Skriptum

Grundlagen der numerischen Thermofluiddynamik

Dipl.-Ing. Thomas Steinbacher

Kilian Förner, M. Sc.

Prof. Wolfgang Polifke, Ph. D.

Winter 2016/ 17

www.tfd.mw.tum.de

12. Oktober 2016

Motivation und Ziele des Praktikums

In Forschung und Entwicklung spielen *computerunterstützte* Methoden eine zentrale Rolle. Im Englischen kennt man in diesem Zusammenhang z.B. *computer aided design* oder *computer aided engineering*, die Abkürzungen CAD und CAE sind auch im Deutschen gebräuchlich. Es gibt natürlich auch computerunterstützte Mathematik. Dieser Begriff ist allerdings nicht weit verbreitet[1], man spricht meist von *numerischer Mathematik* oder allgemeiner *numerischen Methoden*. Ersteres benennt meist die näherungsweise numerische Berechnung der Lösung mathematischer Gleichungen, Letzteres umfasst z.B. auch die Aufbereitung und Visualisierung experimenteller Rohdaten. Eng verwandt und mit zunehmender Leistungsfähigkeit der Computer immer wichtiger ist die *numerische Simulation*. Hier ist das Ziel, Prozesse in Natur und Technik mittels numerischer Modelle detailgetreu und möglichst genau nachzubilden, um mit den so gewonnenen Daten z.B. Vorhersagen zu treffen. Bekanntes Beispiel sind Wetter- und Klimaprognosen. In den Ingenieurwissenschaften ist der Einsatz der numerischen Simulation mit der Erwartung verbunden, dadurch langwierige und teure Experimente an Prototypen zu vermeiden oder zumindest deren Zahl zu reduzieren.

Numerische Methoden sind älter als man vielleicht denkt. Im alten Griechland berechnete Archimedes den Zahlenwert der Kreiszahl π näherungsweise mit einer Genauigkeit von zwei Nachkommastellen. Algorithmen zur numerischen Integralrechnung, also „Verfahrensrezepte“ zur näherungsweisen Berechnung des Zahlenwerts von Flächeninhalten, waren damals ebenfalls schon bekannt. Auch Johann Carl Friedrich Gauß befasste sich mit der numerischen Integration; die „Gaußsche Quadratur“ wird heute noch verwendet. Verfahren zur numerischen Lösung linearer Gleichungssysteme und nichtlinearer Minimierungsprobleme sind ebenfalls mit dem Namen von Gauß verbunden – mehr davon später.

Das Abarbeiten numerischer Algorithmen mit „Papier und Bleistift“ (oder gar Pergament und Federkiel) war freilich mühsam und fehlerbehaftet. Mechanische und ab den 1930ern elektronische Rechenmaschinen haben die weitere Verbreitung und Entwicklung der *Numerik* stark vorangetrieben. So gibt es heute in den Ingenieurwissenschaften für viele Fachgebiete und Aufgabenstellungen hoch entwickelte und komfortabel zu bedienende Softwarelösungen. Nichtsdestotrotz benötigt man zum erfolgreichen Einsatz dieser Werkzeuge ebenso wie zur fachgerechten Interpretation der Ergebnisse sowohl das Verständnis der relevanten physikalischen Phänomene bzw. der zugehörigen Modellvorstellungen, als auch Kenntnisse über Wirkungsweise und Grenzen der verwendeten numerischen Algorithmen.

Das Praktikum *Grundlagen der numerischen Thermofluiddynamik* vermittelt Kenntnisse über die wichtigsten numerischen Algorithmen sowie die Prinzipien guten Programmierens. Die Algorithmen werden von den Teilnehmern in der Programmierumgebung MATLAB implementiert

[1] CAM steht denn auch für *computer aided manufacturing*.

und sogleich benutzt, um Beispielprobleme aus der Thermofluiddynamik und insbesondere der Wärmeübertragung zu behandeln. Die Verfahrensschritte von der Modellformulierung über die Implementierung eines numerischen Algorithmus bis hin zur Validierung und grafischen Aufbereitung werden dabei wiederholt geübt. Anhand der Beispielprobleme werden zentrale Begriffe der Numerik – wie Stabilität, Konsistenz und Konvergenz – vermittelt. Die Vor- und Nachteile unterschiedlicher Verfahren werden ebenfalls diskutiert, was eine differenzierte Interpretation und kritische Bewertung numerischer Ergebnisse ermöglicht.

Nach erfolgreichem Abschluss des Praktikums sind die Teilnehmer in der Lage, grundlegende numerische Algorithmen in der Programmiersprache MATLAB zu implementieren, bzw. bereits in MATLAB implementierte Verfahren mit Sachverstand einzusetzen. Sie können die Lösungswege ausgewählter Probleme der Thermofluiddynamik nachvollziehen und das Erlernte auf analoge Probleme aus allen Bereichen der Ingenieurwissenschaften anwenden. Insgesamt wird so eine solide Basis für den kompetenten Umgang mit Softwarepaketen für die Simulation von z.B. (strömungs-)mechanischen oder thermofluiddynamischen Phänomenen geschaffen.

- Prof. Wolfgang Polifke, Ph.D.

Hinweise zum Ablauf

Es finden acht reguläre Termine statt: Jeder Termin beginnt mit einem einführenden Vortrag, welcher die theoretischen Hintergründe vermittlet sowie Hinweise zur praktischen Umsetzung gibt (ca. 1/5). Ab der zweiten Woche werden die Lehrinhalt der Vorwoche in Kurzvorträgen (5 min.) von den Studierenden wiederholt. Anschließend werden unter Betreuung im Studentenrechnerraum des Lehrstuhls klar definierte Problemstellungen in 2er-Gruppen bearbeitet (ca. 4/5).

Anschließend an diese Termine werden individuelle Abschlussprojekt definiert und bearbeitet: Innerhalb ca. eines Monats werden von den Studierenden gewählte Projekte in 2er-Gruppen selbstständig bearbeitet (Problemdefinition, Implementierung in MATLAB, Aufbereitung der Daten sowie Validierung der Ergebnisse). Im Rahmen von regelmäßigen Sprechstunden können Probleme und Fragen geklärt werden. Die Ergebnisse der Projektarbeiten werden abschließend allen Teilnehmern des Praktikums präsentiert (schriftliche Ausarbeitung und Vortrag).

Inhaltsverzeichnis

1 Grafische Ausgabe und Interpolation

Literaturverzeichnis

[1] Brian P Flannery, William H Press, Saul A Teukolsky und William Vetterling. *Numerical Recipes in C*. 1992. URL: http://www.nr.com/.

Lernziele

- Erstellen von 2D und 3D Plots
- Interpolation/ Extrapolation mittels Polynome
- Grundidee kubischer Splines

Motivation

Die graphische Darstellung von Daten ist oft von großer Bedeutung zum Verständnis eines Phänomens oder zur Präsentation von Ergebnissen. Deshalb ist es besonders wichtig die Erstellung aussagekräftiger Plots zu beherrschen. Im Rahmen dieses Kapitels wird daher zunächst kurz das Plotten von Datenreihen wiederholt, wobei speziell auf den Umgang mit MATLAB eingegangen wird. Der Anspruch hierbei ist es eher bereits vorhandenen Kenntnisse zu festigen, als eine umfassende Einführung in das Erstellen von Grafiken zu bieten.

In der Numerik können immer nur diskrete Datenpunkte verarbeitet werden. Um Aussagen bzgl. eines Funktionsverlaufes zwischen zwei gespeicherten Punkten treffen zu können, ist es notwendig zwischen diesen zu interpolieren. Hierbei werden jedoch implizit Annahmen über einen Funktionsverlauf getroffen, welche man genau kennen sollte. Oftmals wird dieser z.B. als "stetig" und "glatt" angenommen. Aus diesem Grund werden in diesem Kapitel zwei wichtige Obergruppen der Interpolation vorgestellt: Die Interpolation mittels Polynomen und jene mittels sogenannter Splines.

1.1 Plotten von Datenreihen

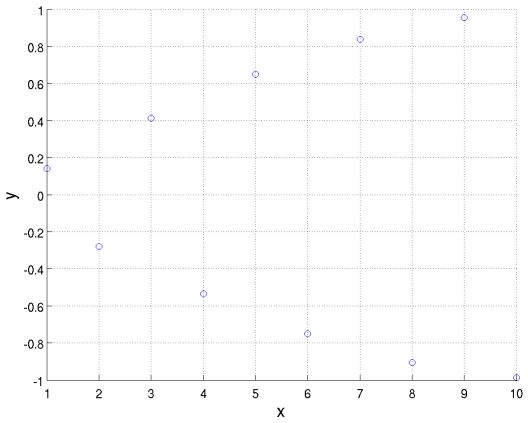
Gegeben seien zwei Datenreihen x und y, welche als Vektoren in MATLAB gespeichert sind und dieselbe Dimension n besitzen. Es wird angenommen, dass alle Elemente aus dem Vektor x über die Funktion f mit dem jeweiligen y-Wert zusammenhängen: $y_i = f(x_i)\ \forall i = 1, 2, \ldots, n$ (in MATLAB: `i=1:n; y(i)=f(x(i))`). Das i-te Element in x, x_i, in MATLAB als `x(i)` schreibbar, korrespondiert dabei mit y_i dem i-ten Element von y bzw. `y(i)`. Dadurch werden n Punkte (x_i, y_i) (MATLAB: `[x(i),y(i)]`) in der x-y-Ebene definiert.

Es bestehen nun verschieden Möglichkeiten diese Datenreihen zu visualisieren: Zum einen können die einzelnen Punkte direkt in die x-y-Ebene eingezeichnet werden. Hierfür kann z.B. die Funktion `scatter(x,y)` verwendet werden. Anhand des Beispiels für `x=0:1:10` und `y=sin(3*x)` ist das Ergebnis hierfür in Abbildung 1.1(a) veranschaulicht.

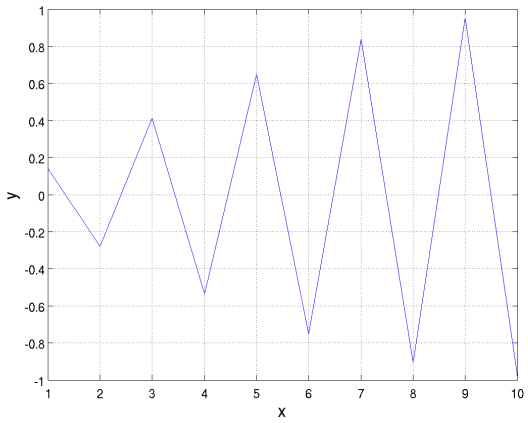
Zum anderen kann ein sogenannter Line-Plot verwendet werden, bei dem die einzelnen Datenpunkte mittels einer Linie verbunden werden. Die Funktion hierfür heißt `plot(x,y)`. Das Ergebnis ist in Abbildung 1.1(b) dargestellt.

Es fällt auf, dass in keinem der beiden Plots die zugrunde liegende Sinus-Funktion $f(x) = sin(3*x)$ erkennbar ist - ganz im Gegenteil - es wird eine periodische Funktion mit wachsender Amplitude suggeriert. Aus diesem Grund soll nun die Auflösung erhöht werden indem der x-Vektor verfeinert wird: `x=0:0.1:10`. Das Ergebnis ist in Abbildung 1.1(c) gezeigt.

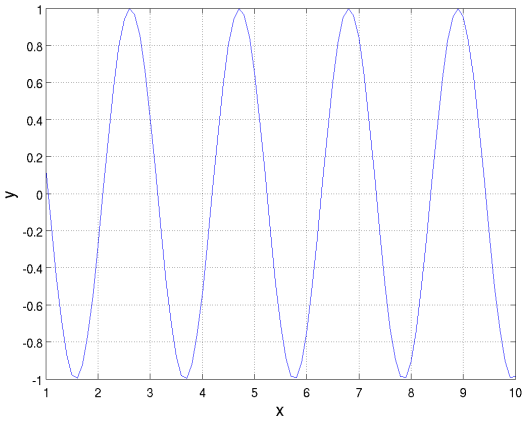
In allen drei eben dargelegten Fällen wurden Datenpunkte geplottet, die jeweils mit ein und der selben Funktion f erzeugt wurden. Die Ergebnisse waren jedoch grundverschieden. Es sollte daher immer im Hinterkopf behalten werden, dass in MATLAB nie Funktionen selbst, sondern immer nur diskrete, u.U. mit einer Linie verbundene Punkte einer Datenreihe dargestellt werden können. Aus diesem Grund kann sich bei einer Erhöhung der Auflösung ein völlig anderes Bild eines Funktionsverlaufes ergeben. Plots sollten also immer kritisch interpretiert werden.



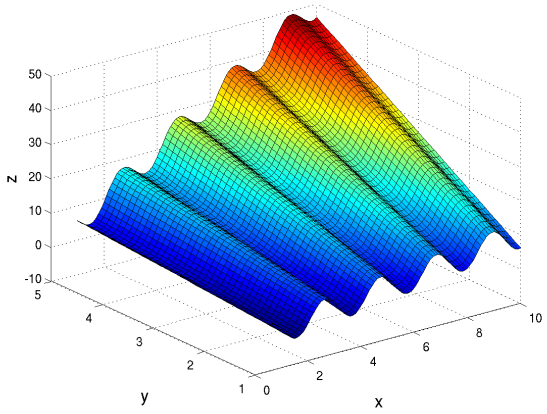
(a) Scatter-Plot von $\sin(3x)$.



(b) Line-Plot von $\sin(3x)$.



(c) Line-Plot mit höherer Auflösung von $\sin(3x)$.



(d) 3D-Plot einer Oberfläche.

Abbildung 1.1: Übersicht über verschiedene Plots.

Um Flächen wie z.B. Ebenen im 3-dimensionalen Raum darzustellen, kann die MATLAB-Funktion `surf(X,Y,Z)` verwendet werden. Hierbei ist zunächst eine Diskretisierung der x-y-Ebene vorzunehmen. Dieser Vorgang wird als Netzgenerierung bezeichnet. In MATLAB steht hierfür die Funktion `[X,Y]=meshgrid(x,y)` zur Verfügung. Es seien x ein Vektor der Länge n und y ein Vektor der Länge m, so erzeugt diese Funktion zwei $n \times m$-Matrizen X und Y. Mithilfe dieser beiden Matrizen kann nun für jedes beliebige Paar an Indices der jeweilig zugehörige x- und y-Wert bestimmt werden: $(X(i,j), Y(i,j))$ definiert für alle $i = 1, \ldots, n$ und $j = 1, \ldots, m$ einen Punkt in dem durch x und y definierten Bereich der x-y-Ebene. Wird der Index i konstant gehalten und j variiert, bewegt man sich auf einer Geraden parallel zur y-Achse und umgekehrt. Genaueres hierzu befindet sich in der MATLAB Hilfe.

Der Vorteil von diesem Netz besteht in MATLAB darin, dass auf diese Weise einfach eine Funktion über der x-y-Ebene ausgewertet werden kann. Um dies zu illustrieren, wird folgender Beispiel-Code zum Plot einer durch $Z(x,y)$ definierten Oberfläche verwendet:

```
[X,Y]=meshgrid(0:0.1:10, 0:0.1:5);
Z=4*sin(3*X)+X.*Y;
surf(X,Y,Z)
```

Es ist zu beachten, dass hier das elementweise Produkt `.*` für die Multiplikation der beiden Matrizen verwendet werden muss. Würde stattdessen `X*Y` geschrieben, so würde eine Matrixmultiplikation durchgeführt. Für die erste Multiplikation `3*X` ist dies unerheblich, da hierbei ein Skalar mit einer Matrix multipliziert wird. Die von MATLAB erzeugte Grafik ist in Abbildung 1.1(d) dargestellt.

Damit wurde eine kurze Einführung in das Plotten mit MATLAB gegeben. Mehr Information kann auf der Hilfe-Seite von Mathworks gefunden werden. Als weiterer Punkt sei die Beschriftung und die Formatierung der erzeugten Plots genannt. Generell arbeitet MATLAB mit Objekten, die bestimmte Eigenschaften („properties") besitzen. Jeder dieser Eigenschaften ist ein Name und ein Wert zugeordnet. Ein Plot ist dabei immer in einem sogenannten Figure-Objekt eingebettet. Wird in MATLAB der Plot-Befehl ausgeführt, so wird automatisch ein Figure-Objekt erzeugt und in dieses schließlich die Grafik eingebettet. Diese besteht wiederum aus Objekten (Text-Objekten, Achsen-Objekten, ...) mit eigenen spezifischen Eigenschaften. Allen Eigenschaften, wie Beschriftung, Schriftgröße, Farbe, etc. werden dabei Standardwerte zugeordnet. Diese können nach Erzeugung des Objekts angepasst werden.

Es existieren dabei Funktionen, die bestimmte Eigenschaften setzen. Hierzu zählt z.B. die Funktion `xlabel()`. Mittels dieser kann die Beschriftung der x-Achse hinzugefügt und formatiert werden:

```
MyLabelX = xlabel('Entropie [kJ/(kg K)]','FontSize', 16);
```

Es wird dabei ein der x-Achse zugeordnetes Text-Objekt erzeugt bzw. verändert, welches die gewünschte Beschriftung enthält. Gleichzeitig wird die Schriftgröße, eine Eigenschaft dieses Objekts, auf 16 Punkte gesetzt. Analog kann mit anderen Eigenschaften verfahren werden. Im obigen Fall wird zusätzlich ein sogenanntes Handle `MyLabelX` zu dem Text-Objekt, welches als Beschriftung der x-Achse definiert ist, erzeugt. Mittels dieses Handles kann auf das jeweilige Objekt bzw. dessen Eigenschaften zugegriffen werden.

Dies geschieht durch die Funktion `set()`:

```
set(MyLabelX                        , ...
   'FontWeight'      , 'bold'       , ...
   'FontName'        , 'Helvetica'  , ...
   'Color'           , 'r'          );
```

In diesem Fall wird als Schriftart für die Achsbeschriftung „Helvetica" gewählt und diese als „fett" definiert. Als Schriftfarbe wird rot verwendet. Andere Objekte besitzen andere spezifische Eigenschaften, die ebenfalls auf diese Weise gesetzt werden können - es muss lediglich das jeweilige Objekt-Handle an die `set()`-Funktion übergeben werden. Alle Möglichkeiten hierzu finden sich auf der Mathworks-Homepage.

Als wichtiges Beispiel eines bereits definierten Handles sei `gca` genannt, das immer auf das aktuell aktive Achsen-Objekt verweist („get current axis"). Damit kann u.a. der zu plottende Bereich definiert werden:

```
set(gca                                  ,...
    'XLim'              , [0,12]         ,...
    'XTick'             , [0:2:12]       );
```

Dieser Befehl bewirkt, dass nur der Bereich von 0 bis 12 auf der x-Achse dargestellt wird. Außerdem werden die Markierungen auf der x-Achse im Abstand von zwei gesetzt. Der Befehle `legend()` fügt dem aktuellen Achsen-Objekt (gca) eine Legende hinzu. Diese kann z.B. wie folgt definiert werden:

```
p1 = plot(...); hold on
p2 = plot(...);
p3 = plot(...);

legend([p1; p2; p3] ,{'Plot 1' 'Plot 2' 'Plot 3'})
```

Es werden in diesem Beispiel zunächst drei Plots in ein einziges Diagramm gezeichnet: Mittels `hold on` werden alle folgenden Plots ebenfalls in demselben Achsen-Objekt wie der erste Plot `p1` dargestellt. Die Funktion `legend()` erzeugt nun im aktuellen Achsen-Objekt eine Legende, wobei den Plots `p1` bis `p3` die in geschweiften Klammern definierten Bezeichnungen der Reihe nach zugewiesen werden. Abgekürzt kann auch mittels
`legend('Plot 1','Plot 2','Plot 3')` eine Legende für obigen Plot erzeugt werden. Hierbei werden die Legendeneinträge der Reihe nach den Plots zugeordnet. Damit kann aber z.B. nicht der Plot `p2` in der Legende ausgelassen werden.

1.2 Interpolation/ Extrapolation

Wie bereits in Kapitel 1.1 für das Plotten beschrieben, liegen in der Numerik Funktionsverläufe immer nur in diskreter Form und für ein abgeschlossenes Gebiet vor. Gleiches gilt für die Auswertung von Messreihen: Es können nicht unendlich viele Messungen vorgenommen werden, um einen Funktionsverlauf bis in das letzte Detail genau bestimmen zu können. Soll also auf Werte zwischen zwei tabellierten zugegriffen werden, so müssen Annahmen bzgl. des Funktionsverlaufes zwischen diesen beiden Werten getroffen werden. Die Auswertung auf Basis dieser Annahme bezeichnet man als **Interpolation** (von Lateinisch „inter-" *zwischen* und „polire" *feilen, glätten*). Besteht Interesse daran, wie die tabellierte Funktion jenseits ihres Beginns und Endes weiterverläuft, so müssen analog Annahmen bzgl. des Funktionsverlaufs vor dem ersten bzw. nach dem letzten tabellierten Wert getroffen werden. In diesem Fall spricht man von **Extrapolation** (von Lateinisch „extra-" *außer*) eines Datensatzes. Im Folgenden werden lediglich 1-dimensionale Datensätze betrachtet, welche einer Folge von Werten x_i Funktionswerte $f(x_i)$ zuordnen ($i = 1, \ldots, n$). Diese tabellierten Wertepaare werden auch als Stützwerte bezeichnet. Die Folge der x_i-Werte ist dabei monoton steigend.

Die Annahmen für den unbekannten Funktionsverlauf jenseits der tabellierten Werte sollte dabei so allgemeingültig wie möglich sein, um nicht für jeden Datensatz zunächst nach einer passenden Modellannahme suchen zu müssen. Häufig werden hierfür Polynome unterschiedlichen Grades verwendet. Der einfachste Spezialfall ist die lineare Interpolation mittels Polynome vom Grad eins.

Bei den bisher genannten Verfahren handelt es sich um lokale Ansätze, welche nur eine bestimmte Anzahl an Stützpunkte um den gesuchten Wert benötigen, um auf dieser Basis den gesuchten Wert mithilfe der jeweils angenommenen Funktion zu bestimmen. Der globale Funktionsverlauf ist jedoch nicht glatt, da diese Methode einfach die lokalen Funktionsverläufe aneinanderreiht. Im Fall der linearen Interpolation befindet sich beispielsweise an jedem tabellierten Punkt ein Knick. Möchte man einen global glatten Funktionsverlauf, so bieten sich sogenannte **kubische Splines** zur Interpolation an. Diese passen sich nicht nur an die lokal tabellierten Werte an, sondern es wird auch sichergestellt, dass der durch Aneinanderreihung der einzelnen Teilstücke entstehende Funktionsverlauf zweimal stetig differenzierbar ist. Ein weiterer Vorteil dieser Methode im Vergleich zu Polynomen ist, dass Polynome höheren Grades ($\gtrsim 5$) zu Oszillationen neigen und somit nicht mehr zur Inter- bzw. Extrapolation geeignet sind. Dies wird durch die Verwendung von Splines vermieden.

Allgemein bestimmt man die Ordnung eines Interpolationsschemas aus der Anzahl der verwendeten Punkte minus Eins. Ein Verfahren 1. Ordnung (linear) benötigt also zwei Datenpunkte, die den gesuchten Wert umschließen. Eine Herleitung dieser Aussage kann - analog zu Kapitel 2 - mithilfe der Taylor-Reihe erfolgen.

1.2.1 Interpolation mittels Polynomen

Wie oben beschrieben, handelt es sich hierbei um ein weit verbreitetes Verfahren zur Interpolation. Speziell der lineare Fall findet in der Interpolation von Hand eine breite Anwendung.

Ein Polynom n-ter Ordnung $P(x)$ ist wie folgt definiert:

$$P(x) = a_0 + a_1 x + a_2 x^2 + \dots a_n x^n = \sum_{i=0}^{n} a_i x^i \tag{1.1}$$

Es besitzt $n+1$ unbekannte Koeffizienten a_i. Um diese zu bestimmen, werden also $n+1$ Gleichungen benötigt. Bei der Interpolation liefert jedes Wertepaar (x_j, f_j) eine Bestimmungsgleichung für diese Koeffizienten, da gefordert wird, dass der Funktionsverlauf des Polynoms jeden dieser Punkte enthält. Demnach werden also zur Bestimmung des Interpolationspolynoms für ein Verfahren n-ter Ordnung $n+1$ Stützstellen benötigt. Damit ergibt sich folgendes lineares Gleichungssystem zur Bestimmung der unbekannten Koeffizienten:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix} \tag{1.2}$$

Dieses System kann für eine geeignete Wahl an tabellierten Werten (x_i, f_i) gelöst (siehe Kapitel 4.2) und das Ergebnis schließlich in Gleichung (1.1) eingesetzt werden. Anschließend kann diese Gleichung für den gewünschten x-Wert ausgewertet werden. Dies soll für den linearen Fall beispielhaft durchgeführt werden. Es ergibt sich hier das einfache System

$$\begin{pmatrix} 1 & x_0 \\ 1 & x_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}. \tag{1.3}$$

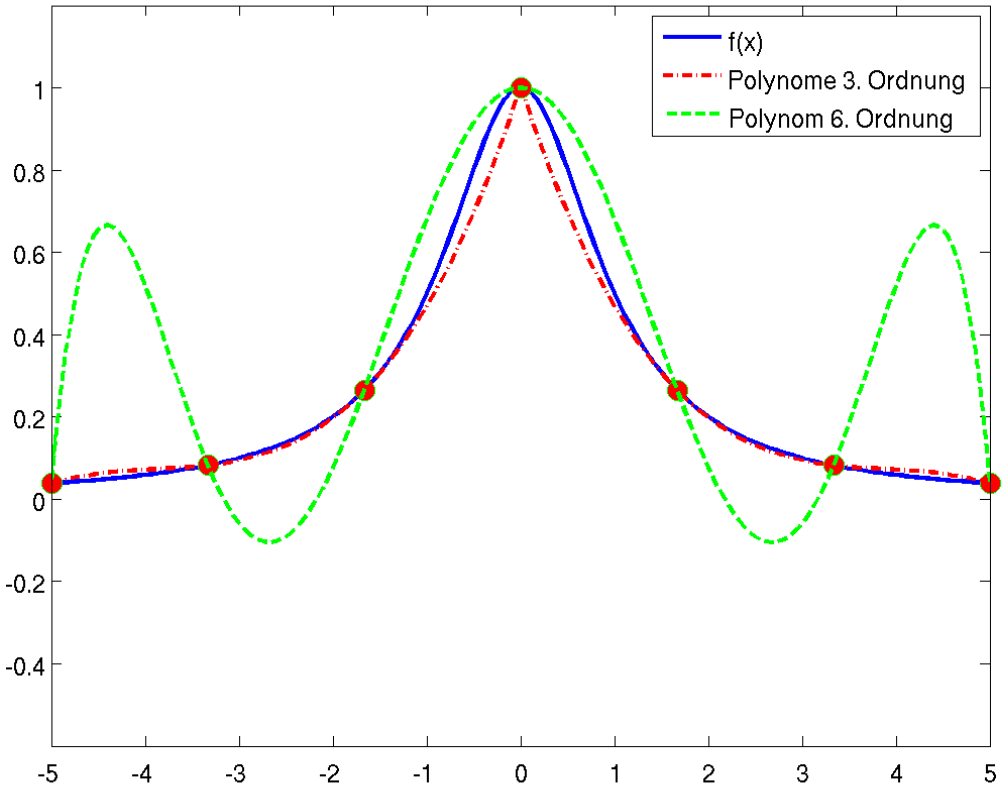


Abbildung 1.2: Approximation einer Funktion $f(x)$ mittels eines Polynoms 6. Grades sowie zwei stückweise bestimmte Interpolationspolynome 3. Grades.

Löst man dieses allgemein nach den Koeffizienten auf, so erhält man

$$a_0 = f_0 + \frac{f_1 - f_0}{x_1 - x_0} x_0 \quad \text{und} \quad a_1 = \frac{f_1 - f_0}{x_1 - x_0}. \tag{1.4}$$

Damit ergibt sich die bekannte Formel der linearen Interpolation:

$$P(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0} (x - x_0) \tag{1.5}$$

Wir werden im Laufe dieses Praktikums noch feststellen, dass viele Probleme der Numerik auf lineare Gleichungssysteme hinauslaufen. Von daher ist es wichtig Methoden zu kennen, diese effizient zu lösen. Eine Methode hierfür, den sogenannten Gauß-Algorithmus, werden wir in Kapitel 4 kennen lernen. Außerdem sei noch angemerkt, dass ein direktes Lösen des Gleichungssystems (1.2) nicht die effizienteste Methode darstellt. Zur Bestimmung des interpolierten Wertes am Ort x ist z.B. der Algorithmus nach Neville zu bevorzugen. Eine Beschreibung hierzu kann z.B. in [1] gefunden werden.

Im Allgemeinen ist die Funktion, mit der die Stützstellen generiert wurden, natürlich unbekannt. Um die Qualität der Interpolation beispielhaft beurteilen zu können, sollen im Folgenden zwischen Stützwerten der bekannten Funktion $f(x) = \frac{1}{1+x^2}$ interpoliert werden. Es werden hierfür sechs Stützstellen gleichmäßig auf dem Intervall $x \in [-5; 5]$ verteilt. Zuerst wird daraus ein Interpolationspolynom 6. Grades berechnet. Anschließend werden für jeweils vier Stützstellen zwei Polynome 3. Grades bestimmt, wobei sich diese die mittlere Stützstelle teilen. Das Ergebnis ist in Abbildung 1.2 dargestellt.

In diesem Fall wird die Approximation der Funktion durch die Erhöhung der Ordnung des Interpolationspolynoms nicht verbessert. Ganz im Gegenteil: Das Polynom 6. Grades weist für die Bereiche, in denen die Funktion $f(x)$ eine geringe Steigung besitzt, starke Überschwinger auf. Die zwei Polynome 3. Ordnung hingegen treffen den Verlauf relativ gut. Ein Verfahren höherer Ordnung bedeutet also nicht unbedingt eine Verbesserung der Qualität der Interpolation. Das Auftreten von Überschwingern wird auch als Runges-Phänomen bezeichnet. Obige Funktion $f(x)$ ist eine pathologische Funktion, für die das Problem der Überschwinger besonders ausgeprägt ist und wird entsprechend als Runge-Funktion bezeichnet. (Fairerweise muss allerdings angefügt werden, dass das Ergebnis anders aussehen würde, falls die Polynome 6. Grades ebenfalls nur für eine Hälfte des betrachteten Bereiches ausgewertet würden. Allerdings würde sich damit auch die Anzahl der benötigten Stützstellen auf 13 erhöhen.)

1.2.2 Interpolation mittels (kubischer) Splines

Betrachtet man den Verlauf der durch Polynome 3. Ordnung angenäherten Kurve in Abbildung 1.2, so wird ersichtlich, dass dieser an der Stelle $x = 0$ einen Knick aufweist und damit nicht glatt ist. Die Ableitung der Funktion ist an dieser Stelle also nicht stetig. Ein Polynom 6. Ordnung würde zwar für die gegebenen Punkte einen beliebig oft differenzierbaren Verlauf liefern, allerdings würde sie durch Überschwinger stark vom gesuchten Verlauf abweichen.

Kubische Spline liefern auf dem gesamten betrachteten Gebiet einen zwei Mal stetig differenzierbaren Funktionsverlauf, der zudem deutlich geringere Überschwinger aufweist, als ein vergleichbares Polynom. Die grundlegende Idee zu deren Bestimmung soll im Folgenden kurz skizziert werden:

Das zu betrachtende Gebiet, für welches die Funktion $f(x)$ angenähert werden soll, sei durch $n+1$ Stützstellen in n-Intervalle aufgeteilt. Für jedes Intervall i soll die gesuchte Spline-Funktion mit einem kubischen Polynom s_i übereinstimmen. Dieses lässt sich als

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \qquad x_{i-1} \leq x \leq x_i \tag{1.6}$$

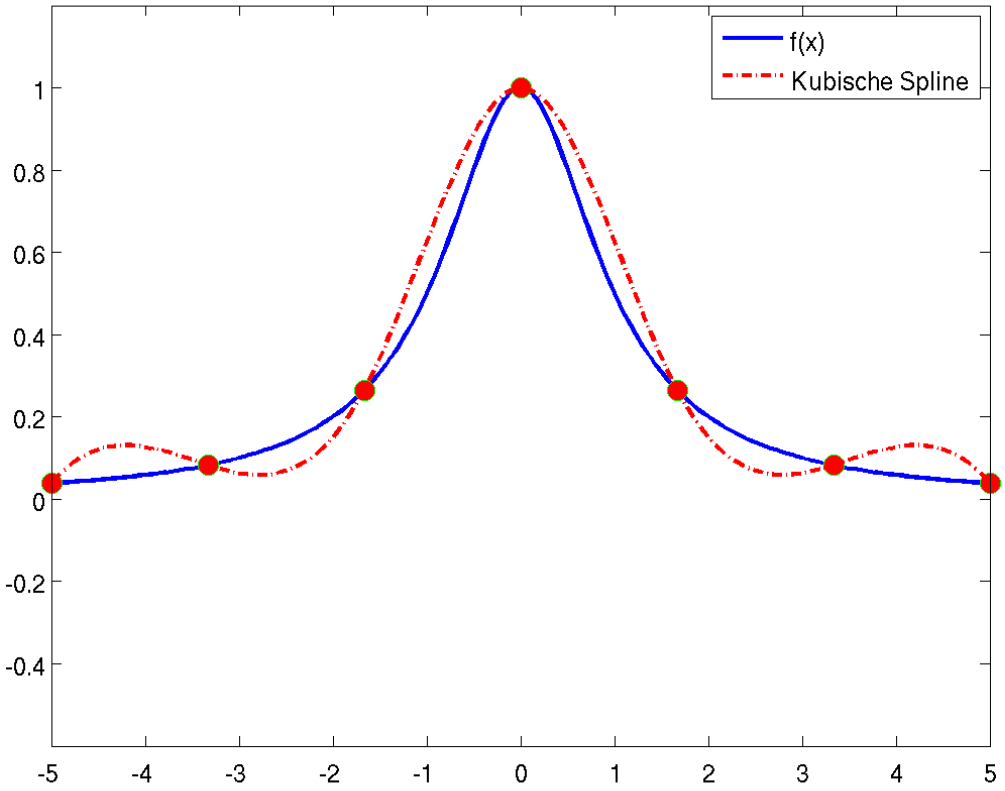


Abbildung 1.3: Approximation einer Funktion $f(x)$ mit Hilfe eines kubischen Splines.

darstellen, wobei $i = 1, 2, \ldots, n$. Für jedes der n Intervalle gilt es also die jeweils vier unbekannten Koeffizienten a_i, b_i, c_i und d_i zu bestimmen. Damit ergeben sich insgesamt $4n$ Unbekannte. Pro Intervall ergeben sich durch die Forderung der Stetigkeit bis zur zweiten Ableitung folgende vier Gleichungen:

$$
\begin{aligned}
s_i(x_{i-1}) &= f_{i-1}, && \text{für} && i = 1, 2, \ldots, n && (1.7)\\
s_i(x_i) &= f_i, && \text{für} && i = 1, 2, \ldots, n && (1.8)\\
s_i'(x_i) &= s_{i+1}'(x_i), && \text{für} && i = 1, 2, \ldots, n-1 && (1.9)\\
s_i''(x_i) &= s_{i+1}''(x_i), && \text{für} && i = 1, 2, \ldots, n-1 && (1.10)
\end{aligned}
$$

Die Gleichungen (1.7) und (1.8) sorgen für Stetigkeit des Funktionsverlaufes, Gleichung (1.9) für die Stetigkeit der ersten und Gleichung (1.10) für die Stetigkeit der zweiten Ableitung. Die Bedingungen (1.7) und (1.8) gelten dabei für alle n Intervalle und liefern damit $2n$ Gleichungen. Die Bedingungen (1.9) und (1.10) hingegen gelten jeweils nur für die ersten $n-1$ Intervalle (da das Intervall $n+1$ nicht mehr existiert) und liefern damit weitere $2n-2$ Gleichungen. Damit das System vollständig bestimmbar ist, fehlen also noch zwei weitere Gleichungen, die Rand-

bedingungen. Diese können auf verschiedene Art und Weisen bestimmt werden. Eine Möglichkeit besteht z.B. darin, die zweiten Ableitungen an den Rändern zu Null zu setzen. Eine andere, den ersten Ableitungen einen fixen Wert zuzuweisen. Die MATLAB-internen Funktionen `spline()` bzw. `interp1(...,'spline')` verwenden die sogenannte „not-a-knot"-Randbedingung, welche für den zweiten und vorletzten Punkt zusätzlich die Kontinuität der dritten Ableitung fordert. Auch für die Spline-Interpolation existieren Algorithmen, die effizient den interpolierten Wert an der gewünschten Stelle bestimmen, ohne direkt das Approximationspolynom auswerten zu müssen.

Für das in Abbildung 1.2 gezeigte Beispiel ergibt sich mittels der Spline-Interpolation der in Abbildung 1.3 dargestellte Verlauf. Es wird deutlich, dass die Überschwinger im Vergleich zu einem Polynom 6. Ordnung reduziert sind und die Funktion gleichzeitig stetig und glatt verläuft.

1.3 Übungsaufgaben

Der erste Teil des heutigen Übungstermines dient zur Auffrischung Ihrer MATLAB-Kenntnisse. Sie erhalten zunächst tabellierte Stoffwerte für Wasser und Ihre Aufgabe wird es sein, daraus 2D- bzw. 3D-Diagramme anzufertigen. Der Schwerpunkt hierbei liegt auf der optischen Aufbereitung sowie der Beschriftung der Plots. Im Rahmen der dritten Aufgabe sollen Sie eine eigene Funktion zur linearen Interpolation erstellen und diese auf die gegebenen Tabellen anwenden. Ihre Ergebnisse sollen Sie zuletzt mit dem exakten Funktionsverlauf, welchen Sie mittels des Programms "XSteam" erhalten, sowie mit den Ergebnissen der MATLAB-internen Funktion zur Interpolation vergleichen.

1.1 Laden Sie die vorgegebenen Tabellen aus der MAT-Datei "Uebung1_2D.mat" und erstellen Sie aus den darin enthaltenen Daten ein T-s-Diagramm mit eingezeichneter Siede- und Taulinie. Tragen Sie auch den kritischen Punkt von Wasser ($T_{krit} = 373.946$ °C, $S_{krit} = 4.4150\ kJ/(kg\ K)$) ein. Plotten Sie in dieses Diagramm den Verlauf der acht gegebenen Isobaren und beschriften Sie diese.

Hinweise:

- Die Variablen *S_siedeT* und *S_tauT* enthalten die Werte der Siede- bzw. der Taulinie in Abhängigkeit der im Vektor *T_vec* gegebenen Temperaturen.
- Beachten Sie, dass alle Linien in Abhängigkeit des gegebenen Temperaturvektors angegeben sind. Dies bedeutet, dass das n-te Element eines Vektors zum n-ten Element des Temperaturvektors gehört. Nichtsdestotrotz soll die Entropie - wie gewohnt - auf der Abszisse (horizontal) und die Temperatur auf der Ordinate (vertikal) aufgetragen werden.
- Die Werte für die Isobaren sind in der Matrix *S_mat* enthalten. Die n-te Zeile dieser Matrix enthält die tabellierten Werte der Entropie der Isobaren für den in der n-ten Zeile des Vektors *P_vec* in bar angegebenen Druck.

1.2 Laden Sie die vorgegebenen Tabellen aus der MAT-Datei "Uebung1_3D.mat" und plotten Sie mittels der darin enthaltenen Daten die Zustandsebene für Wasser in ein T-s-p-Diagramm mit eingezeichneter Siede- und Taulinie. Tragen Sie ebenfalls wieder den kritischen Punkt von Wasser ($p_{krit} = 220.639\ bar$) ein. Beschriften Sie das Diagramm und fügen Sie eine Legende ein.

Hinweise:

- Alle Daten sind jetzt in Abhängigkeit der Entropie und des Druckes gegeben. Erstellen Sie daher mittels `meshgrid()` eine Diskretisierung der p-S-Ebene. Die Daten hierfür sind in den Vektoren *P_vec* und *S_vec* enthalten. In der Matrix *T_mat* sind die entsprechenden Temperatur-Werte der Zustandsebene gespeichert.
- Die Siede- bzw. Taulinie sind als Entropiewerte in Abhängigkeit des Druckes und der Sattdampftemperatur gegeben. D.h. der n-te Eintrag des Druckvektors *P_vec* korrespondiert mit den n-ten Einträgen der Vektoren *Tsat_vec* und *Svapour_vec* bzw. *Sliquid_vec*.

1.3 Erstellen Sie nun eine Funktion zur linearen Interpolation.

1.3.1 Fertigen Sie für die folgende Aufgabe ein Ablaufdiagramm an, bevor Sie mit der Programmierung beginnen.

1.3.2 Erstellen Sie nun eine Funktion, welche für einen gegebenen x-Vektor *x_vec* und einen gegebenen zugehörigen Vektor mit Funktionswerten *f_x* den linear interpolierten Funktionswert an der Stelle *x1* ausgibt.

- `[f_x1] = LinInterpolation(x_vec,f_x,x1)`
  - `f_x1`: Funktionswert an der Stelle `x_1`
  - `x_vec`: Vektor der x-Werte für die zugehörigen Funktionswerte `f_x`
  - `f_x`: Vektor der Funktionswerte
  - `x1`: x-Wert, an dem interpoliert werden soll

Laden Sie anschließend die MAT-Datei "Uebung1_interp.mat" und plotten Sie die darin enthaltenen Datenpunkte der Taulinie (Entropie über Temperatur).

Plotten Sie nun den Verlauf der Taulinie mit 100 Werten in das selbe Diagramm, wobei Sie die Werte zwischen den gegebenen Datenpunkten mit Hilfe Ihrer eben erstellten Funktion interpolieren.

Zum Vergleich führen Sie die Interpolation ebenfalls mit der MATLAB-Funktion `interp1()` durch und verwenden Sie hierbei die Spline-Interpolation. Plotten Sie nun zusätzlich unter Verwendung des Tools XSteam den wahren Verlauf der Taulinie.

Hinweise:

- Ordner, welche MATLAB-Dateien mit Funktionen enthalten und nicht im aktuellen Arbeitsverzeichnis liegen, müssen erst zur PATH-Variablen hinzugefügt werden, bevor Sie auf die darin enthaltenen Funktionen zugreifen können. Dies geschieht mittels des Befehls `addpath PFAD`.
- Zur richtigen Verwendung von XSteam lesen Sie sich die im XSteam-Ordner enthaltene PDF-Datei durch.
- Zum Auffinden zwischen welchen tabellierten Werten *x1* liegt, ist die Funktion `find(Bedingung,1,'last/ first')` hilfreich.

1.4 Wichtige Matlab Befehle

Befehl	Beschreibung
`A<=9, A==9,...`	Logischer Ausdruck: Gibt eins zurück, falls die Bedingung erfüllt ist, andernfalls null. Auch für Vektoren/ Matrizen anwendbar
`abs()`	Bestimmt elementweise den Betrag einer Matrix
`addpath .../Ordner`	Fügt den angegebenen Pfad zur PATH-Variablen hinzu
`disp()`	Gibt den Wert einer Variable im Kommando Fenster aus (Vermeidet eine orangene Markierung)
`doc plot`	Zeigt die Matlab Dokumentation zum Befehl "plot"
`find()`	Sucht die Indices aller Elemente eines Vektors die ungleich Null sind. Kann gut mit einem logischen Ausdruck, z.B. `find(A>3)` kombiniert werden
`interp1()`	Interne Interpolationsroutine von MATLAB
`legend()`	Erzeugt eine Legende
`length()`	Gibt die größte Dimension eines Arrays aus
`linspace()`	Erzeugt einen Vektor mit linear ansteigenden Werten
`load Datei1 Datei2`	Lädt Daten aus einer MAT-Datei
`max() / min()`	Gibt Maximum/ Minimum eines Vektors aus (2. Ausgabe ist dessen Position im Vektor)
`meshgrid()`	Erzeugt ein Gitternetz (für den Befehl `surf()`)
`num2str()`	Konvertiert Zahlen zu Strings
`plot()`	Erzeugt einen 2D-Linienplot
`plot3()`	Erzeugt einen 3D-Linienplot
`set()`	Setzt bestimmte Eigenschaften eines Objekts
`[z,s] = size()`	Gibt die Größe einer Matrix aus
`subplot()`	Positioniert mehrere Plots in einer Figure
`surf()`	3D-Plot einer Oberfläche
`sprintf()`	Formatiert Daten zu einem String (ähnlich printf() aus C)
`text()`	Positioniert Text in einem Plot
`title()`	Erzeugt den Titel eines Plots
`xlabel() / ylabel() / zlabel()`	Erzeugt die Achsbeschriftung eines Plots
`zeros()`	Erzeugt eine Null-Matrix
`axis([])`	Erzeugt Grenzen der Achsen eines Plots
`xlim([])/ ylim([]) / zlim([])`	Erzeugt Grenzen der spezifizierten Achsen
`disp()`	Gibt den Wert einer Variable aus, ohne der Name der Variable zu zeigen

2 Numerische Differentiation

Literaturverzeichnis

[1] Brian P Flannery, William H Press, Saul A Teukolsky und William Vetterling. *Numerical Recipes in C*. Kap. 5.7. 1992. URL: http://www.nr.com/.

[2] Peter Lewis Silveston. "Wärmedurchgang in waagerechten Flüssigkeitsschichten". In: *Forschung auf dem Gebiet des Ingenieurwesens A* 24.1 (1958), S. 29–32.

[3] Karkenahalli Srinivas und Clive Fletcher. *Computational Rechniques for Fluid Dynamics: A Solutions Manual*. Kap. 3. Springer, 1992.

