannungstensor (also der Impulsstrom) auszusehen hat. Hierzu führen wir zunächst die Verschiebungen

$\vec{u}(\vec{r})$ ein, die die Verformung eines Raumpunktes auf Grund der mechanischen Belastung beschrieben. Aus diesen Verschiebungen berechnen wir den Dehnungstensor

$$\underline{\varepsilon} = \frac{1}{2}\left[\nabla\vec{u} + (\nabla\vec{u})^T\right], \tag{13.6}$$

bzw. in Komponentenschreibweise

$$\varepsilon_{ij} = \frac{1}{2}\left(\partial_i u_j + \partial_j u_i\right). \tag{13.7}$$

Anmerkung: Der Ausdruck $\nabla\vec{u}$ ist *nicht* die Divergenz von $\vec{u}$. Diese ist gegeben durch $\nabla\cdot\vec{u}$ – der Punkt ist entscheidend. $\nabla\vec{u}$ ist der Gradient des Vektorfeldes $\vec{u}(\vec{r})$ und damit ein Tensorfeld zweiter Ordnung. Die Komponenten sind gegeben durch

$$[\nabla\vec{u}]_{ij} = \partial_i u_j. \tag{13.8}$$

Die Komponenten des transponierten Tensors sind daher

$$[\nabla\vec{u}]^T_{ij} = [\nabla\vec{u}]_{ji} = \partial_j u_i. \tag{13.9}$$

Expliziter könnte man den Gradienten eines Vektorfeldes mit Hilfe des äußeren Produkts schreiben, es gilt

$$\nabla\vec{u} = \nabla\otimes\vec{u} \tag{13.10}$$

mit äußerem Produkt $[\vec{a}\otimes\vec{b}]_{ij} = a_i b_j$. In der Literatur findet sich allerdings meistens die Form $\nabla\vec{u}$.

Das Hooksche Gesetz beschreibt nun welche Dehnung zu welcher Spannung führt. Für isotrope Festkörper lautet es

$$\underline{\sigma} = \left(\lambda\,\mathrm{tr}\,\underline{\varepsilon}'\right)\underline{1} + 2\mu\underline{\varepsilon}' \quad \text{bzw.} \quad \sigma_{ij} = \lambda\delta_{ij}\varepsilon'_{kk} + 2\mu\varepsilon'_{ij}. \tag{13.11}$$

mit $\varepsilon'_{ij} = \varepsilon_{ij} - \varepsilon_{0,ij}$. Hierbei heißen λ und μ *Lamé-Konstanten*. Die Konstante μ wird auch das Schermodul genannt. Der Ausdruck $\varepsilon'_{kk} \equiv \sum_k \varepsilon'_{kk} = \mathrm{tr}\,\underline{\varepsilon}'$ (Einstein-Konvention!) ist die Spur des Dehnungstensors. Wir haben hier den Dehnungstensor $\underline{\varepsilon}' = \underline{\varepsilon} - \underline{\varepsilon}_0$ eingeführt, der eine Eigendehnung $\underline{\varepsilon}_0$ (engl. "eigenstrain") berücksichtig, die z.B. durch eine thermische Expansion gegeben sein kann.

13.3 Schwache Form

Wir haben ein gekoppeltes lineares Gleichungssystem zweiter Ordnung, welches wir nun mit Hilfe der Methode der finiten Elemente diskretisieren werden. Wir formulieren zunächst die schwache Form von Gl. (15.1),

$$\int_\Omega \mathrm{d}^3 r\, \vec{v}(\vec{r}) \cdot (\nabla \cdot \underline{\sigma}) = 0 \quad \text{bzw.} \quad \int_\Omega \mathrm{d}^3 r\, v_j(\vec{r}) \partial_i \sigma_{ij} = 0, \tag{13.12}$$

wobei $\vec{v}(\vec{r})$ nun ein Testvektor ist. Wir nutzen nun wieder eine Variante der Produktregel,

$$\partial_i \left(\sigma_{ij} v_j\right) = \left(\partial_i \sigma_{ij}\right) v_j + \sigma_{ij} \left(\partial_i v_j\right), \tag{13.13}$$

