Skript zur Vorlesung "Physik auf dem Computer"

JP Dr. A. Arnold
Universität Stuttgart
Institut für Computerphysik

unter Mithilfe von
Dr. O. Lenz

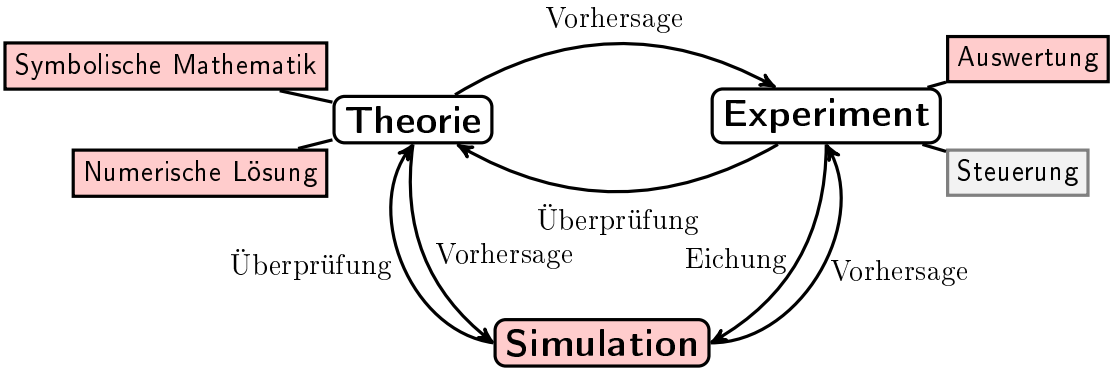
Sommersemester 2012

Dies ist das Skript zur Vorlesung „Physik auf dem Computer“, die von Axel Arnold im Sommersemester 2012 an der Universität Stuttgart gehalten wurde.

Inhaltsverzeichnis

1 Einleitung

In dieser Vorlesung geht es darum, wie der Computer in der modernen Physik eingesetzt wird, um neue Erkenntnisse zu gewinnen. Klassisch war die Physik ein Zusammenspiel aus Experiment und Theorie. Die Theorie macht Vorhersagen, die im Experiment überprüft werden. Umgekehrt kann im Experiment ein neuer Effekt beobachtet werden, für den die Theorie eine Erklärung liefert. Durch den Einsatz von Computern ist dieses Bild komplizierter geworden. In der folgenden Graphik sind die Bereiche farblich hinterlegt, in denen heutzutage Computer zum Einsatz kommen, die hellroten Bereiche werden in dieser Vorlesung behandelt:



Zu den klassischen Säulen Theorie und Experiment ist die *Simulation* als Mittelding zwischen Theorie und Experiment gekommen. Computersimulationen stellen Experimente im Computer nach, ausgehend von bekannten theoretischen Grundlagen. Praktisch alles kann simuliert werden, von Galaxien bis hin zu Elektronen und Quarks. Dazu gibt es eine Vielzahl an unterschiedlichen Methoden. Simulationen erfüllen zwei Hauptaufgaben: Simulationen können einerseits Experimente ziemlich genau reproduzieren, andererseits kann man mit Ihrer Hilfe theoretische Modelle in ihrer vollen Komplexität untersuchen.

Simulationen, die an ein Experiment angepasst (geeicht) sind, können zusätzliche Informationen liefern, die experimentell nicht zugänglich sind. Zum Beispiel kann man dort Energiebeiträge getrennt messen oder sehr kurzlebige Zwischenprodukte beobachten. Außerdem erlauben Simulationen, Wechselwirkungen und andere Parameter gezielt zu verändern, und damit Vorhersagen über zukünftige Experimente zu machen.

Simulationen, die auf theoretischen Modellen basieren, sind oft ein gutes Mittel, um notwendige Näherungen auf Plausibilität zu überprüfen oder um einen ersten Eindruck vom Verhalten dieses Modells zu erhalten. Damit können Simulationen auch helfen, zu entscheiden, ob notwendige Näherungen oder das Modell unvollständig ist, wenn Theorie und Experiment nicht zu einander passen.

In der klassischen theoretischen Physik werden Papier und Bleistift zunehmend vom Computer verdrängt, denn *Computeralgebra* ist mittlerweile sehr leistungsfähig und kann zum Beispiel in wenigen Sekunden komplexe Integrale analytisch lösen. Und falls eine Gleichung doch einmal zu kompliziert ist für eine analytische Lösung, so kann der Computer mit *numerischen Verfahren* oft sehr gute Näherungen finden.

In der experimentellen Physik fallen immer größere Datenmengen an. Der LHC erzeugt zum Beispiel pro Jahr etwa 10 Petabyte an Daten, also etwa 200 Millionen DVDs, was über mehrere Rechenzentren verteilt gespeichert und ausgewertet werden muss. Klar ist, dass nur Computer diese gigantischen Datenmengen durchforsten können. Aber auch bei einfacheren Experimenten helfen Computer bei der *Auswertung und Aufbereitung* der Daten, zum Beispiel durch Filtern oder statistische Analysen. Viele Experimente, nicht nur der LHC, sind aber auch so komplex, dass Computer zur *Steuerung* der Experimente benötigt werden, was wir in dieser Vorlesung aber nicht behandeln können. Die Auswertung und Aufbereitung der Daten hingegen wird besprochen, auch weil dies genauso auch für Computersimulationen benutzt wird.

Neben diesen direkten Anwendungen in der Physik ist der Computer mittlerweile natürlich auch ein wichtiges Mittel für den Wissensaustausch unter Physikern. Quasi alle wissenschaftlichen Arbeiten, wie etwa dieses Skript, werden heute nicht mit der Schreibmaschine und Schablonen erzeugt, sondern auf dem Computer. Die großen Verlage verlangen mittlerweile auch, Manuskripte als elektronische Dokumente zur Publikation einzureichen. Umgekehrt stehen wissenschaftliche Arbeit, vor allem Zeitschriftentexte, normalerweise nur noch in elektronischen Bibliotheken zur Verfügung, dafür aber Texte aus der gesamten Welt. Zur Suche in diesen riesigen Datenmengen dienen wiederum Computer. Und schließlich ist der Computer natürlich auch unverzichtbar, um international zusammenzuarbeiten - Brief und Telefon wären schlicht zu langsam und unflexibel. Diese Aspekte wurden aber schon in den Computergrundlagen behandelt, und sind nicht Teil dieser Vorlesung.