Lernziele

- Diskretisierung der Ableitung
- Ordnung bzw. Abbruchfehler einer Diskretisierung
- Herleitung von Diskretisierungsformeln inklusive Bestimmung deren Ordnung

Motivation

Es sei eine Funktion oder eine Reihe an Messwerten gegeben und es soll numerisch deren Ableitung gebildet werden. A priori sei die mathematische Struktur der abzuleitenden Funktion unbekannt und demnach können keine analytischen Methoden angewandt werden.

Im Fall einer Messreihe sind die Funktionswerte an fest bestimmten Punkten gegeben. Anhand dieser soll die Ableitung bestmöglich approximiert werden. Als Beispiel sei hier die Bestimmung eines Wärmestromes genannt: An bestimmten Messpunkten werden die Temperaturwerte gemessen. Um nun den Wärmestrom aus diesen Daten bestimmen zu können, muss numerisch ein Gradient berechnet werden. Im Rahmen dieses Kapitels wird angenommen, dass die Datenreihe nicht verrauscht ist. Trifft dies nicht zu, so wird auf die in Kapitel 4 beschriebenen Methoden verwiesen.

Oftmals steht Ihnen die abzuleitende Funktion als *Black-Box* zur Verfügung. Sie können diskrete Werte an diese Funktion übergeben und bekommen einen Ausgabewert zurück. Eine derartige Funktion kann z.B. ein in einer beliebigen Programmiersprache geschriebenes Programm sein. Diese *Black-Box* kann an jeder beliebigen Stelle (innerhalb eines definierten Wertebereichs) ausgewertet werden. Da der Aufwand hierfür beliebig groß sein kann (abhängig von der gegebenen Funktion), ist das Ziel mit möglichst wenigen Funktionsaufrufen eine möglichst gute Näherung des Gradienten zu berechnen. Ein Beispiel hierfür ist die Bestimmung der Rayleigh-Zahl für Wasser, welche den thermischen Ausdehnungskoeffizienten $\beta = -1/\rho \cdot (\partial\rho/\partial T)_p$ enthält. Anstatt den Verlauf der Dichte aus Experimenten zu bestimmen, soll eine Funktion verwendet werden, welche für gegebene Temperaturen die Dichte anhand von Näherungsformeln ausgibt. Diese sind jedoch dem Anwender nicht bekannt. Die Stützstellen zur numerischen Approximation der Ableitung können somit frei gewählt werden. Wie eine optimale Wahl dieser Stützstellen aussieht und wie der Approximationsfehler abgeschätzt werden kann, wird im Folgenden erörtert.

2.1 Diskretisierung der Ableitung

2.1.1 Heuristische Herleitung von Näherungsformeln

Das Problem scheint zunächst sehr einfach zu sein: Wir kennen unsere zu untersuchende Funktion $f(x)$ an zwei Stellen, $f(x_i)$ und $f(x_{i+1})$, sowie den (hier für alle x als konstant angenommenen) Abstand zwischen den zugehörigen x-Werten $h = x_{i+1} - x_i = \text{konst.}$. Hierbei gelte $x_i < x_{i+1}$. Wie wir wissen ist die erste Ableitung als

$$\left.\frac{df(x)}{dx}\right|_x = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \tag{2.1}$$

definiert. Da wir den Grenzwert aber nicht numerisch bilden können, ist folgender Ausdruck für den Gradienten an der Stelle x_i naheliegend:

$$\left.\frac{df(x)}{dx}\right|_{x_i} \approx \frac{f(x_{i+1}) - f(x_i)}{h} \tag{2.2}$$

Die Ableitung wurde mittels einer **Vorwärtsdifferenz** bestimmt, da der nachfolgende Stützwert x_{i+1} zur Bestimmung des Differenzenquotienten herangezogen wurde. Genauso gut könnte man stattdessen auch den vorangehenden Punkt x_{i-1} verwenden. Es ergibt sich eine **Rückwärtsdifferenz**

$$\left.\frac{df(x)}{dx}\right|_{x_i} \approx \frac{f(x_i) - f(x_{i-1})}{h} \tag{2.3}$$

Welcher der beiden Ausdrücke liefert nun die verlässlichere Näherung für die gesuchte erste Ableitung? Falls Gleichung (2.2) einen anderen Wert als Gleichung (2.3) liefert, welchem Wert soll vertraut werden? Man könnte sich nun überlegen den Mittelwert aus beiden Ausdrücken zu verwenden und gelangt damit auf die folgende Formulierung einer **zentralen Differenz**

$$\left.\frac{df(x)}{dx}\right|_{x_i} \approx \frac{f(x_i) - f(x_{i-1}) + f(x_{i+1}) - f(x_i)}{2h} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \tag{2.4}$$

2.1.2 Rundungs- und Abbruchfehler

Wir wollen nun systematisch den Fehler dieser Näherungen untersuchen und damit die Qualität obiger Formeln abschätzen. Fehler in der Numerik setzen sich aus zwei Komponenten zusammen: einem Rundungsfehler und einem meist dominierenden Abbruchfehler.

Der **Rundungsfehler** ergibt sich durch die Darstellung der Zahlen mit endlicher Genauigkeit. Hierbei existieren z.B. Zahlen, die sich zwar im Dezimalsystem exakt mit wenigen gültigen Ziffern darstellen lassen, aber im Dualsystem unendlich viele Ziffern benötigen würden. Ab einer bestimmten Ziffer muss der Rechner die Darstellung abbrechen und begeht so einen gewissen Fehler. Ein Beispiel einer solchen Zahl im Dezimalsystem ist 0.1: Im Binärsystem dargestellt, ergibt sich die unendliche Ziffernfolge $0.000110011\ldots$. Ein derartiger Fehler kann bereits bei der Darstellung von h bzw. $x + h$ auftreten. Gerade bei Systemen mit geringer Genauigkeit in ihrer Zahlendarstellung, sollte darauf geachtet werden, dass h bzw. $x + h$ exakt darstellbar sind. In obigen Beispiel wäre es also besser ein h von 0.125 bzw. 0.0625 zu wählen. Für die im Rahmen dieses Praktikums durchgeführten Rechnungen mit Matlab sind diese Überlegungen nicht von besonders großer Bedeutung, da Zahlen intern mit sehr hoher Genauigkeit dargestellt werden können. Im Vergleich zum noch zu erläuternden Abbruchfehler sind die Rundungsfehler i.A. sehr klein.

Der **Abbruchfehler** basiert zum einen darauf, dass Probleme in der Numerik meist iterativ gelöst werden. Ein Ergebnis wird hierbei schrittweise verbessert bis ein definiertes Abbruchkriterium erfüllt ist. Der Fehler skaliert also mit der Anzahl an Iterationen. Als Beispiel wird auf die numerische Integration in Kapitel 3 bzw. die Nullstellensuche in Kapitel 5 verwiesen.

Eine weitere Quelle für den Abbruchfehler ist die Approximation von Funktionsverläufen durch Polynome. In der Numerik sind Funktionsverläufe immer als eine Reihe an Zahlen gegeben. Jedes Wertepaar wird hierbei als Stützstelle bezeichnet. Um Aussagen für den Verlauf jenseits der Stützstellen treffen zu können (Extra- und Interpolation) oder um Ableitungen, Integrale, etc. bilden zu können, muss der Funktionsverlauf zwischen den Stützstellen mit Hilfe einer analytischen Funktion approximiert werden. Die gewünschte Operation - also z.B. die

Interpolation oder das Bilden einer Ableitung - kann anschließend mit Hilfe dieser Näherungsfunktion durchgeführt werden. Meist werden zu diesem Zweck Polynome verwendet, da diese leicht handhabbar und beliebig oft differenzierbar ("glatt") sind. Eine Taylorreihe ist eine solche (lokale) polynomiale Approximation einer Funktion. In der Praxis werden nur die ersten paar Terme dieser Reihe berücksichtigt. Die vernachlässigten stellen dann den Abbruchfehler dieser Näherung dar.

Dieser Fehler soll nun im Folgenden charakterisiert werden. Die oben hergeleiteten, heuristischen Formeln zur Bestimmung des Gradienten können als lineare Näherung des Funktionsverlaufes begriffen werden (zumindest die Vorwärts- und Rückwärtsdifferenz). Das Polynom erster Ordnung lässt sich wie folgt bestimmen:

$$f(x_{i+1}) \approx f(x_i) + \left.\frac{df(x)}{dx}\right|_{x_i} h. \tag{2.5}$$

Durch Auflösen nach der Ableitung ergibt sich Gleichung (2.2). Dieser Zusammenhang kann nun mit einer Taylorentwicklung um x_i verglichen werden:

$$f(x_{i+1}) = f(x_i) + \left.\frac{df(x)}{dx}\right|_{x_i} h + \left.\frac{d^2 f(x)}{dx^2}\right|_{x_i} \frac{h^2}{2} + R(h) \tag{2.6}$$

mit dem Restglied $R(h)$.

Zur qualitativen Bestimmung des Abbruchfehlers in Näherung (2.5) ist es interessant, wie sich das Restglied samt des h^2-Terms für kleine h verhält. Was passiert z.B. wenn h halbiert wird? Um ein solches Verhalten beschreiben zu können, wird die asymptotische obere Schranke eingeführt:

Definition: Asymptotische obere Schranke

$$f = \mathcal{O}(g) \quad \Leftrightarrow \quad \limsup_{h \to a} \left|\frac{f(h)}{g(h)}\right| < \infty \tag{2.7}$$

Die Funktion g ist eine asymptotische obere Schranke der Funktion f für $h \to a$.
Man sagt: „f ist von der Ordnung Groß-O von g."

Im vorliegendem Fall ist das Verhalten für kleine h von Bedeutung, d.h. $a = 0$. Wir haben es mit einem Polynom zu tun, also wählen wir als Vergleichsfunktion $g(h) = h^m$. Wir untersuchen jetzt das asymptotische Verhalten von R:

$$\limsup_{h \to 0} \frac{\left.\frac{d^3 f(x)}{dx^3}\right|_{x_i} \frac{h^3}{3!} + \left.\frac{d^4 f(x)}{dx^4}\right|_{x_i} \frac{h^4}{4!} + \cdots}{h^m} \overset{!}{<} \infty \tag{2.8}$$

Das maximale m, für das obige Ungleichung erfüllt ist, ist $m = 3$, somit gilt $R = \mathcal{O}(h^3)$. Für kleine h gilt also, dass wenn h halbiert wird, sich das Restglied ungefähr achtelt.

Gleichung (2.6) lässt sich also schreiben als

$$f(x_{i+1}) = f(x_i) + \left.\frac{df(x)}{dx}\right|_{x_i} h + \left.\frac{d^2 f(x)}{dx^2}\right|_{x_i} \frac{h^2}{2} + \mathcal{O}\left(h^3\right). \tag{2.9}$$

Vergleicht man nun die Gleichungen (2.5) und (2.9), so stellt man fest, dass deren Unterschied von der Größenordnung $\mathcal{O}\left(h^2\right)$ ist. Um den Fehler für die numerische Ableitung zu bestimmen, wird Gleichung (2.9) nach dieser aufgelöst:

$$\left.\frac{df(x)}{dx}\right|_{x_i} = \frac{f(x_{i+1}) - f(x_i)}{h} - \left.\frac{d^2 f(x)}{dx^2}\right|_{x_i} \frac{h}{2} + \mathcal{O}\left(h^2\right) = \frac{f(x_{i+1}) - f(x_i)}{h} + \mathcal{O}\left(h\right) \tag{2.10}$$

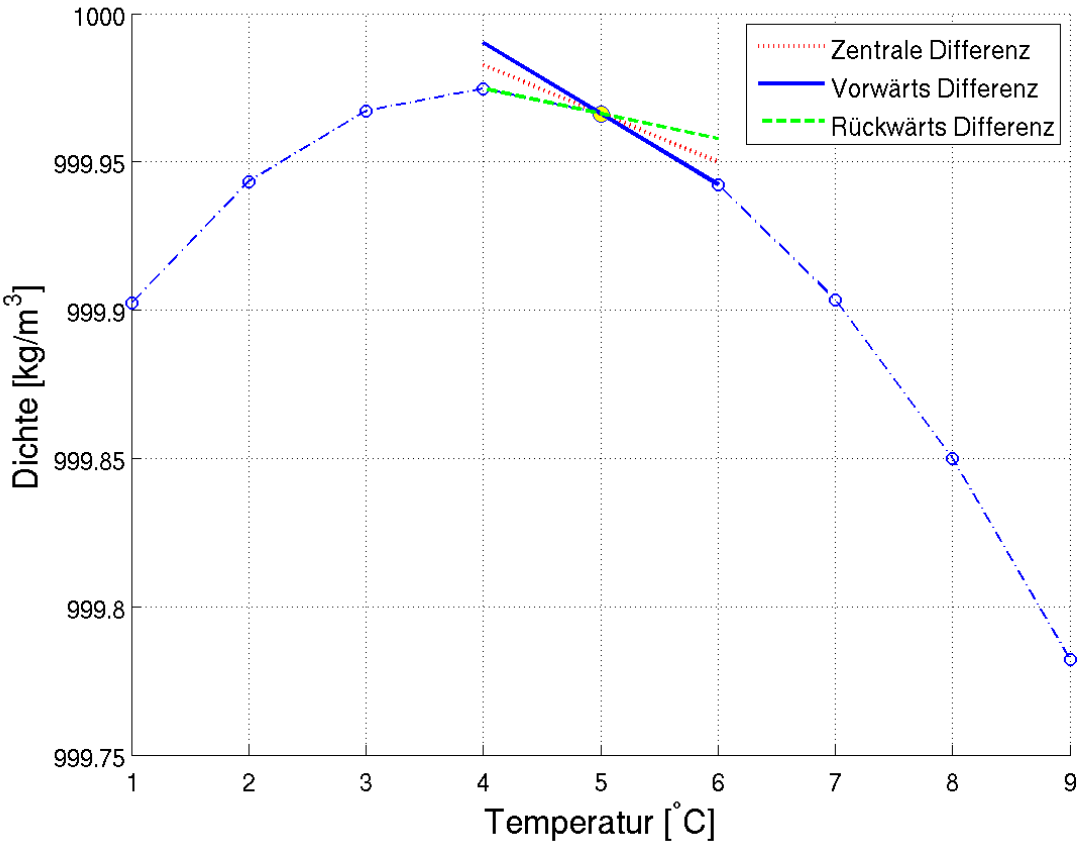


Abbildung 2.1: Visualisierung von drei Möglichkeiten numerisch die erste Ableitung einer Funktion (hier der temperaturabhängige Verlauf der Dichte von Wasser) zu bestimmen.

Der Fehler zur Bestimmung der Ableitung ist demnach von der Ordnung h. Der hier dargestellte Fehler des Diskretisierungsverfahrens wird als Abbruchfehler (engl. truncation error) bezeichnet. Ein Verfahren mit einer derartiger Fehlerordnung, wird als ein Verfahren erster Ordnung bezeichnet. Analog kann für die Rückwärtsdifferenz (2.3) vorgegangen werden:

$$f(x_{i-1}) = f(x_i) - \left.\frac{df(x)}{dx}\right|_{x_i} h + \left.\frac{d^2 f(x)}{dx^2}\right|_{x_i} \frac{h^2}{2} + \mathcal{O}\left(h^3\right), \tag{2.11}$$

$$\left.\frac{df(x)}{dx}\right|_{x_i} = \frac{f(x_i) - f(x_{i-1})}{h} + \left.\frac{d^2 f(x)}{dx^2}\right|_{x_i} \frac{h}{2} + \mathcal{O}\left(h^2\right) = \frac{f(x_i) - f(x_{i-1})}{h} + \mathcal{O}\left(h\right) \tag{2.12}$$

Es zeigt sich wieder die gleiche Größenordnung des Fehlers wie bei der Vorwärtsdifferenz. Zur Ermittlung des Fehlers der zentralen Differenz werden die Taylorentwicklungen nach Gleichung (2.9) und (2.11) in die Definition der zentralen Differenz, Gleichung (2.4), eingesetzt und anschließend nach der Ableitung aufgelöst:

$$\frac{f(x_{i+1}) - f(x_{i-1})}{2h} = \frac{2\left.\frac{df(x)}{dx}\right|_{x_i} h + 2\mathcal{O}\left(h^3\right)}{2h} = \left.\frac{df(x)}{dx}\right|_{x_i} + \mathcal{O}\left(h^2\right) \tag{2.13}$$

$$\left.\frac{df(x)}{dx}\right|_{x_i} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + \mathcal{O}\left(h^2\right) \tag{2.14}$$

Bemerkenswert ist hierbei, dass sich die Terme, welche die zweite Ableitung enthalten, gegenseitig aufheben und somit die Fehlerordnung auf h^2 erhöht wird. Man spricht von einem Verfahren zweiter Ordnung. Tatsächlich beschreibt die zentrale Differenz die Ableitung eines Polynoms zweiten Grades, welches durch die drei Punkte $n-1, n, n+1$ bestimmt ist. Die drei vorgestellten Möglichkeiten der numerischen Differentiation sind in Abbildung 2.1 visualisiert.

Zuletzt sei noch auf das Phänomen der **Auslöschung** hingewiesen, welches eine Folge des oben beschriebenen Fehlers ist. Unter Berücksichtigung aller Fehler werde angemommen die Funktion $f(x)$ liefere den Funktionswert bis auf fünf gültige Ziffern. Zur Approximation der Ableitung soll die Differenz $f(x+h) - f(x)$ gebildet werden. Eben wurde hergeleitet, dass die Näherung umso besser wird, je kleiner h gewählt wird. Je kleiner jedoch h gewählt wird, um so mehr identische Ziffern besitzen $f(x+h)$ und $f(x)$. Für immer kleiner werdendes h wird irgendwann der Punkt kommen bei dem alle fünf gültigen Ziffern dieser beiden Funktionswerte identisch sind. Das Ergebnis der Differenz wird damit fünf Nullen aufweisen, gefolgt von einer Reihe an zufälligen Ziffern ohne jede Aussagekraft. Man sagt alle gültigen Ziffern haben sich ausgelöscht. Der damit gebildete Gradient beinhaltet keinerlei verwertbarer Information mehr. Für jedes Problem exisitiert ein optimales h ab dem der Abbruchfehler des gewählten Verfahrens hinreichend klein ist und gleichzeitig noch keine Auslöschung auftritt. Dieses ist jedoch i.A. unbekannt. Daher sollte immer ein angemessen großer Wert gewählt werden.

Prinzipiell lassen sich numerische Verfahren zur Bestimmung der Ableitung beliebig hoher Ordnung herleiten. Ähnlich wie bei der Interpolation, werden hierfür jedoch zunehmend mehr Stützpunkte benötigt. Der numerische Aufwand für die Auswertung steigt dementsprechend, vor allem wenn es sich um eine numerisch kostspielig auszuwertende Funktion $f(x)$ handelt. Ein allgemeingültiges Vorgehen zur Herleitung für Näherungen beliebiger Genauigkeit und für beliebige Ableitungen soll im folgenden Abschnitt vorgestellt werden.

2.1.3 Systematische Herleitung von Näherungsformeln

Zunächst muss entschieden werden, wie viele Punkte zur Bestimmung der gewünschten Ableitung zur Verfügung stehen. Dabei sollte man beachten, dass i.d.R. die Anzahl der Punkte minus eins die maximale Ordnung des resultierenden Verfahrens ergibt. Es soll nun beispielhaft ein Ausdruck zweiter Ordnung zur Bestimmung der zweiten Ableitung hergeleitet werden. Um die Schreibarbeit etwas zu verringern, wird im Folgenden $f(x_i)$ mit f_i bzw. die Ableitung $df(x)/dx|_{x_i}$ als f_i' bezeichnet. Die zweite Ableitung an der Stelle x_i soll demnach durch eine Linearkombination aus drei Funktionswerten bestimmt werden:

$$f_i'' = af_{i-1} + bf_i + cf_{i+1} + \mathcal{O}\left(h^m\right) \tag{2.15}$$

Die Ordnung m des Verfahrens sei noch nicht bekannt. In diesem Fall handelt es sich um ein zentrales Verfahren, da Werte links und rechts des betrachteten x-Wertes x_i ausgewertet werden. Es können prinzipiell aber auch nur Werte rechts bzw. links davon verwendet werden (Stichwort „Upwind-Verfahren" siehe Numerische Thermofluiddynamik II). Alle fs in dieser Gleichung werden nun mittels einer Taylorentwicklung in Abhängigkeit von f_i bzw. dessen Ableitungen ausgedrückt. Dabei werden alle Terme explizit angegeben deren Ordnung kleiner oder gleich der erwarteten Ordnung des Verfahrens plus Eins ist. Im hier vorliegenden Beispiel also bis zur Ordnung drei:

$$f_{i-1} = f_i - f_i'h + f_i''\frac{h^2}{2} - f_i'''\frac{h^3}{6} + \mathcal{O}\left(h^4\right) \tag{2.16}$$

$$f_i = f_i \tag{2.17}$$

$$f_{i+1} = f_i + f_i'h + f_i''\frac{h^2}{2} + f_i'''\frac{h^3}{6} + \mathcal{O}\left(h^4\right) \tag{2.18}$$

Setzt man diese Gleichungen in (2.15) ein und sortiert nach den Ableitungen von f, so erhält man

$$f_i'' = (a+b+c)f_i + (c-a)hf_i' + (a+c)\frac{h^2}{2}f_i'' + (c-a)\frac{h^3}{6}f_i''' + (a+c)\mathcal{O}\left(h^4\right) + \mathcal{O}\left(h^m\right)). \tag{2.19}$$

Durch Koeffizientenvergleich der beiden Seiten dieser Gleichung, ergeben sich vier linear unabhängige Gleichungen für die vier Unbekannten (a, b, c, m):

$$a + b + c = 0 \tag{2.20}$$

$$(c-a)h = 0 \tag{2.21}$$

$$(a+c)\frac{h^2}{2} = 1 \tag{2.22}$$

$$(c-a)\frac{h^3}{6} = 0 \tag{2.23}$$

$$(a+c)\mathcal{O}\left(h^4\right) + \mathcal{O}\left(h^m\right) = 0 \tag{2.24}$$

Gleichungen (2.21) und (2.23) beinhalten die gleiche Information bzgl. a und c, sind also linear abhängig. Löst man dieses System nun nach den Unbekannten auf, so ergibt sich

$$a = \frac{1}{h^2} \tag{2.25}$$

$$b = -\frac{2}{h^2} \tag{2.26}$$

$$c = \frac{1}{h^2} \tag{2.27}$$

$$O(h^m) = \frac{2\mathcal{O}\left(h^4\right)}{h^2} = \mathcal{O}\left(h^2\right). \tag{2.28}$$

Damit kann also unter Verwendung von Gleichung (2.15) für die zweite Ableitung folgende Näherung zweiter Ordnung angegeben werden:

$$f_i'' = \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} + \mathcal{O}\left(h^2\right). \tag{2.29}$$

Je nachdem was für ein Problem vorliegt, können andere Stellschrauben zur Erhöhung der Genauigkeit eingestellt werden. Liegt beispielsweise eine diskrete Messreihe vor, so kann lediglich die Ordnung des Verfahrens verändert werden. Kann hingegen die zu untersuchende Funktion an beliebigen Stellen ausgewertet werden, so kann zusätzlich der Abstand h verkleinert werden. Dabei kann dieser Abstand zusätzlich an die lokale Krümmung der Funktion angepasst werden ($h = h(x)$): Für Orte mit geringer Krümmung - und damit einer guten Approximierbarkeit durch Polynome geringen Grades - kann ein größerer Abstand h gewählt werden als für Orte hoher Krümmung. Diese Idee wird in Kapitel 7 für das Runge-Kutta-Verfahren mit adaptiven Zeitschritt noch einmal aufgegriffen werden.

Ein weiteres interessantes Detail, welches durch die gezeigte Herleitung deutlich wird, ist, dass die Summe der Koeffizienten eines finite Differenzen-Verfahrens Null ist. Dies folgt unmittelbar aus der Forderung, dass sich Taylor-entwickelte Terme gleicher Größenordnung in h gegenseitig aufheben sollen. Mithilfe dieses Wissens kann schnell die Korrektheit einer finiten Differenzen-Diskretisierung überprüft bzw. ggf. ein Fehler entdeckt werden.

Abschließend sei darauf hingewiesen, dass noch weitere Verfahren zur numerischen Differentiation existieren. Ist z.B. eine Messreihe gegeben, welche Rauschen enthält, so kann zunächst mittels des Least-Square-Ansatzes (siehe Kapitel 4) z.B. ein polynomialer Fit der Messdaten bestimmt und aus diesem im Anschluss die Ableitung berechnet werden. Da somit die gefittete Kurve analytisch vorliegt, liegen auch die Ableitungen analytisch vor. Eine weitere Methode besteht in der Approximation der gegebenen Datenreihe durch Splines. Auch hier kann stückweise eine analytische Form der Ableitung angegeben werden. Für eine detailliertere Darstellung dieser Ideen bzw. für weitere Methoden wird auf [1], [3], [2] verwiesen.

2.2 Übungsaufgaben

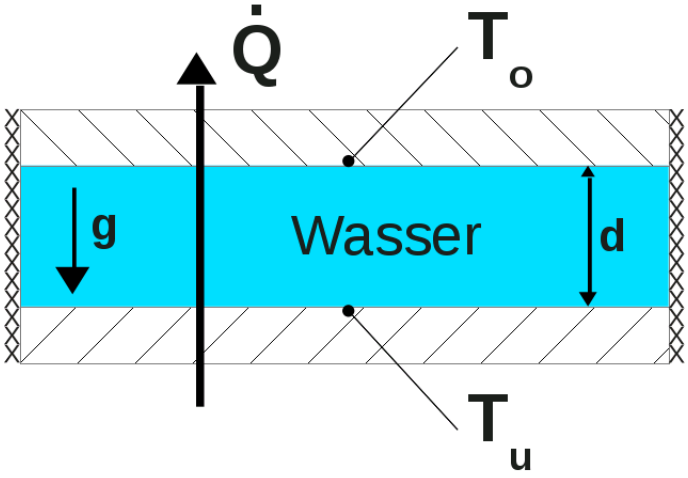


Abbildung 2.1: Versuch zur Bestimmung des Wärmedurchgangs in einer Flüssigkeitsschicht.

Zur Untersuchung des Wärmedurchgangs in einer waagerechten Wasserschicht wird der in Abbildung 2.1 gezeigte Aufbau verwendet. Für eine gegebene Schichthöhe d wird die Temperatur der unteren Platte ausgehend von $T_u = T_o$ erhöht und dabei der sich einstellende Wärmestrom gemessen. Es zeigt sich ein linearer Zusammenhang bis zu einer bestimmten Temperaturdifferenz $\Delta T_{krit} = T_u - T_o$. Ab dieser Differenz nimmt der Wärmestrom überproportional zu. Der Grund hierfür ist die Entstehung einer Konvektionsströmung zwischen den Platten, welche zusätzlich zur normalen Wärmediffusion wirkt. Diese Strömung findet in sogenannten Konvektionszellen statt, in welchen das Fluid zirkuliert.

Treibende Kraft für das Einstellen einer Konvektionsströmung ist das Vorhandensein eines Dichtegradienten innerhalb der Flüssigkeitsschicht in vertikaler Richtung. Aufgrund der Abhängigkeit der Dichte des Wassers von der Temperatur, wird durch einen Temperaturgradienten auch ein Dichtegradient erzeugt.

2.1 Berechnen Sie den Verlauf des Dichtegradienten von Wasser in Abhängigkeit der Temperatur. Nehmen Sie einen Druck von $p = 1\ bar$ an. Verwenden Sie hierfür die von XSteam bereitgestellten Stoffwerte und bestimmen Sie die gesuchten Gradienten mittels numerischer Differentiation (vorwärts, rückwärts und zentral) aus dem Verlauf der Dichte über der Temperatur. Plotten Sie den Verlauf des Gradienten sowie den der Dichte für Temperaturen zwischen ein und neun Grad Celsius. Was stellen Sie fest?

Die Existenz eines Dichtegradienten in der Flüssigkeitsschicht ist lediglich eine notwendige Bedingung für das Auftreten einer Konvektionsströmung. Wie oben beschrieben, zeigt sich erst ab einem bestimmten Temperatur- bzw. Dichtegradienten ein konvektiver Einfluss auf den Wärmeübergang. Dieser Punkt wird mittels einer dimensionslosen Zahl, der Rayleigh-Zahl, bestimmt:

$$Ra = \frac{g\beta(T_u - T_o)d^3}{\nu a} \tag{2.1}$$

Hierbei ist g die Erdbeschleunigung, $\nu = \eta/\rho$ die kinematische Viskosität, $\beta = -1/\rho \cdot (\partial\rho/\partial T)_p$ der thermische Ausdehnungskoeffizient und $a = \lambda/(\rho c)$ die Temperaturleitfähigkeit mit der spezifischen Wärmekapazität c. Nimmt dieses Zahl einen Wert größer als die kritische Rayleigh-Zahl Ra_{krit} an, so liegt keine reine Wärmeleitung mehr vor. Der Wärmedurchgang wird zunehmend von konvektiven Vorgängen bestimmt.

T_0	$25\,°C$
d	$3\,mm$
p	$1\,bar$
Ra_{krit}	1700

2.2 Fertigen Sie für die folgende Aufgabe ein Ablaufdiagramm an, bevor Sie mit der Programmierung beginnen.

2.3 Silveston 1958 [1] gibt die kritische Rayleigh-Zahl zu $Ra_{krit} = 1700$ an. Ermitteln Sie grafisch für eine Schichtdicke von $d = 3\ mm$, einem Druck von $p = 1\ bar$ und einer Temperatur der oberen Platte von $T_o = 25°C$ die kritische Temperaturdifferenz zwischen den Platten, ab der Konvektion auftritt. Entwickeln Sie anschließend ein Verfahren, welches Ihnen den numerischen Wert der gesuchten Temperaturdifferenz (näherungsweise) ausgibt. Zur Bestimmung der Stoffwerte soll die jeweilige mittlere Temperatur $T_m = \dfrac{T_u + T_o}{2}$ verwendet werden.

Hinweise:

- Tipp zum Vorgehen: Nehmen Sie zunächst eine konstante Temperaturdifferenz an und berechnen Sie hierfür die zugehörige Rayleigh-Zahl. Anschließend können Sie mittels einer for-Schleife die Temperatur der unteren Heizplatte T_u variieren.
- Zur Bestimmung der Stoffwerte sollte wieder XSteam verwendet werden.
- Zur einfacheren grafischen Lösung sollten Sie zusätzlich eine horizontale Linie bei der kritischen Rayleigh-Zahl einfügen.

2.3 Wichtige Matlab Befehle

Befehl	Beschreibung
`diff()`	Berechnet die Abstände jeweils zweier benachbarter Elemente eines Vektors
`line()`	Erzeugt eine Linie in einem Plot
`strcmp()`	Untersucht ob zwei Strings identisch sind

3 Numerische Integration

Literaturverzeichnis

[1] Brian P Flannery, William H Press, Saul A Teukolsky und William Vetterling. *Numerical Recipes in C*. Kap. 4. 1992. URL: http://www.nr.com/.

[2] Josef Stoer und Roland Bulirsch. *Introduction to Numerical Analysis*. Bd. 12. Springer, 2002.

Lernziele

- Trapezregel
- Simpsonregel
- Gauß-Quadratur
- Numerische Implementierung der verschiedenen Integrationsverfahren

Motivation

Im ingenieurwissenschaftlichen Alltag ist es relativ häufig erforderlich Integrale auszuwerten. Sei es um einen Mittelwert zu bestimmen, Energien zu berechnen oder Volumen bzw. Flächen zu ermitteln. Oftmals sind diese Integrale analytisch entweder gar nicht oder nur mit großem Aufwand zu lösen. Oder es liegt eine diskrete Messreihe vor und es ist erforderlich das Integral über diese zu berechnen. In all diesen Fällen ist es hilfreich robuste, schnelle und genaue numerische Verfahren zur Integration zu kennen. Vor allem früher wurde die Berechnung von Flächeninhalten oftmals als Quadratur bezeichnet. Auch heute noch ist z.B. das in diesem Kapitel vorgestellte Verfahren von Gauß als die sogenannte Gauß-Quadratur bekannt.

Als konkretes Beispiel sei die Berechnung einer Flammenoberfläche genannt. Oftmals wird eine Flamme als eine Diskontinuität modelliert. Strömungsgrößen, wie z.B. die Dichte oder die Temperatur, verändern hier an der Position der Flamme sprunghaft ihren Wert. Ein derartiges Vorgehen ist vielfach z.B. für laminare Vormischflammen möglich, da hier die Reaktionszone (Zone der Verbrennung) sehr dünn ist und auch die Temperatur, Dichte, etc. nahezu instantan ihre Werte ändern. Im dreidimensionalen Fall ist diese Diskontinuität eine Fläche. Zur Bestimmung der Dynamik der Flamme können Gleichungen hergeleitet werden, die beschreiben wie die als Diskontinuität modellierte Flamme auf die sie umgebende Strömung reagiert. Eine Momentaufnahme einer auf dieser Idee beruhenden Simulation ist in Abbildung 3.1 dargestellt. Die Flamme wurde hierbei zur besseren Darstellung in der Mitte aufgeschnitten. (Aus Gründen der Modellierung fehlt die Spitze der Flamme; dies soll hier allerdings nicht weiter diskutiert werden.)

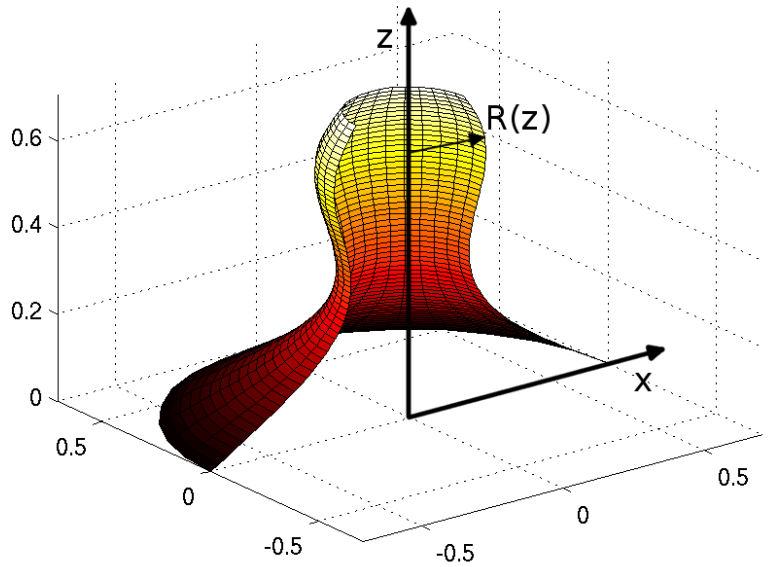


Abbildung 3.1: Flammenoberfläche im Schnitt.

Um den Einfluss der Flamme auf die Akustik oder das sie umgebende Gehäuse abschätzen zu können, muss ihre aktuelle Wärmefreisetzungsrate ermittelt werden. Um deren Bestimmung zu vereinfachen, kann oftmals angenommen werden, dass die Änderung der Wärmefreisetzungsrate proportional zur Änderung der Flammenoberfläche ist. Die Berechnung der Flammenoberfläche macht jedoch für jeden Zeitschritt die numerische Integration des folgenden Ausdruckes erforderlich:

$$A(t) = \iint_F dA = 2\pi \int_0^H R(z,t)\sqrt{1 + \left(\frac{dR(z,t)}{dz}\right)^2}\, dz \tag{3.1}$$

Hierbei beschreibt $A(t)$ die zeitabhängige Flammenoberfläche, H die Höhe der Flamme und $R(z,t)$ den zeitabhängigen Verlauf des Radius der rotationssymmetrischen Flamme. Aus der numerischen Simulation sind nur die Radien an den n Stützstellen z_i bekannt. Abgekürzt können diese mit $R(z_i,t) = R_i(t)$ bezeichnet werden. Wie anhand dieser Informationen Integrale wie das aus Gleichung (3.1) ausgewertet werden können, wird im ersten Teil dieses Kapitels erörtert. Im zweiten Teil werden Methoden und Algorithmen zur effizienten numerischen Implementierung dieser Integrationsverfahren beschrieben. Abschließend wird schließlich die erarbeitete Theorie auf das Problem der Flammenoberfläche angewandt.

3.1 Methoden zur numerischen Auswertung von Integralen

Wie bei der Differentiation soll auch hier zunächst intuitiv vorgegangen werden. Eingeführt wurde die Integration im Rahmen des Schulunterrichts als Grenzwert der Summe von Flächeninhalten einer Treppenfunktion, dem sogenannten Riemannschen Integral (hier am Beispiel der Untersumme mit konstanten Intervallen $h = x_{i+1} - x_i$ gezeigt):

$$\int_a^b f(x)dx = \lim_{h\to 0} h \sum_{i=0}^{n(h)} f(x_i) \tag{3.2}$$

Die meisten Verfahren zur numerischen Integration bauen auf dieser Idee der Approximation mittels summierter Teilflächen auf. Im Allgemeinen lässt sich damit eine Quadraturformel wie folgt darstellen:

$$\int_a^b f(x)dx = h \sum_{i=0}^{n} w_i f(x_i) + E \tag{3.3}$$

Das Integral wird also durch die Summe von endlich vielen gewichteten Funktionswerte an sogenannten Stützstellen angenähert. Diese Stützstellen werden im Folgenden mit x_i und die Gewichtungsfaktoren mit w_i benannt. Je nach Verfahren nehmen sie andere Werte an. Für eine finite Näherung mittels des Riemannschen Ausdrucks aus Gleichung (3.2) beispielsweise ergäben sich die Gewichtungsfaktoren zu eins. Die Position der Stützstellen kann bei den hier vorgestellten Methoden frei gewählt werden. Zur einfacheren Berechnung sollten die Abstände zwischen diesen jedoch äquidistant sein. Einzig die Gauß-Quadratur gibt eine (optimale)

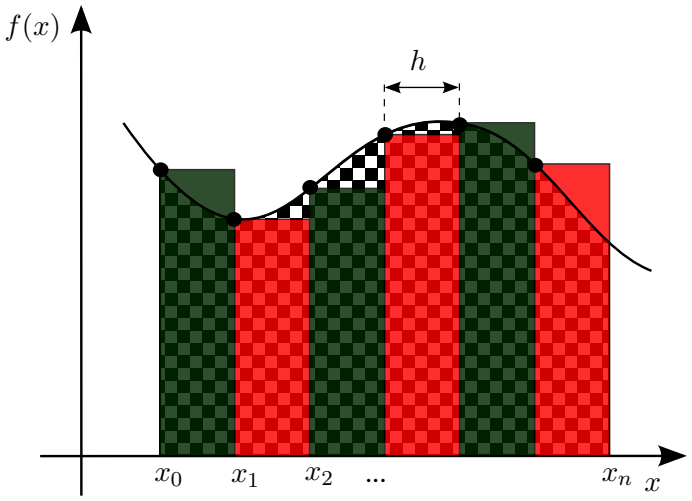


Abbildung 3.2: Veranschaulichung der Definition des Integrals: Für $h \to 0$ stimmt die Fläche der Rechtecke (Untersumme) mit der schachbrettartig gemusterten Fläche unter dem Graph der Funktion überein.

Positionierung der x_i an, so dass dadurch Polynome mit möglichst großer Ordnung exakt integrierbar sind.

Der durch die Näherung verursachte Fehler wird mit E bezeichnet. Wovon dieser abhängt und wie groß dieser ist, ist verfahrensabhängig. Daher wird in diesem Kapitel zu jedem vorgestellten Verfahren eine Abschätzung dieses Fehlers mit angegeben.

Die Faktoren w_i ergeben sich letztendlich aus der konkreten Wahl einer Näherungsfunktion, die den Funktionsverlauf zwischen den Stützstellen beschreiben soll. Es zeigt sich hier wieder, dass es - wie bereits bei der Interpolation und der numerischen Differentiation - notwendig ist, Annahmen über einen Funktionsverlauf zwischen diskreten Punkten zu treffen. Dieser sollte so universell wie möglich sein und dabei den wahren Verlauf möglichst exakt treffen. In allen hier vorgestellten Verfahren werden hierfür Polynome unterschiedlicher Ordnung eingesetzt.

Die ersten zwei im Folgenden vorgestellten Verfahren, die Trapezregel und die Simpsonregel, sind aus der Gruppe der sogenannten abgeschlossenen Newton-Cotes-Formeln. Bei der dritten beschriebenen Methode handelt es sich um die Gauß-Quadratur, ein offenes Integrationsverfahren. Derartige Verfahren zeichnen sich dadurch aus, dass sie die Intervallgrenzen der Integration nicht als Stützstellen enthalten. Abgeschlossene Verfahren hingegen enthalten auch diese. Damit eignen sie sich besonders für itarative Verfahren, wie wir sie in Kapitel 3.2 kennen lernen werden.

3.1.1 Trapezregel

Die Idee der Trapezregel ist es zwischen zwei Punkten x_{i-1} und x_i einen linearen Verlauf anzunehmen und damit die Fläche der Teilintervalle zu bestimmen. Wir integrieren also zwischen zwei Stützstellen über eine lineare Interpolationfunktion $p(x) = p_1(x)$, anstatt über den tat-

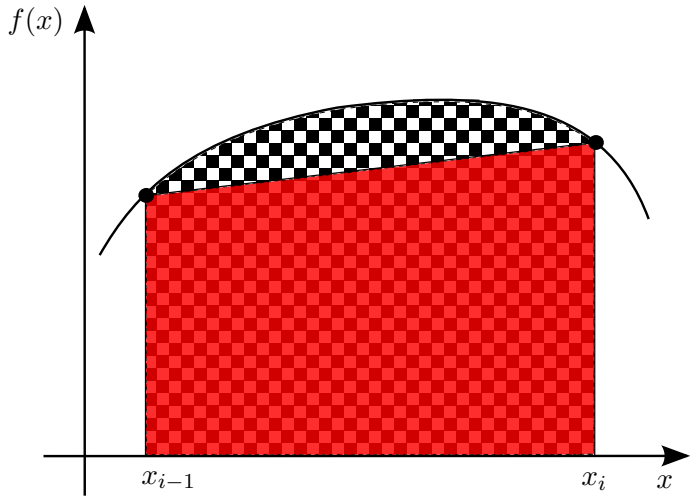


Abbildung 3.3: Veranschaulichung des Flächenstücks $A_{j,1}$ der Breite h.

sächlichen, unbekannten Funktionverlauf $f(x)$. Der Allgemeinheit halber wird zunächst ein allgemeines Polynom $p(x)$ in dem Approximationsansatz verwendet. Erst in einem zweiten Schritt wird es durch das lineare Polynom $p_1(x)$ ersetzt. Der Index 1 zeigt an, dass es sich um ein Polynom ersten Grades handelt. Die dadurch entstehenden Flächen ist ein Trapez, dessen Inhalt sich wie folgt bestimmen lässt:

$$A_{j,1} = \int_{x_{i-1}}^{x_i} f(x)dx = \int_{x_{i-1}}^{x_i} p(x)dx + E \overset{p=p_1}{=} h\frac{f_{i-1}+f_i}{2} + E_1. \tag{3.4}$$

Aufgrund der kürzeren Schreibweise wird $f(x_i)$ durch f_i ersetzt. Die Fläche wird mit dem Index j, mit Hilfe dessen sich Flächen unterscheiden lassen, sowie mit einem Index 1 versehen, der die polynomiale Ansatzfunktion vom Grad Eins beschreibt. Grafisch ist dieses Vorgehen in Abbildung 3.3 dargestellt.

Für den Fehler E_1 kann in Abhängigkeit von h und f folgende Abschätzung getroffen werden [1]:

$$E_1 = \int_{x_{i-1}}^{x_i} (f(x) - p_1(x))dx = \mathcal{O}\left(h^3 f''\right). \tag{3.5}$$

Die Funktion $f(x)$ muss auf den Intervall zweimal stetig differenzierbar sein. Für diese Abschätzung ist der maximale Wert der zweite Ableitung f'' auf den betrachteten Intervall einzusetzen, der allerdings im Allgemeinen unbekannt ist. Es wird deutlich, dass mittels der Trapezregel Polynome bis maximal ersten Grades exakt integriert werden können, da deren zweite Ableitung immer null ist und der Fehler damit verschwindet.

Möchte man nun über einen größeren Bereich numerisch integrieren, so kann die betrachtete Funktion im Allgemeinen nicht auf dem gesamten Integrationsgebiet als linear betrachtet werden. Eine lokale lineare Näherung liefert, wie bereits in den vorangegangenen Kapiteln gezeigt wurde, eine deutlich bessere Näherung. Das gesamte Gebiet wird daher in n gleich breite

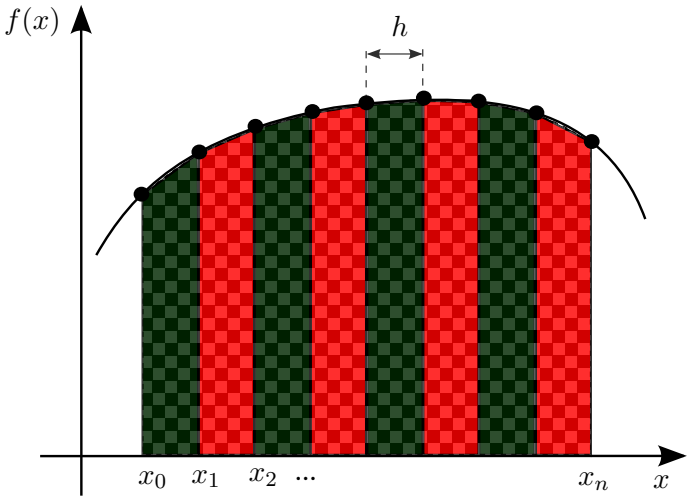


Abbildung 3.4: Veranschaulichung der Summe aus Formel (3.7). Das gesamte Integral wird durch die Summe mehrere Einzeltrapeze $A_{j,1}$ der Breite h approximiert.