um die Ableitung auf die Testfunktion zu überführen. Man erhält

$$\int_\Omega \mathrm{d}^3 r\, \left[\nabla \cdot (\underline{\sigma} \cdot \vec{v}) - (\nabla \vec{v}) : \underline{\sigma}\right] = 0, \tag{13.14}$$

wobei der Doppelpunkt ein doppeltes Skalarprodukt (oder *Kontraktion*) bezeichnet, $\underline{A} : \underline{B} = A_{ij} B_{ij}$. Wir können nun wieder den ersten Term in Gl. (15.12) mit Hilfe des Gaussschen Satzes in ein Oberflächenintegral,

$$\int_\Omega \mathrm{d}^2 r\, \vec{v} \cdot (\underline{\sigma} \cdot \hat{n}) - \int_\Omega \mathrm{d}^3 r\, (\nabla \vec{v}) : \underline{\sigma} = 0, \tag{13.15}$$

überführen, wobei die Tatsache genutzt wurde, dass $\underline{\sigma}$ symmetrisch ist. Dies ist die schwache Form mit verringerter Differenzierbarkeitsanforderung des mechanischen Gleichgewichts. Der Ausdruck $\underline{\sigma} \cdot \hat{n}$ sind wiederum die Flächenlasten auf der Oberfläche.

13.4 Diskretisierung

Wir diskretisieren diese Gleichung nun mit Hilfe der Galerkin-Methode und setzen gleich lineare finite Elemente als Basisfunktionen an. Dies erlaubt es uns, die Diskretisierung direkt mit Hilfe der Formfunktionen und nicht der Basisfunktionen zu schreiben. Wir setzen also innerhalb Element (n) an, dass

$$u_j(\vec{r}) = a_{j,Jn} N_J^{(n)}(\vec{r}), \tag{13.16}$$

wobei Jn für den globalen Knotenindex steht, der dem lokalen Index des Knotens J auf Element (n) entspricht und Summation über J auf Grund des wiederholten Indices impliziert ist.

Unsere Testfunktion $\vec{v}(\vec{r})$ ist nun vektorwertig. Wir benötigen damit einen Satz von DN Testvektoren, wobei D die Dimension des Raumes und N die

Anzahl der Elemente bezeichnet. Wir setzen daher innerhalb des Elements (n)

$$\vec{v}_{i,I}(\vec{r}) = N_I^{(n)}(\vec{r})\hat{e}_i \tag{13.17}$$

mit $i \in [1, \ldots, D]$ als Testvektor an. Hier bezeichnet $\hat{e}_i$ den Vektor, bei dem die i-te Komponente 1 und alle anderen Komponenten 0 sind, also den kanonischen Einheitsvektor in Richtung i.

Mit diesen Ansatzfunktionen erhält man für die Dehnung auf Element (n)

$$\varepsilon_{ij}^{(n)} = \frac{1}{2}\left(a_{i,Jn}\partial_j N_J^{(n)} + a_{j,Jn}\partial_i N_J^{(n)}\right). \tag{13.18}$$

Das Hooksche Gesetz wird damit zu

$$\sigma_{ij}^{(n)}(\vec{r}) = \lambda(\vec{r})\delta_{ij}a_{k,Jn}\partial_k N_J^{(n)} + \mu(\vec{r})\left(a_{i,Jn}\partial_j N_J^{(n)} + a_{j,Jn}\partial_i N_J^{(n)}\right) - \sigma_{0,ij}(\vec{r}) \tag{13.19}$$

mit

$$\sigma_{0,ij}(\vec{r}) = \lambda(\vec{r})\delta_{ij}\varepsilon_{0,kk}(\vec{r}) + 2\mu(\vec{r})\varepsilon_{0,ij}(\vec{r}). \tag{13.20}$$