Um den Computer für Simulationen, Auswertung von Daten oder auch Lösung komplexer Differenzialgleichungen nutzen zu können, sind neben physikalischen Kenntnissen auch solche in Programmierung, numerischer Mathematik und Informatik gefragt. In diesem Skript geht es vor allem um die grundlegenden Methoden und wie diese angewandt werden, daher dominiert die numerische Mathematik etwas. Anders als in einer richtigen Vorlesung zur Numerik stehen hier aber die Methoden und Anwendungen anstatt der Herleitungen im Vordergrund.

1.1 Über dieses Skript

Im folgenden wird eine in der numerischen Mathematik übliche Notation benutzt. Wie auch in den meisten Programmiersprachen werden skalare und vektorielle Variablen nicht durch ihre Schreibweise unterschieden, allerdings werden üblicherweise die Namen i–l für (ganzzahlige) Schleifenindizes benutzt, n und m für Dimensionen. Da Schleifen sehr häufig auftreten, wird hierfür die Kurznotation Anfang(Inkrement)Ende benutzt. Zum

Beispiel bedeuten

$$1(1)n = 1,2,\ldots,n$$
$$n(-2)1 = n, n-2, \ldots, 3, 1.$$

Alle anderen Variablen sind reellwertige Skalare oder Vektoren, $\mathbb{R}^n$ bezeichnet dabei den n-dimensionalen Vektorraum reller Zahlen. Mit e_i wird dabei der Einheitsvektor der i-ten Spalte bezeichnet, mit e_i^T seine Transponierte, also der Einheitsvektor der i-ten Zeile.

Integrale werden mit dem Volumenelement am Ende geschrieben, dessen Dimensionalität sich aus dem Integrationsbereich erschließt. Sehr häufig werden Abschätzungen mit Hilfe der *Landau*-Symbole verkürzt. Wie üblich heißt für zwei Funktionen f und g

$$f = \mathcal{O}_{x\to a}(g) \quad \Longleftrightarrow \quad \lim_{x\to a} \frac{|f(x)|}{|g(x)|} < \infty.$$

In den meisten Fällen ist $a = 0$ oder $a = \infty$ und aus dem Kontext klar, welcher Grenzwert gemeint ist. Dann wird die Angabe weggelassen. Oft wird auch die Notation $f = g + \mathcal{O}(h)$ benutzt, um $f + g = \mathcal{O}(h)$ auszudrücken. $f(x) \doteq g(x)$ schließlich bedeutet $f(x) - g(x) = \mathcal{O}x \to 0(x)$.

Um einzelne Methoden konkret vorstellen zu können, wird in diesem Skript auf die Sprachen Python und C zurückgegriffen. Im Bereich des Hochleistungsrechnens werden vor allem die Sprachen Fortran und C/C++ eingesetzt, weil diese in Verbindung mit guten Compilern sehr effizienten Code ergeben. Allerdings bieten diese Sprachen keine nativen Datentypen wie zum Bespiel Listen oder Wörterbücher und verlangen die explizite Typisierung von Variablen, was Beispiele unnötig verkompliziert. Daher benutzt dieses Skript die Programmiersprache Python[1] mit den Erweiterungen NumPy und SciPy[2], die eine leistungsfähige numerische Bibliothek und umfangreiche Visualisierungsmöglichkeiten bietet. Für elementare Beispiele hingegen greift dieses Skript auf das hardwarenähere C zurück.

Zum Erlernen der Programmiersprachen Python und C sei auf die Materialien der Veranstaltung „Computergrundlagen“ hingewiesen, die im Fachbereich Physik der Universität Stuttgart jährlich angeboten wird.

Die meisten der in diesem Skript vorgestellten numerischen Methoden werden von SciPy direkt unterstützt. Die Qualität dieser Implementationen ist mit eigenem Code nur schwierig zu überbieten. Wo immer möglich, wird daher auf die entsprechend SciPy-Befehle verwiesen. Zum Beispiel wird auf die Funktion „method“ im SciPy-Modul „library“ in der in der Form `scipy.library.method(arg1, arg2, ...)` verwiesen. Trotzdem sind viele Methoden auch als expliziter Code gezeigt, da man natürlich eine Vorstellung davon haben sollte, was diese Methoden tun.

1.2 Beispiel: Fadenpendel

Wir betrachten ein einfaches Beispielsystem, nämlich ein Fadenpendel. Wird dieses Pendel nun ausgelenkt, vollführt es eine periodische Schwingung um die Ruhelage, d.h., den

[1] www.python.org
[2] http://www.scipy.org

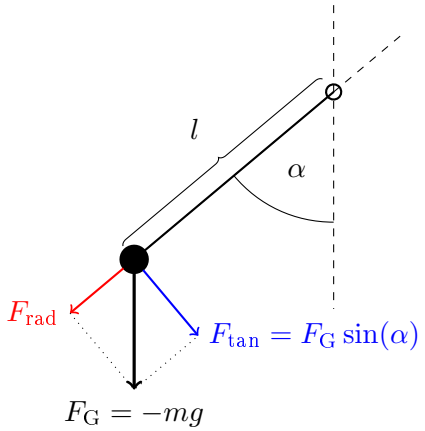


Abbildung 1.1: Schematisches Fadenpendel der Masse m, das an einem masselosen, steifen Faden der Länge l hängt.

tiefsten Punkt. Unser Ziel als Physiker ist nun, die Position der Kugel als Funktion der Zeit vorherzusagen. Das allerdings ist eine unmögliche Aufgabe — man stelle sich zum Beispiel eine stark inhomogene Masse vor (oder ein Fadenpendel als Masse) oder dass der Faden elastisch ist. Daher müssen wir zunächst ein geeignet vereinfachtes *Modell* erstellen, auf das wir dann die bekannten physikalischen Gesetze anwenden können.

1.2.1 Modell

Als Modell wählen wir eine homogene Kugel der Masse m, die an einem masselosen, steifen Faden der Länge l hängt (vergleiche Figur 1.1). Auf diese Kugel wirkt nur eine Gewichtskraft der Größe mg senkrecht nach, alle anderen Kräfte vernachlässigen wir komplett, insbesondere auch die Reibung.