Teilgebiete aufgeteilt. In jedem dieser Gebiete wird über den linearen Funktionsverlauf, wie in Gleichung (3.4) beschrieben, integriert. Damit ergibt sich das gesamte Integral als Summe dieser Teilintegrale:

$$\int_{x_0}^{x_n} f(x)dx = \sum_{j=1}^{n} A_{j,1} = h\left[\left(\frac{f_0}{2}+\frac{f_1}{2}\right)+\left(\frac{f_1}{2}+\frac{f_2}{2}\right)+\cdots+\left(\frac{f_{n-1}}{2}+\frac{f_n}{2}\right)\right]+n\mathcal{O}\left(h^3 f''\right) \tag{3.6}$$

Alle inneren Stützstellen kommen in dieser Summe zwei Mal vor und addieren sich jeweils zu einem Vorfaktor von eins. Da das Integrationsgebiet in n gleichgroße Teilintervalle aufgeteilt wurde, kann für die Größe eines dieser Intervall $h = (x_n - x_0)/n$ geschrieben werden. Setzt man diese Definition in die Formulierung des Fehlers in Gleichung (3.6) ein und vereinfacht, so ergibt sich folgender Ausdruck mit dem sich beliebige Integrale annähern lassen:

Definition: Trapezregel

$$\int_{x_0}^{x_n} f(x)dx = \underbrace{h\left(\frac{f_0+f_n}{2}+\sum_{i=1}^{n-1} f_i\right)}_{=:S_{n,1}} + \mathcal{O}\left(\frac{(x_n-x_0)^3 f''}{n^2}\right) \tag{3.7}$$

In Abbildung 3.4 ist dieses Vorgehen noch einmal veranschaulicht. Der Fehler skaliert in der Definition in der dritten Potenz mit der Größe des Intervalls und linear mit dem Maximum der zweiten Ableitung des gesamten Intervalls. Es werden daher Polynome bis zur ersten Ordnung exakt integriert. Dabei nimmt der Fehler quadratisch mit der Anzahl der Teilintervalle ab.

3.1.2 Simpsonregel

Bei der Simpsonregel erhöhen wir den Grad des Approximationspolynoms in Gleichung (3.4) um eins. Wir nehmen also jetzt ein Polynom zweiten Grades $p_2(x)$ an und integrieren über dieses. Diese Vorgehen ist schematisch in Abbildung 3.5 veranschaulicht und ist in Formeln gegeben als:

$$A_{j,2} = \int_{x_{2i-2}}^{x_{2i}} f(x)dx = \int_{x_{2i-2}}^{x_{2i}} p(x)dx + E \overset{p=p_2}{=} \frac{h}{3} \cdot (f_{i-2} + 4f_{i-1} + f_i) + E_2. \tag{3.8}$$

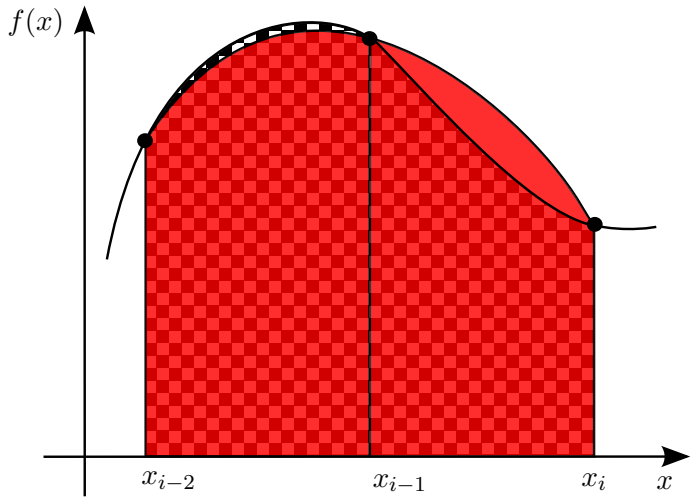


Abbildung 3.5: Veranschaulichung des Flächenstücks $A_{j,2}$.

In diesem Fall muss also die Funktion zusätzlich an einem dritten Punkt ausgewertet werden. Dieser befindet sich in der Mitte des Intervalls. Der Fehler kann hier als proportional zum Produkt der fünften Potenz der Intervallgröße $h = x_{i+1} - x_i$ und der vierten Ableitung der Funktion $f(x)$ angegeben werden [1]:

$$E_2 = \int_{x_{i-2}}^{x_i} (f(x) - p_2(x))dx = \mathcal{O}\left(h^5 f''''\right). \tag{3.9}$$

Zur genaueren Bestimmung eines Integrals kann das gesamte Integrationsgebiet wiederum in Teilintervalle zerlegt werden. Es ist zu beachten, dass für jede Auswertung der Simpsonregel nun zwei Teilintervalle bzw. drei Stützstellen benötigt werden. Aus n ($n \in 2\mathbb{N}$) Teilintervallen lassen sich also $n/2$ Teilintegrale erstellen. Addiert man diese auf, so ergibt sich für das gesamte Integral

$$\begin{aligned}\int_{x_0}^{x_n} f(x)dx = \sum_{j=1}^{n/2} A_{j,2} = \frac{h}{3}\left[(f_0 + 4f_1 + f_2) + (f_2 + 4f_3 + f_4) + \ldots\right. \\ \left. + (f_{n-2} + 4f_{n-1} + f_n)\right] + \frac{n}{2}\mathcal{O}\left(h^5 f''''\right).\end{aligned} \tag{3.10}$$

In dieser Summe kommen alle Punkte mit geraden Indizes doppelt vor, wodurch diese zu einem Vorfaktor 2 addiert werden können. Die Summanden an den ungeraden inneren Punkten kommen jeweils nur einmal vor und behalten damit den Vorfaktor 4. Zur Bestimmung des Fehlers soll wieder die Intervallgröße h durch $(x_n - x_0)/n$ ersetzt werden. Damit ergibt sich die zusammengesetzte Simpsonregel wie folgt:

Definition: Simpson

$$\int_{x_0}^{x_n} f(x)dx = \underbrace{\frac{h}{3}\left(f_0 + 4\sum_{i=1}^{n/2} f_{2i-1} + 2\sum_{i=1}^{n/2-1} f_{2i} + f_n\right)}_{=:S_{n,2}} + \mathcal{O}\left(\frac{(x_n - x_0)^5 f''''}{n^4}\right) \tag{3.11}$$

Das Prinzip ist in Abbildung 3.6 dargestellt. Eine Teilfläche $A_{j,2}$ ist dabei mit einer einheitlichen Farbe gekennzeichnet. n Stützstellen ergeben demnach $(n-1)/2$ Teilflächen der Breite $2h$. Mithilfe dieser Formel lassen sich also Polynome bis zur Ordnung drei exakt integrieren. Damit hat man durch Erhöhung des Approximationspolynoms um lediglich einen Grad zwei Grade bzgl. der exakt zu integrierenden Polynome gewonnen. Es zeigt sich, dass bei Verwendung von Polynomen dritter Ordnung ebenfalls nur Polynome bis zur Ordnung drei exakt integriert werden können. Erst für einen Grad von vier erhöht sich die Fehlerordnung wieder um eins. Allgemein erhöht sich der Grad der exakt zu integrierenden Funktionen nur für Approximationspolynome gerader Ordnung [1].

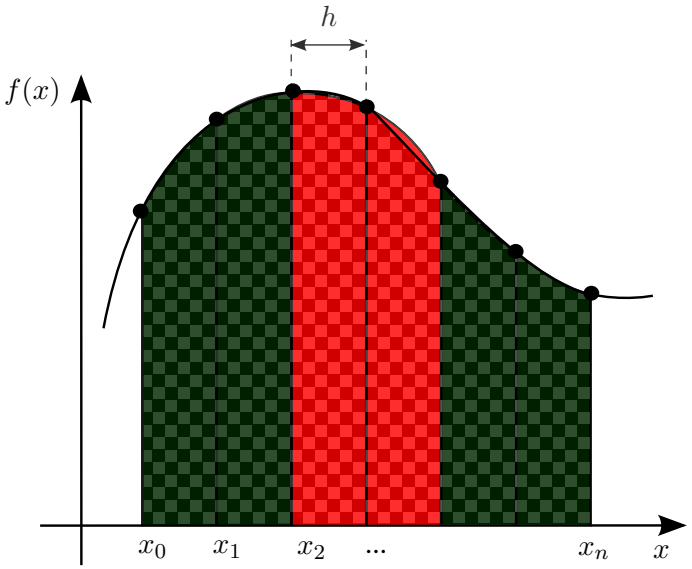


Abbildung 3.6: Veranschaulichung der Summe aus Formel (3.11). Das gesamte Integral wird durch die Summe mehrere Einzelflächen $A_{j,2}$ der Breite $2h$ approximiert.

Es zeigt sich ein in der Praxis nützlicher Zusammenhang zwischen der Trapez- und der Simpsonregel:

$$S_{n,2} = \frac{4}{3}S_{n,1} - \frac{1}{3}S_{n/2,1}. \tag{3.12}$$

Hierbei bezeichnet $S_{n,2}$ den Summenterm der zusammengesetzten Simpsonregel und $S_{n,1}$ den entsprechenden Term der Trapezregel für $n+1$ Stützstellen (Achtung: Die Nummerierung beginnt bei 0, daher ist n der höchste Index!). $S_{n/2,1}$ bezeichnet damit die Trapezregel für $n/2$ Intervalle. Die beiden letztgenannten Terme lassen sich unter Verwendung von Definition (3.7) wie folgt darstellen:

$$S_{n,1} = h\left(\frac{f_0+f_n}{2} + \sum_{i=1}^{n-1} f_i\right) = h\left(\frac{f_0+f_n}{2} + \sum_{i=1}^{n/2} f_{2i-1} + \sum_{i=1}^{n/2-1} f_{2i}\right), \tag{3.13}$$

$$S_{n/2,1} = 2h\left(\frac{f_0+f_n}{2} + \sum_{i=1}^{n/2-1} f_{2i}\right). \tag{3.14}$$

Es ist zu berücksichtigen, dass $S_{n/2,1}$ die doppelte Schrittweite wie $S_{n,1}$ besitzt, also $2h$. Führt man nun die Berechnung nach (3.12) durch, so ergibt sich die Simpsonregel. Die Relevanz dieser Beziehung für die Implementierung dieser Integrationsverfahren wird in Kapitel 3.2 deutlich.

3.1.3 Die Gauß-Quadratur

Dieses Thema soll aufgrund des Umfanges der mit der Gauß-Quadratur assoziierten Theorien und Anwendungen nur angeschnitten werden. Die Gauß-Quadratur erlaubt im Prinzip Integrale über eine Klasse von Funktionen exakt durch eine Summe

$$\int_{-1}^{1} f(z)dz = \sum_{i=1}^{n} w_i f(z_i) \tag{3.15}$$

darzustellen. Hierbei ist es lediglich erforderlich die **Stützstellen** z_i und die zugehörigen **Gewichte** w_i für die jeweilige Klasse an Funktionen zu kennen sowie das gesuchte Integral über das Gebiet $[a,b]$ in ein Integral über das Gebiet $[-1,1]$ zu transformieren. Dies geschieht mittels der Vorschrift

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}z + \frac{a+b}{2}\right)\,dz. \tag{3.16}$$

Damit lässt sich im Prinzip die Gauß-Quadratur anwenden. Das Vorgehen soll im Folgenden anhand der Gauß-Legendre-Integration gezeigt werden:

- Transformation der Integrationsgrenzen nach Gleichung (3.16).
- Anwenden von Formel (3.15):

$$\frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}z+\frac{a+b}{2}\right)dz = \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}z_i+\frac{a+b}{2}\right) \tag{3.17}$$

- Entnehmen der n Werte für w_i und z_i aus Tabelle 3.1 sowie Auswerten der resultierenden Summe.

	z_i	w_i
$n=1$	0	2
$n=2$	$-\sqrt{\frac{1}{3}}$ $\sqrt{\frac{1}{3}}$	1 1
$n=3$	$-\sqrt{\frac{3}{5}}$ 0 $\sqrt{\frac{3}{5}}$	$\frac{5}{9}$ $\frac{8}{9}$ $\frac{5}{9}$

Tabelle 3.1: Gewichte und Gaußpunkte der Gauß-Legendre-Integration.

Mithilfe dieses Vorgehens können Sie Polynome bis zum Grad $2n-1$ exakt integrieren! Anders formuliert: Mit den in Tabelle 3.1 gegebenen Werten für $n=3$ können Integrale über Polynome vom Grad fünf bei nur drei Stützstellen exakt ausgewertet werden. Der Vorteil dieser Methode ist damit klar die überlegene Geschwindigkeit der darauf basierenden Implementierungen. Im Vergleich zu den oben vorgestellten Newton-Cotes-Verfahren ist die Ordnung der exakt integrierbaren Polynome bei gleicher Anzahl an Stützstellen (und damit Funktionsauswertungen) fast doppelt so groß. Die höhere Genauigkeit ergibt sich dabei aufgrund einer optimalen Platzierung der Stützstellen im Integrationsgebiet.

Die Grundidee der Gauß-Quadratur ist dabei, dass die zu integrierende Funktion $f(z)$ auf dem Intervall $[-1,1]$ mit dem Produkt aus einem Polynom $p(z)$ und einer Gewichtsfunktion $w(z)$ angenähert werden kann:

$$\int_{-1}^{1} f(z)dz \approx \int_{-1}^{1} w(z)p(z)dz \tag{3.18}$$

Für diesen Ausdruck kann nun die Gewichtsfunktion $w(z)$ so gewählt werden, dass $f(z)$ sich möglichst gut durch das Produkt aus $w(z)$ mit einem Polynom darstellen lässt. Für die sogenannte Gauß-Legendre-Integration gilt $w(z)=1$. Für $p(z)$ werden Legendre-Polynome verwendet. Mit diesen Ansatz lässt sich dann das Integral über $w(z)p(z)$ exakt als Summe darstellen, wie in Gleichung (3.15) gezeigt. Stimmt nun $f(z)$ noch exakt mit $w(z)p(z)$ überein, so

ist diese Summe eine exakte Darstellung des gesuchten Integrals. Aber auch falls $f(x)$ hinreichend glatt ist, kann eine gute Näherung erzielt werden. Je nach Grad des Polynoms $p(z)$ ergeben sich unterschiedlich viele Summanden. Näheres zur Theorie und zur Berechnung der Gewichte und Stützstellen finden Sie z.B. in [2].

Die Gauß-Quadratur hat allerdings mehr zu bieten als nur die exakte Integration von Polynomen. Es können für verschiedene Klassen von Funktionen Verfahren analog zu dem oben dargestellten entwickelt werden, die für diese exakt sind. Damit ist diese Methode vor allem dann hilfreich, wenn Sie schon vorher ungefähr wissen, wie Ihre zu integrierende Funktion aussieht und Sie nur begrenzte Kapazitäten zur Verfügung haben (z.B. auf Embedded Systems), bzw. ein Integral sehr oft auswerten müssen.

3.2 Implementierung numerischer Integrationsverfahren

In diesem Kapitel sollen Implementierungsmöglichkeiten der oben vorgestellten Verfahren entwickelt und erläutert werden. Es wird lediglich auf die Trapez- und Simpsonregel eingegangen.

Es muss zwischen zwei Fällen unterschieden werden: Ist bereits eine Reihe tabellierter Werte (z.B. Messwerte) vorgegeben, so steht die Anzahl der Stützstellen fest und es kann lediglich der Grad des Approximationspolynoms variiert werden. Die Simpsonregel lässt sich nur dann anwenden, falls eine gerade Zahl an Teilintervallen vorliegt. Andernfalls kann das erste bzw. das letzte Intervall mittels der Trapezregel ausgewertet werden, so dass die Anzahl der übrigen Intervall wieder gerade ist.

In den meisten Fällen liegt allerdings eine Funktion $f(x)$ vor, z.B. in Form einer MATLAB-Funktion, die für beliebige x ausgewertet werden kann. Der Aufwand um dies zu tun kann dabei beliebig groß sein. Daher ist es i.A. sinnvoll $f(x)$ so selten wie möglich zu evaluieren. Der Fehlerterm der Definitionen (3.7) und (3.11) beinhaltet dabei die Anzahl der Stützstellen $n+1$, welche auch der Anzahl der notwendigen Funktionsaufrufen entspricht. Der Fehler der Trapezregel skaliert mit $1/n^2$, der Fehler der Simpsonregel mit $1/n^4$. Für hinreichend glatte Funktionen wird die Simpsonregel daher bei gleicher Genauigkeit mit weniger Teilintervallen auskommen als die Trapezregel.

In der Praxis stellt sich die Frage, wie viele Intervall benötigt werden, damit das Verfahren das gesuchte Integral mit einer definierten Genauigkeit bestimmt. Für eine unbekannte Funktion $f(x)$ kann hierfür nicht von vorne herein eine feste Anzahl an Teilintervallen bzw. Stützstellen angegeben werden. Aus diesem Grund ist es sinnvoll die Anzahl der Intervalle so lange zu erhöhen, bis die dadurch entstehende Änderung der Fläche kleiner als ein hinreichend klein zu wählender Wert ϵ wird, also

$$|S_{n_1,x} - S_{n_2,x}| < \epsilon, \qquad n_1 < n_2. \tag{3.19}$$

Es muss also die Anzahl der Stützstellen iterativ so lange erhöht werden, bis Bedingung (3.19) erfüllt ist. Dabei ist es zweckmäßig die Anzahl der Intervalle in jedem Schritt zu verdoppeln, also jeweils zwischen zwei Stützstellen einen weiteren einzufügen (siehe Abbildung 3.7). Denn betrachtet man die Definition der Trapezregel (3.7), so fällt auf, dass bei einer Verdoppelung

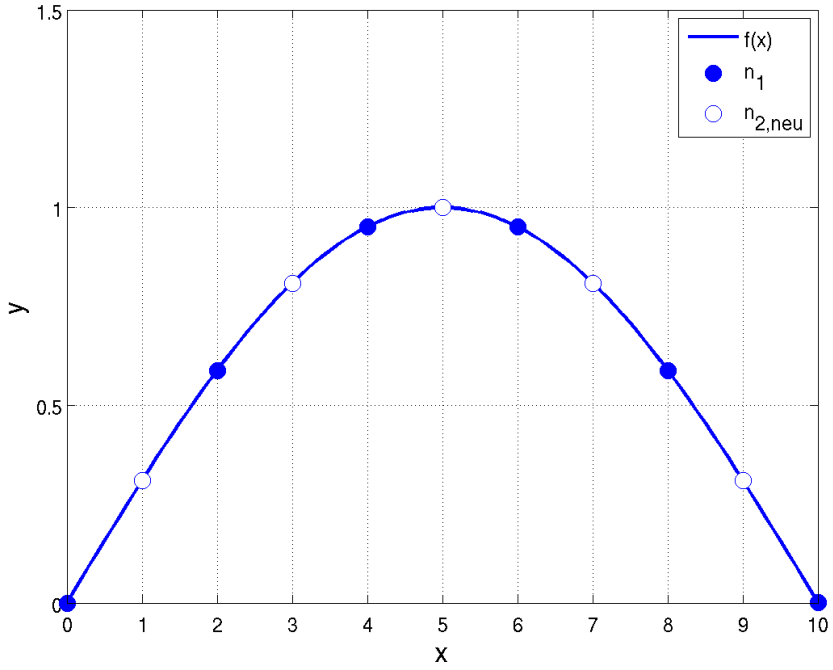


Abbildung 3.7: In blau sind die Stützstellen der Diskretisierung n_1 und in weiß die zusätzlichen Stützstellen der Diskretisierung n_2 aufgetragen.

der Intervalle - und damit einer Erhöhung von n_1 auf n_2 - bereits einige Summanden bekannt sind und somit Funktionsauswertungen entfallen. In Abbildung 3.7 sind die Stützstellen der Diskretisierung n_1 und in weiß die zusätzlichen Stützstellen der Diskretisierung n_2 aufgetragen. Die Summanden an den Stützstellen für n_1 sind bereits bekannt und es müssen daher lediglich die weiß eingezeichneten Punkte ausgewertet werden.

Eine mögliche Strategie zur **Implementierung der Trapezregel** besteht also darin, mit einem Intervall bzw. zwei Stützstellen zu beginnen und iterativ die Anzahl der Punkte so lange zu verdoppeln, bis die Bedingung (3.19) erfüllt ist. Dabei sollten für jeden Schritt nur die neu hinzukommenden Summanden bestimmt werden (also die weißen Punkte in Abb. 3.7). Es ist zu beachten, dass dennoch auch die Summanden des alten Schrittes angepasst werden müssen, da sich die Intervallgröße h ändert! Ein weiterer Funktionsaufruf ist jedoch nicht erforderlich. Außerdem sollte eine Anzahl von Schritten definiert werden, die mindestens zu durchlaufen sind, bevor ein Abbruch aufgrund von Bedingung (3.19) möglich ist. So wird ein fehlerhafter Abbruch bei Funktionen, für die der Algorithmus in den ersten Schritten zu langsam konvergiert, verhindert. Ein Ablaufdiagramm hierfür ist in Abbildung 3.8 dargestellt.

Zur **Implementierung der Simpsonregel** sollte Gleichung (3.12) betrachtet werden. Diese sagt aus, dass durch geschicktes Anwenden der Trapezregel das Ergebnis der Regel nach Simpson erzielt werden kann, welches Polynome bis zur vierten Ordnung exakt integriert. Es ist also lediglich notwendig den Algorithmus zur Integration mittels der Trapezregel dahingehend zu modifizieren, dass zu jedem Schritt das Ergebnis der kleineren und der aktuellen Schrittweite zur Verfügung steht. Dann kann damit komfortabel nach Gleichung (3.12) das gesuchte Inte-

gral bestimmt werden. Dieser Algorithmus ist in Abbildung 3.9 verdeutlicht. Alle Änderungen gegenüber der Trapezregel aus Abbildung 3.8 sind in gelb markiert. Es wird deutlich, dass die Unterschiede beider Algorithmen relativ gering sind.

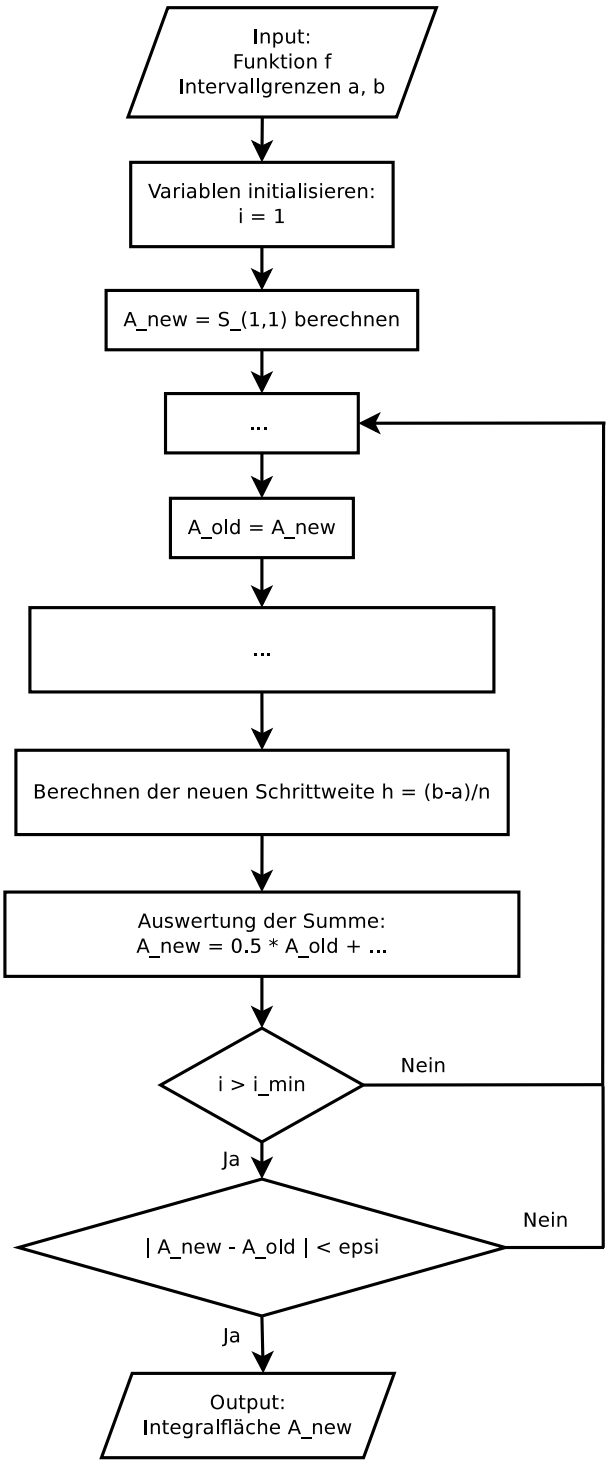


Abbildung 3.8: Ablaufdiagramm des adaptiven Integrationsverfahrens mittels der Trapezregel (mit Lücken).

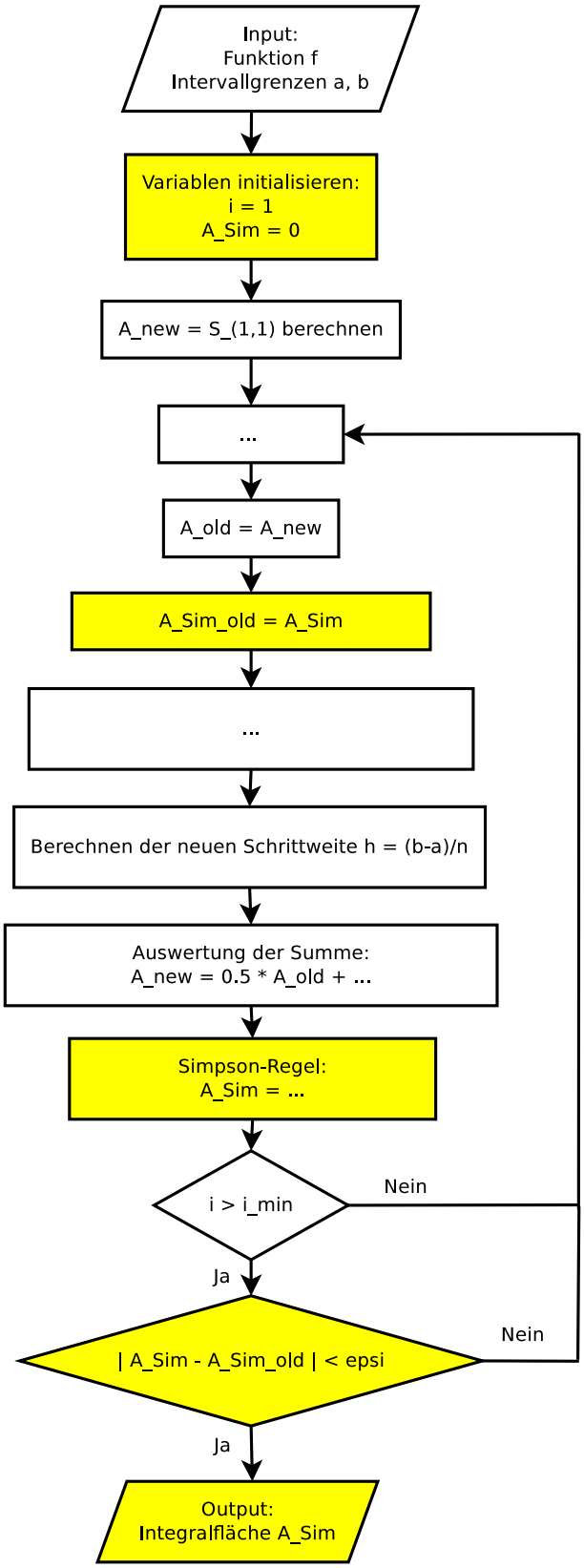


Abbildung 3.9: Ablaufdiagramm des adaptiven Integrationsverfahrens mittels der Simpsonregel (mit Lücken). Änderungen zur Trapezregel sind in gelb markiert.

Diskussion des Beispiels aus der Motivation

Wie zu Beginn dieses Kapitels dargestellt, kann die Oberfläche der zu untersuchenden Flamme mittels

$$A(t) = 2\pi \int_0^H R(z,t)\sqrt{1 + \left(\frac{dR(z,t)}{dz}\right)^2}\,dz \tag{3.20}$$

berechnet werden. Zur Lösung dieses Integrals stehen aus einer numerischen Rechnung die $n+1$ Radien $R_i(t)$ an den $n+1$ Orten z_i zur Verfügung. Der Abstand h zwischen den einzelnen z_i sei äquidistant. Die zu integrierende Funktion kann damit also als

$$f(t,z) = R(z,t)\sqrt{1 + \left(\frac{dR(z,t)}{dz}\right)^2} \tag{3.21}$$

geschrieben werden. Für jeden Wert z_i kann ein Funktionswert berechnet werden:

$$f_i(t) = R_i(t)\sqrt{1 + \left(\frac{dR(z,t)}{dz}\right)_i^2}. \tag{3.22}$$

Die $R_i(t)$ sind direkt aus der numerischen Rechnung verfügbar, während der Gradient z.B. mittels eines zentralen Differenzen-Verfahrens bestimmt werden muss. An den Rändern für $i = 0$ oder $i = n$ muss auf eine Vorwärts- bzw. eine Rückwärtsdifferenz ausgewichen werden, da hier die für eine zentrale Differenz notwendigen Werte R_{-1} bzw. R_{n+1} nicht existieren. Damit ergibt sich also

$$f_i(t) = R_i(t)\sqrt{1 + \left(\frac{R_{i+1}(t) - R_{i-1}(t)}{2h}\right)^2}. \tag{3.23}$$

Damit können nun direkt die oben hergeleiteten Formeln für die Trapez- bzw. Simpsonregel angewandt werden. Als Beispiel wird hier für ein **gerades n** die Simpsonregel gezeigt:

$$A(t) = \int_0^H f(t,z)dz = \frac{h}{3}\left(f_0(t) + 4\sum_{i=1}^{n/2} f_{2i-1}(t) + 2\sum_{i=1}^{n/2-1} f_{2i}(t) + f_n(t)\right) \tag{3.24}$$

Sollte **n** eine **ungerade** Zahl sein, so muss z.B. am unteren Rand ein Mal die Trapezregel verwendet werden:

$$A(t) = \int_0^H f(t,z)dz = \underbrace{\frac{h}{3}\left(f_1(t) + 4\sum_{i=1}^{(n-1)/2} f_{2i}(t) + 2\sum_{i=1}^{(n-1)/2-1} f_{2i+1}(t) + f_n(t)\right)}_{\text{Simpsonregel}} \tag{3.25}$$

$$+ \underbrace{h\left(\frac{f_0(t) + f_1(t)}{2}\right)}_{\text{Trapezregel}}. \tag{3.26}$$

Alternativ kann die Trapezregel auch am oberen Rand angewandt werden:

$$A(t) = \int_0^H f(t,z)dz = \underbrace{\frac{h}{3}\left(f_0(t) + 4\sum_{i=1}^{(n-1)/2} f_{2i-1}(t) + 2\sum_{i=1}^{(n-1)/2-1} f_{2i}(t) + f_{n-1}(t)\right)}_{\text{Simpsonregel}} \tag{3.27}$$

$$+ \underbrace{h\left(\frac{f_{n-1}(t) + f_n(t)}{2}\right)}_{\text{Trapezregel}}. \tag{3.28}$$

Ein Beispiel für frei wählbare Stützstellen x_i kann in der dieses Kapitel abschließenden Übungsaufgabe gefunden werden.

Abschließend wird ein kurzer Ausblick auf weitere numerische Integrationsverfahren gegeben. Wie bei der numerischen Differentiation kann auch für die Integration mithilfe eines Least-Square-Fits (Kapitel 4.1) integriert werden. Dies ist sinnvoll bei verrauschten Messdaten. Alternativ können für Messdaten mit geringem Rauschen kubische Splines eingesetzt werden.

Ein weiteres bekanntes Verfahren, welches einen komplett anderen Ansatz verwendet, besteht in der Monte-Carlo-Integration. Hier wird die gegebene Funktion an zufällig ausgewählten Punkten ausgewertet und anhand der Ergebnisse das Integral abgeschätzt. Dieses Verfahren eignet sich vor allem für höherdimensionale Integrale, da die Qualität der Lösung unabhängig von der Dimension des Problems mit $1/\sqrt{n}$ skaliert. Hierbei ist n die Anzahl der ausgewerteten Punkte [1].

Ein bestimmtes Integral $\int_a^b f(x)dx$ lässt sich durch Differentiation immer in eine Differentialgleichung umschreiben. Hierzu gelte

$$I(x) = \int_a^x f(\tilde{x})d\tilde{x}. \tag{3.29}$$

Damit lässt sich das Integral äquivalent zu folgenden zwei Gleichungen umformen:

$$\frac{dI(x)}{dx} = f(x), \tag{3.30}$$

$$I(a) = 0. \tag{3.31}$$

Diese können nun mithilfe der in Kapitel 6 und 7 vorgestellten Methoden zum Lösen von gewöhnlichen Differentialgleichungen integriert werden.

3.3 Übungsaufgaben

Diese Übung befasst sich mit der numerischen Integration. Anwendungsbeispiel ist die Bestimmung der mittleren Wärmekapazität von Wasser für zwei gegebene Temperaturen. Der Verlauf der Wärmekapazität ist mittels sogenannter NASA-Polynome gegeben, wobei es sich im hier behandelten Fall um Polynome vierten Grades handelt. Die korrekte Bestimmung der Koeffizienten wird Gegenstand des 4. Termins sein.

3.1 Zunächst sollen die in diesem Kapitel beschriebenen Algorithmen zur Integration einer beliebigen Funktion umgesetzt und getestet werden. Dabei soll für jedes der Verfahren eine eigene MATLAB-Funktion erstellt werden.

3.1.1 Vervollständigen Sie die Ablaufdiagramme aus den Abbildungen 3.8 und 3.9 bevor Sie mit der Programmierung beginnen.

3.1.2 Erstellen Sie jeweils eine MATLAB-Funktion zur numerischen Integration mittels der Trapezregel (`Int_Trapez()`), der Simpson-Regel (`Int_Simpson()`) sowie eine zur Integration mit Hilfe des Gauß-Verfahrens (`Int_Gauss()`). Implementieren Sie im Falle der Trapez- und der Simpsonregel ein adaptives Verfahren. Für die minimale Anzahl an Iterationen soll gelten $i_{min} = 5$ und für die maximale absolute noch zulässige Änderung zwischen zwei Iterationschritten gelte $\epsilon = 10^{-6}$.

- `[A,n] = Int_fun(f_x,x_1,x_2)`
  - `A`: aus der Integration resultierende Fläche
  - `n`: Anzahl der Iterationen
  - `f_x`: Funktions-Handle
  - `x_1,x_2`: Integrationsgrenzen

Hinweise:

- Beachten Sie, dass durch die Verdopplung der Anzahl der Stützstellen mit jeder Iteration sehr schnell eine sehr große Anzahl entsteht. Nach zehn Iterationen sind es bereits $2^{10} = 1024$. Aus diesem Grund sollten Sie die Anzahl der maximal durchzuführenden Iterationen auf 20 begrenzen!
- Um den Algorithmus besser zu verstehen, sollten Sie sich für die ersten 2-3 Iterationsschritte eine Skizze der auszuwertenden Stützstellen machen.
- Bei der Implementierung des Gauß-Verfahrens wird die Anzahl der Stützstellen fix vorgegeben (z.B. $n = 3$).

3.2 Berechnen Sie die mittleren Wärmekapazitäten von Wasser $c_p|_{T_0}^{T}$ für eine Bezugstemperatur von $T_0 = 0°C$. Der Temperaturbereich der berechneten Werte sollte hierbei von $T = 100°C$ bis $T = 700°C$ in $50°C$-Schritten gehen. Verwenden Sie zur Evaluation der Wärmekapazität NASA-Polynome vierter Ordnung. Der Druck betrage $p = 1\ bar$.

Hinweise:

- Die mittlere Wärmekapazität berechnet sich zu (siehe Thermodynamik Skriptum "Technische Thermodynamik", Vers. 5.5.12, S. 126 ff.)

$$\bar{c}_p|_{T_0}^{T} = \frac{\int_{T_0}^{T} c_p(T)dT}{T - T_0}. \tag{3.1}$$

- NASA-Polynome vierter Ordnung für die Wärmekapazität sind definiert als

$$\frac{c_p(T)}{R_s} = a_1 + a_2 T + a_3 T^2 + a_4 T^3 + a_5 T^4, \tag{3.2}$$

  wobei R_s die spezifische Gaskonstante des Stoffes ist. Die fünf Koeffizienten a_i sind in der MATLAB-Vorlage für diese Übung gegeben. Die Temperatur ist in Kelvin einzusetzen!

- Zusatzübung: Plotten Sie den durch obiges NASA-Polynom gegebenen Verlauf von c_p über den Temperaturbereich von 200 °C bis 1000 °C. Fügen Sie nun in das selbe Diagramm einen mit Hilfe von XSteam bestimmten Verlauf für c_p ein. Vergleichen Sie beide Kurven! Was für eine Aussage können Sie bzgl. der Gültigkeit der NASA-Polynome treffen?

3.4 Wichtige Matlab Befehle

Befehl	Beschreibung
`abs()`	Bestimmt elementweise den Betrag (komplexer) Zahlen.
`arrayfun()`	Wendet eine (anonyme) Funktion auf jedes Element eines Vektors (bzw. Arrays) an und gibt die Resultate als Vektor aus.
`break`	Beendet for- oder while-Schleife.
`f = @(x) x^2 + 4`	Erstellt eine sogenannte anonyme Funktion, die z.B. via `f(3)` aufgerufen werden kann. Anonyme Funktionen könne auch ineinander verschachtelt werden. Achtung: Damit auch Vektoren als Input verwendet werden könne, müsste in diesem Fall `f = @(x) x.^2 + 4` geschrieben werden!
`integral()`	MATLAB-interne Funktion zur numerischen Integration einer Funktion
`sum()`	Gibt die Summe aller Elemente eines Vektors aus.
`tic`	Startet Zeitmessung
`time_elapsed = toc`	Beendet Zeitmessung und gibt die seit dem `tic`-Befehl vergangene Zeit aus.
`trapz()`	MATLAB-interne Funktion zur numerischen Integration einer Datenreihe mittels der Trapezregel.

4

Methode der kleinsten Fehlerquadrate und Gauß-Algorithmus

Literaturverzeichnis

[1] Brian P Flannery, William H Press, Saul A Teukolsky und William Vetterling. *Numerical Recipes in C*. Kap. 15. 1992. URL: http://www.nr.com/.

[2] R. Isermann und M. Münchhof. *Identification of Dynamic Systems: An Introduction with Applications*. Kap. 8. Springer, 2010. ISBN: 9783540788799.

Lernziele

- Methode der kleinsten Fehlerquadrate
- Gauß-Algorithmus

Motivation

Ein Ingenieur wird oft vor die Aufgabe gestellt ein Modell zur Beschreibung eines Systems zu entwickeln. Ein Beispiel hierfür wäre ein Schalldämpfer: Auf der einen Seite laufen akustische Wellen einer bestimmter Amplitude hinein und treten mit niedriger Amplitude und evtl. anderer Frequenz auf der anderen Seite wieder aus. Ziel ist es den Zusammenhang der ein- und auslaufenden Wellen zu finden. Als Mittel stehen dem Ingenieur Messdaten und sein Wissen über die Physik zur Verfügung.

Im ersten Schritt wird er versuchen ein mathematisches Modell zu entwickeln. Hierfür kann er entweder ein physikalisches Modell herleiten oder er verwendet eine allgemeine, auf das Problem zugeschnittene Modellstruktur. Ein Beispiel für letzteres könnte sein, dass er aus Erfahrung weiß, dass die Amplitude der einlaufenden Well direkt proportional zur auslaufenden ist: $A_{aus} = \mathrm{K}\, A_{ein}$.

Das entwickelte Modell besitzt im Allgemeinen Modellkonstanten, wie im hier genannten Beispiel die Proporionalitätskonstante K. Diese hängen vom System ab, lassen sich aber nicht rein aus pysikalischen Betrachtungen bestimmen oder dies wäre zu aufwändig. In diesem Fall wird der Ingenieur Messdaten erzeugen indem er das System mit verschiedenen Eingangssignalen beaufschlägt und die Ausgangssignale aufzeichnet. Im Beispiel des Dämpfers wird er also den Dämpfer für verschiedene Frequenzen und Amplituden testen und die Ausgangswellen vermessen. Aus diesen Daten kann er dann Rückschlüsse auf die gesuchte Konstante K ziehen.

Messdaten sind jedoch immer fehlerbehaftet (Messfehler). Das fertige Modell kann (und sollte) somit die Messdaten nicht perfekt sondern nur möglichst gut beschreiben. Diese "möglichst gute" Anpassung der Modellkonstanten an gegebene Datenpunkte wird auch als ein *Fit* bezeichnet. Eine Methode hierfür ist die Methode der kleinsten Fehlerquadrate.

Dieses Vorgehen soll nun anhand eines weiteren Beispiels illustriert werden. Diesmal wird die Modellfunktion aus physikalischen Überlegungen gewonnen. Folgendes Modell beschreibt wie in einem idealen Gas, bei konstanter Dichte, der Druck von der Temperatur abhängt:

$$\hat{p}(T, R_s) = R_s \rho T. \tag{4.1}$$

Nehmen wir nun an, wir wüssten nicht mit was für einen Gas wir es zu tun haben. Die spezifische Gaskonstante $R_s = \frac{R}{M_x}$ bzw. die molare Masse M_x des Gases ist uns also unbekannt. Um das Gas zu bestimmen, führen wir n Messungen des Drucks bei n Temperaturen durch. Nun möchten wir die Gaskonstante R_s so bestimmen, dass das durch Gleichung (4.1) definierte Modell einen Druckverlauf liefert, der möglichst optimal die Physik unseres Gases repräsentiert. Das reale Verhalten des Gases ist uns aber nur mittels der Messdaten $p_m(T_i)$ zugänglich, welche im Allgemeinen einen Messfehler e_m aufweisen. Der Messfehler sei dabei nicht systematisch und jeweils zu allen anderen Messfehlern statistisch unabhängig. Es gilt daher für jeden Messpunkt

$$p_m(T_i) = p(T_i) + e_{m,i}. \tag{4.2}$$

Das beschriebene Problem besteht also aus n Messwerten $p_m(T_i)$ bei n verschiedenen Temperaturen, die jeweils aus einem theoretisch wahren Wert (den wir aber nicht kennen) $p(T_i)$ plus

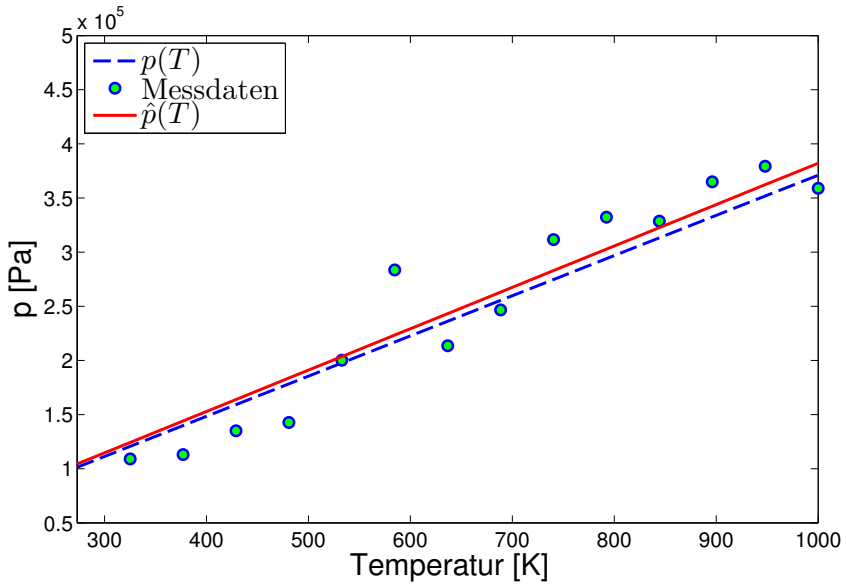


Abbildung 4.1: Messdaten des Druckes für verschiedene Temperaturen (Punktewolke) samt Ausgleichsgeraden $\hat{p}(T)$ und wahren Verlauf des Drucks $p(T)$.

einem Fehler $e_{m,i}$ bestehen. Zusätzlich ist ein parametrisches Modell für den Verlauf des theoretisch wahren Wertes des Druckes verfügbar, nämlich $\hat{p}(T, R_s)$ aus Gleichung (4.1). Dessen Parameter R_s gilt es anhand der Messdaten derart zu bestimmen, dass der modellierte und der wahre Verlauf des Druckes möglichst identisch sind (Anm.: "wahr" bedeutet in diesem Kapitel immer ohne Messfehler). Es sollte also eine Beziehung wie diese gelten:

$$K(R_s) = \sum_{i=1}^{n} (\hat{p}(T_i, R_s) - p_m(T_i))^2 \rightarrow \text{minimal.} \tag{4.3}$$

Hier wird ein Minimum der Summe aller lokalen quadratischen Fehler gefordert. Es sei darauf hingewiesen, dass dies lediglich eine unter mehreren Möglichkeiten ist die sogenannte **Kostenfunktion** K aufzustellen. Allerdings hat sich diese in vielen Fällen bewährt und ist dementsprechend weit verbreitet.