Die diskretisierte Form des Volumenterms der Gleichgewichtsbedingung lautet

$$\begin{aligned}
\int \mathrm{d}^3 r\ (\nabla\vec{v}) : \underline{\sigma} &= \int \mathrm{d}^3 r\ \left(\nabla N_I^{(n)} \otimes \hat{e}_i\right) : \underline{\sigma} \\
&= \int \mathrm{d}^3 r\ \left(\partial_j N_I^{(n)}\right)\sigma_{ji} \\
&= \int \mathrm{d}^3 r\ \Big\{\lambda(\vec{r})a_{k,Jn}\partial_i N_I^{(n)}\partial_k N_J^{(n)} \\
&\quad + \mu(\vec{r})\left(a_{i,Jn}\partial_j N_I^{(n)}\partial_j N_J^{(n)} + a_{j,Jn}\partial_j N_I^{(n)}\partial_i N_J^{(n)}\right) \\
&\quad - \sigma_{0,ji}(\vec{r})\partial_j N_I^{(n)}\Big\}.
\end{aligned} \tag{13.21}$$

Für lineare Elemente sind bis auf die Materialkonstanten alle Terme in dieser Gleichung konstant. Das Integral führt damit effektiv zu einer Mittelung der Materialkonstanten auf den Element. Wir führen die mittleren Materialkonstanten

$$\lambda^{(n)} = \frac{1}{V^{(n)}}\int \mathrm{d}^3 r\, \lambda(\vec{r}) \tag{13.22}$$

und eine äquivalent Gleichung für μ und $\underline{\sigma}_0$ ein. Hierbei ist nun $V^{(n)}$ das Volumen des Elements (n). Wir können Gl. (15.19) schreiben als

$$\begin{aligned}
\int \mathrm{d}^3 r\ (\nabla\vec{v}) : \underline{\sigma} =& \lambda^{(n)}a_{j,Jn}V^{(n)}\partial_i N_I^{(n)}\partial_j N_J^{(n)} + \mu^{(n)}a_{j,Jn}\delta_{ij}V^{(n)}\partial_k N_I^{(n)}\partial_k N_J^{(n)} \\
&+ \mu^{(n)}a_{j,Jn}V^{(n)}\partial_j N_I^{(n)}\partial_i N_J^{(n)} - V^{(n)}\sigma_{0,ij}^{(n)}\partial_j N_I^{(n)}.
\end{aligned} \tag{13.23}$$

Die Elementmatrix hat daher die Komponenten

$$K^{(n)}_{IiJj} = \lambda^{(n)} k^{(n)}_{IiJj} + \mu^{(n)} \delta_{ij} k^{(n)}_{IkJk} + \mu^{(n)} k^{(n)}_{IjJi} \tag{13.24}$$

mit $k^{(n)}_{IiJj} = V^{(n)} \partial_i N^{(n)}_I \partial_j N^{(n)}_J$ und der Beitrag des Elements zum Lastvektor lautet $f^{(n)}_{Ii} = V^{(n)} \sigma^{(n)}_{0,ij} \partial_j N^{(n)}_I$.

Anmerkung: Für unser strukturiertes zweidimensionales Gitter erhalten wir $V^{(n)} = \Delta x \Delta y/2$ und

$$\underline{k} = \frac{1}{2} \begin{pmatrix} \Delta y/\Delta x & 1 & -\Delta y/\Delta x & \cdot & \cdot & -1 \\ 1 & \Delta x/\Delta y & -1 & \cdot & \cdot & -\Delta x/\Delta y \\ -\Delta y/\Delta x & -1 & \Delta y/\Delta x & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ -1 & -\Delta x/\Delta y & 1 & \cdot & \cdot & \Delta x/\Delta y \end{pmatrix} \tag{13.25}$$

sowohl für das untere linke als auch das obere rechte Dreieck. Der Lastvektor wird zu

$$\vec{f} = \frac{1}{2} \begin{pmatrix} -\sigma_{0,xx}\Delta y - \sigma_{0,xy}\Delta x \\ -\sigma_{0,xy}\Delta y - \sigma_{0,yy}\Delta x \\ \sigma_{0,xx}\Delta y \\ \sigma_{0,xy}\Delta y \\ \sigma_{0,xy}\Delta x \\ \sigma_{0,yy}\Delta x \end{pmatrix} \tag{13.26}$$

für das untere linke Dreieck. Der Lastvektor für das obere rechte Dreieck ist $-\vec{f}$. Für $\Delta x = \Delta y$ wird die Elementsteifigkeitsmatrix zu

$$\underline{k} = \frac{1}{2} \begin{pmatrix} 1 & 1 & -1 & \cdot & \cdot & -1 \\ 1 & 1 & -1 & \cdot & \cdot & -1 \\ -1 & -1 & 1 & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ -1 & -1 & 1 & \cdot & \cdot & 1 \end{pmatrix}. \tag{13.27}$$