Da der Faden unendlich steif sein soll, kann sich Kugel lediglich auf einem Kreis mit Radius l um die Aufhängung bewegen, d.h. die Position der Kugel ist durch die Auslenkung α aus dem tiefsten Punkt vollständig beschrieben. Weiter wird die Komponente der Kraft parallel zum Faden komplett von diesem kompensiert, daher bleibt bei Auslenkung α von der Gewichtskraft nur ihre Komponente

$$F_{\text{tan}} = F_{\text{G}} \sin(\alpha) = -mg \sin(\alpha) \tag{1.1}$$

senkrecht zum Faden übrig. Das Newtonsche Gesetz besagt nun, dass die Tangentialbeschleunigung, also die Beschleunigung entlang α

$$l\ddot{\alpha} = F_{\text{tan}}/m = -g \sin(\alpha) \tag{1.2}$$

beträgt. Dies ist jetzt eine Differentialgleichung für die Auslenkung $\alpha(t)$ des Pendels als Funktion der Zeit. Diese wiederum liefert uns die gewünschte Position $(\cos(\alpha)l, \sin(\alpha)l)$ der Kugel relativ zur Aufhängung als Funktion der Zeit. Leider hat selbst diese einfache Differentialgleichung keine geschlossene Lösung, und wir müssen weitere Näherungen einführen, um eine analytische Lösung zu erhalten.

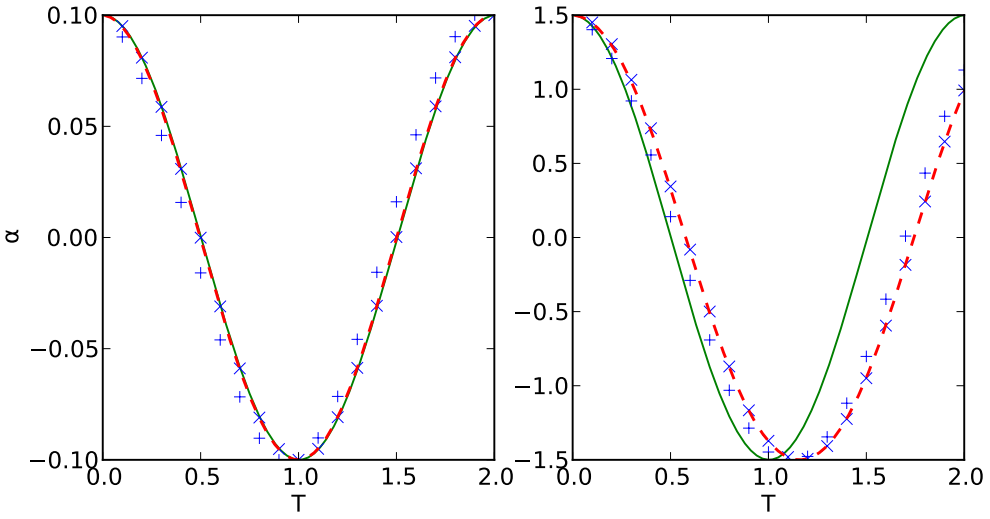


Abbildung 1.2: Lösungen für ein Fadenpendel der Länge $l = 1m$. Im linken Graphen ist die Ausgangslage $\alpha = 1{,}5$, im rechten $\alpha = 0{,}1$; in beiden Fällen ist die Ausgangsgeschwindigkeit 0. Die durchgezogene grüne Linie markiert die analytische Näherungslösung (1.4) für kleine Winkel. $\times$ markiert die Ergebnisse einer Integration mit dem einfachen Vorwärtsschritt (1.11) mit Zeitschritt $0{,}1s$, die gestrichelte rote Linie mit Zeitschritt $0{,}01s$. $+$ markiert die Lösung mit Hilfe des Velocity-Verlet-Algorithmus und Zeitschritt $0{,}1s$.

1.2.2 Näherung: der harmonische Oszillator

Für kleine Winkel gilt $\sin(\alpha) \approx \alpha$, und damit

$$\ddot{\alpha} \approx= -\frac{g}{l}\alpha. \tag{1.3}$$

Diese Differentialgleichung hat die allgemeine Lösung

$$\alpha(t) = A\sin(\omega t + \phi) \tag{1.4}$$

mit $\omega = \sqrt{g/l}$, wie man sich leicht durch Einsetzen überzeugt. Die Größen A und ϕ ergeben sich aus den Anfangsbedingungen, nämlich der Anfangsposition

$$\alpha_0 = A\sin(\phi) \tag{1.5}$$

und -geschwindigkeit

$$v_0 = A\omega\cos(\phi). \tag{1.6}$$

Ist zum Beispiel $v_0 = 0$, so ist $\phi = \pi/2$ und $A = \alpha_0$, im allgemeinen Fall ist

$$\phi = \arctan\left(\frac{\alpha_0\omega}{v_0}\right) \quad \text{und} \quad A = \frac{\alpha_0}{\sin(\phi)}. \tag{1.7}$$

Wir haben nun eine geschlossene Lösung für die Position des Pendels, so lange die Ausgangslage nicht zu sehr ausgelenkt ist. Um diese Lösung zu visualisieren, nutzt man heute üblicherweise den Computer, siehe Graph 1.2.

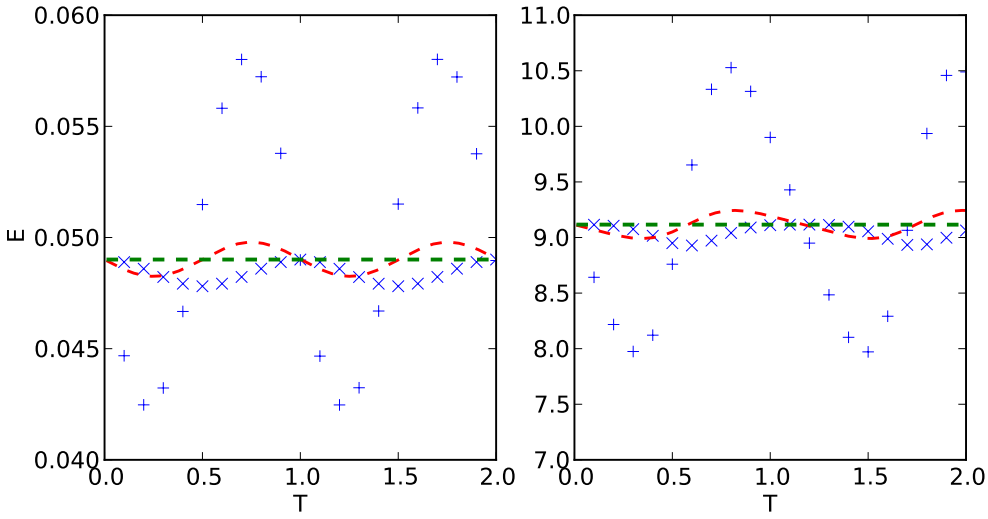


Abbildung 1.3: Energie als Funktion der Zeit, wieder für $l = 1m$, und Ausgangslage $\alpha = 1{,}5$ (links) und $\alpha = 0{,}1$ (rechts) in Ruhe. + markiert die Ergebnisse einer Integration mit dem einfachen Vorwärtsschritt (1.11) mit Zeitschritt $0{,}1s$, die gestrichelte rote Linie mit Zeitschritt $0{,}01s$. $\times$ markiert die Lösung mit Hilfe des Velocity-Verlet-Algorithmus und Zeitschritt $0{,}1s$, und die gestrichelte blaue Linie mit $0{,}01s$.