Prinzipiell sind auch andere Formulierungen für K denkbar, z.B. eine Summe der Beträg der lokalen Fehler. Die quadratische Abweichung bietet jedoch einige Vorteile, z.B. ihre relativ einfache mathematische Handhabbarkeit. Ein weiterer, gewichtigerer, liegt in der verbreiteten Annahme, der Messfehler e_m folge einer Gaußschen Normalverteilung. Damit führt die Verwendung von Gleichung (4.3) auf einen sogenannten Maximum Likelihood-Schätzer. Nähere Informationen hierzu sind im Exkurs weiter unten zusammengefasst.

Intuitiver beschrieben haben wir also, wie in Abbildung 4.1 gezeigt, eine Punktewolke aus Messdaten p_m über T_i gegeben, die „irgendwie" aussieht als würde sie z.B. einer linearen Funktion a_1T+a_2 folgen (in diesem Fall sagt uns dies auch das ideale Gasgesetz). Wir versuchen nun die Parameter a_1 und a_2 dieser Funktion so zu wählen, dass der resultierende Verlauf möglichst gut die Punktewolke beschreibt. Wir konstruieren also in diesem Fall eine Ausgleichsgerade.

Eine Verfahren hierfür bietet die Methode der kleinsten (Fehler-) Quadrate. Das Ergebnis dieser Methode ist ebenfalls in Abbildung 4.1 geplotted ($\hat{p}(T)$).

Diese Methode führt, wie im Folgenden gezeigt, auf eine lineares Gleichungssystem. Die Lösung derartiger Systeme ist für die Numerik fundamental wichtig, da eine große Zahl an numerischen Problemstellungen in einem linearen Gleichungssystem mündet. In diesen Kapitel wird mit dem Gauß-Algorithmus ein klassisches Verfahren zur direkten Lösung vorgestellt.

4.1 Methode der kleinsten Fehlerquadrate

Die Ausgangsgleichung für diese Methode ist Gleichung (4.3), welche in einer allgemeineren Formulierungen

$$K(a_1, a_2, \ldots, a_m) = \sum_{i=1}^{n} \left(\hat{f}(x_i, a_1, a_2, \ldots, a_m) - f_m(x_i) \right)^2 \rightarrow \text{minimal}, \tag{4.4}$$

lautet, worin $\hat{f}(x_i, a_1, a_2, \ldots, a_m)$ die Modellfunktion ausgewertet an den n Stützpunkten darstellt. Diese Funktion sei linear in den m entsprechend der Minimalbedingung zu wählenden Parametern $a_1, a_2, \ldots, a_m$. Die Werte $f_m(x_i)$ bezeichnen die jeweiligen Messwerte an den gegebenen Datenpunkten x_i. Hierbei wird unterstellt

$$f_m(x_i) = f(x_i) + e_{m,i}. \tag{4.5}$$

Der Messfehler $e_{m,i}$ folge zudem einer Gaußschen Normalverteilung und alle Fehler seien untereinander statistisch unabhängig. Dies bedeutet, dass ein Messfehler der Messung i keinen anderen Messwert beeinflusst.

Neben diesen Messfehlern kann im Allgemeinen noch ein Modellfehler $f(x_i) - \hat{f}(x_i)$ auftreten. Nehmen wir z.B. ein lineares Modell an, so werden wir eine Ausgleichsgerade bekommen, die zwar den quadratischen Fehler minimiert, allerdings immer einen inhärenten Modellfehler besitzen wird, falls die gesuchte Funktion nicht linear ist. Ein Beispiel hierzu ist weiter unten gegeben.

Exkurs:
Warum wird ausgerechnet der quadratische Fehler minimiert? Dieses Thema wurde oben bereits kurz angerissen. Ausgangspunkt ist die Frage, für welchen Satz an Modellparametern die Wahrscheinlichkeit am größten ist, dass der Verlauf der Modellfunktion $\hat{f}(x_i)$ gleich dem Verlauf der gesuchten Funktion $f(x_i)$ ist. Hierbei kommt nun die Verteilung des Messfehlers ins Spiel: Es wird angenommen dieser folge einer Gaußschen Normalverteilung. Damit ergibt sich als Wahrscheinlichkeitsdichte für das Auftreten eines Messfehlers $e_{m,i} = \hat{f}(x_i, a_1, \ldots, a_m) - f(x_i)$ in der i-ten Messung

$$p(\hat{f}(x_i, a_1, \ldots, a_m)) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{\hat{f}(x_i, a_1, \ldots, a_m) - f(x_i)}{\sigma} \right)^2 \right).$$

σ bezeichnet die Standardabweichung der Verteilung. Da alle Einzelmessungen stochastisch unabhängig sind, ist die Wahrscheinlichkeitdichte der gesamten Messreihe gerade das Produkt aus allen Einzelwahrscheinlichkeitdichten. Diese Wahrscheinlichkeitdichte soll für die gesuchten Parameter $a_1, \ldots, a_m$ maximal sein, denn dann ist die Wahrscheinlichkeit am größten, dass ein Modell mit diesen Parametern die gesuchte Funktion $f(x)$ beschreibt. Bestimmt man die daraus folgende, zu maximierende Gleichung und löst diese Extremwertaufgabe, so erhält man das gleiche Ergebnis wie bei der Minimierung des quadratischen Fehlers. Für einen normalverteilten Fehler liefert also die Methode des kleinsten (Fehler-) Quadrats jene Lösung, welche mit der größten Wahrscheinlichkeit mit der gesuchten Funktion übereinstimmt. Wichtig ist hierbei, dass nichts über die gewählte Modellfunktion ausgesagt wird. Die Methode passt lediglich die Parameter des ihr vorgegebenen Modells - also z.B. einer linearen Funktion - in optimaler Weise an die ihr gegebenen Datenpunkte an. Weitere Informationen finden sich z.B. in [2]. Um die Güte der Modellfunktionen zu überprüfen bzw. zu vergleichen kann z.B. der Chi-Quadrat-Test verwendet werden [1]. Weitere Aussagen lassen sich durch Analyse der Residuen $\hat{f}(x_i) - f(x_i)$ gewinnen.

Wie bereits erwähnt, beschränken wir uns auf Probleme, die bzgl. der Modellparameter linear sind. Damit lässt sich vorliegendes Problem in Matrix-Vektor-Schreibweise wie folgt darstellen:

$$Aa = b. \tag{4.6}$$

Hierbei enthält der Vektor b die Messdaten, A bestimmt die lineare Abbildung und a ist der Lösungsvektor mit den m Modellparametern. Im Beispiel einer linearen Modellfunktion $\hat{f}(x_i) = a_1 x_i + a_2$ ergibt sich also folgendes System:

$$\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_n & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} f_m(x_1) \\ f_m(x_2) \\ \vdots \\ f_m(x_n) \end{pmatrix} \tag{4.7}$$

Dies ist ein überbestimmtes Gleichungssystem mit $n > m = 2$ Gleichungen (eine je Messung) für die $m = 2$ unbekannten Modellparameter. Für dieses System existiert somit i.A. keine Lösung. Es kann, wie oben beschrieben, also lediglich versucht werden eine Lösung zu finden, die den quadratischen Fehler minimiert:

$$K = ||Aa - b||_2^2 \rightarrow \text{minimal}. \tag{4.8}$$

Es soll nun eine Gleichung für das gesuchte Minimum hergeleitet werden. Hierfür muss gelten

$$\frac{\partial K}{\partial a_i} = 0, \qquad i = 1, 2, \ldots, n. \tag{4.9}$$

Da K eine positiv definite, quadratische Funktion ist, ist diese Forderung notwendig und hinreichend für ein Minimum. Als Ergebnis aus dieser Forderung ergibt sich schließlich die gesuchte Extremalbedingung (genau Herleitung z.B. in [2]):

$$A^T A a = A^T b, \tag{4.10}$$

$$\tilde{A} a = \tilde{b}. \tag{4.11}$$

Dies ist ein $n \times n$-Gleichungssystem und damit lösbar. Unter Verwendung der Inversen von $A^T A$ lässt sich die Lösung formal darstellen als:

Definition: Lösung kleinste Fehler-Quadrat-Methode

$$a = (A^T A)^{-1} A^T b.$$

Die Inverse ist im Allgemeinen allerdings nicht bekannt. Aus diesem Grund muss das Gleichungssystem (4.10) z.B. mit Hilfe des Gauß-Algorithmus, der im folgenden Abschnitt 4.2 beschrieben wird, gelöst werden. Hierfür kann dieses System in $\tilde{A} a = \tilde{b}$ umgeschrieben werden, wobei $\tilde{A} = A^T A$ und $\tilde{b} = A^T b$ gilt.

Es sei an dieser Stelle aber darauf hingewiesen, dass in der Numerik ein Ausdruck $A^T A$ immer mit Vorsicht zu betrachten ist. Wie bereits erwähnt, arbeiten Computer mit endlicher Genauigkeit und demnach enthalten im PC dargestellte Zahlen meist einen gewissen Fehler (Rundungsfehler). Eine numerische Operation kann einen Fehler, der in den Eingangswerten u.U. enthalten ist, weiter verstärken. Man bezeichnet die Abhängigkeit der Lösung von den Abweichungen in den Eingangswerten als **Kondition** eines Problems. Diese ist formal für lineare Abbildungen definiert als:

Definition: Kondition einer linearen Abbildung

$$\kappa(A) = \max_x \frac{||Ax_0 - Ax||}{||x_0 - x||} = \max_x \frac{||A\delta||}{||\delta||} = \left| \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \right|.$$

Die Beziehung, welche den minimalen und den maximalen Eigenwerte $\lambda_{min/max}$ enthält, gilt nur für normale Matrizen ($A^T A = A A^T$). Die Schwierigkeit bei der Bestimmung der Kondition besteht darin, jenes x zu finden, für das obige Beziehung maximal wird. In MATLAB existiert die Funktion `cond()`, welche die Kondition einer Matrix abschätzen kann.

Wir wollen uns den Begriff der Kondition mit Hilfe eines einfachen Beispiels veranschaulichen. Wir definieren uns hierfür zwei Geraden

$$g\text{:} \quad x = k_1 \begin{pmatrix} a \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{4.12}$$

$$h\text{:} \quad x = k_2 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{4.13}$$

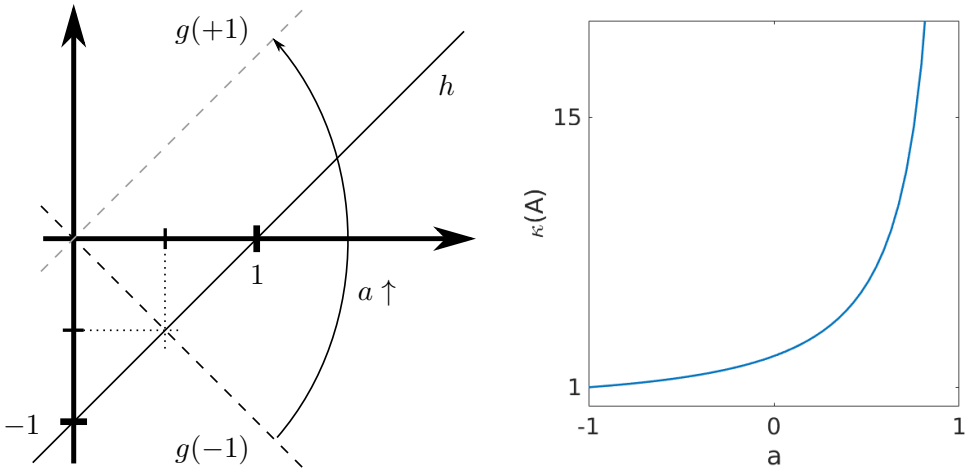


Abbildung 4.2: *Links:* Die Lösung eines Gleichungssystems $Ax = b$ definiert den Schnittpunkt der beiden Geraden $g(a)$ und h, wobei die Steigung der Geraden g vom Parameter $a \in [-1, 1]$ abhängt. *Rechts:* Die Abhängigkeit der Kondition vom Parameter a ist im rechten Bild dargestellt.

wobei erstere von einem Parameter a abhängen soll. Je nachdem wie dieser gewählt wird ändert sich deren Steigung. Für $a = -1$ und $a = +1$ sind die beiden Geraden in Abbildung 4.2 (links) abgebildet. Nun interessiert uns der Schnittpunkt der beiden Geraden. Dessen Berechnung führt uns schließlich auf das Gleichungssystem

$$\begin{pmatrix} a & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{4.14}$$

Bestimmt man nun die Kondition dieser Abbildung in Abhängigkeit des Parameters a, so ergibt sich der in Abbildung 4.2 (rechts) gezeigte Verlauf. Sind die Geraden orthogonal zueinander, so beträgt die Kondition des Systems von Gleichung (4.14) genau eins. Je paralleler die beiden Geraden werden, desto schlechter konditioniert wird das Problem, bis schließlich für $a = 1$ eine Polstelle erreicht ist und das System nicht mehr lösbar ist. Man kann dies direkt nachempfinden, indem man versucht zwei fast parallele Geraden zu zeichnen, die sich in einem definierten Punkt schneiden sollen. Je paralleler beide Geraden sind, desto schwieriger wird diese Aufgabe. Wird beim Zeichnen des obigen Beispiels entweder die Steigung der Geraden (enspricht einer Ungenauigkeit in der Matrix A) oder der Aufpunkt (entspricht einer Ungenauigkeit in der rechten Seite b) nicht exakt getroffen, so ergibt sich für fast parallele Geraden ein großer Fehler im resultierenden Schnittpunkt. Es wird ein immer feinerer Stift benötigt, um eine bestimmte Genauigkeit zu erzielen. Dies bedeutet, je paralleler die Geraden sind und je schlechter damit die Kondition des Problems wird, desto genauer muss die Matrix A und der Vektor b bestimmt werden, um den Schnittpunkt mit einer definierten Genauigkeit bestimmen zu können.

Zuletzt soll auf eine wichtige Problematik mit Ausdrücken der Form $A^T A$ hingwiesen werden. Die Kondition der entstehenden Matrix ist das Quadrat der Kondition der Matrix A: $\kappa(A^T A) =$

$\kappa(\tilde{A}) = \kappa(A)^2$. Damit ist das lineare Gleichungssystem $\tilde{A}a = b$ schlechter konditioniert als das System $Aa = b$. Falls möglich sollten aus diesem Grund derartige Ausdrücke in der Numerik vermieden werden.

Anmerkung:
Die Modellparameter nach der Methode der kleinsten Fehlerquadrate bestimmen sich aus dem Gleichungssystem (4.10). Dieses ist speziell für Polynome höheren Grades schlecht konditioniert. Damit können kleine Änderungen in $\tilde{A}$ oder $\tilde{b}$ eine große Änderung in den Modellparametern verursachen.

Ein weiteres Verfahren zur Lösung des in Gleichung (4.6) beschriebenen Systems, welches in der Literatur oft zu finden ist, besteht in der Berechnung der sogenannten Moore-Penrose-(Pseudo-)Inversen. Hierbei wird direkt der Lösungsvektor des Gleichungssystems durch Bildung der Pseudo-Inversen A^+ gebildet:

$$a = A^+ b. \tag{4.15}$$

Der Fehler des Lösungsvektors a ist dabei wieder im quadratischen Sinne minimal. Damit kann die Methode des kleinsten (Fehler-)Quadrats auch mittels Gleichung (4.15) durchgeführt werden. MATLAB stellt die Funktion `pinv()` zur Bestimmung der Pseudo-Inversen bereit. Unter Verwendung der Pseudo-Inversen kann darüber hinaus die Kondition einer beliebigen Matrix definiert werden:

$$\kappa(A) = ||A|| \cdot ||A^+||. \tag{4.16}$$

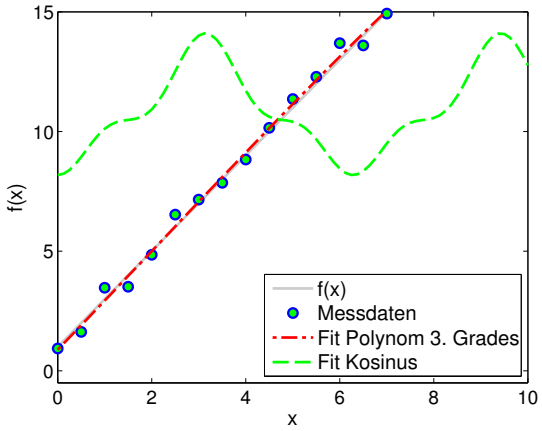
In der Praxis sollte in MATLAB der \-Operator verwendet werden: `a = A \ b` . Handelt es sich bei A um eine quadratische Matrix gibt MATLAB direkt die Lösung des Systems aus. Ist A jedoch rechteckig und das System damit überbestimmt, wie z.B. jenes aus Gleichung (4.6), so gibt MATLAB automatisch jenen Lösungsvektor a zurück, welcher den quadratischen Fehler minimiert (siehe Gl. (4.8)). Eine explizite Bestimmung der Pseudo-Inversen oder des quadratischen Gleichungssystems $A^T A = A^T b$ ist somit überflüsig.

Beispiele

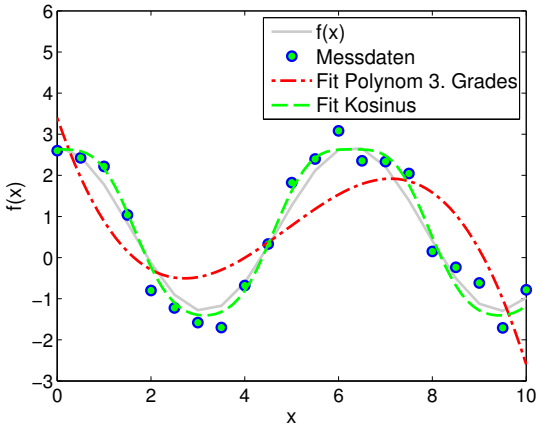
Zur Veranschaulichung der bisher dargelegten Theorie sollen nun vier Beispiele betrachtet werden. Für jedes Beispiel wurde ein wahrer Funktionsverlauf $f(x)$ angenommen und normal verteilt um diesen herum „Messpunkte" generiert (mit der Varianz σ^2). Somit sind genau die in Kapitel 4.1 getroffenen Annahmen bzgl. der Optimalheit eines Fits mittels der Methode der kleinsten Fehler-Quadrate gegeben. Anders als in der Praxis üblich, sind auf diese Weise der wahre Funktionsverlauf und die genaue Verteilung des Fehlers bekannt, wodurch eine qualitative Beurteilung der Verläufe der Fits ermöglicht wird.

Die ersten Datenpunkte in Abbildung 4.3(a) wurden mit ein linear Funktion f erzeugt. Zwei parametrische Modelle sollen an diese Punkte angepasst werden: Dem ersten liegt eine Kosinus-Funktion $\hat{f}(x) = a_1 + a_2 \cos x + a_3 \cos 2x + a_4 \cos 3x$ zugrunde. Das zweite besteht aus einem Polynom dritten Grades. Wendet man nun auf beide Modelle die Methode der kleinsten Fehler-Quadrate an, so ergeben sich die ebenfalls in Abbildung 4.3(a) dargestellten Verläufe. Es zeigt

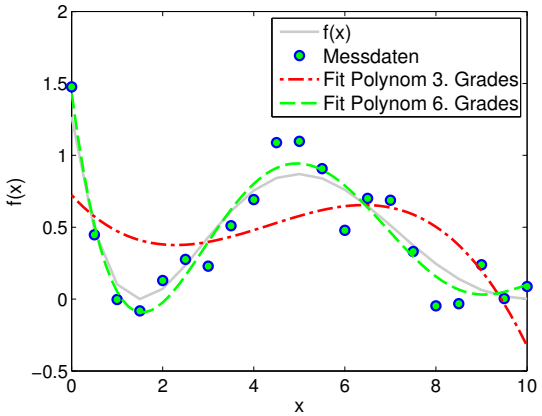
sich, dass das Polynom den wahren Verlauf nahezu perfekt annähert. Dies ist nicht verwunderlich, schließlich ist eine lineare Funktion ein Polynom vom Grad eins und kann damit als Polynom dritten Grades beschrieben werden, dessen Koeffizienten für die nichtlinearen Terme null sind. Die Kosinus-Funktion hingegen erweist sich als schlechte Modellannahme. In diesem Fall dominiert der Modellfehler den gesamten Fehler: Unabhängig von der Wahl der Modellparameter kann niemals der wahre Funktionsverlauf getroffen werden. Das Polynom dagegen konvergiert für eine größer werdende Anzahl an Messpunkten gegen $f(x)$.



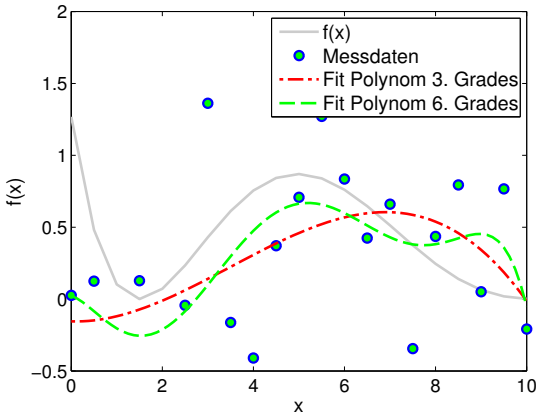
(a) Fit einer Punktewolke welche aus einem linearen Funktionsverlauf $f(x_i)$ und einer normalverteilten Abweichung $e_{m,i}$ generiert wurde.



(b) Fit einer Punktewolke welche aus einem Kosinus als Funktionsverlauf $f(x_i)$ und einer normalverteilten Abweichung $e_{m,i}$ generiert wurde.



(c) Fit einer Punktewolke welche aus einem Polynom 6. Grades als Funktionsverlauf $f(x_i)$ und einer normalverteilten Abweichung $e_{m,i}$ generiert wurde.



(d) Fit einer Punktewolke welche aus einem Polynom 6. Grades als Funktionsverlauf $f(x_i)$ und einer normalverteilten Abweichung $e_{m,i}$ mit großer Standardabweichung generiert wurde.

Abbildung 4.3: Vergleich verschiedener Modellfunktionen zum Least-Square-Fit anhand von vier Beispielen.

Nun wird zur Datengenerierung eine Kosinus-Funktion verwendet. Die Ergebnisse der darauf basierenden Fits sind in Abbildung 4.3(b) abgebildet. Hier zeigt sich, dass die oben genannte Kosinus-Funktion diesmal eine gute Modellannahme darstellt. Für eine zunehmende Anzahl an Messdaten würde diese gegen den realen Verlauf von $f(x)$ konvergieren. Die Modellfunkti-

on besitzt damit eine ausreichende Anzahl an Freiheitsgraden. Auch das Polynom erweist sich als brauchbares Modell, allerdings würde ein Polynom höheren Grades den wahren Verlauf noch besser treffen.

Im dritten Beispiel wurde nun für $f(x)$ ein Polynom sechsten Grades gewählt. Die daraus resultierenden Verläufe sind in Abbildung 4.3(c) dargestellt. Es wird klar, dass ein Polynom dritten Grades nicht oder nur unzureichend als Modellfunktion für ein Polynom sechsten Grades dienen kann. Es besitzt zu wenige Freiheitsgrade, um den wahren Verlauf im betrachteten Bereich treffen zu können. Lokal, also z.B. im Bereich zwischen $2 < x < 4$, kann es durchaus eine gute Näherung darstellen. Dort könnte es sich sogar als die bessere Näherung erweisen, wenn man die unten erwähnte Problematik der Überanpassung mit berücksichtigt.

In Abbildung 4.3(d) wird als letztes Beispiel wieder ein Polynom sechsten Grades verwendet, allerdings wurde hier die Standardabweichung der Messdaten sehr groß gewählt. Dies bedeutet, dass die Messdaten sehr weit um $f(x)$ streuen. Das Polynom sechsten Grades kommt dem wahren Verlauf relativ nahe, allerdings kann bei Unkenntnis von $f(x)$ anhand der Messdaten nur sehr vage auf irgendeinen spezifische Funktionsverlauf geschlossen werden. Jeder Messpunkt enthält ein gewisses Maß an Information zum wahren Funktionsverlauf. Dieser hängt von der Standardabweichung der Messdaten ab: Je größer diese ist, desto weniger Information steckt in jeder Messung. An diesem Beispiel zeigt sich, dass auch für ein perfektes Modell (Polynom vom Grad sechs) zu wenig Information in den Messdaten steckt um den waren Verlauf gut wiedergeben zu können. Um die Schätzung zu verbessern, kann die Anzahl an Messpunkten erhöht werden, da auf diese Weise auch der Informationsgehalt zunimmt. Allgemein gilt:

$$\text{Informationsgehalt in Messdaten} \propto \frac{\text{Anzahl an Messungen}}{\text{Standardabweichung einer Messung}} \tag{4.17}$$

Zuletzt soll auf das Problem der **Überanpassung** („Overfitting") eingegangen werden. Aufgrund der hier vorgestellten Beispiele könnte man z.B. denken es wäre optimal, immer ein Polynom mit sehr vielen Freiheitsgraden, z.B. eines vom Grade 15, als Ansatzfunktion zu verwenden. Es ist zwar auf diese Weise möglich einen Funktionsverlauf zu schätzen, der allen vorhandenen Messpunkten sehr nahe kommt, der allerdings auch aufgrund der Messfehler sehr stark oszilliert. Nehmen wir an wir hätten ein Messreihe bestehend aus 16 Punkten und legen durch diese ein Polynom der Ordnung 15. Alle Messdaten wären von diesem Modell perfekt erklärt. Jetzt soll nur ein einziger zusätzlicher Messwert betrachtet werden. Es zeigt sich, dass dieser nicht besonders gut von dem identifizierten Modell vorrausgesagt werden kann bzw. es ergäbe sich bei nochmaliger Bestimmung der 16 Modellparameter ein ganz anderer Funktionsverlauf als der originale. Derartige Modelle bezeichnet man als überangepasst und sind zu vermeiden.

Der Grund hierfür liegt an der Zusammensetzung der Messdaten aus einem deterministischen (wahrer Funktionsverlauf) und einem nicht-deterministischen (Messfehler) Anteil. Indem wir ein parametrisches Modell, wie oben beschrieben, wählen, entscheiden wir uns für ein deterministisches Modell. Die Methode der kleinsten Fehlerquadrate soll nun eingesetzt werden um den zufälligen Anteil in den Messdaten zu eliminieren. Je mehr Freiheitsgrade das gewählte Modell besitzt, desto wahrscheinlicher wird es, dass nicht-deterministische Anteile mit modelliert werden. Diese können jedoch mit dem gewählten Modell nicht abgebildet werden und

führen damit lediglich zu einer schlechteren Schätzung des wahren Funktionsverlaufes. Modelle mit so wenig Parametern wie möglich und nur so vielen wie nötig sind daher robuster und sagen i.A. mehr über die vorhandenen Daten bzw. deren funktionalen Zusammenhang aus.

Zusammengefasst muss für die Methode der kleinsten Fehler-Quadrate folgendes beachtet werden:

- Je mehr Messpunkte - deren Fehler statistisch unabhängig ist - betrachtet werden und je kleiner die Standardabweichung pro Messpunkt ist, desto näher kommt die gefittete Kurve dem wahren Funktionsverlauf (Informationsgehalt $\uparrow$).
- Folgt der Fehler einer Normalverteilung, so liefert die Methode der kleinsten Fehlerquadrate eine optimale Schätzung der Modellparameter.
- Ausreißer verändern den Verlauf eines Fits stark: In der implizit angenommenen Normalverteilung sind diese beliebig unwahrscheinlich und werden daher zu hoch gewichtet.
- Muss mit Ausreißern gerechnet werden, so ist zu überprüfen ob die implizite Annahme eines normalverteilten Fehlers gerechtfertigt ist. Falls nicht, muss eine alternative Kostenfunktion ermittelt werden (z.B. mit Hilfe der Maximum-Likelihood-Methode) oder es müssen durch z.B. Mittelung mehrerer Messwerte pro Messpunkt die Ausreißer beseitigt werden.
- Das verwendete Modell sollte möglichst dem wahren Funktionsverlauf entsprechen. Ist dieser nicht bekannt, so haben sich in der Praxis Polynome ausreichend hohen Grades bewährt.
- Der Grad des Polynoms sollte nicht zu hoch gewählt werden, um das Problem der Überanpassung zu vermeiden (=Modellierung des nicht deterministischen Messfehlers).

4.2 Gaußsches Eliminationsverfahren

Das Gaußsche Eliminationsverfahren ist ein sogenanntes direktes Verfahren zum Lösen linearer Gleichungssysteme. Ein solches System, bestehend aus m Gleichungen für die n Unbekannten x_j, läßt sich allgemein wie folgt schreiben:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\ \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{4.18}$$

Es ist genau dann eindeutig lösbar, falls es aus $m = n$ linear unabhängigen Gleichungen besteht. Definiert wird das System durch die Wahl der n^2 Koeffizienten a_{ij} sowie der rechten Seiten b_i. Zur einfacheren Handhabung wollen wir das Gleichungssystem (4.18) in Vektor- Matrixform angeben:

$$Ax = b. \tag{4.19}$$

A ist somit eine quadratische $n \times n$ Matrix, welche die Koeffizienten a_{ij} beinhaltet, und b ein $n \times 1$ Vektor mit der rechten Seite des Systems:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{4.20}$$

Das Gaußsche Eliminationsverfahren besteht aus zwei Schritten: Zunächst wird die Matrix A durch Umformungen des Systems, welche den Lösungsvektor x nicht verändern, auf sogenannte Stufenform gebracht (**Vorwärtselimination**). Diese Form ist dadurch charakterisiert, dass alle Koeffizienten unterhalb der Diagonalen null sind. Am Beispiel eines Gleichungssystems für vier Unbekannte sieht diese wie folgt aus:

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{pmatrix}. \tag{4.21}$$

Im zweiten Schritt kann dieses neue System, beginnend mit der letzten Zeile und nach oben fortschreitend, direkt gelöst werden (**Rücksubstitution**):

$$x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{n} a'_{ij} x_j \right), i < n. \tag{4.22}$$

Zum Erreichen der Stufenform werden lediglich drei elementaren Umformungen benötigt:

- Addition des Vielfachen einer Zeile zu einer anderen Zeile
- Multiplikation einer Zeil mit einem Skalar (ungleich null)
- Vertauschen zweier Zeilen

Zwei (eindeutig lösbare) Gleichungssysteme, welche nur durch beliebiges Anwenden dieser drei Transformationen ineinander überführt werden können, besitzen die gleiche Lösung. Es ist zu beachten, dass alle Transformationen nicht nur auf die Matrix A, sondern auch auf die rechte Seite b angewandt werden müssen!

Im Detail läuft das Gaußsche Eliminationsverfahren wie folgt ab:

- **Vorwärtselimination**

  Der Algorithmus geht spaltenweise vor. Er versucht für jede Spalte, beginnend bei der ersten, jeweils alle Elemente unterhalb der Diagonalen zu eliminieren (zu null zu setzen). Hierbei werden die ersten zwei oben genannten Umformungen benötigt.

  Dies soll am Beispiel der ersten Spalte veranschaulicht werden. Ausgangspunkt ist die Matrix und die rechte Seite aus Gl. (4.20). Es wird zunächst ein vielfaches der erste Zeile so zur zweiten addiert, dass das erste Element der zweiten Zeile zu null wird. Auf diese Weise wird für jede weitere Zeile unterhalb der Diagonalen verfahren. Es ergibt sich das folgende neue Gleichungssystem:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} - \frac{a_{21}}{a_{11}} a_{12} & \dots & a_{2n} - \frac{a_{21}}{a_{11}} a_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n2} - \frac{a_{n1}}{a_{11}} a_{12} & \dots & a_{nn} - \frac{a_{n1}}{a_{11}} a_{1n} \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{4.23}$$

  Dieses Vorgehen erfordert eine Division durch das Diagonalelement a_{11}, welches auch Pivotelement (franz. pivot - Dreh- und Angelpunkt) genannt wird. Diese Tatsache bringt zwei Probleme mit sich: Erstens darf das Pivotelement nicht zu null werden. In so einem Fall würde der Algorithmus abbrechen. Zweitens sollte in der Numerik die Division durch eine (betragsmäßig) sehr kleine Zahl vermieden werden, da dadurch eine sehr große Zahl entsteht. Um diese beiden Fälle zu verhindern, wird in der Praxis der Gauß-Algorithmus meist mit einer sogenannten **Pivotisierung** durchgeführt.

  Für jede Spalte muss hierfür ein geeignetes Pivotelement gesucht werden. Um eine Division durch allzu kleine Zahlen oder null zu vermeiden, wird jeweils der betragsmäßig größte Koeffizient unterhalb des aktuellen Diagonalelements als Pivotelement gewählt. Hierzu wird die entsprechende Zeile des gewünschten Pivotelements mit der Zeile des Diagonalelements getauscht. Dieser Prozess wird auch als Pivotisierung bezeichnet. Nur auf diese Weise findet der Algorithmus für alle lösbaren Systeme den Lösungsvektor. Erkauft wird dies jedoch mit einem erhöhten Rechenaufwand, da das Suchen des geeignetsten Pivotelements und das Tauschen der Zeilen zusätzlichen Aufwand bedeutet.

Sind alle Elemente der ersten Spalte unterhalb der Diagonalen gleich null, so kann analog mit jeder weiteren Spalte bis zur vorletzten verfahren werden. Übrig bleibt die gesuchte Matrix in Stufenform wie sie in Gl. (4.21) dargestellt ist

- **Rücksubstitution**

  Wurde die Stufenform der Matrix A erreicht, kann durch schrittweises Einsetzen von $i = n$ bis $i = 1$ unter Verwendung von Gleichung (4.22) der Lösungsvektor x bestimmt werden.

4.2.1 LR-Zerlegung

Der numerische Aufwand des Gaußschen Eliminationsverfahren lässt sich zu $\mathcal{O}\left(n^3\right)$ angeben. Dies bedeutet, die Lösung eines doppelt so großen Gleichungssystems benötigt in etwa acht Mal soviel Zeit. Der Aufwand wird dabei von der Vorwärtselimination mit Pivotisierung, welche von der Ordnung $\mathcal{O}\left(n^3\right)$ ist, dominiert. Der Aufwand der Rücksubstitution dagegen ist von der Ordnung $\mathcal{O}\left(n^2\right)$. Die Motivation für die Anwendung der sogenanten LR-Zerlegung (im Englischen auch LU-Zerlegung genannt) beruht auf dieser Tatsache.

Soll beispielsweise ein Gleichungssystem $Ax = b$ für viele verschiedene b Vektoren gelöst werden, so zeigt sich, dass nur das erste Mal eine Vorwärtselimination notwendig ist. Diese wird als sogenannte LR-Zerlegung ausgeführt, welche alle notwendigen Umformschritte des oben vorgestellten Eliminationsverfahrens in einer Matrix L und einem Vektor p speichert. Die Lösungen für alle übrigen rechten Seiten können dann unter Verwendung von L und p alleine mit Hilfe einer Permutation des Vektors b, einer Vorwärts- sowie einer Rückwärtssubstitution berechnet werden, die jeweils einen Aufwand der Ordnung $\mathcal{O}\left(n^2\right)$ besitzen. Der Aufwand für jedes weitere zu lösende System reduziert sich daher von $\mathcal{O}\left(n^3\right)$ auf $3 \cdot \mathcal{O}\left(n^2\right)$. Bezugnehmend auf das vorherige Kapitel ist dies beispielsweise relevant, falls Fits für verschiedene Messdaten ($A^T b$) an den gleichen Messpunkten bestimmt werden sollen.

Wir wollen nun definieren was unter einer LR-Zerlegung zu verstehen ist. Es handelt sich dabei um ein Gaußsches Eliminationsverfahren bei dem alle Umformungsschritte (Vertauschung zweier Zeilen, Multiplikation einer Zeile mit einem Skalar,...) mit gespeichert werden. Am Ende ergeben sich dabei eine untere Dreiecksmatrix L (*links*), deren Diagonalelemente eins sind (per Definition), eine obere Dreiecksmatrix R (*rechts*) sowie ein Permutationsvektor p:

$$Ax = b \quad \Rightarrow \quad LRx = b_1 \tag{4.24}$$

wobei gilt

$$L = \begin{pmatrix} 1 & 0 & \dots & 0 \\ a'_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ a'_{n1} & a'_{n2} & \dots & 1 \end{pmatrix} \quad R = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1n} \\ 0 & a'_{22} & \dots & a'_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & a'_{nn} \end{pmatrix}. \tag{4.25}$$

Zu beachten ist, dass die rechte Seite b_1 mit Hilfe des Permutationsvektors p aus b berechnet werden kann: $b_1 = b(p)$.

Die Matrix R ist die bereits aus dem Gaußschen Eliminationsverfahren bekannte obere Dreiecksmatrix. Die Matrix L beinhaltet alle zur Berechnung von R notwendingen Umformungsschritte, ausgenommen des Tauschens zweier Zeilen. Es sollte beachtet werden, dass bei der Vorwärtselimination in der L-Matrix nur entsprechende Einträge unterhalb der Diagonalen mitgetauscht werden dürfen, falls zwei Zeilen in R vertauscht werden! Alle Tauschmodifikationen werden im Permutationsvektor p gespeichert. Wurden L,R und p für eine gegebene Matrix A bestimmt, so lässt sich das zugehörige Gleichungssystem alleine durch das Anwenden der im Permutationsvektor gespeicherten Permutationen auf den Vektor b sowie einer Vorwärts- und einer Rückwärtssubstitution lösen:

$$\begin{aligned} b_1 &= b(p) && \text{Permutation von } b \\ Ly &= b_1 && \text{Vorwärtssubstitution} \\ Rx &= y && \text{Rücksubstitution} \end{aligned}$$

Die Vorwärtssubstitution ist identisch zur Rücksubstitution, mit dem Unterschied, dass erstere von oben nach unten im Lösungsvektor vorgeht und letztere genau umgekehrt. Der Vektor y ist hierbei lediglich eine Zwischengröße

$$L \underbrace{Rx}_{=:y} = b_1. \tag{4.26}$$

Zuletzt bleibt noch das Aussehen des Permutationsvektors zu klären. Initialisiert wird dieser als eine aufsteigende Folge natürlicher Zahlen von 1 bis n:

$$p = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ n \end{pmatrix}. \tag{4.27}$$

Jedes Vertauschen von Zeilen wird nun auch auf diesen Vektor angewandt. Am Ende des Algorithmuses kann dann abgelesen werden, welches Element von b in welcher Zeile von b_1 stehen muss. In Matlab kann die Permutation von b direkt durch `b_1=b(p)` umgesetzt werden. Theoretisch kann anstatt eines Permutationsvektors auch eine Permutationsmatrix P definiert werden. Es gilt dann $b_1 = Pb$. Da diese Matrix jedoch viele Nulleinträge enthalten würde, ist es aufgrund des geringeren Speicherbedarfs sinnvoller lediglich einen Vektor zu speichern. Die vorgestellte Theorie soll nun im Folgenden anhand eines Beispiels veranschaulicht werden.

4.2.2 Beispiel LR-Zerlegung für ein 4x4-Gleichungsystem

Folgendes Gleichungssystem soll mit Hilfe der LR-zerlegung gelöst werden:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 10 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 24 & 14 & 15 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 70 \\ 183 \\ 252 \\ 422 \end{pmatrix}. \tag{4.28}$$

Die Matrix A wird nun als Produkt mit der Einheitsmatrix geschrieben:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 10 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 24 & 14 & 15 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 10 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 24 & 14 & 15 \end{pmatrix} \tag{4.29}$$

Alle Modifikationen der Matrix A sollen nun durch Änderungen der Einheitsmatrix kompensiert werden, so dass der b Vektor unverändert bleibt ($LRx = b$). Damit wird nach Beendigung des Verfahrens die Einheitsmatrix zur unteren Dreiecksmatrix L und A zur oberen Dreiecksmatrix R. Wie oben erwähnt, ist eine Pivotisierung - also ein Vertauschen von Zeilen - zwingend notwendig damit das Verfahren sicher eine Lösung liefert. Diese Tauschoperationen werden im Vektor p gespeichert, der mit den Zahlen von 1 bis 4 initialisiert wird. Dieser wird im Folgenden ganz rechts mit angegeben.

- **Schritt 1** Vorwärtselimination der 1. Spalte: Das betragsmäßig größte Element wird gesucht (Pivotelement). Dies ist in Zeile 4 zu finden, weshalb Zeile 1 und 4 getauscht werden. Ebenso wird mit dem Permutationsvektor verfahren. Die Matrix L bleibt unverändert, da für diesen Zeilentausch keine Elemente unterhalb der Diagonalen getauscht werden können:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 12 & 24 & 14 & 15 \\ 5 & 10 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 1 & 2 & 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 422 \\ 183 \\ 252 \\ 70 \end{pmatrix} \left\| \begin{pmatrix} 4 \\ 2 \\ 3 \\ 1 \end{pmatrix} \right. . \tag{4.30}$$

- **Schritt 2** Nun wird die erste Zeile durch 12 geteilt und ihr 5- bzw. 8- bzw. 1-faches von der 2., 3. und 4. Zeile subtrahiert. Dies wird durch entsprechende Einträge in der linken Matrix kompensiert, so dass sich der korrekte b-Vektor ergibt:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 5/12 & 1 & 0 & 0 \\ 2/3 & 0 & 1 & 0 \\ 1/12 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 12 & 24 & 14 & 15 \\ 0 & 0 & 1/6 & 3/4 \\ 0 & -7 & 2/3 & 1 \\ 0 & 0 & 11/6 & 11/4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 422 \\ 183 \\ 252 \\ 70 \end{pmatrix} \left\| \begin{pmatrix} 4 \\ 2 \\ 3 \\ 1 \end{pmatrix} \right. . \tag{4.31}$$

- **Schritt 3** Vorwärtselimination der 2. Spalte: Hier ist jetzt der Fall eingetreten, dass das Diagonalelement gleich null ist. Eine Pivotisierung ist also zwingend notwendig. Das betragsmäßig größte Element unterhalb der Diagonalen ist in der 3. Zeile zu finden, weshalb wir diese mit der 2. tauschen. Ebenso wird mit dem Permutationsvektor verfahren. In der Matrix L finden sich lediglich in der ersten Spalte zwei Elemente unterhalb der Diagonalen die getauscht werden müssen:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2/3 & 1 & 0 & 0 \\ 5/12 & 0 & 1 & 0 \\ 1/12 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 12 & 24 & 14 & 15 \\ 0 & -7 & 2/3 & 1 \\ 0 & 0 & 1/6 & 3/4 \\ 0 & 0 & 11/6 & 11/4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 422 \\ 252 \\ 183 \\ 70 \end{pmatrix} \left\| \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix} \right. . \tag{4.32}$$

- **Schritt 4** Die Elemente unterhalb der Diagonalen in der 2. Spalte sind bereits null, daher ist nichts weiter zu tun:

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 2/3 & 1 & 0 & 0 \\ 5/12 & 0 & 1 & 0 \\ 1/12 & 0 & 0 & 1 \end{array}\right) \cdot \left(\begin{array}{cccc} 12 & 24 & 14 & 15 \\ 0 & -7 & 2/3 & 1 \\ 0 & 0 & 1/6 & 3/4 \\ 0 & 0 & 11/6 & 11/4 \end{array}\right) \cdot \left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\right) = \left(\begin{array}{c} 422 \\ 252 \\ 183 \\ 70 \end{array}\right) \left| \left(\begin{array}{c} 4 \\ 3 \\ 2 \\ 1 \end{array}\right) \right. . \quad (4.33)$$

- **Schritt 5** Vorwärtselimination der 3. Spalte: Der betragsmäßig größte Koeffizient unterhalb der Diagonalen befindet sich in der 4. Zeile, weshalb wir diese mit der 3. tauschen. In der L-Matrix werden entsprechende Elemente der ersten und zweiten Spalte (beide Null!) vertauscht:

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 2/3 & 1 & 0 & 0 \\ 1/12 & 0 & 1 & 0 \\ 5/12 & 0 & 0 & 1 \end{array}\right) \cdot \left(\begin{array}{cccc} 12 & 24 & 14 & 15 \\ 0 & -7 & 2/3 & 1 \\ 0 & 0 & 11/6 & 11/4 \\ 0 & 0 & 1/6 & 3/4 \end{array}\right) \cdot \left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\right) = \left(\begin{array}{c} 422 \\ 252 \\ 70 \\ 183 \end{array}\right) \left| \left(\begin{array}{c} 4 \\ 3 \\ 1 \\ 2 \end{array}\right) \right. . \quad (4.34)$$

- **Schritt 6** Die 3. Zeile wird durch $11/6$ dividiert und dann ihr $1/6$-faches von der 4. abgezogen:

$$\underbrace{\left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 2/3 & 1 & 0 & 0 \\ 1/12 & 0 & 1 & 0 \\ 5/12 & 0 & 1/11 & 1 \end{array}\right)}_{=L} \cdot \underbrace{\left(\begin{array}{cccc} 12 & 24 & 14 & 15 \\ 0 & -7 & 2/3 & 1 \\ 0 & 0 & 11/6 & 11/4 \\ 0 & 0 & 0 & 1/2 \end{array}\right)}_{=R} \cdot \left(\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\right) = \underbrace{\left(\begin{array}{c} 422 \\ 252 \\ 70 \\ 183 \end{array}\right)}_{=b_1} \left| \underbrace{\left(\begin{array}{c} 4 \\ 3 \\ 1 \\ 2 \end{array}\right)}_{=p} \right. . \quad (4.35)$$

- **Schritt 7** Vorwärtssubstitution: Der Vektor y wird durch die Lösung der Gleichung $Ly = b_1$ berechnet. Wie b_1 aus b bestimmt werden kann, steht im Vektor p:

$$\begin{aligned} y_1 &= 422 \\ y_2 &= 252 - 2/3 \cdot 422 &&= -88/3 \\ y_3 &= 70 - (1/12) \cdot 422 &&= 209/6 \\ y_4 &= 183 - (5/12) \cdot 422 - (1/11) \cdot (209/6) &&= 4 \end{aligned}$$

$$y = \left(\begin{array}{c} 422 \\ -88/3 \\ 209/6 \\ 4 \end{array}\right) \quad (4.36)$$

- **Schritt 8** Rückwärtssubstitution: Entsprechend der Gleichung $Rx = y$ wird nun der Lösungsvektor berechnet:

$$\begin{aligned} x_4 &= 4 \cdot 2 &&= 8 \\ x_3 &= (209/6 - 11/4 \cdot 8)/(11/6) &&= 7 \\ x_2 &= (-88/3 - (2/3) \cdot 7 - 1 \cdot 8)/(-7) &&= 6 \\ x_1 &= (422 - 24 \cdot 6 - 14 \cdot 7 - 15 \cdot 8)/12 &&= 5. \end{aligned}$$

$$x = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} \tag{4.37}$$

Auf diese Weise kann für jeden beliebigen b-Vektor die Lösung berechnet werden. Formelmäßig wird in Schritt 7 und 8 der Lösungsvektor wie in Gl. (4.22) beschrieben bestimmt, wobei die Richtung der Laufvariable in beiden Fällen gerade entgegengesetzt ist.