Wenn wir hier die Spur von Untermatrizen der Größe 2×2 berechnen, erhalten wir die Matrix des Laplace-Operators, Gl. (14.10).

Die Dehnung auf Element (n) wird zu

$$\varepsilon_{xx}^{(n)} = a_{x,Jn}\partial_x N_J^{(n)} = -a_{x,0n}/\Delta x + a_{x,1n}/\Delta x \tag{13.28}$$

$$\varepsilon_{yy}^{(n)} = a_{y,Jn}\partial_y N_J^{(n)} = -a_{y,0n}/\Delta_y + a_{y,2n}/\Delta y \tag{13.29}$$

$$\begin{aligned}\varepsilon_{xy}^{(n)} &= \frac{1}{2}\left(a_{x,Jn}\partial_y N_J^{(n)} + a_{y,Jn}\partial_x N_J^{(n)}\right) \\ &= \frac{1}{2}\left(-a_{x,0n}/\Delta y + a_{x,2n}/\Delta y - a_{y,0n}/\Delta x + a_{y,1n}/\Delta x\right)\end{aligned} \tag{13.30}$$

für das linke untere Dreieck.

Kapitel 14

Zeitabhängige Probleme

Kontext: Viele Probleme denen wir begegnen, wie z.B. der bereits diskutierte Diffusionsprozess, sind zeitabhängig. Wir haben bislang nur die stationäre Lösung von linearen Problemen behandelt. Eine Behandlung des Anfangswertproblems, welches die Zeitabhängigkeit *explizit* beinhaltet, benötigt entsprechende Integrationsalgorithmen.

14.1 Anfangswertprobleme

Typische zeitabhängige PDGLs haben die Form,

$$\frac{\partial u}{\partial t} + \mathcal{L}u(\vec{r}, t) = f(\vec{r}, t), \tag{14.1}$$

wobei $\mathcal{L}$ ein irgendwie gearteter Operator ist, z.b. $\mathcal{L} = -\nabla \cdot D\nabla$ für Diffusionsprozesse. Die dazugehörige stationäre Lösung, die üblicherweise für $t \to \infty$ erreicht wird, ist durch $\mathcal{L}u = 0$ gegeben. Wir haben in den letzten Kapiteln gelernt, wie man eine solche stationäre Lösung für lineare Probleme numerisch berechnen kann.

Gleichung (12.1) ist ein *Anfangswertproblem* (engl. "initial value problem"), weil man das Feld $u(\vec{r}, t)$ zu einem Zeitpunkt (üblicherweise $t = 0$) festlegen muss. Man integriert dann diese Anfangswertprobleme von diesem Zeitpunkt in die Zukunft. Eine Auswahl solcher Zeitintegrationsalgorithmen (engl. "time marching") werden in diesem Kapitel besprochen. Bevor wir dazu kommen, müssen wir jedoch zunächst noch einmal zur Diskretisierung der räumlichen Ableitungen kommen.

14.2 Räumliche Ableitungen

Wir approximieren nun wiederum die (nun zeitabhängige) Funktion $u(\vec{r}, t)$ durch eine Reihenentwicklung $u_N(\vec{r})$. Im Unterschied zu den vorhergehenden Kapiteln nehmen wir nun an, dass die Koeffizienten nicht mehr konstant sondern zeitabhängig sind. D.h. wir schreiben

$$u_N(\vec{r}, t) = \sum_n a_n(t)\varphi_n(\vec{r}). \tag{14.2}$$

Nun multiplizieren wir die gesamte zeitabhängige PDGL Gl. (12.1) mit den Basisfunktionen $\varphi_n(\vec{r})$ der Reihenentwicklung,

$$\frac{\partial}{\partial t}(\varphi_n, u_N) + (\varphi_n, \mathcal{L}u_N) = (\varphi_n, f), \tag{14.3}$$