1.2.3 Numerische Lösung

Was passiert nun, wenn das System stärker ausgelenkt ist? Mit sehr viel mehr Aufwand lässt sich auch für diesen Fall eine analytische Lösung finden, allerdings in Form einer Reihe, die nicht mehr so einfach zu zeichnen ist. Eine Alternative ist, die Differentialgleichung (1.2) mit Hilfe des Computers zu berechnen. Wir sagen, wir „simulieren“ das Pendel. Dazu fixieren wir ein Einheitensystem, zum Beispiel eine Sekunde als Zeiteinheit und einen Meter als Längeneinheit. In diesem System ist also $l = 1$, $g \approx 9{,}81$ und $\omega \approx 3{,}13$, falls des Pendel einen Meter lang ist.

Zunächst müssen wir das Problem aber für den Computer anpassen, der ja nur mit (endlich vielen) gewöhnlichen Zahlen rechnen kann, wir müssen das Problem *diskretisieren*. Wir betrachten nur die Zeitpunkte

$$t_n = n\delta t, n = 1(1)N, \tag{1.8}$$

wobei der Zeitschritt δt frei wählbar ist. Je kleiner δt, desto genauer können wir $\alpha(t)$ bestimmen, allerdings steigt natürlich die Anzahl der Schritte, die nötig sind, um eine feste Gesamtzeit zu erreichen. Unsere Lösung, die Funktion $\alpha(t)$ wird also durch ihre Werte $\alpha(t_n)$ an den diskreten Zeitpunkten dargestellt.

Um Gleichung (1.2) auf den Computer zu bringen, müssen wir uns allerdings noch überlegen, wie wir mit der Ableitung verfahren. Da wir die Ausgangsposition und -ge-

schwindigkeit gegeben haben, liegt es nahe, die Gleichung zu integrieren:

$$v(t+\delta t) = \dot{\alpha}(t+\delta t) = v(t) + \int_t^{t+\delta t} -\omega^2 \sin\alpha(\tau)\, d\tau. \tag{1.9}$$

Da δt aber unser Zeitschritt ist, wir also nichts weiter über $\alpha(\tau)$ wissen, bietet sich die folgende Näherung an:

$$v(t+\delta t) \approx v(t) - \omega^2 \sin\alpha(t)\delta t. \tag{1.10}$$

Analog ergibt sich dann durch nochmalige Integration:

$$\alpha(t+\delta t) \approx \alpha(t) + v(t)\delta t. \tag{1.11}$$

Ausgehend von

$$\alpha(0) = \alpha_0 \quad \text{und} \quad v(0) = v_0 \tag{1.12}$$

lässt sich damit also $\alpha(t)$ numerisch bestimmen. Der Quellcode 1.1 zeigt, wie eine einfach Implementation in Python aussehen könnte.

Wie kann man nun überprüfen, ob diese Lösung tatsächlich korrekt ist? Da das System abgeschlossen ist, muss seine Energie

$$E = \frac{1}{2} l^2 v(t)^2 + gl(1 - \cos(\alpha(t))) \tag{1.13}$$

erhalten sein. Lässt man sich diese allerdings ausgeben, stellt man fest, dass $E(t)$ erheblich schwankt, vergleiche Graph 1.3. Dies lässt sich durch Verringern des Zeitschritts beheben, das kostet aber entsprechend mehr Rechenzeit.

Eine bessere Alternative ist, den Algorithmus zu verbessern, was wiederum etwas analytische Arbeit erfordert. Wir betrachten die Taylorentwicklungen

$$\alpha\left(t+\delta t\right) = \alpha\left(t+\frac{\delta t}{2}\right) + \frac{\delta t}{2} v\left(t+\frac{\delta t}{2}\right) + \frac{\delta t^2}{8} F\left(t+\frac{\delta t}{2}\right) + \mathcal{O}(\delta t^3) \tag{1.14}$$

und

$$\alpha\left(t\right) = \alpha\left(t+\frac{\delta t}{2}\right) - \frac{\delta t}{2} v\left(t+\frac{\delta t}{2}\right) + \frac{\delta t^2}{8} F\left(t+\frac{\delta t}{2}\right) - \mathcal{O}(\delta t^3). \tag{1.15}$$

Durch Subtraktion ergibt sich

$$\alpha(t+\delta t) = \alpha(t) + \delta t\, v\left(t+\frac{\delta t}{2}\right) + \mathcal{O}(\delta t^4). \tag{1.16}$$

Die Geschwindigkeiten an den halben Zeitschritten erhält man einfach durch $v(t+\delta t/2) = v(t) + \delta t F(t)/2$. Zusammengefasst ergibt sich der folgende *Velocity-Verlet-Algorithmus*:

$$v(t+\frac{\delta t}{2}) = v(t) + \frac{\delta t}{2} F(t) \tag{1.17}$$

$$\alpha(t+\delta t) = \alpha(t) + v(t+\frac{\delta t}{2}) \tag{1.18}$$

$$v(t+\delta t) = v(t+\frac{\delta t}{2}) + \frac{\delta t}{2} F(t+\delta t), \tag{1.19}$$

der anders als die direkte Vorgehensweise vorher numerisch stabil ist und quasi keine Energieschwankungen aufzeigt, vergleiche Graph 1.3. Im Quellcode 1.1 ist alternativ auch dieser Integrator implementiert. Obwohl er nur unwesentlich komplizierter ist als der einfache Integrator zuvor, erreicht etwa dieselbe Genauigkeit wie dieser mit einem Zehntel der Zeitschritte.