Der Gauß-Algorithmus stellt eine wichtige Möglichkeit dar, lineare Gleichungssysteme zu lösen. Es soll jedoch erwähnt werden, dass für spezielle Matrizen deutlich effizientere Methoden existieren. Der numerische Aufwand des Gauß-Algorithmus skaliert dabei mit $\mathcal{O}\left(n^3\right)$, wobei n die Anzahl der betrachteten Gleichungen darstellt. Für spezielle, dünnbesetzte Matrizen, wie sie z.B. beim finite Differenzen-Verfahren (Kapitel 8) auftreten, gibt es deutlich effizientere Methoden, deren Aufwand mit bis zu $\mathcal{O}\left(n \log n\right)$ - also fast linear - skaliert. Diese Verfahren sind jedoch keine direkten Verfahren mehr, sonderen sogenannte iterative. Sie verbessern eine Schätzung der Lösung iterativ solange, bis ein definiertes Konvergenzkriterium erfüllt wird. Dies wird im Kurs "Computational Thermo-Fluid Dynamics" genauer thematisiert.

4.3 Übungsaufgaben

Diese Übung widmet sich der Implementierung der Methode der kleinsten Fehlerquadrate ("Least Square") zur optimalen Anpassung von Modellparametern. Als Beispiel dient wiederum das NASA-Polynom zur Bestimmung der spezifischen Wärmekapazität aus Übung 3. Das Ergebnis des Least Square Ansatzes ist ein lineares Gleichungssystem. Zur Lösung dieses Systems wird in der ersten Aufgabe der sogenannte Gauß-Algorithmus implementiert, bevor in der zweiten Aufgabe das eigentliche Least-Square Problem gelöst wird.

4.1 Implementieren Sie das Gaußsche Eliminationsverfahren mit Hilfe einer LR-Zerlegung.

4.1.1 Fertigen Sie für die folgende Aufgabe ein Ablaufdiagramm an, bevor Sie mit der Programmierung beginnen.

4.1.2 Erstellen Sie eine MATLAB-Funktion `Gauss()`, die für eine gegebene Matrix A und einen gegebenen Vektor b das zugehörige lineare Gleichungssystem $Ax = b$ löst. Die Ausgabe der Funktion ist der Lösungsvektor x.

- `[x] = Gauss(A,b)`
  - `x`: Lösungsvektor (m×1)
  - `A,b`: Matrix (m×m) und Vektor des Gleichungssystems (m×1)

Hinweise:

- Nutzen Sie bereits vorhandene MATLAB-Funktionen wie `find()`, `[n,m]=max()`, etc.
- Berücksichtigen Sie, dass Sie mit dem `:`-Operator auf ganze Zeilen/ Spalten einer Matrix zugreifen können.
- Verwenden Sie so wenig `for`-Schleifen wie möglich. Diese lassen sich oftmals durch Vektormultiplikationen `v*v` ersetzen. Achten Sie hierbei jedoch auch auf eine einfache Lesbarkeit Ihres Programmes, die ab und zu die Verwendung einer `for`-Schleifen rechtfertigen kann.
- Testen Sie Ihre Funktion mit Hilfe eines einfachen Beispiels, welches Sie von Hand nachrechnen können bevor Sie mit der nächsten Aufgabe fortfahren (z.B. jenes in Kapitel 4.2.2)!

4.1.3 Erweitern Sie Ihre Funktion so, dass sie mehrere b-Vektoren als Eingabe akzeptiert, also eine b-Matrix. Die Ausgabe ist dann eine entsprechend große x-Matrix. Nutzen Sie die Vorteile der LR-Zerlegung! Vergleichen Sie die Laufzeiten, wenn Sie zunächst Ihre Funktion mit einer b-Matrix bestehend aus k b-Vektoren aufrufen und anschließend die gleiche Funktion k-Mal mit jeweils einem b-Vektor ausführen. Was ist zu erwarten? Warum?

4.2 Zur Approximation des Verlaufes der spezifischen Wärmekapazität kann für einen bestimmten Temperaturbereich, wie bereits aus Aufgabe 3 bekannt, das sogenannte NASA-Polynom verwendet werden. Es lässt sich als

$$\frac{c_p(T)}{R_s} = a_1 + a_2 T + a_3 T^2 + a_4 T^3 + a_5 T^4 \tag{4.1}$$

schreiben. Die Koeffizienten dieses Polynoms werden dabei mit Hilfe der Methode der kleinsten Fehlerquadrate bestimmt. Es wird gefordert, dass die Summe aller Fehler im Quadrat minimal sein soll:

$$\min_{a_i} \sum_{n=1}^{k} ||c_{p,Modell}(T_n, a_i) - c_{p,Messung}(T_n)||^2 \iff \frac{\partial}{\partial a_i} \sum_{n=1}^{k} ||c_{p,Modell}(T_n, a_i) - c_{p,Messung}(T_n)||^2 = 0. \tag{4.2}$$

Hierbei wird der Fehler als Differenz zwischen Messung und Modell an k Messdaten ausgewertet.

4.2.1 Bestimmen Sie die fünf Koeffizienten a_i des NASA-Polynoms derart, dass der quadratische Fehler der Näherung zu den k Messdaten minimal wird. Überprüfen Sie die Qualität Ihres "Fits" grafisch. Erstellen Sie hierfür ein Diagramm, welches Ihre Messwerte sowie den "wahren" und den gefitteten Verlauf von c_p über der Temperatur enthält. Verwenden Sie zur Lösung des entstehenden linearen Gleichungssystems den in Aufgabe 4.1 programmierten Gauß-Algorithmus.

4.2.2 Bestimmen Sie mittels der MATLAB-Funktion `cond()` die Kondition Ihrer Matrix A sowie des Produktes $A^T A = \tilde{A}$. Berechnen Sie auch das Produkt $A^T b = \tilde{b}$. Was fällt Ihnen auf?

4.2.3 Lösen Sie das Minimierungsproblem auch mit dem MATLAB-internen Operator "\":

$$x = A \backslash b. \tag{4.3}$$

Hinweise:

- Überlegen Sie sich zunächst, wie Ihre Matrix A sowie Ihre Vektoren x und b für das gegebene Problem aussehen, so dass es wie folgt formuliert werden kann:

$$\min_x ||Ax - b||^2. \tag{4.4}$$

- Aus Gl. (4.4) ergibt sich folgendes, lineares Gleichungssystem:

$$\tilde{A}x = \tilde{b}. \tag{4.5}$$

- Die "Messdaten" können Sie mittels der Funktion `Create_measurements()` erhalten. Diese Funktion berechnet die c_p-Werte mit Hilfe von XSteam bei $p = 1\ bar$ und überlagert den Ergebnissen ein weißes Rauschen. Auf diese Weise streuen die künstich erzeugten Messwerte um den "wahren" Verlauf von c_p. Diese Funktion können Sie wie folgt verwenden:

```
[c_p_noise,c_p]=Create_measurements(T_lower,T_upper,delta,noiseLevel)
```

  Sie geben eine untere sowie ein obere Temperatur vor (`T_lower` und `T_upper`). Die Funktion berechnet Ihnen dann für den gewählten Temperaturbereich die gesuchten c_p-Werte im Abstand `delta`. Hierbei enthält die erste Ausgabe - `c_p_noise` - die Werte samt weißen Rauschens und die zweite Ausgabe - `c_p` - den "wahren" Verlauf. Mit dem Parameter `noiseLevel` bestimmen Sie, wie stark die Messwerte um den "wahren" Wert schwanken. Ein guter Wert ist hierfür 40. Beachten Sie, dass Sie für jeden Aufruf der Funktion neue Messwerte erhalten.

4.4 Wichtige Matlab Befehle

Befehl	Beschreibung
`\`	Operator zum Lösen von linearen Gleichungssystemen: `x = A \ b`. Gibt für überbestimmte Systeme automatisch die im Sinne der kleinsten Fehlerquadrate optimale Lösung zurück.
`A([a b],:)=A([b a],:)`	Tauscht die Zeilen a und b einer Matrix A.
`B=A(a:b,c:d)`	Schneidet eine Teilmatrix B aus der Matrix A. Diese ergibt sich aus Zeile a bis b und Spalte c bis d von A.
`cond()`	Gibt eine Näherung der Kondition einer Matrix aus.
`fit()`	MATLAB-interne Funktion zum Fitten von Daten.
`inv()`	Berechnet die Inverse einer Matrix.
`norm()`	Berechnet die 2-Norm einer Matrix.
`pinv()`	Berechnet die Moore-Penrose Pseudoinverse einer Matrix.

5 Numerische Nullstellenbestimmung

Lernziele

- Bisektions-Verfahren
- Sekanten-Verfahren
- Newton-Verfahren

Motivation

Die Nullstellensuche ist eine weitere grundlegende Aufgabe in der Numerik. Gegeben sei ein System an (nichtlinearen) Gleichungen

$$F(x) = 0, \tag{5.1}$$

wobei F das System darstellt und x ein Vektor ist. Für lineare Gleichungssysteme lässt sich dies auch als

$$Ax - b = 0 \tag{5.2}$$

schreiben und kann, wie in Kapitel 4.2 beschrieben, mit Hilfe des Gauß-Algorithmus gelöst werden. Man bezeichnet dies als ein direktes Verfahren, da das System durch Auflösen nach den Unbekannten gelöst wird.

Für nichtlineare Systeme muss nach anderen Methoden gesucht werden, da sich diese i.A. nicht einfach nach den gesuchten Größen auflösen lassen. Zur Veranschaulichung wird die Berechnung der Rohrreibungszahl λ betrachtet (siehe auch Moody-Diagramm). Diese ist eine dimensionslose Kennzahl zur Bestimmung des Druckverlustes in einem Rohr. Im Falle turbulenter Strömungen ergeben sich, abhängig von der Reynolds-Zahl Re und der Rauigkeit der Rohrinnenwand, verschiedene Korrelationen um λ zu bestimmen. Für hydraulisch glatte Rohre ergibt sich beispielsweise die Formel nach Prandtl:

$$\frac{1}{\sqrt{\lambda}} = 2\log_{10}\left(Re\sqrt{\lambda}\right) - 0.8. \tag{5.3}$$

Dabei handelt es sich um eine sogenannte transzendente Gleichung, d.h. es kann keine analytische Lösung für λ angegeben werden. Das Problem ist zudem eindimensional. Schreibt man alle Terme auf eine Seite des Gleichheitszeichen, so lässt sich das Problem in eine Nullstellensuche, wie sie in Gleichung (5.1) definiert wurde, umformen. Bei Betrachtung der Gleichung werden sofort einige Probleme offenbar: Der Logarithmus ist im Reellen nicht für negative Zahlen definiert, genausowenig wie die Quadratwurzel. Außerdem liegt eine Polstelle bei $\lambda = 0$ vor. All dies schränkt das Gebiet ein in dem nach der Nullstelle gesucht werden kann. Leider stehen a priori numerischen Algorithmen diese Informationen nicht zur Verfügung, weswegen dies u.U. zu Problemen führen könnte.

Eine weitere Schwierigkeiten mit nichtlinearen Systemen besteht darin, dass die Anzahl der Lösungen - sofern überhaupt eine existiert - i.A. unbekannt ist. Aus diesen Gründen gibt es im nichtlinearen keine universell anwendbare Methode zur Nullstellensuche, wie es der Gauß Algorithmus für lineare Probleme war. Jedes Problem muss für sich betrachtet werden. Nichtsdestotrotz existieren einige Standardverfahren mit denen eine Vielzahl an Problemen gelöst werden können. Die drei bekanntesten, das Bisektions-Verfahren, das Sekanten-Verfahren und das Newton-Raphson-Verfahren, werden in diesem Kapitel vorgestellt. Jedes dieser Verfahren besitzt dabei individuelle Vor- und Nachteile sowie einige Parameter (Startwert, Unterrelaxationsfaktor) die korrekt gewählt werden müssen.

Wie erwähnt, können nichtlineare Systeme i.A. nicht direkt durch Umstellen gelöst werden. Daher sind muss iterativ vorgegangen werden: Man beginnt mit einer Schätzung der Lösung (Startwert) und verbessert diese sukzessive bis ein Abbruchkriterium erfüllt ist. Die Konvergenz eines Verfahrens lässt sich a priori kaum voraussagen, daher sollte immer eine maximale Zahl an Iterationen festgelegt werden. Mögliche Abbruchkriterien sind im eindimensionalen:

$$\begin{aligned} &(1) \qquad |x_i - x_{i-1}| < \epsilon_1, \\ &(2) \qquad |f(x_i)| < \epsilon_2. \end{aligned}$$

Hierbei bezeichnet x_i die vom Algorithmus berechnete Position der Nullstelle nach dem i-ten Schritt. $f(x)$ ist die Funktion deren Nulldurchgang gefunden werden soll. Nach Kriterium (1) wird also abgebrochen, wenn sich die berechnete Position der Nullstelle für einen Iterationsschritt weniger als ϵ_1 ändert. Im Fall (2) bricht der Algorithmus ab, sobald der Betrag des Funktionswerts an der potentiellen Nullstelle kleiner als ϵ_2 ist. Mit letzt genannter Methode kann eingeschätzt werden, ob wirklich eine Nullstelle gefunden wurde, während erst genanntes Kriterium die Konvergenz des Verfahrens bewertet. Im Zweifelsfall sollten beide Methoden gleichzeitig angewandt werden.

Für alle im Folgenden vorgestellten Verfahren gilt, dass sie jeweils nur eine einzige Nullstelle finden können. Die Existenz weiterer Nullstellen ist damit nicht ausgeschlossen. Es sollte daher sichergestellt werden, dass der gefundene Nulldurchgang auch tatsächlich dem gesuchten Ergebnis entspricht.

5.1 Bisektions-Verfahren

Ist bekannt, dass ein bestimmtes Intervall genau eine Nullstelle ungerader Vielfachheit enthält, so kann das Bisketions-Verfahren zur Suche verwendet werden. Die Idee dieser Methode beruht dabei auf dem sukzessiven Eingrenzen der Lösung durch Ausschluss der Bereiche in denen die Lösung auf keinen Fall zu finden ist. Dies lässt sich anhand eines Beispiels veranschaulichen:

Nehmen wir an, wir besäßen N Steine gleichen Gewichtes sowie einen Stein der etwas schwerer als die anderen ist. Wie finden wir diesen einen Stein möglichst schnell wieder wenn wir ihn unter die anderen mischen und uns lediglich eine Balkenwaage zur Verfügung steht? Eine Möglichkeit besteht darin alle Steine der Reihe nach mit einem Vergleichsstein zu wiegen und zu hoffen, dass der gesuchte Stein möglichst bald gefunden ist. Im schlechtesten Fall müssten wir so $N - 1$ Mal wiegen. Alternativ können wir aber auch jeweils $N/2$ Steine links und rechts auf die Waage packen. Ist die linke Seite schwerer, so wissen wir, dass der gesuchte Stein unter diesen sein muss. Anschließend teilen wir diese Steine wiederum auf und es bleiben nach dem Wiegen $N/4$ Steine als mögliche Kandidaten für den schwereren Stein übrig, usw. Nach dem n-ten Wiegen muss der gesuchte Stein also unter $N/2^n$ Steinen zu finden sein. Damit also nur noch genau 1 Stein übrig bleibt, müssen wir dementsprechend $\log_2 N$ Mal wiegen. Gegenüber dem erst genannten Verfahren ist dies eine deutliche Zeitersparnis. Zufallstreffer sind hier allerdings nicht möglich. *(Im vorliegenden Beispiel könnten wir sogar lediglich* $\log_3 N$ *Wiegevorgänge schaffen, falls die Anzahl der Steine eine Potenz von drei ist.)*

Dieses Verfahren kann nun auf die Nullstellensuche übertragen werden: Wir nehmen an, dass wir ein Intervall kennen, in dem eine Nullstelle liegt. Das Vorzeichen des Funktionswertes am linken Punkt des Intervalls unterscheidet sich von dem Vorzeichen am rechten Punkt. Nun wählen wir einen Punkt in der Mitte des Intervalls und vergleichen das Vorzeichen des Funktionswertes dort mit dem Vorzeichen des linken Wertes. Ist es gleich, so muss die Nullstelle in der rechten Hälfte liegen, andernfalls in der linken. Damit haben wir die Nullstelle weiter eingegrenzt und können auf das neue Intervall wieder die gleiche Methodik anwenden. In Abhängigkeit dieser Iterationsschritte n, kennen wir also die Position des Nulldurchgangs mit einer Genauigkeit von $|x_2 - x_1|/2^n$. Hierbei bezeichnen x_1 und x_2 die Grenzen des Anfangsintervalls. D.h. um eine Genauigkeit von ϵ zu erzielen, müssen

$$n = \log_2 \frac{|x_2 - x_1|}{\epsilon} \tag{5.4}$$

Iterationsschritte ausgeführt werden. Dabei werden wir immer ein Ergebnis bekommen, sofern bereits zu Beginn an den Intervallgrenzen verschiedene Vorzeichen vorlagen. Nullstellen von einer geraden Vielfachheit können auf diese Weise also nicht bestimmt werden.

5.2 Sekanten-Verfahren

Ein weiteres relativ einfaches Verfahren zur Nullstellensuche ist das Sekanten-Verfahren. Wie in Abbildung 5.1 verdeutlicht, müssen zunächst zwei Startwerte x_0 und x_1 gewählt werden. Die zugehörigen Punkte auf dem Funktionsverlauf werden in einem ersten Schritt durch eine Sekante miteinander verbunden, welche durch die Gleichung

$$y = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1) + f(x_1) \tag{5.5}$$

bestimmt ist. Diese schneidet an der Stelle x_2 die x-Achse:

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}. \tag{5.6}$$

Nun wird dieser Algorithmus mit diesem neuen Wert wiederholt: Es wird die Sekante zwischen den Funktionswerten an x_1 und x_2 bestimmt und deren Schnittpunkt mit der x-Achse ergibt den Punkt x_3. Dies wird solange durchgeführt bis sich die Nullstelle der letzten Sekante nur noch um einen kleinen Wert ϵ_1 im Vergleich zur vorletzten verändert, bzw. - alternativ - bis der Funktionswert $f(x_n)$ kleiner als ein Wert ϵ_2 ist. Die Nullstelle dieser letzten Sekante ist dann eine Näherung der gesuchten Nullstelle.

Dieses Verfahren konvergiert für glatte Funktionen meist schneller gegen das Ergebnis als das Bisektions-Verfahren. Allerdings ist nicht sicher gestellt, dass es überhaupt konvergiert, da die Nullstelle - anders als beim Bisektionsverfahren - nicht zwischen den Werten x_i und x_{i-1} eingeschlossen ist. Ein Vorteil gegenüber dem im Folgenden vorgestellten Newton-Raphson-Verfahren besteht darin, dass die Funktion $f(x)$ pro Schritt nur einmal ausgewertet werden muss, was für rechenintensive Funktionen ein deutlicher Vorteil sein kann.

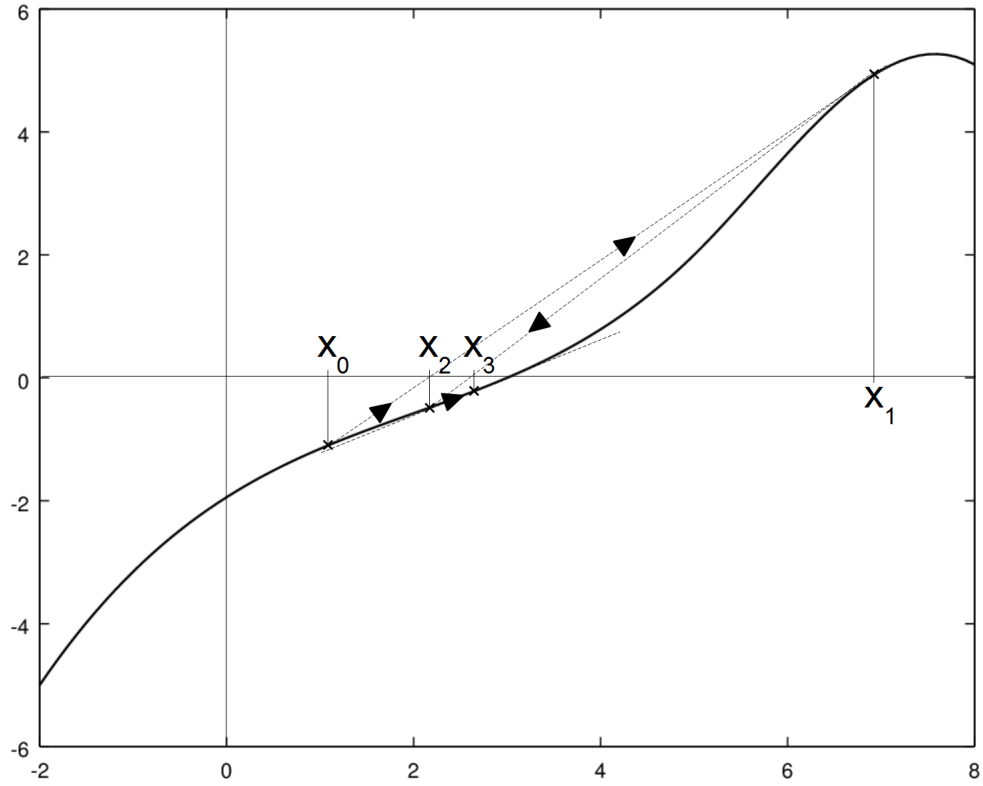


Abbildung 5.1: Veranschaulichung des Algorithmus des Sekanten-Verfahrens.

Um die Konvergenz sicherzustellen, kann das Verfahren dahingehend verändert werden, dass nur dann wie beschrieben vorgegangen werden soll, wenn sich für den neuen Punkt das Vorzeichen der Funktion an diesem Ort ändert. Ist dies nicht der Fall, wird der erste Punkt zur Konstruktion der neuen Sekante beibehalten und für den zweiten x-Wert der neue Punkt verwendet. Somit konvergiert diese als **Regula-falsi-Verfahren** bekannte Methode in jedem Fall, wenn auch langsamer als das originale Sekanten-Verfahren.

5.3 Newton-Raphson-Verfahren

Für dieses Verfahren ist - im Gegensatz zu den bisher beschriebenen - die Auswertung der ersten Ableitung, bzw. für mehrdimensionale Probleme der Jakobi-Matrix, notwendig. Da diese i.A. nicht analytisch bestimmt werden kann, muss sie numerisch angenähert werden. Strategien hierfür wurden bereits in Kapitel 2.1 diskutiert. Die Auswertung ist dabei u.U. sehr rechenintensiv, da zur Bestimmung der Ableitung die Funktion mindestens zwei Mal ausgewertet werden muss.

Ausgangspunkt dieser Methode ist ein nichtlineares Gleichungssystem mit n Gleichungen und

n Unbekannten:

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0 \\ f_2(x_1, x_2, ..., x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, ..., x_n) &= 0. \end{aligned} \tag{5.7}$$

X_1 sei nun ein Vektor mit den Einträgen x_i, $i = 1, \ldots, n$, der als Startwert zur Lösungssuche gewählt wird. Um nun die Position der gesuchten Nullstelle zu bestimmen, wird zunächst an der Stelle X_1 die Tangente an die Funktion bestimmt. Deren Nullstelle wird als erste Abschätzung für die Position der Nullstelle des Gleichungssystems verwendet.

Die Tangente $T_i(X)$ zu berechnen bedeutet die Funktion in der Umgebung von X_1 mittels einer Taylor-Reihe zu approximieren, die nach dem zweiten Glied abgebrochen wird:

$$T_i(X) = f_i(X_1) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \delta x_j. \tag{5.8}$$

Hierbei werden die in der Summe vorkommenden Ableitungen in der sogenannten Jakobi-Matrix zusammengefasst:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}. \tag{5.9}$$

Die Bestimmung der Nullstelle dieser Tangente resultiert in einem linearen Gleichungssystem

$$f_i(X_1) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \delta x_j = 0, \tag{5.10}$$

welches mit den bereits bekannten Mitteln nach $\delta X = (\delta x_1, \delta x_2, \ldots, \delta x_n)$ gelöst werden kann. Damit kann die Position der Nullstelle der Tangente bestimmt werden: $X_2 = X_1 + \delta X$. Im Prinzip wurde damit eine um den Startvektor X_1 linearisierte Version des gegebenen Gleichungssystems gelöst. Der Vektor X_2 befindet sich nun idealerweise näher an der gesuchten Nullstelle des nichtlinearen Systems. Um diesen kann das Gleichungssystem wiederum linearisiert werden und in einem weiteren Schritt analog X_3 berechnet werden. Dies wird nun sooft durchgeführt, bis die Nullstelle mit einer gewünschten Genauigkeit bestimmt ist. Dieses Vorgehen ist in Abbildung 5.2 für drei Schritte anhand eines eindimensionalen Beispiels veranschaulicht.

Die Grundidee des Verfahrens besteht darin, dass die gegebene Funktion für positive Funktionswerte zur Nullstelle hin stetig abfällt bzw. für negative analog stetig zunimmt. Ist dies gewährleistet konvergiert das Newton-Raphson-Verfahren schneller als die beiden bisher vorgestellten Verfahren. In der Umgebung einer Nullstelle muss es immer einen Punkt geben, für den dieses Verfahren konvergiert. Dieser Punkt ist jedoch a priori nicht bekannt, da der Funktionsverlauf ebenfalls unbekannt ist. Die Kunst ist es nun den Startwert so zu wählen, dass

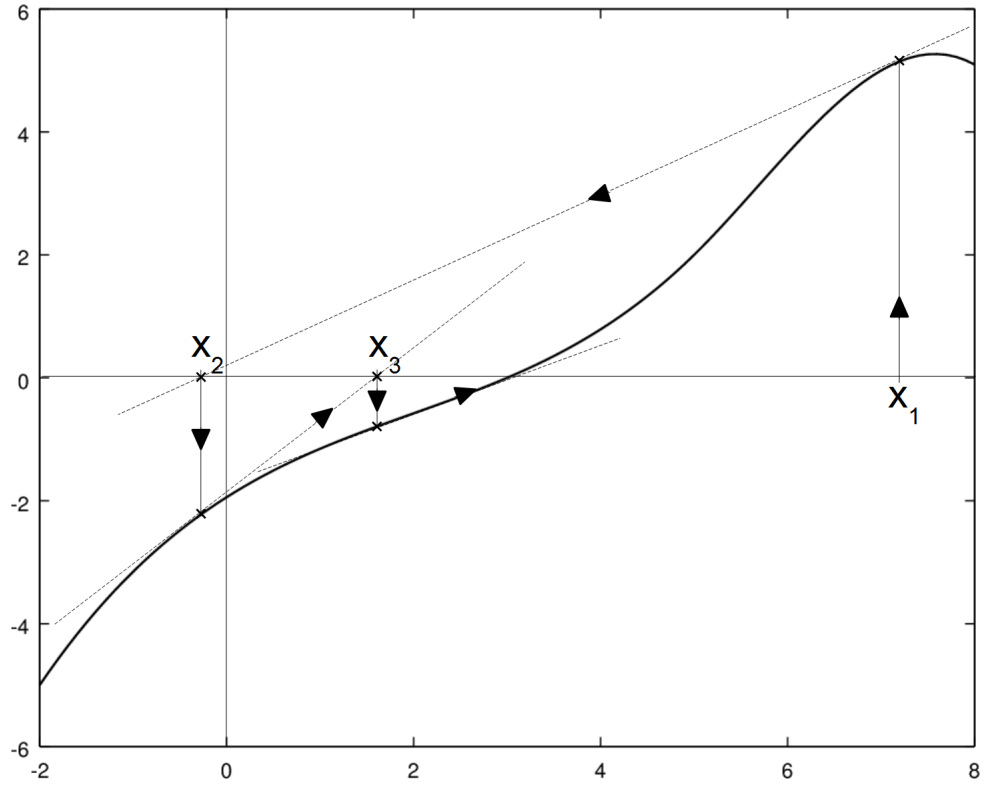


Abbildung 5.2: Veranschaulichung Newton-Raphson-Verfahrens.

das Verfahren gegen die Nullstelle konvergiert. Wie hierzu genau vorgegangen werden sollte, ist abhängig vom Problem. Es kann z.B. ein vereinfachter physikalischer Zusammenhang ausgewertet werden, auf Messdaten zurückgegriffen werden oder grafisch der Funktionsverlauf für ausgewählte Punkte geplottet werden. Wird z.B. der Startpunkt für die Funktion in Abbildung 5.2 rechts neben dem eingezeichneten Maximum oder sogar auf diesem gewählt („worst case"), so wird entweder eine andere Nullstelle gefunden oder das Verfahren divergiert im schlechtesten Fall.

Die Newton-Raphson Methode hat den Vorteil, dass sie sehr übersichtlich und sehr allgemein anwendbar ist. Durch den Trick der Linearisierung des Gleichungssystems, lässt sich ein nichtlineares Gleichungssystem mit linearen Gleichungslösern lösen. Ein Großteil der Rechenzeit nimmt die Bestimmung der sogenannten Jacobi-Matrix in Anspruch, die für jeden Iterationsschritt neu berechnet werden muss.

Wie oben erwähnt ist die Newton-Raphson Methode nicht unbedingt stabil, sprich sie konvergiert nicht zwangsläufig. Vielmehr ist es zum einen, sehr wichtig eine geeignete Startposition X_1 in der Nähe der erwarteten Lösung zu wählen. Zum anderen kommt man fast nie ohne eine sogenannte **Unterrelaxation** aus. Hierfür hat sich bewährt zu überprüfen ob die folgende Bedingung erfüllt ist:

$$\sum_{i=1}^{n} |f_i(X_n)| \leq \sum_{i=1}^{n} |f_i(X_{n+1})|. \tag{5.11}$$

Es wird hierbei getestet, ob man sich weiter von 0 entfernt (0 wäre ja die Lösung). Ist Bedingung (5.11) erfüllt, so wird die Schrittweite δX proportional zum sogenannten Unterrelaxationsfaktor $\gamma \in (0, 1)$ verringert und es wird nur der kleinere Schritt $\gamma\ \delta X$ ausgeführt. Andernfalls wird der volle Schritt gegangen.

Von allen hier vorgestellten Verfahren konvergiert das Newton-Raphson-Verfahren am schnellsten, d.h. pro Schritt wird die Anzahl der korrekten Ziffern der Nullstelle am schnellsten erhöht. Allerdings ist gleichzeitig auch die Tendenz zu divergieren am größten. Des Weiteren ist es jedoch in Punkto Geschwindigkeit im eindimensionalen Fall fast immer dem Sekanten-Verfahren unterlegen, da es pro Schritt zwei Evaluationen der Funktion benötigt. Dieser Nachteil kann durch die geringere Anzahl an benötigten Schritten nicht aufgewogen werden. Für höherdimensionale Probleme stellt es jedoch immer, auch mangels Alternativen, ein in Erwägung zu ziehendes Verfahren dar.

In der Praxis wird oftmals auch eine Kombination aus den Verfahren verwendet: Es kann z.B. mit Hilfe des Bisektionsverfahrens ein Vorzeichen-Wechsel gefunden und bis zu einer gewissen Intervallbreite die Nullstelle eingeschlossen werden. Anschließend wird die genau Position der mit Hilfe des Sekanten Verfahrens bestimmt.

5.4 Übungsaufgaben

Die heutige Übung behandelt die Nullstellensuche. Im Allgemeinen handelt es sich hierbei um Systeme nichtlinearer Gleichungen, welche gelöst werden sollen. Um den Rahmen der Übung nicht zu sprengen, wird im Folgenden eine einzelne, nichtlineare Gleichung behandelt. Hierzu werden zunächst drei ausgewählte Verfahren implementiert und diese dann auf ein physikalisches Problem angewandt.

5.1 Zunächst sollen wieder die in diesem Kapitel beschriebenen Algorithmen zur Nullstellensuche für beliebige Funktionen umgesetzt und getestet werden. Als Eingabeparameter sollen diese zum einen die untersuchte Funktion als anonyme Funktion ("function handle") und zum anderen einen Startwert bzw. ein Startintervall erhalten. Erstellen Sie zur Bearbeitung der Teilaufgaben 3 bis 6 auf einem Blatt Papier eine Tabelle (wie hier abgebildet) und tragen Sie Ihre Ergebnisse dort ein.

Funktion	Verfahren	Startwerte x_1 und x_2	Nullstelle	Laufzeit	Anzahl Iterationen
$x^3 + x + 1$	Newton	...	...	...	...
	Sekanten	...	...	...	...
...	...	...	...	...	...

5.1.1 Fertigen Sie für das Bisektions-Verfahren ein Ablaufdiagramm an, bevor Sie mit der Programmierung beginnen.

5.1.2 Implementieren Sie jeweils in einer eigenen MATLAB-Funktion das Sekanten-, das Bisektions- sowie das Newton-Verfahren und testen Sie Ihre Implementierungen mit Hilfe analytisch auswertbaren Funktionen.

- `[x_out,n]= Bisektion(f,x_1,x_2)`
  `[x_out,n]= Sekanten(f,x_1,x_2)`
  `[x_out,n]= Newton(f,x_1)`
  - `x_out`: Nullstelle
  - `n`: Anzahl der Iterationen
  - `f`: Funktions-Handle
  - `x_1,x_2`: Startwerte

5.1.3 Ihr Algorithmus benötigt einen Startwert von dem aus er die Suche nach einer Nullstelle beginnen kann. Was für einen Wert wählen Sie hier? Ist ihr Ergebnis unabhängig von der Wahl dieses Wertes?

5.1.4 Führen Sie Messungen der Laufzeit für die einzelnen Verfahren durch. Mitteln Sie hierbei über meherer Durchläufe (ca. 100). Welches Verfahren ist das schnellste? Wo liegt Ihrer Meinung nach der zeitkritischste Punkt der Verfahren, speziell im Hinblick auf größere Systeme? Verwenden Sie auch den Profiler (siehe wichtige Matlab Befehle).

5.1.5 Vergleichen Sie Laufzeit und Anzahl der benötigten Iterationen der einzelnen Verfahren (bei vergleichbarer Genauigkeit des Ergebnisses). Variieren Sie auch Ihre Testfunktionen (nichtlineare Terme). Benötigt das Verfahren mit der geringsten Laufzeit auch die kleinste Anzahl an Iterationen?

5.1.6 Wenn Sie Ihre bisherigen Beobachtungen betrachten, wo liegen die Vor- und Nachteile der einzelnen Verfahren (nicht nur auf die Laufzeit bezogen)? Wie könnte man die Verfahren sinnvoll kombinieren?

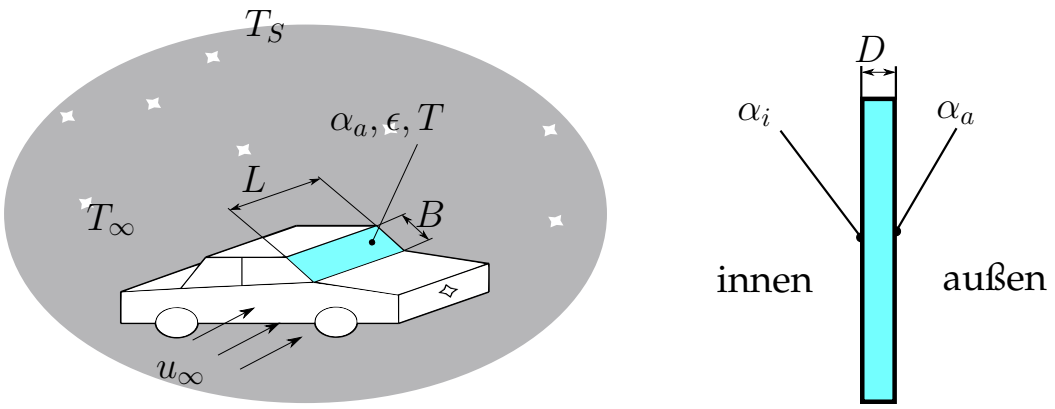


Abbildung 5.1: Windschutzscheibe eines Autos, welches über Nacht draußen geparkt wurde. Links eine Prinzipskizze, rechts die Windschutzscheibe im Schnitt.

In diesem Beispiel soll die Temperatur einer Windschutzscheibe eines Autos bestimmt werden, welches über Nacht draußen geparkt wurde. Hierzu sollen drei Einflüsse betrachtet werden: Zum einen soll der Wärmestrom via Strahlung berücksichtigt werden, der sich aufgrund der unterschiedlichen Temperaturen des Körpers und der Umgebung einstellt. Der Nachthimmel soll als schwarzer Körper der Temperatur T_S ("S" für Sky) betrachtet werden. Der Emissionsgrad der Scheibe betrage ϵ. Zum anderen soll der Wärmeübergang an die Umgebung simuliert werden. Dieser ergibt sich an der Außenseite der Windschutzscheibe aufgrund einer vom Wind erzwungenen Konvektion. Die Scheibe kann als eine mit der Geschwindigkeit u_∞ längsangeströmte ebene Platte betrachtet werden. Hieraus lässt sich der Wärmeübergangskoeffizient α_a abschätzen. An der Innenseite der Scheibe kann der Wärmeübergang mittels eines Koeffizienten α_i berechnet werden, der sich unter Annahme einer freien Konvektionsströmung ergibt. Die Scheibe kann hierfür als senkrechte Platte modelliert werden.

Das Temperaturfeld in der Scheibe soll als homogen angenommen werden. Nimmt man zusätzlich an die Temperatur im Inneren des Autos betrage T_∞, so ergibt sich mit Hilfe des 1. Hauptsatzes für den stationären Fall folgende nichtlineare Gleichung für die Temperatur T der Scheibe:

$$(\alpha_i(T) + \alpha_a)(T_\infty - T) + \sigma\epsilon(T_S^4 - T^4) = 0. \tag{5.1}$$

Die effektive Temperatur des Nachthimmels kann abhängig vom Bewölkungsgrad N als

$$T_S = 280\ K \cdot N + 260\ K \cdot (1 - N) \tag{5.2}$$

modelliert werden. Für $N = 1$ ist der Himmel komplett bewölkt, für $N = 0$ liegt eine klare Nacht vor. Die Parameter sowie Stoffdaten sind wie folgt:

Temperaturen	
T_∞	$275\ K$
T_S	$280\ K \cdot N + 260\ K \cdot (1 - N)$
T_{tau}	$271\ K$
Geometrie	
L	$1.5\ m$
B	$1\ m$
Geschwindigkeiten	
u_∞	$1\ \frac{m}{s}$
Strahlung	
ϵ	0.93
σ	$5.670e-8 \frac{\mathrm{W}}{\mathrm{m^2K^4}}$

5.2 Ist für die gegebenen Größen damit zu rechnen, dass die Windschutzscheibe vereist? Die Taupunkttemperatur betrage $T_{tau} = 271\ K$.

5.2.1 Berechnen Sie zur Beantwortung der Frage die Temperatur T der Windschutzscheibe in einer klaren Nacht ($N = 0$) im stationären Gleichgewicht. Lösen Sie hierfür Gleichung (5.1) numerisch. Überprüfen Sie Ihr Ergebnis anschließend graphisch.

5.2.2 Was für einen Einfluss hat der Bewölkungsgrad? Erstellen Sie zur Beantwortung dieser Frage ein N-T-Diagramm. Zeichnen Sie in dieses für zehn verschiedene $N \in [0, 1]$ jeweils die zugehörige Gleichgewichtstemperatur der Windschutzscheibe. Tragen Sie ebenfalls die Taupunkttemperatur als horizontale Linie in dieses Diagramm ein.

Hinweise:

- Eine MATLAB-Funktion zur Ermittlung der konvektiven Wärmeübergangskoeffizienten α_a und α_i wird bereitgestellt. Die hierfür umgesetzte Theorie finden Sie im Buch *Wärmeübertragung* von W. Polifke und J. Kopitz (2005) auf den Seiten 240 und 189.
- Möchten Sie eine anonyme Funktion für jeden einzelnen Wert eines Vektors $\vec{x}$ auswerten, so können Sie wie folgt vorgehen:

```
y_vec = arrayfun(test,x_vec);
```

  Es wird hierbei nacheinander jeder Wert von `x_vec` in die Funktion `test` gegeben und die Ausgabe im Vektor `y_vec` gespeichert.

Zusatzinfo: `varargin`

- Ist für eine Funktion die Anzahl der zu erwartenden Eingaben unbekannt oder kann sie variieren, so empfiehlt es sich die letzte Eingangsgröße einer Funktion als `varargin` zu definieren. Alle zusätzlichen Eingabeparameter werden dann in einem cell array `varargin` zusammengefasst und können mit Hilfe von `varargin{i}` abgerufen werden, wobei `i` für den i-ten zusätzliche Parameter steht. Folgendes Beispiel soll den Einsatz veranschaulichen:

```
function [ b ] = test( a , varargin )
% Der Parameter c kann optional vom Nutzer verändert werden.
% Der Default-Wert ist 0

ind = find(strcmpi(varargin,'c'),1,'first');
if ~isempty(ind)
  % Verwende Benutzereingabe
  c = varargin{ind+1};
else
  % Verwende Default-Wert
  c = 0;
end

b = a + c;

end
```

- Ein möglicher Einsatzzweck ist die Übergabe von Steuerparameter, wie z.B. die Abbruchkriterien $\epsilon_{1/2}$ bei der Nullstellensuche.

5.5 Wichtige Matlab Befehle

Befehl	Beschreibung
`fsolve()`	MATLAB-interne Funktion zum Lösen von nichtlinearen Gleichungssystemen.
`fplot()`	Funktion zum Plotten von function handles oder per String spezifizierten Funktionen.
`fzero()`	MATLAB-interne Funktion zum Auffinden von Nullstellen nichtlinearer Funktionen.
`mean()`	Berechnet den Mittelwert eines Vektors.
`profile on`	Startet den Profiler.
`profile off`	Beendet den Profiler.
`profile viewer`	Öffnet eine Übersicht aller Ergebnisse des Profilers.
`sign()`	Signum Funktion zur Extraktion des Vorzeichens.

6

Lösen gewöhnlicher Differentialgleichungen

Literaturverzeichnis

[1] P. Deuflhard und F. Bornemann. *Numerische Mathematik 2.: Integration gewöhnlicher Differentialgleichungen*. Kap. 4. De Gruyter, 1994. ISBN: 9783110139372.

[2] Wolfgang Polifke und Jan Kopitz. *Wärmeübertragung: Grundlagen, analytische und numerische Methoden*. Kap. 24.1. Pearson Deutschland GmbH, 2009.

Lernziele

- Euler-Verfahren
- Explizite und implizite Verfahren
- Stabilität und Genauigkeit der Verfahren

Motivation

Die Entwicklung dynamischer Prozesse wird normalerweise mithilfe von Differentialgleichungen (DGL) mathematisch beschrieben. Hierbei wird eine Gleichung dafür angegeben, um welchen Wert sich eine Größe in Abhängigkeit einer anderen, z.B. der Zeit, verändert. Gilt diese Gleichung auch dann für alle betrachteten Zeiten, wenn diese zwei Zeitpunkte beliebig nahe zusammenrücken, so handelt es sich um eine Differentialgleichung. In allgemeiner Form lässt sich eine derartige, gewöhnliche Differentialgleichung wie folgt darstellen:

$$\frac{df(t)}{dt} = g(t, f). \tag{6.1}$$

Der gesuchte Funktionsverlauf wird durch die Funktion f beschrieben. Bei der Funktion g handelt es sich um eine beliebige Funktion, die von der Zeit t sowie von f abhängen kann. Durch Angabe der Differentialgleichung ist das Problem jedoch noch nicht eindeutig bestimmt. Hierfür muss zusätzlich eine Anfangsbedingung

$$f(0) = a \tag{6.2}$$

vorgegeben werden. Probleme die sich auf diese Weise formulieren lassen, werden im Folgenden behandelt.

Das spezielle Vorgehen richtet sich dabei nach der Art der Funktion g. Diese kann nichtlinear in f wie auch in t sein (was wahrscheinlich ist, ansonsten müsste man nicht numerisch nach einer Lösung suchen). Die Grundidee dabei ist, die zeitliche Entwicklung der Lösung durch algebraische Gleichungen anzunähern und diese zu lösen.