In den vorhergehenden Kapiteln, haben wir bereits gelernt, wie wir die beiden rechten Terme dieser Gleichung ausrechnen müssen,

$$(\varphi_n, \mathcal{L}u_N) = \sum_k K_{nk}a_k(t), \tag{14.4}$$

$$(\varphi_n, f) = f_n(t) \tag{14.5}$$

wobei $\underline{K}$ die bekannte Systemmatrix ist und $\vec{f}(t)$ der (nun potentiell zeitabhängige) Lastvektor. Nun ziehen wir die Zeitableitung in das Skalarprodukt hinein. Man erhält damit

$$\sum_k M_{nk}\frac{\mathrm{d}a_k}{\mathrm{d}t} + \sum_k K_{nk}a_k(t) = f_n(t), \tag{14.6}$$

mit $M_{nk} = (\varphi_n, \varphi_k)$, der *Massenmatrix*. Dies ist ein System gekoppelter *gewöhnlicher* Differentialgleichungen. Wir haben also die PDGL durch die räumliche Diskretisierung in ein *System* von GDGLs umgewandelt. Für eine Fourier-Basis ist die Massenmatrix diagonal, für eine finite-Elemente Basis ist diese dünnbesetzt aber nicht mehr diagonal. Wir können aber Gl. (12.6) formal von links mit $\underline{M}^{-1}$ multiplizieren und erhalten,

$$\frac{\mathrm{d}\vec{a}}{\mathrm{d}t} = -\underline{M}^{-1} \cdot \underline{K} \cdot \vec{a}(t) + \underline{M}^{-1} \cdot \vec{f}(t) \equiv \vec{g}(\vec{a}(t), t). \tag{14.7}$$

Anmerkung: Da sich die Massematrix in der Zeitentwicklung nicht ändert, kann $\underline{M}^{-1}$ hier einmal vorberechnet werden. Da $\underline{M}$ üblicherweise

dünnbesetzt ist, kann es aber auch numerisch effizienter sein, in jedem Schritt das entsprechende Gleichungssystem zu lösen. Dies liegt daran, dass die Inverse einer dünnbesetzten Matrix nicht mehr dünnbesetzt ist. Damit braucht die Multiplikation mit $\underline{M}^{-1} \sim N^2$ Operationen, während das Lösen des Gleichungssystems nur $\sim N$ Operationen benötigt. Die Anzahl der benötigten Operationen nennt man die *Komplexität* eines Algorithmus.

Beispiel: Für die Basis der finite Elemente kann die Massematrix wieder als Summe über entsprechende Elementmatrizen $\underline{M}^{(n)}$ ausgedrückt werden. Die Komponenten dieser Elementmassematrizen sind durch

$$M_{IJ}^{(n)} = (N_I^{(n)}, N_J^{(n)}) \tag{14.8}$$

gegeben. Man erhält beispielsweise

$$\begin{aligned} M_{11}^{(0)} = M_{22}^{(0)} &= \Delta x \Delta y \int_0^1 \mathrm{d}\xi \int_0^{1-\xi} \mathrm{d}\eta \, \xi^2 \\ &= \Delta x \Delta y \int_0^1 \mathrm{d}\xi \, \xi^2 (1-\xi) \\ &= \Delta x \Delta y / 12, \end{aligned} \tag{14.9}$$

wobei die Integrationsgrenzen für das Integral über das Dreieck gewählt sind. Die weiteren Integrale können entsprechend ausgeführt werden. Hiermit bekommt man

$$\underline{M}^{(n)} = \frac{\Delta x \Delta y}{24} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}. \tag{14.10}$$

Der Ausdruck ist für beide Elementmassematrizen identisch. Die globale Systemmassematrix erhält man auf dem gleichen Weg wie die Systemmatrix.