Als weiterer Test bietet sich an, bei kleinen Auslenkungen mit der analytisch bekannten Lösung zu vergleichen, die gut reproduziert wird, siehe Graph 1.2. Bei größeren Anfangsauslenkungen oder -geschwindigkeiten ist die Abweichung allerdings sehr groß, weil hier die analytische Näherung versagt. Im Rahmen ihrer Genauigkeit erlaubt also die numerische Lösung, das vorgegebene Modell in einem größeren Parameterraum auf sein Verhalten hin zu untersuchen, als analytisch möglich wäre.

```
# Simulation der Bahn eines Fadenpendels
##########################################
import scipy as sp
import matplotlib.pyplot as pyplot

# Laenge des Pendelarms
l=1
# Erdbeschleunigung
g = 9.81
# Zeitschritt
dt = 0.01
# Zeitspanne
T = 2
# Methode, "simple" oder "velocity-verlet"
integrator="velocity-verlet"
# (Start-)Position
a = 0.1
# (Start-)Winkelgeschwindigkeit
da = 0
# Zeit
t = 0

# Tabellen fuer die Ausgabe
tn, an, En = [], [], []

# Kraft, die auf die Kugel wirkt
def F(a):
    return -g/l*sp.sin(a)

while t < T:
    if integrator == "simple":
        da += F(a)*dt
        a += da*dt
    elif integrator == "velocity-verlet":
        da += 0.5*F(a)*dt
        a += da*dt
        da += 0.5*F(a)*dt
    t += dt
    tn.append(t)
    an.append(a)
    En.append(0.5*(l*da)**2 + g*(l - l*sp.cos(a)))

# Ausgabe von Graphen
ausgabe = pyplot.figure(figsize=(8,4))

loesung = ausgabe.add_subplot(121)
loesung.set_xlabel("T")
loesung.set_ylabel("Winkel")
loesung.plot(tn, an)

energie = ausgabe.add_subplot(122)
energie.set_xlabel("Zeit")
energie.set_ylabel("Energie")
energie.plot(tn, En)

pyplot.show()
```

Listing 1.1: Python-Code zum Fadenpendel mit graphisch aufbereiteter Ausgabe mit Hilfe der `matplotlib`.

2 Lineare Algebra I

Lineare Gleichungssysteme sind die einfachste Art von Gleichungssystemen, für die sich zum Beispiel leicht bestimmen lässt, ob und wieviele Lösungen es gibt. Daher führt man auch die Lösung komplexerer Probleme, wie zum Beispiel Differentialgleichungen, oft auf die Lösung eines Satzes von linearen Gleichungssystemen zurück. Lineare Gleichungssysteme sind in diesem Sinne eine der wesentlichen Grundlagen der numerischen Mathematik. Der händischen Lösung der Systeme steht dabei vor allem ihre Größe im Weg — Finite-Elemente-Rechnungen können leicht die Lösung von Gleichungssystemen mit Millionen von Variablen erfordern. Mit modernen Algorithmen und Computern lassen sich solche Gleichungssysteme allerdings schnell und zuverlässig lösen. In diesem Kapitel lernen wir die grundlegende Methode zum Lösen von Gleichungssystemen kennen, nämlich die allgemeine, aber langsame Gaußelimination. Daneben lernen wir noch die LU-Zerlegung und die Choleskyzerlegung kennen, die mit etwas Vorarbeit eine effizientere Lösung erlauben und im folgenden oft zum Einsatz kommen werden.

Wir betrachten also folgendes Problem: Sei $A = (a_{ik}) \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Gesucht ist die Lösung $x \in \mathbb{R}^n$ des Gleichungssystems

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m \end{array} \tag{2.1}$$

oder kurz $Ax = b$. In dieser allgemeinen Form ist weder garantiert, dass es eine Lösung gibt (z.B. $A = 0$, $b \neq 0$), noch, dass diese eindeutig ist ($A = 0$, $b = 0$).

2.1 Dreiecksmatrizen

Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt eine *rechte obere Dreiecksmatrix*, wenn sie quadratisch ist und $a_{ij} = 0$ für $i > j$. Analog kann man auch die linken unteren Dreiecksmatrizen definieren, mit $a_{ij} = 0$ für $i < j$. In jedem Fall bilden rechte obere und linke untere Dreiecksmatrizen jeweils Unteralgebren der Matrixalgebra, d.h., sie sind abgeschlossen unter Addition und Multiplikation. Die Schnittmenge dieser Algebren ist wiederum die Algebra der *Diagonalmatrizen*.

Ist A eine rechte obere Dreiecksmatrix, so hat das Gleichungssystem die Form

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ & & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & a_{nn}x_n & = & b_n. \end{array} \tag{2.2}$$

Dieses Gleichungssystem hat genau dann eine Lösung, wenn A regulär ist, also $\det A = \prod_{i=1}^{n} a_{ii} \neq 0$. Die Lösung kann dann durch *Rücksubstitution* direkt bestimmt werden:

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{k=i+1}^{n} a_{ik} x_k \right) \quad \text{für } i = n(-1)1. \tag{2.3}$$

Für reguläre *linke untere Dreiecksmatrizen* ergibt sich die Lösung entsprechend durch *Vorwärtssubstitution*:

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{k=1}^{i-1} a_{ik} x_k \right) \quad \text{für } i = 1(1)n. \tag{2.4}$$

Für Diagonalmatrizen ist die Situation natürlich einfacher, es gilt

$$x_i = \frac{1}{a_{ii}} b_i \quad \text{für } i = 1(1)n. \tag{2.5}$$

SciPy stellt für Dreiecksmatrizen spezielle Löserroutinen zur Verfügung, `scipy.linalg.solve_triangular(A, b, lower=False)`, wobei `lower` angibt, ob `A` eine linke untere statt rechte obere Dreiecksmatrix ist.

2.2 Gaußelimination

Die Gaußelimination ist ein Verfahren, um eine beliebiges Gleichungssystem $Ax = b$, mit $A \in \mathbb{R}^{m \times n}$, auf die äquivalente Form

$$\begin{pmatrix} R & K \\ 0 & 0 \end{pmatrix} x' = b' \tag{2.6}$$

zu bringen, wobei R eine reguläre rechte obere Dreiecksmatrix und $K \in \mathbb{R}^{k \times l}$ beliebig ist, und x' eine Permutation (Umordnung) von x. Dieses Gleichungssystem hat offenbar nur dann eine Lösung, wenn $b'_i = 0$ für $i = m - k + 1(1)m$.

Diese ist im allgemeinen auch nicht eindeutig, vielmehr können die freien Variablen $x_{\mathrm{K}} = (x'_i)_{i=n-k+1}^{n}$ frei gewählt werden. Ist $x_{\mathrm{R}} = (x'_i)_{i=1}^{n-k}$ der Satz der verbleibenden Lösungsvariablen, so gilt also

$$x_{\mathrm{L}} = R^{-1} b' - R^{-1} K x_{\mathrm{K}}.$$

Die Lösungen ergeben sich daraus als

$$x' = \begin{pmatrix} R^{-1} b' \\ 0 \end{pmatrix} + \left\langle \begin{pmatrix} -R^{-1} K_i \\ e_i \end{pmatrix} \right\rangle, \tag{2.7}$$

wobei K_i die i-te Spalte von K und $\langle\rangle$ den aufgespannten Vektorraum bezeichnet. Die Ausdrücke, die R^{-1} enthalten, können durch Rücksubstitution bestimmt werden.