Zur allgemeinen Behandlung von Differentialgleichungen ist es ausreichend lediglich Systeme erster Ordnung zu betrachten, da sich Gleichungen höherer Ordnung immer auf ein System von Differntialgleichungen erster Ordnung reduzieren lassen:

$$\frac{d^2 f(t)}{dt^2} + \frac{df(t)}{dt} = g(t, f). \tag{6.3}$$

Substituieren wir $df(t)/dt$ mit z, so ergibt sich ein System gewöhnlicher Differentialgleichungen erster Ordnung:

$$\begin{bmatrix} \frac{df(t)}{dt} \\ \frac{dz(t)}{dt} \end{bmatrix} = \begin{bmatrix} z(t) \\ g(t, f) \end{bmatrix}. \tag{6.4}$$

Diese Technik kann sukzessiv auch für DGLs höherer Ordnung angewendet werden. Systeme von DGLs höherer Ordnung lassen sich ebenfalls in (höherdimensionale) Systeme erster Ordnung transformieren. Wird also die Numerik für DGL-Systeme erster Ordnung beherrscht, können damit auch immer Systeme höherer Ordnung integriert werden!

6.1 Verfahren zur Zeitdiskretisierung I

Wie bereits in Kapitel 2.1 erörtert wurde, können Differentiale in der Numerik nur als Differenzen dargestellt werden. Es wurden entsprechende Näherungsverfahren angegeben. Die jetzt vorliegende Aufgabe ist allerdings etwas anders: Es soll nicht der Wert der Ableitung bestimmt werden, sondern - umgekehrt - aus dem Wert der Ableitung der Funktionsverlauf. Denn wir kennen zu einem Zeitpunkt t_1 den Wert von $g(t_1, f(t_1))$, der nach Gleichung (6.1) gerade der zeitlichen Ableitung von f zu diesem Zeitpunkt entspricht. Ziel ist es daher mit Hilfe der bereits bekannten Varfahren (Vorwärts- / Rückwärtsdifferenz) die Zeitableitungen zu diskretisieren und anschließend daraus den Funktionswert zum nächsten Zeitpunkt zu berechnen. Nähere Informationen sowie alternative Methoden zur Herleitung und Interpretation der Verfahren sind in Kapitel 7 sowie in [2] und [1] zu finden.

Zunächst soll die **Vorwärtsdifferenz** zur Approximation der zeitlichen Ableitung verwendet werden. Damit ergibt sich für die DGL (6.1) folgende Form:

$$\frac{f(t^{n+1}) - f(t^n)}{t^{n+1} - t^n} = \frac{f^{n+1} - f^n}{h} = g^n + \mathcal{O}(h). \tag{6.5}$$

Hierbei wurde eine verkürzte Schreibweise verwendet, welche die verschiedenen Zeitpunkte als t^n bezeichnet - t^0 ist dabei der Anfangszeitpunkt. Für $f(t^n)$ wird der Ausdruck f^n verwendet und analog dazu wird für die rechte Seite $g(t^n, f^n) = g^n$ geschrieben. Der Zeitschritt $t^{n+1} - t^n$ wird als h bezeichnet. Das asymptotische Verhalten des Fehlers der Diskretisierung $\mathcal{O}(h)$ wurde ebenfalls mit eingefügt.

Verwendet man Formel (6.5) kann eine explizite Gleichung für den Funktionswert f^{n+1} angegeben werden. Verfahren dieser Art werden deshalb auch **explizite** Verfahren genannt. Es lassen sich damit bei gegebenem h schrittweise alle Funktionswerte f^n bis zum gewünschten Endzeitpunkt berechnen. Dieses Verfahren bezeichnet man als das explizite Euler-Verfahren:

Definition: Explizites Euler-Verfahren

$$f^{n+1} = f^n + hg^n + \mathcal{O}\left(h^2\right) \tag{6.6}$$

Nehmen wir an, wir wollen den zeitlichen Verlauf der Funktion $f(t)$ vom Zeitpunkt t_0 bis zum Zeitpunkt t_{end} wissen. Dazu diskretisieren wir den Zeitbereich in $N = 1 + (t_{end} - t_0)/h$ einzelne Zeitpunkte. Wir können nun, indem wir iterativ Gleichung (6.6) N Mal lösen, den gesuchten Funktionsverlauf approximieren. Hierbei ist zu berücksichtigen, dass wir für jeden einzelnen Schritt einen lokalen Fehler der Ordnung $\mathcal{O}(h^2)$ begehen.

Es kann zur Näherung der Ableitung auch die **Rückwärtsdifferenz** verwendet werden. Damit ergibt sich für die DGL:

$$\frac{f^n - f^{n-1}}{h} = g^n + \mathcal{O}(h). \tag{6.7}$$

Wir möchten nun, dass diese Gleichung, analog der Vorwärtsdifferenz, auch für $n = 0$ angewendet werden kann (negative n existieren nicht). Dies kann erreicht werden, indem allen n um eins erhöht werden. Damit lässt sich Gleichung (6.7) als

$$\frac{f^{n+1} - f^n}{h} = g(t^{n+1}, f^{n+1}) + \mathcal{O}(h) \tag{6.8}$$

schreiben. Es wurde für die rechte Seite wieder die ausführliche Schreibweise verwendet, um zu verdeutlichen, dass diese eine Funktion von f^{n+1} ist. Kurz würde diese mit g^{n+1} bezeichnet werden. Diese Gleichung kann für jeden Zeitschritt z.B. mittels des Sekanten- oder des Newton-Verfahrens gelöst werden (Kapitel 5). Man spricht hierbei von einem **impliziten** Verfahren, da die gesuchte Funktion nur implizit gegeben ist.

Definition: Implizites Euler-Verfahren

$$f^{n+1} = f^n + hg^{n+1} + \mathcal{O}\left(h^2\right) \tag{6.9}$$

Der lokale Fehler für dieses Verfahren ist ebenfalls wieder von der Ordnung $\mathcal{O}(h^2)$.

Beide soeben hergeleiteten Verfahren nähern den gesuchten Funktionsverlauf linear an, liefern dabei jedoch im Allgemeinen verschiedene Ergebnisse. Das explizite Verfahren benutzt hierfür die Tangente an den Funktionsverlauf zum aktuellen, das implizite die Tangente zum folgenden Zeitpunkt. Mittelt man diese beiden Tangenten und benutzt das Ergebnis als lineare Näherung, so ergibt sich das sogenannte **Crank-Nicolson-Verfahren**.

Definition: (Implizites) Crank-Nicolson-Verfahren

$$f^{n+1} = f^n + \frac{h}{2}\left(g^n + g^{n+1}\right) + \mathcal{O}\left(h^3\right). \tag{6.10}$$

Da f^{n+1} auch in der rechten Seite innerhalb von g^{n+1} der Gleichung vorkommt, gehört das Crank-Nicolson-Verfahren zur Gruppe der impliziten Verfahren. Zur Berechnung der bereits angegebenen Fehlerordnung wird die DGL aus Gleichung (6.1) über einen Zeitschritt der Länge h integriert. Es ergibt sich folgender, exakter Ausdruck:

$$f^{n+1} = f^n + \int_{nh}^{(n+1)h} g(t, f)dt \tag{6.11}$$

Wendet man zur Näherung des Integrals die bereits in Kapitel 3.1 vorgestellte Trapezregel an, so ergibt sich das sogenannte Crank-Nicolson-Verfahren samt Angabe zum asymptotischen Verhalten.

Auch die beiden Euler-Verfahren lassen sich anhand dieses Integrals herleiten, wenn der Funktionsverlauf durch Treppenstufen angenähert wird (Polynom der Ordnung Null). Das explizite

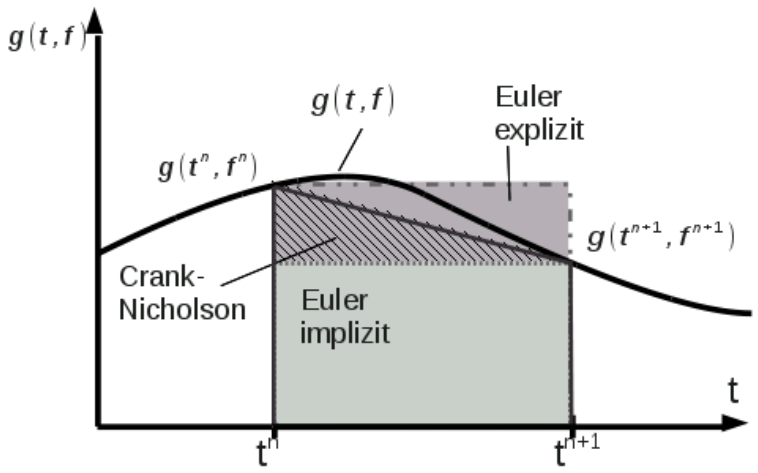


Abbildung 6.1: Veranschaulichung der drei vorgestellten zeitlichen Diskretisierungsmöglichkeiten.

Euler-Verfahren verwendet jeweils den linken Funktionswert eines Intervalls (g^n; Untersumme) während das implizite Verfahren den jeweils rechten Wert (g^{n+1}; Obersumme) zur Näherung des Verlaufes verwendet. Alle drei vorgestellten zeitlichen Integrationsmöglichkeiten sind in Abbildung 6.1 grafisch veranschaulicht.

Dies sind die drei einfachsten Schemata zur Zeitdiskretisierung. In Kapitel 7 wird gezeigt werden, dass diese sich alle einer Gruppe von Verfahren zuordnen lassen, den sogenannten Runge-Kutta-Verfahren.

6.2 Wichtige Grundbegriffe

Bisher wurden drei numerische Verfahren zur Integration gewöhnlicher Differentialgleichungen vorgestellt. Neben Rundungsfehlern beinhalten alle diese Verfahren einen Diskretisierungsfehler, dessen lokales - also für jeden einzelnen Zeitschritt - asymptotisches Verhalten bekannt ist. Egal was wir anstellen, numerisch werden wir diese Fehler niemals los. Ein Ergebnis ist jedoch nur dann wertvoll, wenn eingeschätzt werden kann wie genau es ist ist. Um numerische Verfahren systematisch analysieren zu können, werden im Folgenden die Konzepte der Konvergenz, Stabilität und Konsistenz vorgestellt.

6.2.1 Konvergenz

Ziel der Konstruktion eines numerischen Verfahrens ist es, globale **Konvergenz** zu erzielen. Darunter versteht man, dass die numerische Lösung und die exakte Lösung der DGL für unendlich kleine Schrittweite zusammenfallen. Man kann zeigen:

Konsistenz zusammen mit Stabilität ist hinreichend und notwendig für die globale Konvergenz eines Verfahrens.

Diese beiden Konzepte werden im Folgenden erläutert.

6.2.2 Stabilität

Beim Konzept der Stabilität wird untersucht, wie sich eine (kleine) Abweichung in den Anfangsdaten auf die mit dem numerische Verfahren berechnete Lösung auswirkt. Es werden also zwei numerische Lösungen miteinander verglichen: f^n mit den Anfangsdaten f^0 und F^n mit den leicht gestörten Anfangsdaten $f^0+\epsilon^0$. Für jeden Zeitschritt erhält man also einen Fehler ϵ^n mit der Beziehung

$$F^n = f^n + \epsilon^n. \tag{6.12}$$

Bleibt der Fehler ϵ^n eines Verfahrens für alle Zeitschritte n konstant, so ist das Verfahren **neutralstabil**. Nimmt der Fehler von Zeitschritt zu Zeitschritt ab, so hat man es mit einem **stabilen** Verfahren zu tun. Nimmt er jedoch zu, so spricht man von einem **instabilen** Verfahren. Damit ein Verfahren für alle Zeiten und Anfangsbedingung stabil ist muss also

$$\left|\frac{\epsilon^{n+1}}{\epsilon^n}\right| < 1 \qquad \forall n \tag{6.13}$$

gelten. Für eine lineare, diskretisierte Gleichung muss nach dem Superpositionsprinzip auch der Fehler ϵ^n dieser genügen. Es gilt damit für das explizite Euler-Verfahren mit einer linearen rechten Seite g:

$$\epsilon^{n+1} = \epsilon^n + hg(t^{n+1}, \epsilon^{n+1}). \tag{6.14}$$

Hierzu soll ein einfaches Beispiel angeführt werden: Gegeben sei die Differentialgleichung

$$\frac{df}{dt} = -cf \qquad c > 0. \tag{6.15}$$

Diskretisiert man dieses Verfahren mittels des expliziten Euler-Verfahren, ergibt sich für den Fehler

$$\epsilon^{n+1} = \epsilon^n - hc\epsilon^n. \tag{6.16}$$

Damit folgt aus der Stabilitätsbedingung (6.13)

$$|1 - hc| < 1, \tag{6.17}$$

woraus sich $0 < h < 2/c$ als Stabilitätsbereich für die Zeitschrittweite ergibt. Das Verfahren ist also nur für Schrittweiten kleiner $2/c$ auf jeden Fall stabil.

Wir können diesen Sachverhalt auch mit Papier und Bleistift überprüfen. Hierzu nehmen wir für die Schrittweite einmal einen stabilen Schritt von $h_1 = 2/(3c)$, einmal einen neutral-stabilen von $h_2 = 2/c$ und einmal einen instabilen von $h_3 = 3/c$ an. Des Weiteren nehmen wir an, die Anfangsbedingung sei null und besäße den Fehler ϵ, d.h. $f_0 = \epsilon$. Damit ergibt sich für die folgenden Zeitschritte:

	f_0	f_1	f_2	f_3	...
$h_1 = \frac{2}{3c}$	ϵ	$\frac{\epsilon}{3}$	$\frac{\epsilon}{9}$	$\frac{\epsilon}{27}$	...
$h_2 = \frac{2}{c}$	ϵ	$-\epsilon$	ϵ	$-\epsilon$	...
$h_3 = \frac{3}{c}$	ϵ	-2ϵ	4ϵ	-8ϵ	...

Für die stabile Schrittweite nimmt der Fehler also mit jedem Zeitschritt weiter ab. Für den neutral-stabilen Schritt schwingt er hingegen für alle Zeitschritte zwischen $+\epsilon$ und $-\epsilon$. Für den instabilen Zeitschritt h_3 schwingt der Fehler ebenfalls, die Amplitude dieser Schwingung erhöht sich jedoch.

Führt man das gleiche für das implizite Euler-Verfahren durch, so ergibt sich

$$\left|\frac{\epsilon^{n+1}}{\epsilon^n}\right| = \left|\frac{1}{1+hc}\right|. \tag{6.18}$$

Da sowohl h als auch c größer null sind, ist dieses Verfahren für alle Zeitschrittweiten h stabil und konvergiert für beliebig große h gegen Null. Damit bietet das Verfahren eine sichere Möglichkeit DGLs zu lösen. Ein Nachteil ist jedoch der größere numerische Aufwand gegenüber dem expliziten Euler-Verfahren, da für jeden Zeitschritt eine u.U. nichtlineare Gleichung gelöst werden muss.

Zuletzt soll das Crank-Nicolson-Verfahren auf Stabilität überprüft werden. Für die gegebene DGL (6.15) ergibt sich folgende Stabilitätsbedingung:

$$\left|\frac{\epsilon^{n+1}}{\epsilon^n}\right| = \left|\frac{1-\frac{hc}{2}}{1+\frac{hc}{2}}\right| < 1. \tag{6.19}$$

Auch dieses Verfahren ist somit für dieses Problem für alle Schrittweiten stabil, für beliebig große h konvergiert der Betrag dieses Verhältnisses jedoch nicht - wie beim impliziten Euler-Verfahren - gegen Null, sondern gegen eins. Dies bedeutet im schlechtesten Fall einer (theoretisch) unendlich großen Schrittweite wird das Verfahren maximal neutral stabil.

Zur genauen Beurteilung der Stabilität von Verfahren wurden spezielle Kriterien entwickelt. Aufgrund der Einfachheit wurde die eben behandelte Differentialgleichung (6.15) als Referenzgleichung definiert. Je nachdem wie sich ein Verfahren für diese schlägt, wird es kategorisiert. Hierzu wird der bereits hergeleitete Faktor zur Fehlerfortpflanzug näher betrachtet (Gleichungen (6.18) und (6.19)). Ist dieser Quotient für alle Werte $hc > 0$ kleiner als eins, so bezeichnet man das Verfahren als **A-Stabil**. Ein Beispiel hierfür ist das Crank-Nicolson Verfahren. Störungen werden stetig gedämpft. Gilt zusätzlich, dass dieser Quotient für große Werte von hc gegen

null konvergiert, also $\lim_{hc\to\infty} \left|\epsilon^{n+1}/\epsilon^n\right| = 0$ gilt, dann bezeichnet man das Verfahren als **L-Stabil**. Zu dieser Klasse zählt das implizite Euler-Verfahren.

Wichtig anzumerken ist, dass die hier getroffenen Aussagen zunächst nur für lineare Differentialgleichungen gelten. Im nichtlinearen Fall ist es deutlich schwieriger verlässliche Stabilitätsaussagen zu treffen. Eine Möglichkeit die beschriebene Methode dennoch anwenden zu können besteht darin, die DGL zu linearisieren und anschließend diese Version auf Stabilität zu überprüfen.

Differentialgleichungen, die für die expliziten Verfahren zur Instabilität tendieren, werden als **steife Differentialgleichungen** bezeichnet.

Die Stabilitätsuntersuchung eines numerischen Verfahrens, basiert auf der Annahme einer exakten Rechenarithmetik. In der Realität gibt es jedoch zusätzlich in jedem Zeitschritt einen Rundungsfehler, der ebenfalls in der Zeit weiterpropagiert. Eine schlechte Kondition von eingesetzten Operationen kann diesen zusätzlich verstärken. Stabile Verfahren dämpfen diesen sich fortsetzenden Fehler, wohingegen instabile ihn weiter vergrößern.

6.2.3 Konsistenz

Wir wenden uns jetzt dem lokale Diskretisierungsfehler zu. Wie bereits erwähnt, muss in der Numerik eine Differntialgleichungen in ein System an Differenzengleichungen überführt werden. Eine Methode hierfür bezeichnet man als konsistent, falls für $h \to 0$ die diskretisierte gegen die analytische DGL konvergiert. Darüber hinaus lässt sich eine sogenannte Konsistenzordnung p definieren, die das asymptotische Konverganzverhalten beschreibt.

Diese soll nun am Beispiel der in Gleichung (6.15) gegebenen DGL veranschaulicht werden. Wendet man auf diese zur Diskretisierung das explizite Euler-Verfahren an, so ergibt sich:

$$\frac{f^{n+1} - f^n}{h} + cf^n = \mathcal{O}(h). \tag{6.20}$$

Die selbe Gleichung diskretisiert mit dem Crank-Nicolson-Verfahren führt auf

$$\frac{f^{n+1} - f^n}{h} + \frac{c}{2}\left(f^n + f^{n+1}\right) = \mathcal{O}\left(h^2\right). \tag{6.21}$$

Beide Methoden gehen für $h \to 0$ in die analytische Lösung über, da der Fehlerterm auf der rechten Seite verschwindet. Damit sind beide Methoden konsistent. Der Fehlerterm des Crank-Nicolson-Verfahrens verschwindet asymptotisch jedoch schneller ($\mathcal{O}(h^2)$) als der des Euler-Verfahrens ($\mathcal{O}(h)$). Die Ordnung dieses Verhaltens bezeichnet man als Konsistenzordnung p. Die Euler-Verfahren sind damit von der Ordnung $p = 1$, das Crank-Nicolson-Verfahren von der Ordnung $p = 2$.

Wie eingangs festgehalten, ist die Konsistenz zusammen mit der Stabilität äquivalent zur globalen Konvergenz eines Verfahrens. Wie eben gezeigt sind beide vorgestellte Verfahren für die Beispiel-DGL (6.15) konsistent. Des Weiteren sind sie bei Einhaltung der in Kapitel 6.2.2 hergeleiteten Bedingungen auch stabil. Damit konvergieren beide Verfahren innerhalb der Stabilitätsgrenzen auch global gegen die analytische Lösung.

6.3 Übungsaufgaben

In dieser Übung wird die zeitliche Entwicklung des in Übung 5 behandelten Problems betrachtet. Die Zeitableitung wird in der Leistungsbilanz nicht mehr zu Null gesetzt, wodurch sich folgende DGL ergibt:

$$\frac{dT}{dt} = \frac{1}{D\rho_S c_S}\left((\alpha_i + \alpha_a)(T_\infty - T) + \sigma\epsilon(T_S^4 - T^4)\right). \tag{6.1}$$

Die Dicke der Scheibe wird zu $D = 0.01\ m$ angenommen. Der Parameter ρ_S stellt die Dichte und c_S die spezifische Wärmekapazität des Scheibenmaterials dar. Diese gewöhnliche Differentialgleichung gilt es in der Zeit zu integrieren. Hierfür sollen die zwei einfachsten Verfahren zur Zeitdiskretisierung, das sogenannte explizite und implizite Euler-Verfahren sowie das Crank-Nicolson-Verfahren, eingesetzt werden.

Die Windschutzscheibe aus Übung 5 besitze zum Zeitpunkt t_0 eine Temperatur von $T_{ini} = 284\ K$. Die Stoffwerte sollen durch die von Quarzglas angenähert werden:

$\rho_S = 2.201e3\ kg/m^3$, $c_S = 1052\ J/(kgK)$.

Die übrigen Parameter können aus Aufgabe 5 übernommen werden.

6.1 Explizites Euler Verfahren:

6.1.1 Erstellen Sie ein Ablaufdiagramm für das explizite Euler-Verfahren, bevor Sie mit der Programmierung beginnen.

6.1.2 Berechnen Sie den instationären Verlauf der Temperaturentwicklung der Windschutzscheibe mit Hilfe eines expliziten Euler Verfahrens und plotten Sie das Ergebnis. Wie lange dauert es, bis sich ein stationärer Zustand einstellt?

- `[y_vec,t_vec] = EulerFW(g,t_1,t_2,y_ini,N)`
  - `y_vec`: Vektor mit y-Werten zu den Zeiten `t_vec`
  - `t_vec`: Vektor mit allen Zeitschritten
  - `g`: Funktions-Handle
  - `t_1,t_2`: Start- und Endzeit
  - `y_ini`: y-Startwert zur Zeit `t_1` (Anfangsbedingung)
  - `N`: Anzahl an Zeitschritten

6.1.3 Vergleichen Sie die stationäre Temperatur mit dem Ergebnis aus Übung 5. Was stellen Sie fest?

6.1.4 Nachdem Sie die stationäre Lösung gefunden haben, testen Sie was passiert, wenn Sie das Temperaturniveau erhöhen:

$$T_{S,2} = T_S + niveau \tag{6.2}$$
$$T_{\infty,2} = T_\infty + niveau \tag{6.3}$$
$$T_{ini,2} = T_{ini} + niveau \tag{6.4}$$

Wählen Sie hierzu eine Simulationszeit von $t_2 = 20000\ s$ und 50 Zeitschritte. Erhöhen Sie $niveau$ nun in 100er-Schritten, beginnend bei Null. Was beobachten Sie? Warum?

6.2 Implizites Euler Verfahren:

6.2.1 Berechnen Sie den instationären Verlauf der Temperaturentwicklung der Windschutzscheibe mit Hilfe eines impliziten Euler Verfahrens und plotten Sie das Ergebnis. Wie lange dauert es bis sich ein stationärer Zustand einstellt?

- `[y_vec,t_vec] = EulerBW(g,t_1,t_2,y_ini,N)`

6.2.2 Verwenden Sie zur Lösung Ihrer impliziten Zeitschrittbedingung Ihre in Übung 5 erstellte Funkion zur Nullstellensuche (z.B. Newton-Verfahren). Definieren Sie wieder ein function handle zur effizienten Übergabe Ihrer Schrittbedingung an die Funktion zur Nullstellensuche.

6.2.3 Nachdem Sie die stationäre Lösung gefunden haben, testen Sie auch für dieses Verfahren was passiert, wenn Sie das Temperaturniveau wie oben beschrieben schrittweise erhöhen. Was beobachten Sie nun? Warum?

6.3 Crank-Nicolson Verfahren:

6.3.1 Berechnen Sie den instationären Verlauf der Temperaturentwicklung der Windschutzscheibe mit Hilfe eines Crank-Nicolson Verfahrens und plotten Sie das Ergebnis. Wie lange dauert es bis sich ein stationärer Zustand einstellt?

- `[y_vec,t_vec] = CrankNicol(g,t_1,t_2,y_ini,N)`

6.3.2 Nachdem Sie die stationäre Lösung gefunden haben, testen Sie auch für dieses Verfahren was passiert, wenn Sie das Temperaturniveau wie oben beschrieben schrittweise erhöhen. Was beobachten Sie nun? Warum?

Hinweise:

- Beim Crank-Nicolson-Verfahren handelt es sich um ein implizites Verfahren. Um am schnellsten zur Lösung zu kommen, verwenden Sie Ihre Funktion für das implizite Euler Verfahren und passen es entsprechend an.

6.4 Nun soll das numerische Ergebnis verifiziert werden. Ein wichtiger Punkt ist hierbei der Einfluss der Zeitdiskretisierung. Zunächst muss ein geeignetes Konvergenzkriterium definiert werden. Wir sind an einem möglichst globalen Kriterium interessiert. Daher soll mit Hilfe von $N_1 + 1$ über den Rechenzeitraum gleichverteilten Punkten (entspricht N_1 Intervallen) der mittlere quadratische Unterschied zwischen zwei Diskretisierungen betrachtet werden:

$$\Delta_{conv}(N_1) = \frac{1}{N_1 + 1} \sum_{n=0}^{N_1} \left(f_{N_1}(t_n) - f_{N_2}(t_n) \right)^2 \ , \quad t_n = n\, h_{N_1} \ . \tag{6.5}$$

Hierbei bezeichnet $f_{N_1/N_2}(t_n)$ den Wert von f zum Zeitpunkt t_n berechnet mit $N_1 + 1$ bzw. $N_2 + 1$ Stützstellen, wobei $N_2 = 2N_1$ gelten soll. Die Zeitschrittweite h_{N_1} bezieht sich dabei auf die Diskretisierungen N_1. Plotten Sie für jedes Ihrer drei Verfahren den Wert des Konvergenzkriteriums Δ_{conv} über der Anzahl an Stützstellen N_1. Beginnen Sie mit $N_1 = 20$ und verdoppeln Sie N_1 sechs Mal. Was stellen Sie fest?

Hinweise:

- Machen Sie sich zunächst klar warum $N_1 + 1$ Stützstellen zu N_1 Intervallen führen.
- Verdoppeln Sie die Anzahl an Intervallen, so bekommen Sie $2N_1 + 1$ Stützstellen. Die alten N_1 Stützstellen sind eine Teilmenge der $2N_1$ neuen Stützstellen. Vollziehen Sie diesen Zusammenhang grafisch nach.

6.5 Nach welcher Zeit müssen Sie also mit einer Vereisung der Winschutzscheibe rechnen, wenn die Taupunkttemperatur wieder $T_{tau} = 271\ K$ betrage? Lösen Sie diese Aufgabe grafisch und verwenden Sie eine Ihrer Meinung nach konvergierte Lösungskurve (Aufgabe 6.4).

6.4 Wichtige Matlab Befehle

Befehl	Beschreibung
`ode45()`	MATLAB-interne Funktion zum Lösen nichtsteifer DGLs.

7

Runge-Kutta-Verfahren mit adaptiven Zeitschritt

Literaturverzeichnis

[1] Brian P Flannery, William H Press, Saul A Teukolsky und William Vetterling. *Numerical Recipes in C*. Kap. 16. 1992. URL: http://www.nr.com/.

[2] Wolfgang Polifke und Jan Kopitz. *Wärmeübertragung: Grundlagen, analytische und numerische Methoden*. Kap. 13.5. Pearson Deutschland GmbH, 2009.

Lernziele

- Kennenlernen der Familie der Runge-Kutta-Verfahren
- Butcher-Tableau
- Adaptive Zeitschrittweitensteuerung mittels eingebetteter Verfahren

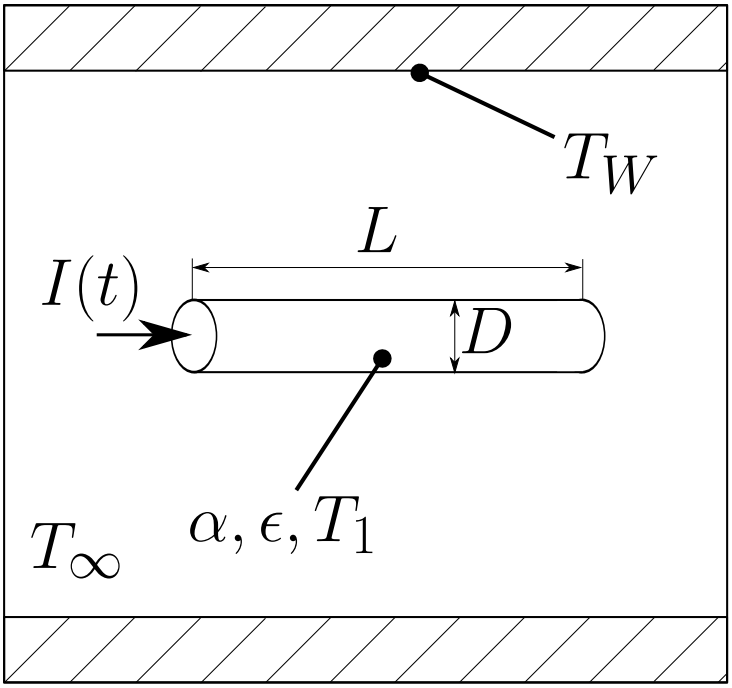


Abbildung 7.1: Skizze des betrachteten Beispiels: Ein stromdurchflossener Draht in einem großen Rohr.

Motivation

Im Kapitel 6.1 wurden bereits drei Verfahren zur Integration gewöhnlicher Differentialgleichungen vorgestellt. In der Praxis werden jedoch häufig Verfahren höherer Ordnung verwendet. Eine Verfahrensfamilie stellen die sogenannten Runge-Kutta-Verfahren dar, zu der auch die bereits betrachten Methoden (Euler expizit/ implizit, Crank-Nicolson) gehören. Aus dieser könne Verfahren fast beliebiger Ordnung gewonnen werden.

In der Praxis wird darüber hinaus fast immer eine variable Zeitschrittweite eingesetzt, da sich auf diese Weise der Fehler kontrollieren und ggf. auch Rechenzeit einsparen lässt. Der lokale Fehler wird dabei durch die Differenz aus einem Verfahren m-ter und $(m+1)$-ter Ordnung abgeschätzt. Ziel des Algorithmus ist es nun den größtmöglichen Zeitschritt zu finden, der dennoch die gewählte Genauigkeitsanforderung erfüllt. Diese beiden wichtigen Konzepte werden in diesem Kapitel vorgestellt.

Bevor in die hierzu gehörige Theorie eingeführt wird, soll das Vorgehen zunächst anhand eines Beispiels motiviert werden. Betrachtet wird ein von einem Strom $I(t)$ durchflossener Konstantan-Draht der Dicke D und der Länge L (Abbildung 7.1). Dieser befindet sich in einem im Vergleich zur Drahtdicke großem Rohr der Temperatur T_W. Die Luft im Rohr besitze die Temperatur T_∞. Der Draht wird also vom Stromfluss erhitzt und steht im thermischen Austausch mit der Umgebung, wobei lediglich die Wärmeübertragung durch Strahlung und Konvektion betrachtet werden soll. Hierfür sei der Wärmeübergangskoeffizient α bekannt, der sich aufgrund von freier Konvektion ergibt (siehe hierzu auch [2]). Zur Berechnung des

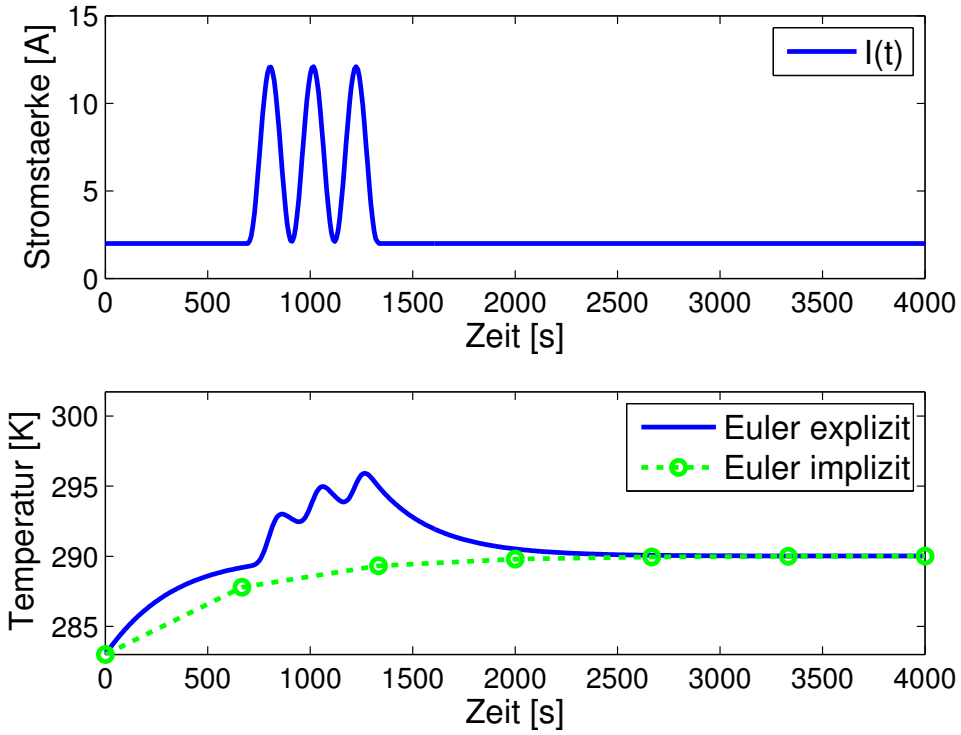


Abbildung 7.2: Zeitlicher Verlauf der Stromstärke im Draht sowie die dazu korrespondierende Temperaturänderung des Drahtes, jeweils berechnet mit einem expliziten/ impliziten Euler-Verfahren. Der Zeitschritt des expliziten Verfahrens ist dabei wesentlich geringer als der des impliziten!

durch Strahlung übertragenen Wärmestroms können diffus graue Strahler angenommen und der Austauschkoeffizient kann durch $\sum_{12} = \epsilon\sigma$ approximiert werden. σ ist hierbei die Stefan-Boltzmann-Konstante und ϵ der Emissionsgrad des Leiters. Unter Verwendung dieser Annahmen ergibt sich folgende DGL für die Temperatur T des Drahtes:

$$\frac{dT}{dt} = \frac{4}{D\rho c}\left[\alpha\left(T_\infty - T\right) + \epsilon\sigma\left(T_W^4 - T^4\right)\right] + \frac{4P_{V,el}}{\pi D^2 L\rho c}. \tag{7.1}$$

Hierin gibt ρ die Dichte und c die spezifische Wärmekapazität von Konstantan an. $P_{V,el}$ ist die elektrische Verlustleistung, welche den Draht erhitzt. Diese kann näherungsweise durch

$$P_{V,el} = RI(t)^2 = \rho_W \frac{4L}{\pi D^2} I(t)^2 \tag{7.2}$$

berechnet werden. R steht für den elektrischen Widerstand, ρ_W entsprechend für den elektrischen spezifischen Widerstand von Konstantan. Die Zahlenwerte aller Parameter und Stoffgrößen sind in der Übungsaufgabe zu diesem Kapitel gegeben.

Dieses System soll nun simuliert werden. Zum Zeitpunkt $t = 0$ besitze der Draht die Umgebungstemperatur T_∞. Wie schon angedeutet, wird die Stromstärke hierfür nicht als gleich-

förmig angenommen. Vielmehr soll diese zunächst die ersten 700 Sekunden konstant $2\,\mathrm{A}$ betragen. Anschließend wird für ca. $630\,\mathrm{s}$ ein harmonisch um $7\,\mathrm{A}$ schwankender Verlauf angenommen bevor dieser wieder konstant $2\,\mathrm{A}$ beträgt. Die transiente Änderung von $\mathrm{I}(t)$ sowie der per expliziten/ impliziten Euler-Verfahren simulierte Verlauf der Drahttemperatur ist in Abbildung 7.2 gezeigt.

Es wird deutlich, dass sich der Draht zunächst aufgrund des per Strahlung übertragenen Wärmestroms von der heißeren Rohrwand sowie aufgrund der elektrischen Verlustleistung langsam erhitzt. Bei $t = 700\,\mathrm{s}$ beginnt die Stromstärke zu oszillieren. Im Resultat erhitzt sich der Draht, wobei die Temperaturänderung in diesem Bereich ebenfalls schwingt. Ab ca. $t = 1330\,\mathrm{s}$ weist der Strom wieder konstant $2\,\mathrm{A}$ auf. Entsprechend kühlt sich der Draht wieder ab und strebt gegen die Gleichgewichtstemperatur von ca. $290\,\mathrm{K}$.

Um den transienten Verlauf der Drahttemperatur abbilden zu können, wurde für das explizite Euler-Verfahren ein relativ kleiner Zeitschritt verwendet. Dieser ist jedoch nur im Bereich von $700\,\mathrm{s} < t < 1330\,\mathrm{s}$ erforderlich, da die Temperatur hier relativ starke Änderungen aufweist. Im übrigen Bereich könnte ein größerer Zeitschritt angewandt und somit Rechenzeit gespart werden. Wir kennen jedoch bisher keine Möglichkeit den Zeitschritt zur Laufzeit entsprechend der Lösung zu variieren. Genau hier setzen die in diesem Kapitel vorgestellten Verfahren mit adaptivem Zeitschritt an: Diese passen den Zeitschritt an den lokalen Verlauf der Lösung an und sparen somit Rechenzeit in Bereichen geringer Änderung ohne in Bereichen großer Änderung an Genauigkeit zu verlieren. Bevor diese Verfahren jedoch eingeführt werden können, muss zunächst die Familie der sogenannten Runge-Kutta-Verfahren - auf der die hier gezeigten adaptiven Verfahren beruhen - behandelt werden. Aufgrund dieser Tatsache ergibt sich die Zweiteilung dieses Kapitels.

Zuletzt sei noch auf den mit Hilfe des impliziten Euler-Verfahren bestimmten Temperaturverlauf in Abbildung 7.2 hingewiesen. Der Zeitschritt wurde hier wesentlich größer gewählt als für das explizite Verfahren - insofern können die Ergebnisse qualitativ nicht verglichen werden. Bemerkenswert ist jedoch, dass das implizite Verfahren, wie in Kapitel 6.2.2 gezeigt, auch für sehr große Zeitschritte stabil ist. Für derartige Zeitschritte kann zwar keine Aussage bzgl. des Zeitverlaufes der Temperatur getroffen werden, der stationäre Endzustand jedoch wird korrekt wiedergegeben. Auf diese Weise kann also mit relativ wenig Rechenaufwand der stationäre Wert bestimmt werden (in diesem Fall als Alternative zum Lösen der nichtlinearen stationären Gleichung).

7.1 Verfahren zur Zeitdiskretisierung II

Bisher wurden drei Verfahren behandelt: Das explizite und das implizite Euler-Verfahren sowie das Crank-Nicolson-Verfahren. Diese lassen sich auf drei verschiedene Arten veranschaulichen:

- Alle Verfahren können als Näherungen für das Integral aus Gleichung (6.11) betrachtet werden (siehe auch Abbildung 6.1).
- Die Euler-Verfahren können als mit den in Kapitel 2 vorgestellten Methoden diskretisierte Gleichung gesehen werden.

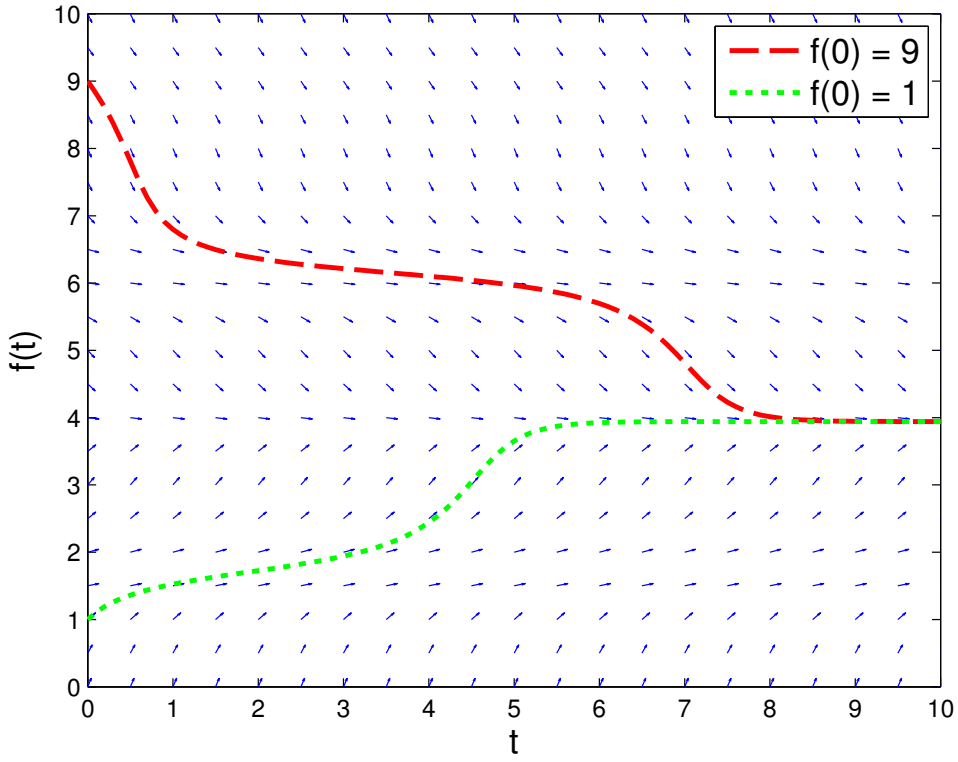


Abbildung 7.3: Darstellung des durch Gleichung (7.3) definierten Gradientenfeldes sowie zweier Lösungstrajektorien für die zwei Anfangsbedingungen $f(0) = 1$ und $f(0) = 9$.

- Die rechte Seite $g^n = g(t^n, f^n)$ kann als Tangente an die Funktion $f(t)$ zum Zeitpunkt t^n betrachtet werden.

Nach der letzt genannten, geometrischen Deutungsmöglichkeit bedeutet dies, dass sich die Lösungskurve einer DGL darin auszeichnet, dass für jeden Punkt auf ihr mittels einer Auswertung der rechten Seite $g(f, t)$ eine Tangente an diese gefunden werden kann. Man könnte die Lösungskurve als Stromlinie betrachten und das durch $g(f, t)$ erzeugte Feld als Strömungsfeld. Um dies zu verdeutlichen soll beispielhaft folgende (frei erfundene) rechte Seite betrachtet werden:

$$g(f, t) = g(f) = \cos(2f) - 0.5f + 2. \tag{7.3}$$

Da diese Funktion nicht explizit von t abhängt, handelt es sich um eine sogenannte *autonome* DGL. Mit Hilfe dieser Funktion lässt sich nun zum einen genanntes 2D-Gradientenfeld und zum anderen für eine Anfangsbedingung $f(0)$ die Lösungskurve der zu Gleichung (7.3) gehörigen DGL bestimmen. Beides ist in Abbildung 7.3 gezeigt. Es wurden hierin zwei Lösungskurven für die Anfangsbedingungen $f(0) = 1$ und $f(0) = 9$ mittels eines expliziten Euler-Verfahrens ausgewertet.

Die Lösungskurve entspricht damit gerade jener Trajektorie die ein masseloser Partikel eingebracht am Ort 1 bzw. 9 zurücklegen würde, wenn man das Gradientenfeld als Strömungsfeld

betrachtet. Beide Kurven streben gegen den Wert 3.9416. Man bezeichnet die Kurve $f(t) = 3.9416$ auch als Attraktor, da dieser alle Lösungskurven in seiner Umgebung anzieht. Dies ist der Fall, da der Gradient der Lösung auf dieser Kurve nach Gleichung (7.3) null sein muss. Es handelt sich dabei also um eine stabile stationäre Lösung des Systems.

Ziel aller numerischer Verfahren zur Integration von Differentialgleichungen ist es nun den in Abbildung 7.3 gezeigten Weg zu finden. Hierfür kann die rechtes Seite $g(t, f)$ lediglich zu diskreten Zeitpunkten ausgewertet werden. Um den Rechenaufwand zu minimieren, sollte diese Auswertung so selten wie möglich geschehen. Das explizite Euler-Verfahren versucht den Verlauf durch den Gradienten der Lösung am momentanen Punkt abzuschätzen. Es wird also eine lineare Extrapolation durchgeführt. Das implizite Euler-Verfahren sucht einen Punkt zum nächsten Zeitschritt, dessen durch g^{n+1} definierte Tangente den Punkt zum aktuellen Zeitschritt trifft. Dies ist für beide Verfahren in Abbildung 7.4 gezeigt.

Ebenfalls in dieser Abbildung ist die sogenannte Mittelpunktregel eingetragen. Hierbei wird der neue Funktionswert in zwei Stufen erzeugt: Zunächst wird, wie im Euler-Verfahren, anhand der Tangente ein Funktionswert in der Mitte des Intervalls bei $t^n + h/2$ bestimmt. Hierzu wird zunächst die Zunahme der Funktion f^n über diesem Intervall bestimmt (k_1) und dann daraus der gesuchte Wert $f^{n+1/2}$:

$$k_1 = hg(t^n, f^n) \tag{7.4}$$

$$f^{n+1/2} = f^n + \frac{1}{2}k_1 \tag{7.5}$$

An diesem Punkt kann nun die rechte Seite der DGL ausgewertet werden, woraus sich (näherungsweise) die Tangente an die Funktion in der Mitte des Intervalls ergibt. Diese Tangente wird in einem zweiten Schritt dafür verwendet, die Änderung der Funktion f^n über dem ganzen Intervall zu bestimmen (k_{RK}). Der gesuchte Funktionswert zum Zeitschritt t^{n+1} ergibt sich dann aus genau diesem Wert:

$$k_{RK} = hg\left(t^n + \frac{h}{2}, f^{n+1/2}\right) = hg\left(t^n + \frac{h}{2}, f^n + \frac{1}{2}k_1\right) \tag{7.6}$$

$$f^{n+1} = f^n + k_{RK} \tag{7.7}$$

Es wurde also ein Testschritt unternommen, um zu sehen, wie sich die Ableitung entwickelt. Der dabei erhaltene Wert wird dann für den gesamten Schritt verwendet. Ist für das Euler-Verfahren der lokale Fehlerterm noch von der Ordnung $\mathcal{O}\left(h^2\right)$ (Konsistenzordnung 1), so verringert sich dieser für die Mittelpunktregel auf $\mathcal{O}\left(h^3\right)$ (Konsistenzordnung 2).