14.3 Runge-Kutta Methoden

14.3.1 Euler-Verfahren

Gleichung (12.7) kann in der Zeit propagiert werden. Wir nehmen an $a_n(t)$ sei bekannt, dann können wir $a_n(t + \Delta t)$ um t in eine Taylorreihe entwickeln. Dies ergibt

$$\vec{a}(t + \Delta t) = \vec{a}(t) + \Delta t \vec{g}(\vec{a}(t), t) + \mathcal{O}(\Delta t^2), \tag{14.11}$$

wobei $\mathcal{O}(\Delta t^2)$ für quadratische und höhere Terme steht, die hier vernachlässigt werden. Gleichung (12.11) kann direkt eingesetzt werden, um die Koeffizienten $\vec{a}$ einen Schritt Δt in die Zukunft zu propagieren. Der Euler-Algorithmus ist allerdings nicht sonderlich stabil und verlangt sehr kleine Zeitschritte Δt.

14.3.2 Heun-Verfahren

Basierend auf der Euler Integration kann ein einfaches Verfahren mit höherer Konvergenzordnung konstruiert werden. Die Konvergenzordnung besagt, wie sich der Fehler verringert wenn die Schrittweite reduziert wird. Bei einem Verfahren erster Ordnung reduziert sich der Fehler linear mit der Schrittweite, bei einem Verfahren zweiter Ordnung quadratisch.

Im Heun-Verfahren schätzt man zunächst den Funktionswert zum Zeitpunkt $t + \Delta t$ mit dem Euler-Verfahren ab. Man berechnet also

$$\tilde{\vec{a}}(t + \Delta t) = \vec{a}(t) + \Delta t \vec{g}(\vec{a}(t), t), \tag{14.12}$$

und benutzt dann die Trapezregel mit diesem abgeschätzten Funktionswert um die Funktion einen Zeitschritt Δt zu integrieren:

$$\vec{a}(t + \Delta t) = \vec{a}(t) + \frac{\Delta t}{2} \left(\vec{g}(\vec{a}(t), t) + \vec{g}(\tilde{\vec{a}}(t + \Delta t), t) \right) \tag{14.13}$$

Das Heun-Verfahren hat quadratische Konvergenzordnung. Verfahren die zunächst Funktionswerte schätzen und dann korrigieren nennt man auch *Predictor-Corrector* Verfahren.

14.3.3 Automatische Schrittweitenkontrolle

Mit Hilfe zweier Integrationsverfahren mit unterschiedlicher Konvergenzordnung lässt sich eine automatische Schrittweitenkontrolle realisieren, in der der Zeitschritt Δt so angepasst wird, dass ein bestimmer Fehler nicht überschritten wird. Die Verfahren sind insbesondere dann interessant, wenn die Berechnungen der niedrigen Fehlerordnung in der Berechnung der höheren Fehlerordnung

wiederverwendet werden kann, wie dies z.B. bei dem Heun-Verfahren der Fall ist.

Für eine Kombination zweier Verfahren (z.B. Euler und Heun), ergibt sich eine Abschätzung des Fehlers aus der Differenz der beiden Integrationen. Unter Vorgabe einer globalen Fehlerschranke, kann dann der Zeitschritt so angepasst werden, dass der Fehler immer unterhalb dieser Schranke bleibt.

Anmerkung: Sowohl die Euler-Integration als auch das Heun-Verfahren gehören zur Klasse der *Runge-Kutta Methoden*. Es gibt eine ganze Reihe von Runge-Kutta Methoden mit unterschiedlichen Konvergenzordnungen. Interessant sind insbesondere Methoden mit automatische Schrittweitenkontrolle wie hier beispielhaft beschrieben. In `scipy` sind insbesondere Verfahren mit den Konvergenzordnungen 2/3 und 4/5 implementiert. Diese können über die Funktion `scipy.integrate.solve_ivp` genutzt werden.

14.4 Stabilitätsanalyse

Zeitpropagationsverfahren werden bei zu hohen Zeitschritten instabil. Eine Schrittweitenkontrolle ist eine automatisierte Methode solche Instabilitäten zu verhindern.