Um das System $Ax = b$, das wir im folgenden als $A|b$ zusammenfassen, auf diese Form zu bringen, stehen folgende Elementaroperationen zur Verfügung, die offensichtlich die Lösung nicht verändern:

1. Vertauschen zweier Gleichungen (Zeilentausch in $A|b$)
2. Vertauschen zweier Spalten in x und A (Variablenaustausch)
3. Addieren eines Vielfachen einer Zeile zu einer anderen
4. Multiplikation einer Zeile mit einer Konstanten ungleich 0

Die Gausselimination nutzt nun diese Operationen, um die Matrix spaltenweise auf die gewünschte Dreiecksform zu bringen. Dazu werden Vielfache der ersten Zeile von allen anderen abgezogen, so dass die Gleichung die Form

$$\left(\begin{array}{cccc|c} a_{11}^{(0)} & a_{12}^{(0)} \dots & a_{1n}^{(0)} & & b_1^{(0)} \\ 0 & a_{22}^{(1)} \dots & a_{2n}^{(1)} & & b_2^{(1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{m2}^{(1)} \dots & a_{mn}^{(1)} & & b_m^{(1)} \end{array}\right) =: A^{(1)}|b^{(1)} \tag{2.8}$$

annimmt, wobei

$$\begin{array}{rcll} a_{ik}^{(1)} & = & a_{ik}^{(0)} - l_i^{(1)} a_{1k}^{(0)} & \text{für } i = 2(1)n, k = 1(1)m \\ b_i^{(1)} & = & b_i^{(0)} - l_i^{(1)} b_1^{(0)} & \text{für } i = 2(1)n \qquad \text{mit } l_i^{(1)} = \dfrac{a_{i1}^{(0)}}{a_{11}^{(0)}}. \\ a_{1k}^{(1)} & = & a_{1k}^{(0)}, \quad b_1^{(1)} = b_1^{(0)} & \text{sonst} \end{array} \tag{2.9}$$

Mit dem verbleibenden Resttableau wird nun genauso weiter verfahren:

$$\begin{array}{rcll} a_{ik}^{(r)} & = & a_{ik}^{(r-1)} - l_i^{(r)} a_{r,k}^{(r-1)} & \text{für } i = r+1(1)n, k = r(1)m \\ b_i^{(r)} & = & b_i^{(r-1)} - l_i^{(r)} b_r^{(r-1)} & \text{für } i = r+1(1)n \qquad \text{mit } l_i^{(r)} = \dfrac{a_{ir}^{(r-1)}}{a_{rr}^{(r-1)}}. \\ a_{ik}^{(r)} & = & a_{ik}^{(r-1)}, \quad b_i^{(r)} = b_i^{(r-1)} & \text{sonst} \end{array} \tag{2.10}$$

Das Verfahren ist beendet, wenn das Resttableau nur noch eine Zeile hat.

Ist während eines Schrittes $a_{rr}^{(r-1)} = 0$ und

1. nicht alle $a_{ir}^{(r-1)} = 0$, $i = r+1(1)m$. Dann tauscht man Zeile r gegen eine Zeile i mit $a_{ir}^{(r-1)} \neq 0$, und fährt fort.
2. alle $a_{ir}^{(r-1)} = 0$, $i = r(1)m$, aber es gibt ein $a_{ik}^{(r-1)} \neq 0$ mit i,$k \geq r$. Dann vertauscht man zunächst Zeile r mit Zeile i, tauscht anschließend Spalte k mit Spalte r, und fährt fort.
3. alle $a_{ik}^{(r-1)} = 0$ für i,$k \geq r$. Dann hat $A^{(r-1)}|b^{(r-1)}$ die gewünschte Form (2.6) erreicht, und das Verfahren terminiert.

Das Element $a_{rr}^{(r-1)}$ heißt auch *Pivotelement*, da es sozusagen der Dreh- und Angelpunkt des iterativen Verfahrens ist. In der Praxis ist es numerisch günstiger, wenn dieses Element möglichst groß ist. Das lässt sich erreichen, in dem wie in den singulären Fällen verfahren wird, also Zeilen oder Spalten getauscht werden, um das betragsmäßig maximale $a_{ik}^{(r-1)}$ nach vorne zu bringen. Folgende Verfahren werden unterschieden

- *kanonische Pivotwahl*: es wird stets $a_{rr}^{(r-1)}$ gewählt und abgebrochen, falls dieses betragsmäßig zu klein wird. Diese Verfahren scheitert schon bei einfachen Matrizen (z.B. $\left(\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right)$), und kann daher nur eingesetzt werden, wenn die Struktur der Matrix sicherstellt, das $a_{rr}^{(r-1)}$ stets hinreichend groß ist.

- *Spaltenpivotwahl*: es wird wie oben im 1. Fall nur in der Spalte maximiert, d.h. wir wählen als Pivotelement
$$i_0 = \text{argmax}_{i>r}|a_{ir}^{(r-1)}| \tag{2.11}$$
und tauschen Zeilen i_0 und r; die Variablenreihenfolge bleibt unverändert. Ist die Matrix A quadratisch, bricht das Verfahren genau dann ab, wenn A singulär ist.

- *Totalpivotwahl*: wie oben im 2. Fall wird stets das maximale Matrixelement im gesamten Resttableau gensucht, also
$$i_0,k_0 = \text{argmax}_{i,k>r}|a_{ik}^{(r-1)}|. \tag{2.12}$$
Dann vertauscht man zunächst Zeile r mit Zeile i_0, und tauscht anschließend Spalte k_0 mit Spalte r, wobei man sich noch die Permutation der Variablen geeignet merken muss, zum Beispiel als Vektor von Indizes.

Unabhängig von der Pivotwahl benötigt die Gaußelimination bei quadratischen Matrizen im wesentlichen $\mathcal{O}(n^3)$ Fließkommaoperationen. Das ist relativ langsam, daher werden wir später bessere approximative Verfahren kennenlernen. Für Matrizen bestimmter Struktur, zum Beispiel Bandmatrizen, ist die Gaußelimination aber gut geeignet. NumPy bzw. SciPy stellen daher auch keine Gaußelimination direkt zur Verfügung. `scipy.linalg.solve(A, b)` ist ein Löser für Gleichungssysteme $Ax = b$, der immerhin auf der LU-Zerlegung durch Gaußelimination basiert. Dieser Löser setzt allerdings voraus, dass die Matrix nicht singulär ist, also eindeutig lösbar.