Die Konsistenzordnung des Verfahrens lässt sich weiter erhöhen, indem mehrere Testschritte (auch an anderen Stellen als an der Intervallmitte) durchgeführt und an diesen Stellen schließlich die Tangenten ermittelt werden (also die rechten Seiten evaluiert werden). Der finale Schritt kann dann aus einer geschickten Linearkombination dieser Tangenten ermittelt werden. Geschickt bedeutet in diesem Fall, dass der finale Schritt z.B. derartig konstruiert wird, dass eine möglichst hohe Konsistenzordnung erzielt wird. Pro Evaluierung der rechten Seite spricht man

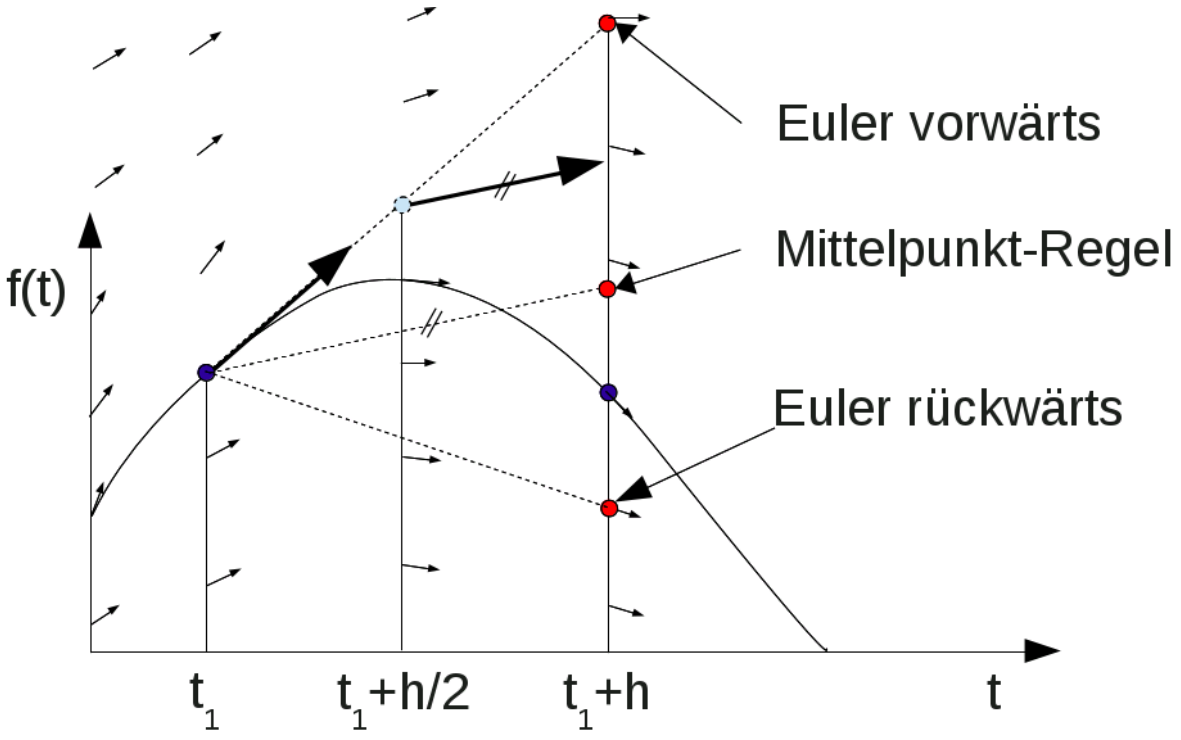


Abbildung 7.4: Veranschaulichung des expliziten Euler-Verfahrens (1 Stufen Runge-Kutta) und der Mittelpunktregel (2 Stufen Runge-Kutta).

dabei von einer *Stufe*. Das Euler-Verfahren ist demnach ein Runge-Kutta-Verfahren der Stufe eins und die Mittelpunktregel eines der Stufe zwei.

Ein Beispiel mit vier Stufen und der Konsistenzordnung vier ist das sogenannte **klassische Runge-Kutta-Verfahren**. Hierbei wird zunächst ein Testschritt zur Intervallmitte durchgeführt und mit der dabei erhaltenen Tangente ein weiterer Schritt zur Intervallmitte getätigt. Es wird dort abermals die rechte Seite ausgewertet und mit diesem Wert dann ein Schritt über das gesamte Intervall durchgeführt, wo letztmals die rechte Seite evaluiert wird. Aus diesen vier Schritten wird dann mittels einer gewichteten Summe der sogenannte Runge-Kutta-Schritt zur finalen Bestimmung des Funktionswertes zum Zeitpunkt t^{n+1} berechnet. In Formeln ausgedrückt ergibt sich:

$$\begin{aligned} k_1 &= hg(t^n, f^n) \\ k_2 &= hg(t^n + h/2, f^n + 1/2 \cdot k_1) \\ k_3 &= hg(t^n + h/2, f^n + 1/2 \cdot k_2) \\ k_4 &= hg(t^n + h, f^n + k_3) \\ f^{n+1} &= f^n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{aligned} \tag{7.8}$$

Hierbei ist auf die korrekte Definition der k-Werte zu achten. Egal wie groß die Schrittweite eines Testschrittes war, die rechte Seite wird immer mit der vollen Intervalllänge h multipliziert.

Die Schrittweite wird erst bei der Ermittlung der folgenden Stufe berücksichtigt (Vorfaktor der k_i). Eine klare Konvention ist wichtig, da es eine große Anzahl an Runge-Kutta-Methoden gibt und diese alle standardisiert in sogenannten Butcher-Tableaus notiert werden.

Das Butcher-Tableau

Allen Runge-Kutta-Verfahren ist gemein, dass sie Testschritte unternehmen, anhand derer, durch geschickte Wahl von Gewichtungen, ein möglichst guter Gesamtschritt konstruiert wird. Eine Möglichkeit, alle möglichen derartigen Methoden darzustellen ist das **Butcher-Tableau**. Um dieses zu verstehen, wird im Folgenden eine allgemeine Form für Runge-Kutta-Verfahren angegeben. Alle oben genannten Verfahren sind Spezialfälle davon.

Die Grundgleichung ist dabei

$$f^{n+1} = f^n + \sum_{i=1}^{s} b_i k_i = f^n + k_{RK}. \tag{7.9}$$

Hierbei ist s die Stufenzahl des jeweiligen Verfahrens und k_{RK} der Runge-Kutta-Schritt, den es zu bestimmen gilt. Dieser setzt sich aus den s Teilschritten k_i zusammen, die jeweils mit einer Gewichtung b_i multipliziert werden. Des Weiteren muss gespeichert werden, wie groß die einzelnen Testschritte sind. Diese können alle eine unterschiedliche Länge besitzen (sie müssen nicht zwingend der halben Intervalllänge entsprechen wie oben). Dies wird in einem Vektor c gespeichert. Abhängig davon ergeben sich die Vorfaktoren der k_is. Sie werden mit a_{ij} bezeichnet und in einer Matrix A gespeichert. Es ergibt sich folgendes Gleichungssystem:

$$\begin{aligned}
k_1 &= hg(t^n + c_1 h, f^n + a_{11}k_1 + a_{12}k_2 + \ldots + a_{1,s}k_s) \\
k_2 &= hg(t^n + c_2 h, f^n + a_{21}k_1 + a_{22}k_2 + \ldots + a_{2,s}k_s) \\
k_3 &= hg(t^n + c_3 h, f^n + a_{31}k_1 + a_{32}k_2 + \ldots + a_{3,s}k_s) \\
&\vdots \\
k_s &= hg(t^n + c_s h, f^n + a_{s1}k_1 + a_{s2}k_2 + \cdots + a_{s,s}k_s)
\end{aligned} \tag{7.10}$$

Alle diese Größen lassen sich in dem genannten Butcher-Tableau zusammenfassen, welches die Form

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

aufweist. Ausgeschrieben ergibt sich:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \ldots & a_{1s} \\ c_2 & a_{21} & a_{22} & \ldots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \ldots & a_{ss} \\ \hline & b_1 & b_2 & \ldots & b_s \end{array}$$

Ein Runge-Kutta-Verfahren ist nur dann konsistent, wenn gilt

$$\sum_{i=1}^{s} b_i = 1. \tag{7.11}$$

Zusätzlich gilt für in den Ingenieurwissenschaften übliche Verfahren

$$\sum_{j=1}^{s} a_{ij} = c_i. \tag{7.12}$$

Diese Bedingung gilt auf jeden Fall dann, wenn eine *autonome* Differentialgleichung vorliegt, d.h. wenn die rechte Seite keine explizite Funktion von t ist, sich also als $g(f)$ darstellen lässt. Wichtig zu wissen ist außerdem, dass für explizite Verfahren alle Koeffizienten a_{ij}, für die gilt $j \geq i$, null sind. Nur Einträge unterhalb der Diagonalen von A sind damit ungleich null. Dies ergibt sich bei Betrachtung des Gleichungssystems (7.10). Wird z.B. zur Bestimmung von k_2 auch k_3 verwendet, ist also a_{23} ungleich null, so taucht ein noch unbekannter Wert in der Bestimmungsgleichung von k_2 auf. Damit können nicht alle s Gleichungen der Reihe nach explizit gelöst werden. Butcher-Tableaus **expliziter Verfahren** haben damit die Form:

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
$\vdots$	$\vdots$	$\vdots$	$\ddots$		
c_s	a_{s1}	a_{s2}	$\dots$	$a_{s,s-1}$	
	b_1	b_2	$\dots$	b_{s-1}	b_s

Ziel dieses Praktikums ist es nicht, Runge-Kutta-Verfahren herzuleiten. Vielmehr sollen Sie lediglich in die Lage versetzt werden, anhand eines gegebenen Butcher-Tableaus das dazu korrespondierende Verfahren zu implementieren. Um diese Fähigkeit zu entwickeln, werden im Folgenden Beispiele zu einigen wichtigen Verfahren gegeben.

Beispiele

- Explizites Euler-Verfahren

$$f^{n+1} = f^n + hg^n$$

0	0
	1

- Implizites Euler-Verfahren

$$f^{n+1} = f^n + hg^{n+1}$$

1	1
	1

- Crank-Nicolson-Verfahren

$$f^{n+1} = f^n + \frac{h}{2}\left(g^n + g^{n+1}\right)$$

0	0	0
1	1/2	1/2
	1/2	1/2

- Mittelpunktregel

$$f^{n+1} = f^n + hg^{n+1/2}$$

0	0	0
1/2	1/2	0
	0	1

- Klassisches Runge-Kutta-Verfahren

$$\begin{aligned} k_1 &= hg(t^n, f^n) \\ k_2 &= hg(t^n + h/2, f^n + 1/2 \cdot k_1) \\ k_3 &= hg(t^n + h/2, f^n + 1/2 \cdot k_2) \\ k_4 &= hg(t^n + h, f^n + k_3) \\ f^{n+1} &= f^n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{aligned}$$

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

Das klassische Runge-Kutta-Verfahren ist ein Runge-Kutta-Verfahren der Stufe und der Ordnung vier. Verfahren dieser Stufe sind in der Praxis die am weit verbreitetsten Verfahren. Ein Grund hierfür lässt sich ableiten, wenn man die maximal mögliche Ordnung eines Verfahrens in Abhängigkeit seiner Stufe betrachtet. Eine Gegenüberstellung hiervon ist in Tabelle 7.1 gezeigt.

Es wird ersichtlich, dass die mögliche Konsistenzordnung nur bis zur vierten Stufe proportional zur Stufe ansteigt. Ein Verfahren der Stufe fünf kann keine Verbesserung bzgl. der Ordnung

Stufe	1	2	3	4	5	6	7
max. Ordnung	1	2	3	4	**4**	**5**	**6**

Tabelle 7.1: Maximal mögliche Konsistenzordnung eines expliziten Runge-Kutta-Verfahrens angegeben in Abhängigkeit der Stufenzahl.

bringen. Zieht man nun in Betracht, dass jede zusätzliche Stufe Rechenzeit benötigt, wird klar, dass sich ein Verfahren höherer Ordnung nur dann lohnt, wenn bei gleicher Genauigkeit ein größerer Schritt h realisiert werden kann. Die Mittelpunktregel ist z.B. ein Verfahren der Stufe zwei, es muss also pro Zeitschritt zwei Mal die Funktion $g(t, f)$ ausgewertet werden. Ein Verfahren vierter Ordnung benötigt vier solcher Evaluationen. Damit kann dieses nur effizienter sein, wenn mindestens ein doppelt so großer Zeitschritt ausgeführt werden kann ohne an Genauigkeit zu verlieren. Dies ist in vielen Fällen gegeben, allerdings nicht zwingend der Fall.

Für Verfahren mit mehr als vier Stufen wird pro zusätzlicher Stufe nicht immer eine bessere Ordnung des Fehlers erzielt. Es kann also sein, dass trotz mehr Evaluationen, also mehr Rechenaufwand, keine Verbesserung in der Genauigkeit erzielt werden kann. In Kapitel 7.2 werden dennoch solche Verfahren vorgestellt. Hier wird die höhere Ordnung allerdings zum Abschätzen des lokalen Fehlers verwendet, was letztlich der Bestimmung einer optimalen Zeitschrittweite h dient. In der Praxis erweisen sich derartige Verfahren durchaus als effizient.

7.2 Runge-Kutta-Verfahren mit adaptivem Zeitschritt

Bisher ist der Zeitschritt auf einen fixen Wert festgelegt. Je glatter die Lösung ist (entspricht kleiner Krümmung), desto besser kann sie von den oben beschriebenen Verfahren erkundet werden. Der Fehler pro Zeitschritt ist hier also sehr klein. An dieser Stelle kann die Performance erhöht werden, ohne dass die Genauigkeitsanforderungen verletzt werden: Wird das Verfahren in Gebieten mit glattem Verlauf genauer als ein vorher festgelegter Wert ϵ, so kann der Zeitschritt erhöht werden. In Gebieten jedoch, die größere Veränderungen in $f(t)$ aufweisen, wird der Zeitschritt wieder solange verkleinert bis die geforderte (lokale) Genauigkeit erreicht wird. Hierbei können zwei Probleme auftreten: (1) Die Schrittweite kann unter Umständen so klein werden, dass der Algorithmus auf der Stelle tritt, da die Änderung über einen Zeitschritt aufgrund von Rundungsfehlern als Null betrachtet wird. (2) Umgekehrt kann es auch passieren, dass die Schrittweite zu groß gewählt wird. Der Funktionsverlauf lässt sich damit nun nicht mehr verlässlich abschätzen und das berechnete Maß für die Genauigkeit wird somit wertlos.

Um ein solches Verfahren effizient und stabil zu implementieren, müssen folgende drei Fragen beantwortet werden:

- Wie kann der lokale Fehler, bzw. die lokale Krümmung, effizient abgeschätzt werden?
- Wie kann der maximal zulässige lokale Fehler festgelegt werden?
- Wie kann der Zeitschritt in Abhängigkeit dieser Abschätzung angepasst werden?

Abschätzung des lokalen Fehlers

Der Fehler eines Runge-Kutta-Verfahrens ist umso größer je weniger glatt der gesuchte Funktionsverlauf ist. Verfahren höherer Ordnung können gekrümmten Bereichen besser folgen als Verfahren niedriger Ordnung. Aus diesem Grund kann die Differenz der Runge-Kutta Schritte

eines Verfahrens der Konsistenzordnung p_1 und eines der Ordnung $p_2 < p_1$ als Maß für die lokale Krümmung und damit für den Fehler interpretiert werden.

Zur effizienten Implementierung von zwei Verfahren unterschiedlicher Konsistenzordnung wurden von dem deutschen Mathematiker Erwin Fehlberg sogenannte eingebettete Methoden eingeführt. Wie oben beschrieben, ist für ein sechsstufiges (explizites) Runge-Kutta-Verfahren maximal ein Verfahren der Ordnung fünf erzielbar. Dies bedeutet, ein Verfahren niedriger Ordnung als fünf ist mit sechs Stufen ebenfalls realisierbar. Es können nun die Koeffizienten derart gewählt werden, dass bei Verwendung der gleichen Matrix A und eines identischen Vektors c, für einen Vektor b_1 ein Verfahren fünfter Ordnung und für einen Vektor b_2 ein Verfahren vierter Ordnung entsteht. Man sagt in einem solchen Fall, in das Verfahren der Ordnung fünf ist ein Verfahren der Ordnung vier eingebettet. Das zugehörige Butcher-Tableau hat die Form:

$$\begin{array}{c|c} c & A \\ \hline & b_1^T \\ & b_2^T \end{array}$$

Mit einem derartigen Schema kann der vorliegende Fehler als Differenz der Verfahren verschiedener Ordnung abgeschätzt werden:

$$\Delta_{err,ist} = \left\| f_{\text{5.Ord}}^{n+1} - f_{\text{4.Ord}}^{n+1} \right\|_2 = \| (b_1 - b_2) \bullet k \|_2 \tag{7.13}$$

Hierin bezeichnet k einen Vektor, dessen i-tes Element k_i ist, und der Operator $\bullet$ kennzeichnet das Skalarprodukt.

Alte Implementierungen bestimmen den auszuführenden Runge-Kutta-Schritt durch Verwendung des Verfahrens vierter Ordnung ($f_{\text{4.Ord}}^{n+1}$), denn für diese wurde der Fehler nach Gleichung (7.13) abgeschätzt. Heute ist es allerdings gebräuchlich mit dem Verfahren höherer Ordnung zu rechnen ($f_{\text{5.Ord}}^{n+1}$), da die Fehlerabschätzung ungenau ist und zudem keine Informationen über den globalen Fehler liefert. Sie kann als Maß für die lokale Krümmung betrachtet werden und dient lediglich der Anpassung der Schrittweite an den lokalen Funktionsverlauf $f(t)$. Von dieser Anpassung profitiert auch das Verfahren höherer Ordnung.

Neben der klassischen Fehlberg-Methode wurden weitere Verfahren entwickelt, welche verschiedene Verbesserungen enthalten. Neuere Verfahren wurden z.B. dahingehend entwickelt, den Fehler der Lösung fünfter statt vierter Ordnung zu minimieren und den führenden Fehlerterm der Lösung vierter Ordnung dominanter gegenüber den anderen Termen zu machen. Ein bekanntes Verfahren, welches auch von MATLAB verwendet wird, wurde 1980 von Dormand und Prince entwickelt [1].

Definition des maximal zulässigen Fehlers

Somit steht uns eine Abschätzung des lokalen Fehlers für eine gegebe Zeitschrittweite zur Verfügung. Diese Abschätzung muss nun mit einem maximal zulässigen Fehler verglichen werden, um zu entscheiden ob die gewählte Zeitschrittweite veringert werden muss oder ob die geforderte Toleranz bereits eingehalten wird.

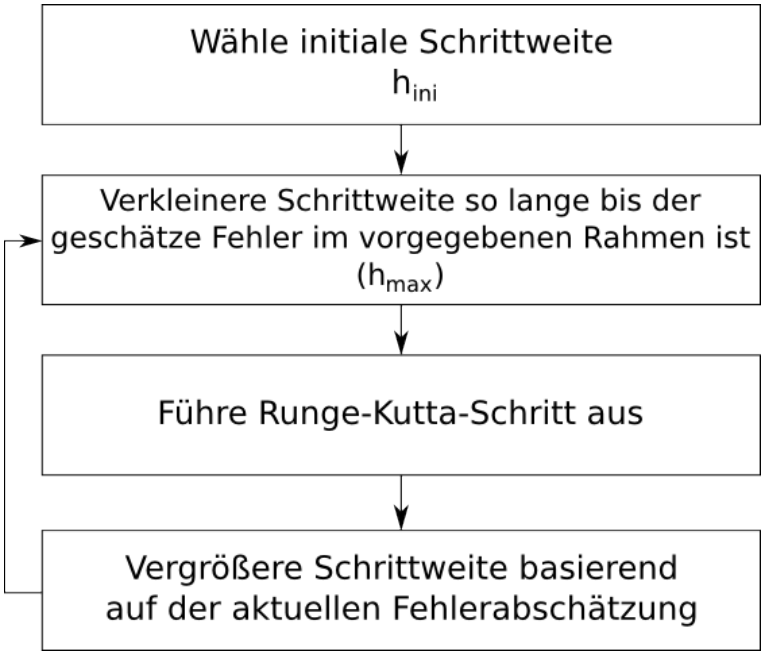


Abbildung 7.5: Grundprinzip der adaptiven Schrittweitensteuerung.

Bei der Definition des maximal zulässigen Fehlers sollte beachtet werden, dass es einen Unterschied bzgl. der geforderten Genauigkeit macht von welcher Größenordnung die gesuchte Funktion f ist. Aus diesem Grund ist es sinnvoll einen relativen Fehler zu definieren. Dieser besteht aus einer Skalierung f_{skal} und einer relativen Toleranz ϵ_r. Zur Skalierung kann z.B. der Betrag des aktuelle f-Wertes verwendet werden. Damit ergibt sich:

$$\Delta_{err,max} = \epsilon_r f_{skal} = \epsilon_r |f| \tag{7.14}$$

Auf diese Weise wurde ein Kriterium aufgestellt, für das es keine Rolle spielt ob f von der Größenordnung Giga oder Mikro ist. Besitzt f einen Nulldurchgang, so würde dieses Verfahren jedoch eine beliebig kleine Schrittweite vorschlagen. Aus diesem Grund wird zusätzlich eine absolute Toleranz ϵ_a definiert:

$$\Delta_{err,max} = \epsilon_r |f| + \epsilon_a. \tag{7.15}$$

Anpassung der Zeitschrittweite

Basierend auf der oben vorgestellten Fehlerabschätzung und der Definition des maximal zulässigen Fehlers soll nun für jeden Zeitschritt die größt mögliche Schrittweite bestimmt werden, welche noch die Toleranz erfüllt. Um ein derartiges Verhalten zu erreichen, wollen wir nach dem in Abbildung 7.5 dargestellten Algorithmus vorgehen. Zunächst wird eine initiale Schrittweite h_{ini} vorgegeben. Diese wird nun iterativ solange verkleinert, bis der geschätzte Fehler kleiner als der maximal zulässige ist. Die Schrittweite welche als erstes die Bedingung

$$\Delta_{err,ist}\left(h_{max}\right) \leq \Delta_{err,max} \tag{7.16}$$

erfüllt, nennen wir h_{max}. Der Runge-Kutta-Schritt $f^{n+1} = f^n + k_{RK}$ wird jetzt mit dem Verfahren höherer Ordnung und dem Zeitschritt h_{max} ausgeführt. An dieser Stelle muss nun eine Möglichkeit eingebaut werden, dass der Zeitschritt wieder größer werden kann. Es soll dabei angenommen werden, dass dieser umso stärker zunehmen kann, je kleiner die Abschätzung des momentanen Fehlers im Vergleich zum geforderten maximalen ist. Die Schrittweite wird also vergrößert und auf diese Weise das h_{ini} des zweiten Zeitschritts abgeschätzt. Der Algorithmus beginnt nun von vorne.

Wie kann die zu wählende Schrittweite h möglichst optimal bestimmt werden?

Ziel ist es für jeden Zeitschritt den Runge-Kutta-Schritt mit der optimalen Zeitschrittweite h_{max} auszuführen. Wir kennen jedoch keinen analytischen Zusammenhang, wie groß diese jeweils zu wählen ist. Daher verfolgen wir, wie Abbildung 7.5 zu entnehmen ist, einen iterativen Ansatz: Wir beginnen mit einer zu großen Zeitschrittweite h_{ini}, welche wir so lange verkleinern, bis unser Fehlermaß innerhalb des Toleranzbereiches ist. Doch wie sieht eine optimale Strategie zur Verkleinerung aus, welche möglichst wenige Iterationen benötigt? Denn bei jeder Iteration müssen wir ein neues $\Delta_{err,ist}$ bestimmen, was Rechenzeit kostet. Wir suchen also ein Verfahren, das die Schrittweite abhängig vom momentan geschätzen Fehler verkleinert oder vergrößert. Eine derartige Strategie soll im folgenden vorgestellt werden.

Der lokale Fehler eines Verfahrens der Konsitenzordnung p ist von $\mathcal{O}\left(h^{p+1}\right)$. Demnach skaliert der lokale Fehler des hier verwendeten Verfahrens vierter Ordnung ungefähr mit h^5. Mit dieser Näherung kann der Fehler für die momentan verwendete Schrittweite als $\Delta_{err,ist} \propto h_{ist}^5$ geschrieben werden. Zusätzlich kann eine Fehlerobergrenze $\Delta_{err,max}$ gewählt werden, welcher bei einer noch unbekannten Schrittweite h_{max} entsteht. Für diese gilt analog der Zusammenhang $\Delta_{err,max} \propto h_{max}^5$. Setzt man diese beiden Fehler ins Verhältnis, kann die maximalen Schrittweite welche noch die gewählte Genauigkeitsanforderung erfüllt abgeschätzt werden:

$$h_{max} = \left(\frac{\Delta_{err,max}}{\Delta_{err,ist}}\right)^{\frac{1}{5}} h_{ist} \tag{7.17}$$

Damit wurde eine Abhängigkeit des zu wählenden Zeitschrittes von dem geschätzten Fehler des momentanen Zeitschrittes hergeleitet. Der Zeitschritt sollte hierbei hinreichend klein sein, da ansonsten die Fehlerabschätzung nicht in dieser Weise gültig ist. Es sei daher darauf hingewiesen, dass diese Herleitung nicht mehr aus rein mathematisch gerechtfertigten Schlussfolgerungen besteht, sondern auch Erfahrungswerte aus der Praxis enthält. Demnach ist die hier vorgestellte Methode nur als Beispiel einer Schrittweitensteuerung zu sehen.

Wird in diesen Ausdruck Gleichung (7.17) eingesetzt und zusätzlich ein Sicherheitsfaktor S eingeführt, ergibt sich schließlich für ein Verfahren der Ordnung p

$$h_{max} = S\left(\frac{\epsilon_r f_{skal} + \epsilon_a}{\Delta_{err,ist}}\right)^{\frac{1}{p+1}} h_{ist}. \tag{7.18}$$

Der Sicherheitsfaktor soll dabei sicherstellen, dass der tatsächlich gemachte Fehler nicht unterschätzt wird. Damit der Algorithmus nicht zu ineffizient wird, sollte ein Wert knapp unter Eins verwendet werden. Optimale Werte für die absolute und die relative Toleranz müssen problemangepasst gefunden werden. Es sollten einerseits keine zu kleinen Werte gesetzt werden, da es ansonsten dazu kommen kann, dass eine Schrittweite von nahezu Null verwendet wird. Dies minimiert zwar den Fehler, allerdings kann dies dazu führen, dass der Algorithmus auf der Stelle tritt. Aus diesem Grund ist es sinnvoll eine untere Grenze für den Zeitschritt zu definieren.

Zusätzlich sollte die maximal Schrittweitenerhöhung bzw. -abnahme limitiert werden. Als praktikabel hat es sich dabei erwiesen die maximale Zunahme auf 5 (pro Zeitschritt) und die maximale Abnahme auf 10 (pro Iteration nicht pro Zeitschritt!) zu deckeln. Dies bedeutet, die neue Schrittweite kann maximal das fünf-fache und - pro Iteration der Schrittweitenverkleinerung - minimal das 0.1-fache der vorherigen Schrittweite betragen.

Zuletzt soll darauf hingewisen werden, dass Formel (7.18) zur Verkleinerung wie auch zur Vergrößerung der Zeitschrittweite eingesetzt werden kann. Ist der für den momentanen Zeitschritt geschätzte Fehler $\Delta_{err,ist}$ kleiner als der maximal zulässige, so ist der Quotient

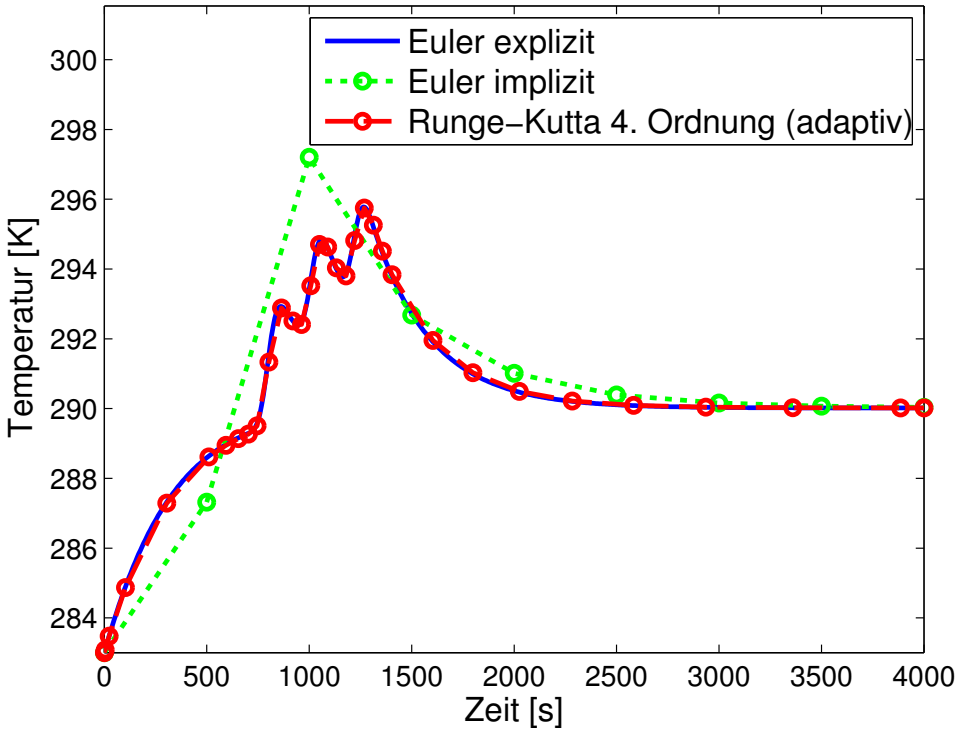
$$\frac{\epsilon_r |f| + \epsilon_a}{\Delta_{err,ist}} \tag{7.19}$$

kleiner Eins. Die Anwendung der Formel führt damit zu einer Verkleinerung der Schrittweite. Dies ist die Situation im zweiten Block von oben in Abbildung 7.5. Wurde jene Schrittweite h_{max} bestimmt, die gerade die definierte Toleranz erfüllt, wird dieser Quotient einen Wert größer Eins aufweisen. Ein weiteres Anwenden der Formel führt also zu einer Vergrößerung der Zeitschrittweite. Diese Situation liegt im untesten Block von Abbildung 7.5 vor.

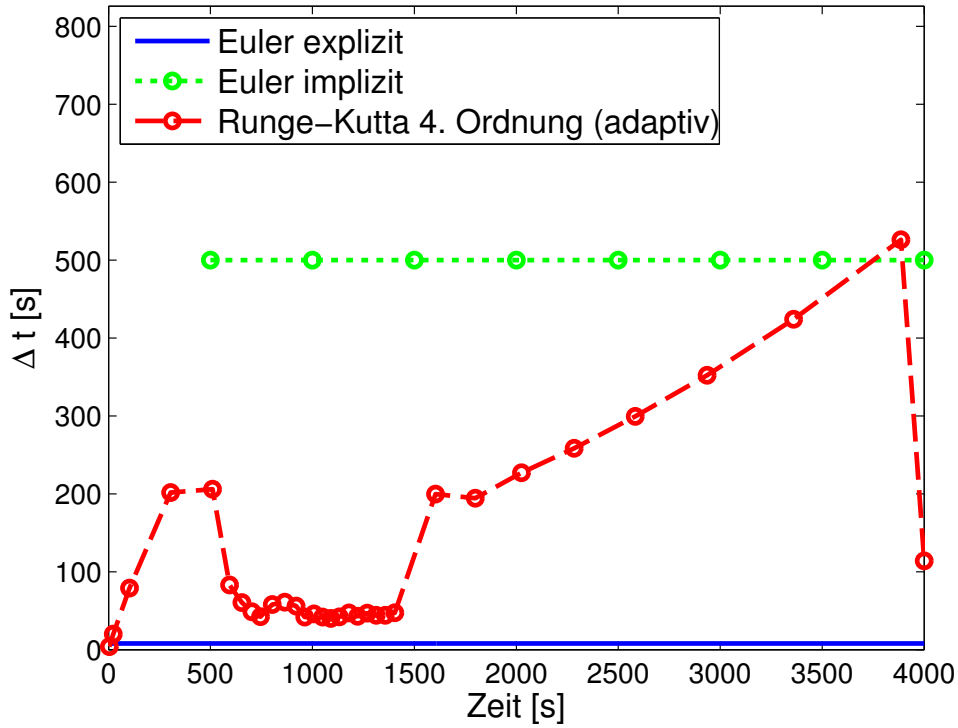
In Abbildung 7.6(a) ist noch einmal der simulierte Temperaturverlauf des in der Motivation zu diesem Kapitel vorgestellten Beispiels gezeigt. Hier wurde zusätzlich zu den Lösungskurven des expliziten und impliziten Euler-Verfahrens noch die Lösung des eben behandelten adaptiven Runge-Kutta-Verfahrens geplottet. Der Zeitschritt des impliziten Verfahrens ist hierbei etwas kleiner als in der einleitend gezeigten Lösung in Abbildung 7.2. Darüber hinaus wird in Abbildung 7.6(b) der Verlauf der Zeitschrittweite $h(t)$ gezeigt.

Es zeigt sich, dass das adaptive Verfahren deutlich weniger Zeitschritten als das explizite Verfahren benötigt, um einen ähnlichen Lösungsverlauf berechnen zu können. Im Bereich mit starkem Temperaturschwankungen wird die Zeitschrittweite reduziert, während sie im übrigen Bereich deutlich vergrößert wird. Dies ist genau das von uns erwünschte Verhalten. Dabei muss für den unmittelbaren Vergleich der Verfahren zusätzlich berücksichtigt werden, dass die Konsistenzordnung der Verfahren verschieden ist, das Euler-Verfahren also bereits von daher benachteiligt ist.

Abschließend sei noch darauf hingewiesen, dass das adaptive Verfahren unter Umständen den Zeitschritt soweit erhöht, dass es instabil wird (denn es handelt sich um ein explizites Verfahren). Dies sollte vermieden werden. Um das zu verhindern kann im Zweifelsfall eine maximal zulässige Schrittweite definiert werden.



(a) Zeitverlauf der Temperatur eines abkühlenden Körpers



(b) Zeitschrittweite geplottet über der Zeit.

Abbildung 7.6: Lösung des in der Motivation zu diesem Kapitel vorgestellten Beispiels mittels eines adaptiven Runge-Kutta-Verfahrens vierter Ordnung (Cash-Karp).

7.3 Übungsaufgaben

Heute werden Sie ein verbreitetes Verfahren zur Zeitdiskretisierung, das explizite (klassische) Runge-Kutta-Verfahren vierter Ordnung, implementieren. Hierfür wird zunächst eine konstante Zeitschrittweite verwendet. In der zweiten Aufgabe werden Sie zur Effizienzsteigerung sowie zur Kontrolle der Genauigkeit Ihres Verfahrens eine adaptive Zeitschrittweitensteuerung programmieren. Hierbei wird das in der Motivation zu Kapitel 7 vorgestellte physikalische Problem betrachtet.

Die Parameter sowie Stoffdaten sind wie folgt:

Temperaturen	
T_{ini}	$283\ K$
T_∞	$283\ K$
T_W	$300\ K$
Geometrie	
D	$5\ mm$
L	$0.1\ m$
Stoffdaten Konstantan	
ρ	$8900\ \frac{kg}{m^3}$
c	$410\ \frac{J}{kg\ K}$
λ	$23\ \frac{W}{m\ K}$
ρ_W	$4.9e-7\ Ohm\ m$
Strahlung	
ϵ	0.91
σ	$5.670e-8\ \frac{W}{m^2\ K^4}$

Eine Funktion zur Bestimmung des Wärmeübergangskoeffizienten α ist in den Hilfsdateien zu dieser Übung gegeben.

7.1 Berechnen Sie den instationären Verlauf der Temperaturentwicklung des Konstantandrahtes für gegebenes $I(t)$ mit Hilfe des **klassischen Runge-Kutta-Verfahrens 4. Ordnung** und plotten Sie das Ergebnis. Das Problem werde durch Gleichung (7.1) beschrieben. Simulieren Sie 4000 Sekunden.

Hinweise:

- Die rechte Seite der zu integrierenden DGL hängt auch von der Zeit und der Temperatur ab (damit ist diese nicht mehr autonom). Für den Runge-Kutta-Schritt sollten Sie nicht die Multiplikation mit den c_is vergessen.
- Das Problem wird einfacher, wenn Sie zunächst eine konstante Stromstärke annehmen.
- Das Butcher-Tableau des klassischen Runge-Kutta-Verfahrens sieht wie folgt aus:

0				
$1/2$	$1/2$			
$1/2$	0	$1/2$		
1	0	0	1	
	$1/6$	$1/3$	$1/3$	$1/6$

- Es ist zweckmäßig das Butcher-Tableau in einer Matrix `a_mat` sowie in den zwei Vektoren `b_vec` und `c_vec` zu speichern. Somit können Sie leicht Ihr Runge-Kutta-Verfahren anpassen, ohne Ihre gesamte Implementierung ändern zu müssen.
- Der Verlauf der Stromstärke $I(t)$ ist in der Hilfsdatei zu dieser Aufgabe gegeben.
- Teilen Sie die Implementierung Ihres Verfahrens nach Möglichkeit in zwei Funktionen: Eine Hauptfunktion `RungeKutta4()`, welche die Steuerung übernimmt und eine Funktion `RungeKutta4Step()`, welche für eine gegebene Zeitschrittweite h genau den Schritt k_{RK} entsprechend des Runge-Kutta-Verfahrens 4. Ordnung zurückgibt (siehe Gleichung (7.9)). Auf diese Weise können Sie Ihre Funktion einfach erweitern (wie in Aufgabe 7.3 erforderlich). Außerdem ist das Butcher Tableau nur in der Hilfsfunktion enthalten. Sie müssten also nur eine andere Hilfsfunktion verwenden um das RK-Schema zu wechseln.
  - `[y_vec,t_vec]=RungeKutta4(g,t_1,t_2,y_ini,N)`
    - `y_vec`: Vektor mit y-Werten zu den Zeiten `t_vec`
    - `t_vec`: Vektor mit allen Zeitschritten
    - `g`: Funktions-Handle
    - `t_1,t_2`: Start- und Endzeit
    - `y_ini`: y-Startwert zur Zeit `t_1` (Anfangsbedingung)
    - `N`: Anzahl an Zeitschritten
  - `[k_RK]=RungeKutta4Step(g,y,h,t)`
    - `k_RK`: Schritt
    - `g`: Funktion-Handle
    - `y`: aktueller y-Wert
    - `h`: Schrittweite
    - `t`: aktuelle Zeit

7.2 Fertigen Sie ein Ablaufdiagramm für das adaptive Runge-Kutta-Verfahren der nächsten Aufgabe an, bevor Sie mit der Programmierung beginnen.

7.3 Berechnen Sie den instationären Verlauf der Temperaturentwicklung des Konstantandrahtes mit Hilfe eines **expliziten Runge-Kutta-Verfahrens 4. Ordnung** mit **adaptiver Zeitschrittweitensteuerung**. Verwenden Sie hierfür die Methode nach Cash-Karp. Plotten Sie das Ergebnis sowie den Verlauf des Zeitschrittes über der Zeit. Simulieren Sie wieder 4000 Sekunden.

Hinweise:

- Starten Sie mit Ihrer Implementieren des klassischen Runge-Kutta Verfahrens aus Aufgabe 7.1 und erweitern Sie diese (Kopie!). Beachten Sie die oben vorgeschlagene Aufteilung in eine Hauptfunktion `RungeKutta45()` und eine Hilfsfunktion `RungeKutta45Step()`.
  - `[y_vec,t_vec]=RungeKutta45(g,t_1,t_2,y_ini)`
    - `y_vec`: Vektor mit y-Werten zu den Zeiten `t_vec`
    - `t_vec`: Vektor mit allen Zeitschritten
    - `g`: Funktions-Handle
    - `t_1,t_2`: Start- und Endzeit
    - `y_ini`: y-Startwert zur Zeit `t_1` (Anfangsbedingung)

  - `[k_RK,error]=RungeKutta45Step(g,y,h,t)`
    * `k_RK`: Schritt
    * `error`: Geschätzter lokaler Fehler
    * `g`: Funktion-Handle
    * `y`: aktueller y-Wert
    * `h`: Schrittweite
    * `t`: aktuelle Zeit
- Das Butcher-Tableau der Cash-Karp-Methode ist in den Hilfsdateien zu diesem Übungstermin gegeben.
- Verwenden Sie für die absolute Toleranz $\epsilon_a = 1\text{e} - 6$, für die relative $\epsilon_r = 1\text{e} - 6$ und einen Sicherheitsfaktor von $\text{S} = 0.9$.
- Da Sie - anders als bei einer festen Zeitschrittweite - die Anzahl der Zeitschritte nicht im Voraus kennen, müssen Sie sich zur Speicherung Ihres zeitlichen Temperaturverlaufes eine alternative Idee überlegen! Tipp: Eine Anpassung der Dimension des Lösungsvektors zur Laufzeit wird nicht empfohlen (Warum?).
- Beachten Sie, dass der letzte Zeitschritt genau auf die Endzeit der Simulation $t_{end} = 4000\ s$ treffen sollte.

7.4 Für Verfahren mit festen Zeitschritt konnten Sie die Unabhängigkeit der Lösung von der Größe des Zeitschrittes durch Vergleich der Lösung zweier Diskretisierungen zeigen. Wie können Sie diesen Nachweis im Falle eines adaptiven Verfahrens erbringen? Können Sie damit Ihre Lösung verifizieren?

7.4 Wichtige Matlab Befehle

Befehl	Beschreibung
`heaviside()`	Heaviside-Funktion (Stufen-Funktion).
`vector(n:m) = []`	Löscht den Bereich zwischen der n-ten und m-ten Stelle im Vektor `vector`.

8 Lösen partieller Differentialgleichungen

Literaturverzeichnis

[1] H.D. Baehr und K. Stephan. *Wärme- und Stoffübertragung*. Kap. 2.4. Springer Berlin Heidelberg, 2013. ISBN: 9783642365577. URL: http://books.google.de/books?id=HI6TngEACAAJ.

Lernziele

- Räumliche und zeitliche Diskretisierung verschiedener Konsistenzordnung
- Implementierung der wichtigsten Randbedingungen
- Bedingung für Stabilität

Motivation

Physikalische Vorgänge werden meist mithilfe von partiellen Differentialgleichungen modelliert. In den Ingenieurwissenschaften treten hierbei i.A. Zeitableitungen und räumliche Ableitungen auf. Ein gängiges Vorgehen zur numerischen Behandlung besteht darin, diese in ein System von Differentialgleichungen zu transformieren, welches nur noch Zeitableitungen maximal erster Ordnung enthält. Dann kann dieses System wie in den Kapiteln 6 und 7 beschrieben in der Zeit integriert werden, wobei alle Ortsableitungen in dem Term auf der rechten Seite enthalten sind. Um ein System aus gewöhnlichen Differentialgleichungen zu erhalten, müssen in einem zweiten Schritt die Orstableitungen diskretisiert werden. Hierfür können die in Kapitel 2 verwendeten Näherungsformeln verwendet werden. Ein derartiges Vorgehen bezeichnet man als Finite-Differenzen-Verfahren.

Ein Beispiel hierfür ist die zweidimensionale Ausbreitung von (Schall-) Wellen. Eine Momentaufnahme einer numerischen Simulation mit Hilfe von Finiten-Differenzen ist in Abbildung 8.1 dargestellt. Am Ende dieses Kapitels werden wir alles Rüstzeugs zusammen haben, um einen Löser für eine derartige Simulation selbst zu programmieren.