Um zu verstehen, warum solche Instabilitäten auftreten analysieren wir nun beispielhaft die eindimensionale Diffusionsgleichung,

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2}. \tag{14.14}$$

Eine Diskretisierung der räumlichen Ableitung mit linearen finiten Elementen führt zu

$$\frac{\partial c}{\partial t} = \frac{D}{\Delta x^2}\left(c(x-\Delta x) - 2c(x) + c(x+\Delta x)\right). \tag{14.15}$$

Anmerkung: Eigenlich müsste auf der linken Seite von Gl. (12.15) die Massematrix auftauchen. Wir vernachlässigen diese hier und approximieren $\underline{M} = \underline{1}$. Diese Art diskretisierter Gleichung erhält man durch eine Diskretisierung mit der Methode der finiten Differenzen.

Wir schreiben nun die Funktion $c(x)$ als Entwicklung in eine Fourier-Basis, also

$$c(x) = \sum_n c_n \exp(ik_n x). \tag{14.16}$$

Damit werden Term der Form $c(x + \Delta x)$ zu

$$c(x + \Delta x, t) = \sum_n c_n(t) \exp\left[ik_n(x + \Delta x)\right] = \sum_n \exp(ik_n \Delta x) c_n(t) \exp(ik_n x). \tag{14.17}$$

Wir schreiben nun Gl. (12.15) als

$$\begin{aligned} \frac{\partial c_n}{\partial t} &= \frac{D}{\Delta x^2} \left(\exp(-ik_n \Delta x) - 2 + \exp(ik_n \Delta x)\right) c_n(t) \\ &= \frac{2D}{\Delta x^2} \left(\cos(k_n \Delta x) - 1\right) c_n(t), \end{aligned} \tag{14.18}$$

Diese Gleichung können wir aber analytisch für ein Zeitinterval Δt lösen,

$$c_n(t + \Delta t) = c_n(t) \exp\left[\frac{2D}{\Delta x^2} \left(\cos(k_n \Delta x) - 1\right) \Delta t\right], \tag{14.19}$$

wohingegen Euler Integration

$$c_n(t + \Delta t) \approx \left[1 + \frac{2D}{\Delta x^2} \left(\cos(k_n \Delta x) - 1\right) \Delta t\right] c_n(t) \tag{14.20}$$

liefert. Der Wert des Terms $\cos(k_n \Delta x) - 1$ liegt zwischen -2 und 0. D.h. wir können Gl. (12.19) für beliebige Δt propagieren, ohne dass die Konzentration $c_n(t)$ zeitlich divergiert. Außer für $k_n = 0$ werden die Koeffizienten $c_n(t)$ zeitlich kleiner.

Für das Euler-Verfahren, Gl. (12.20), ist dies aber nur den Fall, wenn

$$\mu = \frac{D \Delta t}{\Delta x^2} < \frac{1}{2}. \tag{14.21}$$

Für $\mu > 1/2$ wachsen einige der Koeffizienten $c_n(t)$ mit t an, der Algorithmus wird instabil. Die dimensionslose Zahl μ nennt sich eine Courant-Friedrich-Lewy-(CFL)-Zahl und die Bedingung Gl. (12.21) eine *CFL-Bedingung*. Die exakte Form der CFL-Bedingung hängt von der PDGL und dem Algorithmus ab.

Anmerkung: Die CFL-Bedingung sagt, dass der maximale Zeitschritt

$$\Delta t < \frac{1}{2D} \Delta x^2 \tag{14.22}$$

von der räumlichen Diskretisierung Δx abhängt. D.h. wenn wir die räumliche Diskretisierung feiner machen, müssen wir auch einen kleineren Zeitschritt wählen. Eine Halbierung der räumlichen Diskretierung

braucht einen Zeitschritt der um ein viertel kleiner ist. Dies erhöht die Kosten einer Simulation der gleichen Simulationsdauer um einen Faktor 8. Fein aufgelöste Simulationen werden also schnell numerisch aufwändig. Für Verfahren mit automatischer Schrittkontrolle passiert diese Anpassung des Zeitschritts natürlich automatisch.

Literaturverzeichnis

J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, New York, 2000. 05628.

A. Einstein. Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Ann. Phys.*, 17:549, 1905.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd ed edition, 2006.