2.3 Matrixinversion

Ist $A \in \mathbb{R}^{n\times n}$ regulär, so liefert die Rücksubstitution implizit die Inverse von A, da für beliebige b das Gleichungssystem $Ax = b$ gelöst werden kann. Allerdings muss das für jedes b von neuem geschehen. Alternativ kann mit Hilfe der Gaußelimination auch die Inverse von A bestimmt werden. Dazu wird das Tableau $A|I$ in das Tableau $I|A^{-1}$ transformiert, wobei I die $n\times n$-Einheitsmatrix bezeichnet. Die Vorgehensweise entspricht zunächst der Gaußelimination mit Spaltenpivotwahl. Allerdings werden nicht nur die Elemente unterhalb der Diagonalen, sondern auch oberhalb eliminiert. Zusätzlich wird die Pivotzeile noch mit $1/a_{ii}^{(i-1)}$ multipliziert, so dass das A schrittweise die Form
$$\begin{pmatrix} 1 & 0 & a_{12}^{(2)} & \dots & a_{1n}^{(2)} \\ 0 & 1 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & 0 & a_{32}^{(2)} & \dots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n2}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix} \tag{2.13}$$

annimmt. Das Verfahren ist allerdings numerisch nicht sehr stabil, und generell sollte die explizite Berechnung der Inversen wann inner möglich vermieden werden. SciPy stellt die Matrixinversion als Funktion **`scipy.linalg.inv(A)`** zur Verfügung.

Eine Ausnahme bilden Matrizen der Form $I + A$ mit $\|A\| = \max\|Ax\|/\|x\| < 1$. Dann ist

$$(I + A)^{-1} = I - A + A^2 - A^3 + \cdots \tag{2.14}$$

eine gut konvergierende Näherung der Inversen.

2.4 LU-Zerlegung

Eine weitere Anwendung der Gaußelimination ist die LU-Zerlegung von bestimmten quadratischen Matrizen. Dabei wird eine Matrix $A \in \mathbb{R}^{n\times n}$ so in eine ein linke untere Dreiecksmatrix L und eine rechte obere Dreiecksmatrix U zerlegt, dass $A = L \cdot U$. Um die LU-Zerlegung eindeutig zu machen, vereinbart man üblicherweise, dass $l_{ii} = 1$ für $i = 1(1)n$. Das U steht übrigens für das englische „upper right“ und L für „lower left“. Im Deutschen findet sich vereinzelt noch der Begriff LR-Zerlegung, wobei hier L für eine linke untere und R für eine rechte obere Matrix steht.

Ist eine solche Zerlegung einmal gefunden, lässt sich das Gleichungssystem $Ax = b$ für beliebige b effizient durch Vorwärts- und Rücksubstitution lösen:

$$Ly = b, Ux = y \quad \Longrightarrow \quad Ax = LUx = Ly = b. \tag{2.15}$$

Zunächst wird also y durch Vorwärtssubstitution berechnet, anschließend x durch Rückwärtssubstitution. Die Inverse lässt sich so auch bestimmen:

$$Ly_i = e_i, Ux_i = y_i \quad \text{für } i = 1(1)n \quad \Longrightarrow \quad A^{-1} = (x_1, \ldots, x_n)\,. \tag{2.16}$$

Die Determinante von $A = L \cdot U$ ist ebenfalls einfach zu bestimmen:

$$\det A = \det L \, \det U = \prod_{i=1}^{n} u_{ii} \tag{2.17}$$

Um die LU-Zerlegung zu berechnen, nutzen wir wieder die Gaußelimination. Kann bei $A \in \mathbb{R}^{n\times n}$ die Gaußelimination in kanonischer Pivotwahl durchgeführt werden, so ist die LU-Zerlegung von A durch $U = A^{(n-1)}$, also die finale, auf rechte obere Dreiecksform transformierte Matrix, und durch die Matrix

$$L = \begin{pmatrix} 1 & & & & 0 \\ l_1^{(0)} & 1 & & & \\ l_2^{(0)} & l_2^{(1)} & 1 & & \\ \vdots & & \ddots & \ddots & \\ l_n^{(0)} & \ldots & \ldots & l_n^{(n-1)} & 1 \end{pmatrix} \tag{2.18}$$

der Updatekoeffizienten aus (2.10) gegeben.

Wie bereits gesagt, ist die Voraussetzung, dass die Gaußelimination mit kanonischer Pivotwahl durchgeführt werden kann, stark, und schließt selbst einfache Matrizen wie $\left(\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right)$ aus. Wie man sich leicht überlegt, besitzt diese Matrix allerdings keine LU-Zerlegung.

Für manche Anwendungen ist es günstiger, wenn L und U normiert sind. Dann benutzt man die LDU-Zerlegung $A = LDU$, mit L linker unterer Dreiecksmatrix, D Diagonalmatrix und U rechter oberer Dreiecksmatrix. Jetzt müssen $l_{ii} = 1$ und $r_{ii} = 1$ sein. Die LDU-Zerlegung ergibt sich aus der LU-Zerlegung durch $d_{ii} = u_{ii}$ und $u'_{ik} = u_{ik}/u_{ii}$.

In SciPy ist die LU-Zerlegung in den Funktionen **`scipy.linalg.lu_factor(A)`** oder **`scipy.linalg.lu(A)`** (zur Zerlegung der Matrix A) und **`scipy.linalg.lu_solve((lu,piv), b)`** (zum Lösen des LGS) implementiert.

2.5 Cholesky-Zerlegung

Wir betrachten im folgenden nur symmetrische, positiv definite Matrizen, wie sie gerade in der Physik oft vorkommen. Auch in der Optimierung spielen diese eine wichtige Rolle. Sei $A = LDU$ eine LDU-Zerlegung einer symmetrischen Matrix, dann gilt

$$LDU = A = A^T = (LDU)^T = U^T D L^T. \tag{2.19}$$

Da die LDU-Zerlegung aber eindeutig ist und U^T eine normierte, linke untere Dreiecksmatrix und L^T eine normierte, rechte obere Dreiecksmatrix, so gilt $U = U^T$, und damit

$$A = U^T DU = \widehat{U}^T \widehat{U} \quad \text{mit } \widehat{U} = \operatorname{diag}(\sqrt{d_{ii}})U. \tag{2.20}$$