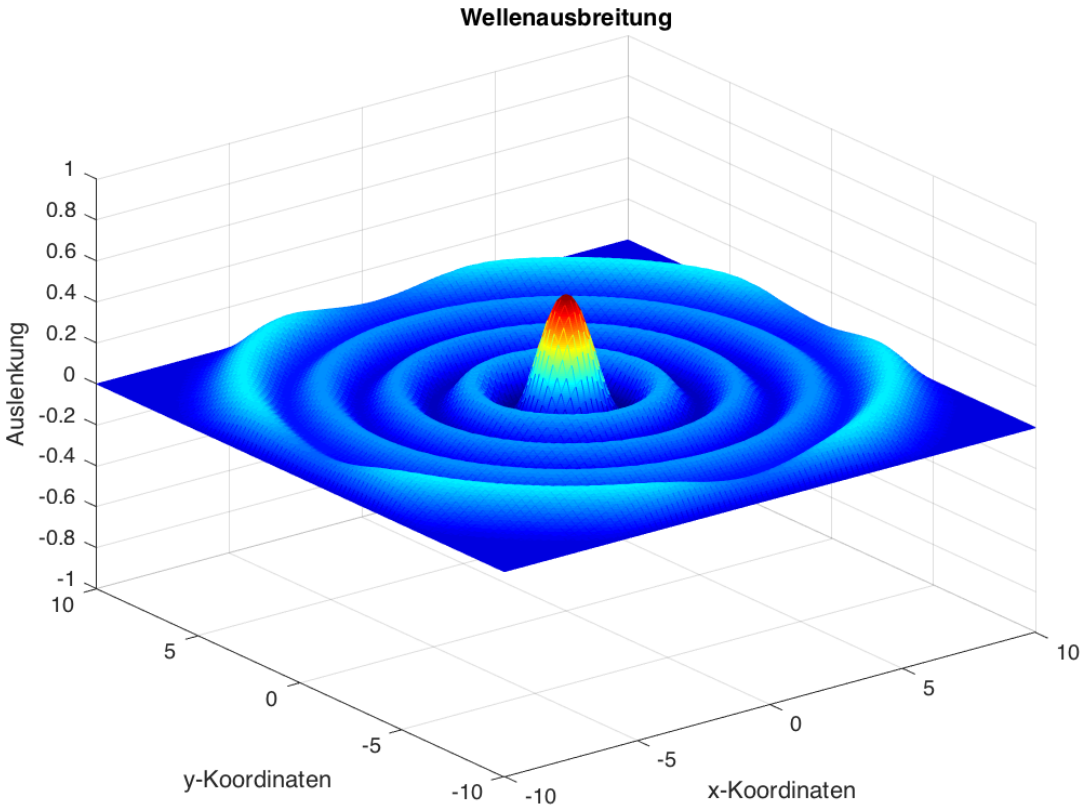


Abbildung 8.1: Momentaufnahme einer Simulation der zweidimensionalen Wellenausbreitung mit Finiten-Differenzen.

Das Feld der partiellen Differentialgleichungen (PDGL) ist sehr weitläufig, weshalb es sich als sinnvoll erwiesen hat die Gleichungen nach verschiedenen Kriterien zu klassifizieren. Ein naheliegender Weg, dies zu tun besteht darin, PDGLs nach der höchsten vorkommenden Ableitung zu charakterisieren. PDGLs zweiter Ordnung enthalten demnach maximal zweite Ableitungen. Sehr praxisrelevant ist auch die Einteilung in lineare und nichtlineare PDGLs. Linear ist eine PDGL genau dann, wenn die unbekannten Funktionen in allen Ableitungen nur linear vorkommen. Nichtlineare Probleme sind im Allgemeinen um einiges aufwändiger zu behandeln als lineare. Als Beispiel für ein nichtlineares Phänomen sei die Turbulenz in Strömungen genannt. Zuletzt können PDGLs auch nach dem Verlauf ihrer Charakteristiken geordnet werden. Man unterscheidet dann zwischen *elliptischen*, *parabolischen* und *hyperbolischen* Differentialgleichungen. Je nach Typ müssen andere Kombinationen aus Rand- und Anfangsbedingungen verwendet werden um ein korrekt gestelltes Problem zu erhalten, welches eindeutig lösbar ist. Genaueres hierzu wird im Kurs „Computational Thermo-Fluid Dynamics" behandelt.

Für die numerische Behandlung ist die Unterscheidung in Anfangswertprobleme und Randwertprobleme wichtig. In ersterem Fall wird die zeitliche Entwicklung eines Anfangszustandes betrachtet, wobei zu jeder Zeit alle Randbedingungen erfüllt sein müssen. Beispiel hierfür ist die Temperaturentwicklung in einem Stab, der zu einem bestimmten Zeitpunkt auf einer Seite erhitzt wird. Im Fall des Randwertproblems wird ein stationärer Zustand betrachtet. Eine Frage hier könnte z.B. sein, welche Temperaturverteilung sich in einem einseitig erhitzen Stab, der an der gegenüberliegenden Seite gekühlt wird, nach langer Zeit einstellt. Diese Probleme sind in Bezug auf Stabilität und Rechenaufwand i.A. einfacher numerisch zu lösen.

8.1 Finite-Differenzen-Methode

Gegeben sei eine PDGL, die wieder in der allgemeinen Form

$$\frac{\partial f(t,x,y,\dots)}{\partial t} = g\left(t,x,y,\dots,f(t,x,y,\dots),h(t,x,y,\dots),\dots\right) \tag{8.1}$$

angegeben werden kann. Wie bereits erwähnt, lässt sich jede DGL höherer Ordnung auf ein System erster Ordnung bzgl. der Zeitableitung reduzieren. Dies ist auch für PDGLs möglich. Der Unterschied in diesem Kapitel ist nun, dass erstens f auch von anderen Größen als t abhängen kann und dass zweitens die rechte Seite neben Ableitungen von f auch Ableitungen anderer Funktionen enthalten kann (hier mit $h(t,x,y,\dots)$ angedeutet).

Zusätzlich zu der PDGL müssen Rand- und Anfangsbedingungen gegeben sein, um ein eindeutig lösbares Problem zu formulieren. Es existieren dabei drei gängige Randbedingungen, die im Folgenden für den 1D Fall angegeben werden:

- **Dirichlet-Randbedingung:** Hierbei wird der Funktionswert auf dem Rand x_R festgesetzt.

$$f(x_R,t) = c \tag{8.2}$$

- **Neumann-Randbedingung:** Hier wird die erste Ableitung der Funktion normal zum Rand festgesetzt.

$$\left.\frac{\partial f(x,t)}{\partial x}\right|_{x_R} = c \tag{8.3}$$

- **Schiefe Randbedingung:** Auch Robin-Randbedingung genannt. Diese ist eine Kombination der beiden zuvor genannten.

$$a \left.\frac{\partial f(x,t)}{\partial x}\right|_{x_R} + b f(x_R,t) = c \tag{8.4}$$

Anfangsbedingungen geben einen Funktionsverlauf zum Zeitpunkt t_0 vor. Dieser sollte kompatibel mit den gewählten Randbedingungen sein. Um die Diskussion hier nicht zu abstrakt werden zu lassen, soll als Beispiel die 1D-Wärmeleitungsgleichung (Stab) für konstante Stoffwerte betrachtet werden:

$$\frac{\partial T(x,t)}{\partial t} = a \frac{\partial^2 T(x,t)}{\partial x^2}. \tag{8.5}$$

Als Anfangsbedingung sei $T(x,0)$ gegeben. Am linken Rand sei die Temperatur konstant (Dirichlet-Randbedingung) und am rechten Rand sei der Temperaturgradient als $-\dot{q}_r/\lambda$ (Neumann-Randbedingung) festgesetzt.

8.1.1 Raumdiskretisierung

In einem ersten Schritt wird nun der Raum diskretisiert, wozu ein Rechengitter über der Geometrie erzeugt werden muss. Beispielhaft sei dies für das vorliegende Problem in Abbildung 8.2 veranschaulicht. Der Grund weshalb links ein Gitterpunkt genau auf dem Rand liegt und rechts der Rand von zwei Gitterpunkten eingeschlossen wird, wird weiter unten bei der Behandlung der Randbedingung nachgeliefert. Für folgende Betrachtung soll zunächst angenommen werden, dass auch der rechte Gitterpunkt x_N genau auf dem Rand liegt und x_H nicht existiert.

Bezüglich dieses Gitters werden nun die Raumableitungen mit den in Kapitel 2 vorgestellten Verfahren dargestellt. Die Aussagen zum Diskretisierungsfehler oder zur Verfahrensordnung gelten analog. Es soll ein Verfahren zweiter Ordnung zum Einsatz kommen, womit sich

$$\frac{dT(x_i,t)}{dt} = a \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} + \mathcal{O}\left(\Delta x^2\right) \tag{8.6}$$

ergibt. Zur Verkürzung der Schreibweise wurde auf der rechten, diskretisierten Seite $T(x_i,t)$ durch T_i ersetzt. Damit liegt das Problem nun in der in Kapitel 6 eingeführten Standardform für gewöhnliche Differentialgleichungen vor. Die rechte Seite ist jetzt nur noch eine Funktion von $T(x_i,t)$, die Zeit ist noch kontinuierlich (daher wird auch keine partielle Ableitung $\partial/\partial t$ mehr verwendet, sondern die „normale" Ableitung d/dt). Allerdings ist durch die Diskretisierung T nun ein Vektor, der die Temperaturwerte an den Stellen x_1 bis x_N enthält. Damit besteht die

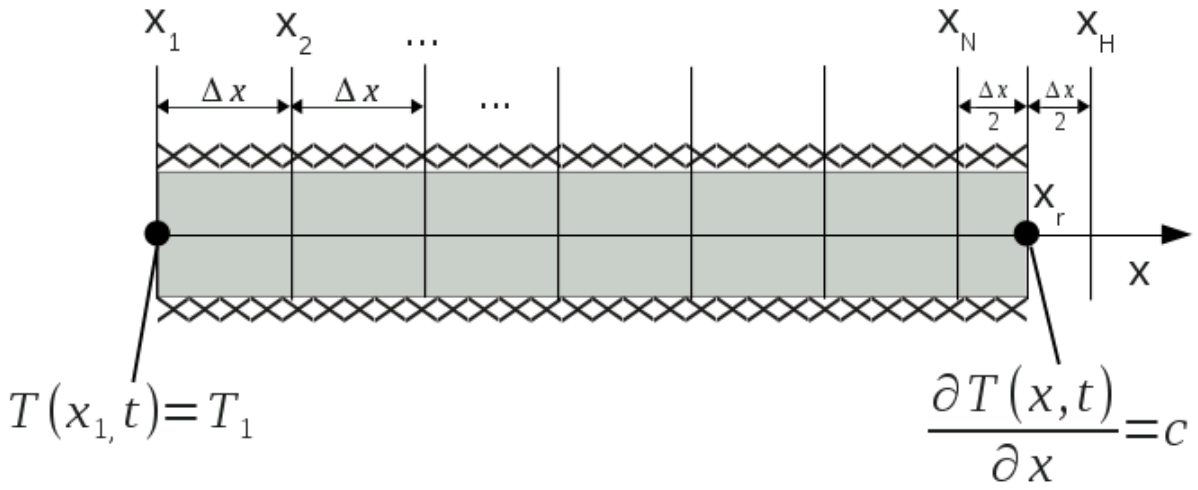


Abbildung 8.2: Diskretisierung der Stabgeometrie

linke Seite der Gleichung aus einem Vektor der Länge $N-2$, während die rechte Seite sich durch eine Matrix-Vektor-Multiplikation mit einem Vektor der Länge N darstellen lässt:

$$\frac{d}{dt}\begin{pmatrix} T_2 \\ T_3 \\ T_4 \\ \vdots \\ T_{N-1} \end{pmatrix} = A \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{pmatrix} = \frac{a}{\Delta x^2} \begin{pmatrix} 1 & -2 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & & & & \ddots & & \\ 0 & 0 & 0 & \dots & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{pmatrix} \tag{8.7}$$

Es sind also $N-2$ gewöhnliche Differentialgleichungen entstanden. Dabei ist zu beachten, dass auf der linken Seite die Temperaturen an den Rändern ausgespart wurden, da dort Gleichung (8.6) nicht anwendbar ist - ein 0-ter bzw. $N+1$-ter Eintrag im T-Vektor existiert nicht. Dieser wäre jedoch zur Berechnung von dT_1/dt bzw. dT_N/dt erforderlich. Damit ist die Matrix A in dieser Darstellung nicht quadratisch: Sie besitzt mehr Spalten als Zeilen. Dies ist nicht wünschenswert, da ein solches Gleichungssystem unterbestimmt ist und i.A.nicht gelöst werden kann. Um das Porblem zu schließen, also um die Matrix quadratisch zu bekommen, müssen extra Gleichungen mithilfe der Randbedingungen bestimmt werden. Dabei muss nach dem Typus des jeweiligen Randes unterschiedlich vorgegangen werden.

8.1.2 Randbedingungen

Es gibt mehrere Möglichkeit die Randbedingungen in das Gleichungssystem (8.7) zu integrieren. Die im Folgenden vorgestellte Methode hat sich jedoch bewährt und zeigt ein Prinzip nach dem vorgegangen werden kann. Es sollte unabhängig vom gewählten Vorgehen immer darauf geachtet werden, dass sich die Ordnungen der Diskretisierungsverfahren am Rand und im Inneren nicht zu stark unterscheiden (Empfehlung ist höchstens eine Ordnung Unterschied).

- **Dirichlet-Randbedingung:** Das Gitter wird so gewählt, dass der Randpunkt des Gitters genau auf dem Rand des Rechengebietes liegt (siehe Abbildung 8.2 links). Für eine Dirichlet-Randbedingung am linken Rand wird also eine Gleichung für die Zeitableitung des ersten Punktes T_1 gesucht. Da dem Randpunkt ein konstanter Wert zugeordnet wird, ist die zeitliche Änderung gerade null:

$$\frac{dT_1}{dt} = 0. \tag{8.8}$$

- **Neumann-Randbedingung:** Liegt eine derartige Randbedingung vor, so sollte der Rand in der Mitte zwischen dem ersten bzw. letzten Gitterpunkt und einem „Hilfspunk" x_H liegen. Das ist am rechten Rand der Konfiguration von Abbildung 8.2 illustriert. Auf diese Weise kann die Ableitung der Randbedingung leicht mittels einer zentralen Differenz gebildet werden, was ebenfalls ein Verfahren zweiter Ordnung ist:

$$-\lambda \left.\frac{\partial T(t,x)}{\partial x}\right|_{x_r} = \dot{q}_r \tag{8.9}$$

$$\frac{T_H - T_N}{\Delta x} = -\frac{\dot{q}_r}{\lambda} \tag{8.10}$$

$$T_H = T_N - \frac{\dot{q}_r \Delta x}{\lambda} \tag{8.11}$$

Damit lässt sich nun ein Ausdruck für $\frac{dT_N}{dt}$ herleiten, der am Ende des Gleichungssystems (8.7) hinzugefügt werden kann. Die letzte Gleichung des Rechengebietes für den Punkt N schreibt sich also

$$\frac{dT_N}{dt} = \frac{a}{\Delta x^2}\left(T_{N-1} - 2T_N + T_H\right) = \frac{a}{\Delta x^2}\left(T_{N-1} - T_N\right) - \frac{a\dot{q}_r}{\lambda \Delta x}. \tag{8.12}$$

Hier tritt jetzt ein konstanter Term $\frac{a\dot{q}_r}{\lambda\Delta x}$ auf, der zur Aufstellung des gesamten Gleichungssystems gesondert mittels eines b-Vektors berücksichtigt werden muss (s.u.).

- **Schiefe-Randbedingung:** In diesem Fall sollte der erste bzw. letzte Gitterpunkt auf dem Rand liegen und zusätzlich wird wieder ein „Hilfspunkt" x_H außerhalb des Rechengebietes hinzugefügt, wie es in Abbildung 8.3 für den rechten Rand gezeigt ist. Analog zum Vorgehen für Neumann-Randbedingungen wird dieser Punkt nicht in den Lösungsvektor aufgenommen, sondern wird lediglich zur Herleitung der Randbedingung verwendet. Es wird die Randbedingung für den konvektiven Wärmeübergang

$$-\lambda \left.\frac{\partial T(t,x)}{\partial x}\right|_{x_r} = \alpha(T(t,x_r) - T_\infty) \tag{8.13}$$

hergeleitet, einem Spezialfall einer schiefen Randbedingung:

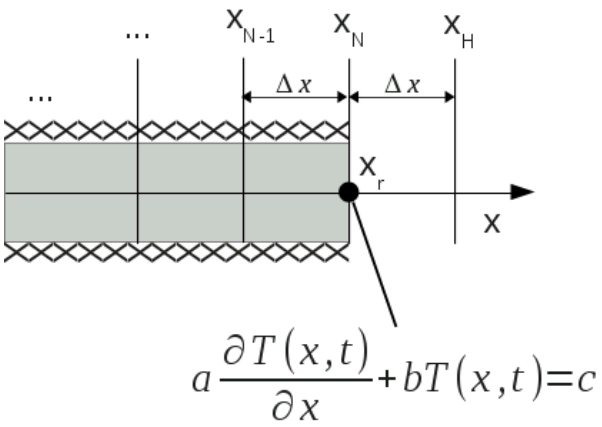


Abbildung 8.3: Diskretisierung der Stabgeometrie bei einer schiefen Randbedingung.

$$-\lambda \frac{T_H - T_{N-1}}{2\Delta x} = \alpha(T_N - T_\infty), \tag{8.14}$$

$$T_H = T_{N-1} - \frac{2\Delta x \alpha}{\lambda} T_N + \frac{2\Delta x \alpha}{\lambda} T_\infty. \tag{8.15}$$

Dieser Ausdruck kann wiederum in die letzte Gleichung des Gleichungssystems eingesetzt werden:

$$\frac{dT_N}{dt} = \frac{a}{\Delta x^2}\left(T_{N-1} - 2T_N + T_H\right) = \frac{a}{\Delta x^2}\left[2T_{N-1} - \left(2 + \frac{2\Delta x \alpha}{\lambda}\right) T_N\right] + \frac{2a\alpha}{\lambda \Delta x} T_\infty \tag{8.16}$$

Auch hier ergibt sich wieder ein konstanter Term, der in einen b-Vektor gespeichert werden muss.

Wendet man das gezeigte Vorgehen auf das oben genannte Problem an, so ergibt sich bei den gegebenen Randbedingungen (Dirichlet links und Neumann rechts) folgendes Gleichungssystem:

$$\frac{\partial}{\partial t}\begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{pmatrix} = \frac{a}{\Delta x^2}\begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & & & \ddots & & \\ 0 & 0 & \dots & 0 & 1 & -1 \end{pmatrix}\begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ -\frac{a\dot{q}_r}{\lambda \Delta x} \end{pmatrix} \tag{8.17}$$

Damit ist die Matrix nun symmetrisch und es kommen nur Temperaturen vor, die innerhalb des Stabes liegen. Möchte man die Temperatur auf dem rechten Rand wissen, so muss z.B. linear zwischen T_N und T_H interpoliert werden, wobei T_H mit Hilfe von Gleichung (8.11) berechnet werden kann. Das System (8.17) lässt sich in Matrix-Vektorform schreiben als

$$\frac{\partial T}{\partial t} = AT + b, \tag{8.18}$$

wobei der Vorfaktor $\frac{a}{\Delta x^2}$ mit in die Matrix A gezogen wurde.

8.1.3 Zeitdiskretisierung

Die rechte Seite des Systems wurde zu N algebraischen Ausdrücken umgeformt und lässt sich durch $AT + b$ ausdrücken. Nun kann die Zeitdiskretisierung, wie in Kapitel 6 beschrieben, erfolgen. Hierzu werden im Folgenden die drei dort vorgestellten Verfahren durchgeführt. Es soll auch weiterhin die Matrix-Vektor-Schreibweise angewandt werden.

- **Das explizite Euler-Verfahren:**

$$\frac{T^{n+1} - T^n}{h} = AT^n + b \tag{8.19}$$

$$T^{n+1} = h\left(AT^n + b\right) + T^n \tag{8.20}$$

Beispielhaft wird hier der Ausdruck für den i-ten Eintrag des Temperaturvektors T^{n+1} angegeben:

$$T_i^{n+1} = h\left(\left(AT^n\right)_i + b_i\right) + T_i^n \tag{8.21}$$

- **Das implizite Euler-Verfahren:**

$$\frac{T^{n+1} - T^n}{h} = AT^{n+1} + b \tag{8.22}$$

$$(I - hA)T^{n+1} = T^n + hb \tag{8.23}$$

$$T^{n+1} = (I - hA)^{-1}(T^n + hb) \tag{8.24}$$

Hierbei bezeichnet I die Einheitsmatrix bzw. die Identitätsmatrix, welche auf der Diagonalen mit Einsern und ansonsten mit Nullen besetzt ist. Gleichung (8.23) ist demnach ein lineares Gleichungssystem mit der Matrix $(I - hA)$, dem Lösungsvektor T^{n+1} und der rechten Seite $T^n + hb$. Dieses gilt es zu lösen. Formell geschieht dies durch Invertierung der Matrix $(I - hA)$, wie dies in Gleichung (8.24) dargestellt ist. In MATLAB würde man dies jedoch niemals so implementieren. Hier sollte immer der \-Operator verwendet werden!

Im Gegensatz zum im Kapitel 6.1 beschriebenen impliziten Euler-Verfahren handelt es sich hier um ein *lineares* Gleichungs*system* (da wir jetzt nur noch lineare PDGLs betrachten wollen). Die Anwendung eines Algorithmuses zur Nullstellensuche ist somit nicht erforderlich. Es müssen vielmehr Verfahren zum Lösen derartiger Gleichungssysteme, wie z.B. der Gauß-Algorithmus aus Kapitel 4.2, eingesetzt werden.

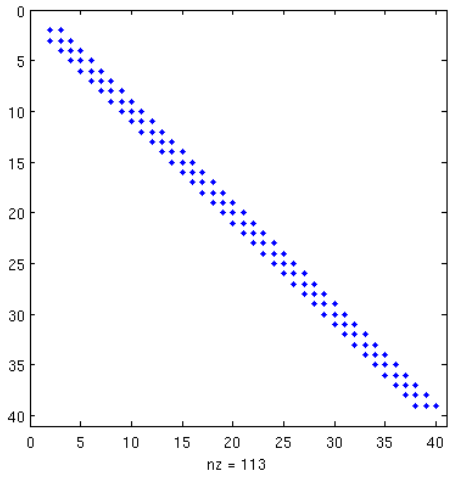


Abbildung 8.4: Visualisierung der Struktur einer Systemmatrix des finiten Differenzen-Verfahrens: Elemente ungleich null werden durch ein gefülltes Rechteck gekennzeichnet.

- **Das Crank-Nicolson-Verfahren:**

$$T^{n+1} = T^n + \frac{h}{2}\left[(AT^n + b) + (AT^{n+1} + b)\right] \tag{8.25}$$

$$(I - \frac{h}{2}A)T^{n+1} = (I + \frac{h}{2}A)T^n + hb \tag{8.26}$$

$$T^{n+1} = (I - \frac{h}{2}A)^{-1}\left[(I + \frac{h}{2}A)T^n + hb\right] \tag{8.27}$$

Gleichung (8.25) leitet sich direkt aus Gleichung (6.10) ab. Statt der rechten Seite $g(t, f)$ wurde hier die rechte Seite aus Gleichung (8.18) verwendet. Löst man diese Bedingung analog zum impliziten Euler-Verfahren auf, so bekommt man das zu lösende Gleichungssystem (8.26) sowie dessen Lösung (8.27).

Kann ein Problem durch Diskretisierung des Raumes auf eine Gleichung der Form (8.18) gebrachte werden, so kann daraus nach obigen Formeln einfach die Zeitdiskretisierung abgeleitet werden. Die in Kapitel 7 vorgestellten Runge-Kutta-Verfahren höherer Ordnung können natürlich ebenfalls verwendet werden. Allerdings ist deren Herleitung etwas komplizierter als jene für die drei oben vorgestellten Verfahren.

Die System-Matrizen der impliziten Verfahren ($(I - hA)$ Euler bzw. $(I - \frac{h}{2}A)$ Crank-Nicolson) besitzen immer eine besondere Struktur: Lediglich auf der Diagonalen sowie den jeweils darüber und darunter liegenden Diagonalen (Nebendiagonalen) besitzen sie Einträge ungleich null. Dies ist in Abbildung 8.4 visualisiert, in der Einträge ungleich null durch ein gefülltes Viereck dargestellt werden. Für höherdimensionale Probleme wäre es ineffizient diese Systeme mit einem normalen Gauß-Algorithmus zu lösen. Besser sind hier iterative Verfahren wie sie im Kurs „Computational Thermo-Fluid Dynamics" vorgestellt werden.

Zur Herleitung eines numerischen Verfahrens zur Zeitdiskretisierung muss nicht zwingend die eben vorgestellte Matrix-Vektor-Schreibweise verwendet werden. Genauso kann Gleichung (8.6) als Grundlage verwendet werden. Das Ergebnis ist dabei jedoch nicht so allgemein anwendbar. Das **explizite Euler-Verfahren** ließe sich in dieser Schreibweise z.B. darstellen als

$$\frac{T_i^{n+1} - T_i^n}{h} + \mathcal{O}(h) = a\frac{T_{i-1}^n - 2T_i^n + T_{i+1}^n}{\Delta x^2} + \mathcal{O}\left(\Delta x^2\right) \tag{8.28}$$

$$T_i^{n+1} = T_i^n + ah\frac{T_{i-1}^n - 2T_i^n + T_{i+1}^n}{\Delta x^2} + \mathcal{O}\left(h\Delta x^2\right) + \mathcal{O}\left(h^2\right) \tag{8.29}$$

Die Randbedingungen sind wie oben beschrieben zu behandeln. Dies führt schließlich ebenfalls auf die in Gleichung (8.21) dargestellte Lösung. Hierbei wurden die Abbruchfehler der Diskretisierungen mit eingefügt, um den Fehler der Näherung abschätzen zu können. Damit ist noch einmal verdeutlicht, dass auch die Gleichungen für die Randbedingungen einen Fehler der gleichen Ordnung haben sollten, da ansonsten der gesamte Fehler von den Rändern dominiert würde.

Das **implizite Euler-Verfahren** ließe sich analog darstellen als

$$\frac{T_i^{n+1} - T_i^n}{h} + \mathcal{O}(h) = a\frac{T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}}{\Delta x^2} + \mathcal{O}\left(\Delta x^2\right) \tag{8.30}$$

$$T_i^n = -\frac{ah}{\Delta x^2}T_{i-1}^{n+1} + \left(\frac{2ah}{\Delta x^2} + 1\right)T_i^{n+1} - \frac{ah}{\Delta x^2}T_{i+1}^{n+1} + \mathcal{O}\left(h\Delta x^2\right) + \mathcal{O}\left(h^2\right). \tag{8.31}$$

Die Koeffizienten vor den Temperaturen zum neuen Zeitschritt $n+1$ bilden die Einträge der Matrix $(I - hA)$ aus Gleichung (8.23). Diese besitzt demnach drei Diagonalen. Der Vektor hb ergibt sich aus der Neumann-Randbedingung und hat nur für die letzte Temperatur T_N^n einen Eintrag.

8.1.4 Stabilität

Für gewöhnliche Differentialgleichungen wurde bereits eine Stabilitätsbedingung in Abschnitt 6.2 formuliert. Auch für PDGLs betrachten wir wieder zwei numerische Lösungen der diskretisierten Differentialgleichung, zum einem f mit den Anfangswerten f^0 und zum andern F, welches die leicht gestörte Anfangswerte $f^0 + \epsilon^0$ besitzt. Dadurch erhält man einen zeitabhängigen Fehler ϵ_i^n, der bei PDGLs zusätzlich vom Ort abhängt:

$$F_i^n = f_i^n + \epsilon_i^n \tag{8.32}$$

Für die Stabilität wird auch hier gefordert, dass der Fehler in der Zeit abnimmt:

$$\frac{||\epsilon^{n+1}||}{||\epsilon^n||} < 1 \qquad \forall n. \tag{8.33}$$

Bzgl. der Fehlerentwicklung ist es schwer allgemeine Aussagen zu treffen. Wenn aber periodische Randbedingungen auf dem Rechengebiet $[0, L]$ angenommen werden, lässt sich der Fehler als Fourier-Reihe

$$\epsilon(x,t) = \sum_{m=1}^{\infty} b(t) e^{Ik_m x} \tag{8.34}$$

$$\epsilon_i^n = \sum_{m=1}^{\infty} b^n e^{Ik_m x_i} \tag{8.35}$$

darstellen mit der Wellenzahl $k_m = \pi m/L$ und der komplexen Einheit I. Dadurch werden sogenannte räumliche Moden $e^{Ik_m x}$ mit einer zeitabhängigen Amplitude b^n definiert.

Für das Beispiel aus Gleichung (8.5) ergibt sich unter Verwendung der oben gezeigten räumlichen Diskretisierung und eines expliziten Euler-Verfahrens

$$T_i^{n+1} = T_i^n + ah\frac{T_{i-1}^n - 2T_i^n + T_{i+1}^n}{\Delta x^2}. \tag{8.36}$$

Für **lineare PDGL** muss auch der Fehler die diskretisierte Form der PDGL erfüllen. Im Folgenden betrachten wir nur noch eine einzige Mode ($k_m = k$) und setzten den Fehler in Gleichung (8.36) ein. Multipliziert man diese Gleichung mit e^{-Ikx_i} ergibt sich:

$$b^{n+1} = b^n + \frac{ah}{\Delta x^2}\left(b^n e^{-Ik\Delta x} - 2b^n + b^n e^{Ik\Delta x}\right) \tag{8.37}$$

Hierbei ist zu beachten, dass $\epsilon_{i\pm1}^n = b^n e^{Ik(x_i \pm \Delta x)}$ gilt. Diese Gleichung hängt nicht mehr vom Ort ab. Lediglich die Zeit und die Diskretisierung spielen eine Rolle. Durch Division mit b^n lässt sich ein Ausdruck für die Stabilitätsbedingung herleiten:

$$\left|\frac{b^{n+1}}{b^n}\right| = \left|1 + \frac{ah}{\Delta x^2}(\underbrace{e^{-Ik\Delta x} + e^{Ik\Delta x}}_{=2\cos(k\Delta x)} - 2)\right| \tag{8.38}$$

$$= \left|1 + 2\frac{ah}{\Delta x^2}(\cos(k\Delta x) - 1)\right| < 1 \tag{8.39}$$

Dabei ergibt sich eine sinnvolle Bedingung, falls der Term im Betrag als negativ angenommen wird:

$$-\left(1 + 2\frac{ah}{\Delta x^2}(\cos(k\Delta x) - 1)\right) < 1, \tag{8.40}$$

$$\frac{ah}{\Delta x^2}(1 - \cos(k\Delta x)) < 1. \tag{8.41}$$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	$\sum_{i=1}^{9} \lvert x_i \rvert$
t^0	0	0	0	0	ϵ	0	0	0	0	ϵ
t^1	0	0	0	$\frac{\epsilon}{2}$	0	$\frac{\epsilon}{2}$	0	0	0	ϵ
t^2	0	0	$\frac{\epsilon}{4}$	0	$\frac{\epsilon}{2}$	0	$\frac{\epsilon}{4}$	0	0	ϵ
t^3	0	$\frac{\epsilon}{8}$	0	$\frac{3\epsilon}{8}$	0	$\frac{3\epsilon}{8}$	0	$\frac{\epsilon}{8}$	0	ϵ

Tabelle 8.1: Fehlerausbreitung für den neutral-stabilen Fall: $\frac{ah}{\Delta x^2} = \frac{1}{2}$.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	$\sum_{i=1}^{9} \lvert x_i \rvert$
t^0	0	0	0	0	ϵ	0	0	0	0	ϵ
t^1	0	0	0	ϵ	$-\epsilon$	ϵ	0	0	0	3ϵ
t^2	0	0	ϵ	-2ϵ	3ϵ	-2ϵ	ϵ	0	0	9ϵ
t^3	0	ϵ	-3ϵ	6ϵ	-7ϵ	6ϵ	-3ϵ	ϵ	0	27ϵ

Tabelle 8.2: Fehlerausbreitung für den instabilen Fall: $\frac{ah}{\Delta x^2} = 1$.

Ist diese Bedingung für alle $k \in \mathbb{R}^+$ erfüllt, ist auch insbesondere Gleichung (8.33) erfüllt. Dies folgt daraus, dass jede mögliche Mode sich durch Variation von k bilden lässt. Ist die lineare PDGL für alle Moden erfüllt, ist sie es folglich auch für die Summe dieser Moden. Der Term links in Gleichung (8.41) ist dann am größten, falls der Kosinus zu -1 wird. Gilt die Ungleichung für diesen Fall, so gilt sie auch für alle möglichen k. Mit dieser Überlegung ergibt sich die finale Stabilitätsbedingung:

$$\frac{ah}{\Delta x^2} < \frac{1}{2}. \tag{8.42}$$

Solange dieser Zusammenhang erfüllt ist, handelt es sich bei Gleichung (8.36) um ein stabiles Verfahren. Die maximale Größe des Zeitschrittes ist direkt proportional zum Quadrat der geometrischen Schrittweite. Damit muss für eine halb so großen Ortsschrittweite der Zeitschritt geviertelt werden, bei ansonsten gleichen Bedingungen. Damit wird klar, dass eine feine Ortsauflösung eine sehr feine Zeitauflösung und damit eine über die Maßen größeren Rechenaufwand erfordert, als man dies naiv (doppelte Ortsauflösung entspricht doppeltem Rechenaufwand) erwarten würde. Je stärker die Zeitauflösung mit der Ortsauflösung zum Erhalt eines stabilen Verfahrens skaliert, als desto steifer wird eine Differentialgleichung bezeichnet. Es sei darauf hingewiesen, dass auch die Randbedingungen einen Einfluss auf das Stabilitätsverhalten haben. Z.B. führt eine schiefe Randbedingung, wie sie oben eingeführt wurde, zu einem schlechteren Stabilitätsverhalten als das durch Gleichung (8.42) gegebene (vgl. [1]).

Die hier vorgestellte Stabilitätsanalyse ist als **von Neumann Analyse** bekannt. Genauso wie für gewöhnliche Differentialgleichungen, kann auch hier das Verhalten eines Verfahrens mit Papier und Bleistift nachvollzogen werden. Hierzu nimmt man als Anfangsbedingung an, alle Gitterpunkte besäßen den Wert null. Einzig im mittleren Punkt soll eine Störung ϵ vorliegen. Beispielhaft ist dies in Tabelle 8.1 einmal für den neutral-stabilen Fall und einmal in Tabelle 8.2 für den instabilen Fall vorgeführt. Zu beachten ist hier, dass keine Randbedingungen betrachtet wurden, sondern es wurde zur Auswertung der diskretisierten PDGL die zwei Randpunkte $x_0 = x_{10} = 0$ verwendet.

8.1.5 Konsistenz

Die Konsistenz bzw. die Konsistenzordnung eines Verfahren kann direkt aus der Diskretisierung der PDGL abgelesen werden, falls die Fehlerterme mit berücksichtigt werden. Für ein explizites Euler-Verfahren und eine Verwendung zentraler Differenzen im Raum ergibt sich für die 1D-Wärmeleitungsgleichung folgende Beziehung:

$$\frac{T_i^{n+1} - T_i^n}{h} - a\frac{T_{i-1}^n - 2T_i^n + T_{i+1}^n}{\Delta x^2} = \mathcal{O}(h) + \mathcal{O}\left(\Delta x^2\right). \tag{8.43}$$

Dieses Verfahren besitzt also die Konsistenzordnung 1 in der Zeit und die Konsistenzordnung 2 im Raum. Betrachtet man das Crank-Nicolson-Verfahren in analoger Weise, so ergibt sich die Konsistenzordnung 2 in der Zeit:

$$\frac{T_i^{n+1} - T_i^n}{h} - \frac{a}{2\Delta x^2}\left(\underbrace{T_{i-1}^n - 2T_i^n + T_{i+1}^n}_{g^n} + \underbrace{T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}}_{g^{n+1}}\right) = \mathcal{O}\left(h^2\right) + \mathcal{O}\left(\Delta x^2\right). \tag{8.44}$$

Diese Angaben gelten natürlich nur, falls auch die Randbedingungen mit entsprechenden Konsistenzordnungen implementiert werden!

8.2 Übungsaufgaben

Im Rahmen der letzten Übung dieses Praktikums wird eine Methode zur Lösung partieller Differentialgleichungen (PDGL), das sogenannte Finite-Differenzen-Verfahren, behandelt. Neben einer Zeitdiskretisierung (siehe Übungen 6 und 7), ist hierbei zusätzlich eine Diskretisierung des Raumes erforderlich. Dieses Verfahren wird im Folgenden anhand eines 1-Dimensionalen Beispiels veranschaulicht.

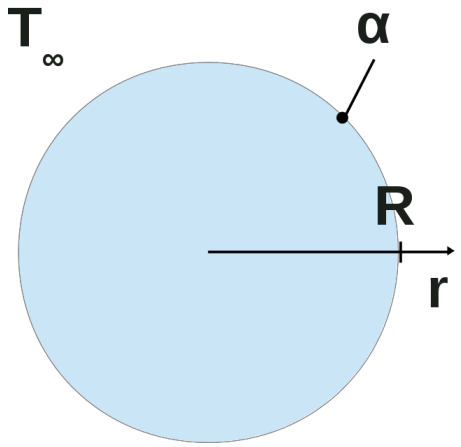


Abbildung 8.1: Zu härtende Kugel mit Radius R.

Bei der Herstellung von Kugellagern werden die enthaltenen Wälzkörper zur Verbesserung der Laufeigenschaften sowie zur Verlängerung der Haltbarkeit gehärtet. Als Werkstoff kommt hierbei typischer Weise der Stahl 100Cr6 zum Einsatz.

In einem ersten Schritt werden die zu härtenden Werkstücke zunächst auf eine Temperatur $\theta_{ini} \approx 800\ °C - 830\ °C$ gebracht. Diese Temperatur wird solange gehalten, bis innerhalb des Werkstücks eine annähernd gleichmäßige Temperaturverteilung vorliegt. Das Gefüge von 100Cr6 besteht in diesem Bereich vornehmlich aus Austenit (sogenannte γ-Mischkristalle). Anschließend wird in einem zweiten Schritt das Werkstück in ein Wasser- oder Ölbad gebracht, wodurch sich dessen Temperatur schlagartig auf die Temperatur des Bades T_∞ abkühlt. Dieser Vorgang wird auch als Abschrecken bezeichnet. Wird das Werkstück dabei schnell genug auf eine Temperatur unterhalb der sogenannte Martensitstarttemperatur $M_S = 250\ °C$ abgekühlt, so bildet sich eine harte Martensitphase aus. Dabei ist die (lokale) Abkülgeschwindigkeit maßgebend dafür ob ein voll martensitisches Gefüge entsteht. Die kleinste Geschwindigkeit bei der dies der Fall ist, wird als kritische Abkülgeschwindigkeit $v_{krit,AK}$ bezeichnet. Diese gibt an, wie schnell das Werkstück von der Temperatur $\theta_1 = 800\ °C$ auf die Temperatur $\theta_2 = 500\ °C$ abgekühlt wurde:

$$v_{krit,AK} = \frac{\theta_2 - \theta_1}{t(\theta_2) - t(\theta_1)}. \tag{8.1}$$

Der Prozess des Abkühlens soll im Folgenden für eine Kugel des Durchmessers $D = 70\ mm$ betrachtet werden. Diese besitze zum Zeitpunkt $t_0 = 0\ s$ eine homogene Temperaturverteilung

mit einem Wert von $T_{ini} = 1080\ K$. Die Temperatur des Fluides (Wasser) betrage $T_\infty = 293\ K$. Zur Berechnung des instationären Temperaturverlaufs soll lediglich der konvektive Wärmeübergang von der Kugel an das sie umgebende Fluid betrachtet werden. Hier soll vereinfacht angenommen werden, der Wärmeübergangskoeffizient betrage konstant $\alpha = 2000\ W/(m^2K)$. Der Effekt einer Dampfblasen- bzw. -filmbildung soll also vernachlässigt werden. Die Stoffdaten sowie Abmessungen sollen zudem als konstant angenommen werden.

Die Annahme eines konstanten Wärmeübergangskoeffizienten stellt eine grobe Vereinfachung dar, da die Wärmeübertragung vom Werkstück an das umgebende Fluid zunächst stark von einer Schicht an verdampftem Fluid eingeschränkt wird. Diese bildet eine Art Isolierschicht. Dieses Phänomen ist auch als Leidenfrost-Effekt bekannt. Erst wenn dieser schützende Film zerfällt nimmt die Wärmeübertragung stark zu und ermöglicht somit ein schnelleres Abkühlen. Durch Zusätze im Fluid bzw. durch Aufprägen einer Strömung wird versucht den Einfluss dieses Effektes zu vermindern.

Das quasi 1D instationäre Temperaturfeld T einer Kugel kann durch die PDGL

$$\frac{\partial T}{\partial t} = a\frac{\partial^2 T}{\partial r^2} + \frac{2a}{r}\frac{\partial T}{\partial r} \tag{8.2}$$

beschrieben werden. Die Variable a bezeichnet die Temperaturleitfähigkeit und ist definiert als $a = \lambda/(\rho c)$; r bezeichnet die radiale Koordinatenachse.

Die Parameter sowie Stoffdaten sind zusammengefasst wie folgt:

Temperaturen	
T_{ini}	$1080\ K$
T_∞	$293\ K$
θ_1	$800\ °C$
θ_2	$500\ °C$
M_S	$250\ °C$
Geometrie	
D	$70\ mm$
Stoffdaten Konstantan	
ρ	$7830 \frac{kg}{m^3}$
c	$470\ \frac{J}{kg\ K}$
λ	$46\ \frac{W}{m\ K}$
Sonstiges	
$v_{krit,AK}$	$30\ \frac{K}{s}$
α	$2000\ \frac{W}{m^2 K}$

8.1 Der Abkühlprozess soll für die gegebenen Randbedingungen mittels eines **expliziten Euler-Verfahrens** und zentraler Differenzen zur Raumdiskretisierung bis zu einer Zeit $t_1 = 60\ s$ simuliert werden.

8.2.1 Implementieren Sie ein derartiges Verfahren und stellen Sie die Temperaturentwicklung grafisch dar (r-t-T-Diagramm). Erstellen Sie außerdem ein Video der Entwicklung des Temperaturverlaufs. Verwenden Sie hierzu die Funktion `Polarkoordinaten_plot()` aus den Hilfsdateien zu diesem Termin.

8.2.2 Zur Verifikation müssen Sie nun die Unabhängigkeit der Lösung von der räumlichen wie auch der zeitlichen Diskretisierung nachweisen. Finden Sie nach Augenmaß eine geeignete Zeitschrittweite sowie räumliche Auflösung.

8.2.3 Berechnen Sie, bis zu welchem Radius r_{krit} die mittlere kritische Abkülgeschwindigkeit $v_{krit,AK}$ zwischen $\theta_1 = 800\ °C$ und $\theta_2 = 500\ °C$ größer als $30\ K/s$ ist. Prüfen Sie, ob Ihr Ergebniss von der Diskretisierung unabhängig ist!

8.2.4 Nach welcher Zeit ist an jeder Stelle der Kugel die Temperatur kleiner als M_S?

Hinweise:

- Erstellen Sie zunächst ihre Systemmatrix `A_mat` bzw. A ohne Berücksichtigung der Randbedingungen. Mit `spy(A_mat)` können Sie die Struktur Ihrer Matrix einfach graphisch darstellen.
- In der Mitte der Kugel ist der Wärmestrom aufgrund der Symmetrie gleich Null.
- An der Oberfläche der Kugel liegt eine sogenannte Randbedingung dritter Art vor.
- Überlegen Sie sich, wie Sie die Randbedingungen in Ihr System integrieren können. Ihr System sollte folgende im Skript angegebene Struktur besitzen:

$$\left[\frac{\partial T}{\partial t}\right]^n = AT^n + b \tag{8.3}$$

- Wenn Sie Ihr System nun wie in Gleichung (8.3) beschrieben aufgestellt haben, können Sie leicht die Zeitdiskretisierung vornehmen. Beachten Sie hierfür die Regeln der Rechnung mit Matrizen und Vektoren!
- Sie können sich anhand des Stabilitätskriteriums der gegebenen DGL für das explizite Euler Verfahren die minimale stabile Zeitschrittweite berechnen. Sie sollten allerdings einen etwas kleineren Wert als den berechneten verwenden.

8.2 Implementieren Sie für oben genanntes Poblem ein **(implizites) Crank-Nicolson-Verfahren** und vergleichen Sie die Ergebnisse sowie Rechenzeiten. Was stellen Sie fest?

Hinweise:

- Ihr zu lösendes Gleichungssystem (8.4) samt Raumdiskretisierung (Matrix A) haben Sie bereits in Aufgabe 8.1 aufgestellt. Sie können nun einfach eine Zeitdiskretisierung nach Crank-Nicolson angeben:

$$\frac{T^{n+1} - T^n}{h} = \frac{1}{2}\left[(AT^n + b) + (AT^{n+1} + b)\right]. \tag{8.4}$$

  Diese gilt es nun nach T^{n+1} aufzulösen.
- Um Rechenzeit zu sparen sollten Sie so viele Matrizen und Vektoren wie möglich außerhalb der Iterationsschleife über der Zeit berechnen (z.B. die Matrix $\left(I - \frac{h}{2}A\right)$ oder den Vektor hb).

8.3 Bestimmen Sie die mittlere Temperatur der Kugel nach 60 Sekunden. Werten Sie hierzu das Integral

$$\bar{T} = \frac{1}{V}\int_V T dV = \frac{1}{V}\int_0^{2\pi}\int_0^{\pi}\int_0^{D/2} T r^2 \sin(\theta) dr d\theta d\phi = \frac{4\pi}{V}\int_0^{D/2} r^2 T dr \tag{8.5}$$

mit Hilfe der Trapezregel aus. Hierbei bezeichnet V das Volumen Ihrer Kugel.

8.3 Wichtige Matlab Befehle

Befehl	Beschreibung
`diag()`	Erstellt eine Diagonalmatrix oder gibt die Diagonalelemente einer Matrix aus.
`sparse(A)`	Transformiert die Matrix A zu einer sogenannten *sparse* Matrix welche nur noch die Werte ungleich Null speichert. Optimiert den Speicherbedarf und die Rechenzeit bei schwach besetzten Matrizen.
`spy()`	Plottet die Struktur einer Matrix wie in Abb. 8.4 gezeigt (Einträge ungleich null).

Index