Dies ist die Cholesky-Zerlegung. Anstatt die Gaußelimination durchzuführen, lässt sich die Zerlegung aber auch direkt mit Hilfe des *Cholesky-Verfahrens* bestimmen: Sei $A = \widehat{R}^T \widehat{R}$ eine Cholesky-Zerlegung. Da $\widehat{R}$ unterhalb der Diagonalen nur 0 enthält, gilt

$$a_{ik} = \sum_{l=1}^{i} \hat{r}_{li}\hat{r}_{lk} \quad \text{für } i = 1(1)n, k = 1(1)n. \tag{2.21}$$

Daraus lässt sich die erste Zeile von $\widehat{R}$ direkt ablesen:

$$\hat{r}_{11} = \sqrt{a_{11}} \quad \text{und } \hat{r}_{1k} = \frac{a_{1k}}{\hat{r}_{11}} \quad \text{für } k = 2(1)n. \tag{2.22}$$

Die nächsten Zeilen lassen sich analog bestimmen, da für jedes i

$$a_{ii} = \sum_{l=1}^{i} \hat{r}_{li}^2 \quad \Longrightarrow \quad \hat{r}_{ii} = \sqrt{a_{ii} - \sum_{l=1}^{i-1} \hat{r}_{li}^2}. \tag{2.23}$$

Für die restlichen Elemente der Zeile gilt

$$\hat{r}_{ik} = \frac{1}{\hat{r}_{ii}} \left(a_{ik} - \sum_{l=1}^{i-1} \hat{r}_{li}\hat{r}_{lk} \right) \quad \text{für } k = i + 1(1)n \tag{2.24}$$

Das Cholesky-Verfahren ist wie die Gaußelimination von der Ordnung $\mathcal{O}(n^3)$, braucht aber nur halb so viele Operationen. In SciPy ist die Cholesky-Zerlegung als **`scipy.linalg.cholesky(A)`** implementiert.

2.6 Bandmatrizen

Im folgenden werden wir oft mit k-Bandmatrizen zu tun haben, also Matrizen, bei denen nur die Diagonale und einige Nebendiagonalen besetzt sind. Diagonalmatrizen sind also 1-Bandmatrizen, eine *Dreibandmatrix* hat die Form

$$\begin{pmatrix} d_1 & t_1 & & & 0 \\ b_1 & d_2 & t_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & d_{n-1} & t_{n-1} \\ 0 & & & b_{n-1} & d_n \end{pmatrix}. \tag{2.25}$$

Für Matrizen dieser Form ist die Gaußelimination mit kanonischer Pivotwahl sehr effizient, da pro Iteration jeweils nur die erste Zeile des Resttableaus verändert werden muss. Dadurch ist der Rechenaufwand nur noch linear in der Matrixgröße bzw. Länge der Bänder. Die Dreiecksmatrizen L und U der LU-Zerlegung sind zusätzlich (Drei-) Bandmatrizen, wobei L nur auf der Haupt und der unteren Nebendiagonalen von Null verschiedene Einträge hat, U nur auf der Diagonalen und der Nebendiagonalen oberhalb.

SciPy stellt für Bandmatrizen ebenfalls spezielle Löserroutinen zur Verfügung, `scipy.linalg.solve_banded((l,u), A, b)`, wobei `l` und `u` die Anzahl der Nebendiagonalen oberhalb und unterhalb angeben, und `A` die Matrix in Bandform angibt.

3 Darstellung von Funktionen

Auch moderne Prozessoren beherrschen nur die Grundrechenarten. Wie kann man also auf einem Computer kompliziertere Funktionen berechnen, wie z.B. die Sinusfunktion?

Beispielsweise könnte man die Funktionen als Vektor von Funktionswerten speichern. Für die graphische Darstellung reicht das aus, aber um Funktionen mit wenigstens sechs Stellen Genauigkeit im Computer bereitzustellen, wären Millionen von Stützstellen nötig.

Daher müssen bessere Darstellungen für Funktionen genutzt werden. Um beliebige Funktionen auf dem Computer berechnen zu können, führt man diese meist auf (stückweise definierte) Polynome zurück, die nur mit Hilfe der Grundrechenarten berechnet werden können. Dies ist selbst dann der Fall, wenn ein Prozessor gewisse Funktionen scheinbar in Hardware implementiert hat; tatsächlich führt dieser intern die notwendigen elementaren Operationen durch.

3.1 Horner-Schema

Die naive Auswertung eines Polynoms $\sum_{i=0}^{n} c_i x^i$ mit $n+1$ Termen bzw. vom Grad n benötigt n Additionen und $2n$ Multiplikationen sowie einen Zwischenspeicher für die Potenzen x^i des Arguments x. Besser ist die Auswertung des Polynoms nach dem Horner-Schema:

$$\sum_{i=0}^{n} c_i x^i = c_0 + x(c_1 + x(c_2 + x(\ldots(c_{n-1} + xc_n)))). \tag{3.1}$$

Wird dieser Ausdruck von rechts nach links ausgewertet, so muss das Ergebnis in jedem Schritt nur mit x multipliziert und der nächste Koeffizient addiert werden, was nur n Multiplikationen und Additionen benötigt. Auch muss kein Zwischenwert gespeichert werden, was Prozessorregister spart. Als C-Code sieht die Auswertung des Hornerschemas so aus:

```
double horner(double *series, int n, double x)
{
    double r = c[n];
    for(int i = n-1; i >= 0; --i)
        r = r*x + c[i];
    return r;
}
```

Die Polynomauswertung stellt NumPy als **`numpy.polyval(x, c)`** zur Verfügung. `c` bezeichnet die Koeffizienten des Polynoms und `x` das Argument, für das das Polynom ausgewertet werden soll.

Eine weitere Anwendung des Hornerschemas ist die Polynomdivision durch lineare Polynome der Form $x - x_0$, die zum Beispiel wichtig für die iterative Bestimmung von Nullstellen ist. Es gilt nämlich

$$P(x) = \sum_{i=0}^{n} c_i x^i = \left(\sum_{i=0}^{n-1} d_{i+1} x^i \right) (x - x_0) + d_0, \tag{3.2}$$

wobei $d_i = c_i + x_0(c_{i+1} + x_0(\ldots(c_{n-1} + x_o c_n)))$ die Zwischenterme des Hornerschemas bei Auswertung an der Stelle x_0 sind. d_0 ist dabei der Divisionsrest; ist $P(x)$ durch $x - x_0$ teilbar, so ist $d_0 = 0$.

Dies zeigt man durch Induktion: für $P(x) = c_1 x + c_0$ ist offenbar $P(x) = c_1(x - x_0) + c_0 + x c_1 = d_1(x - x_0) + d_0$. Für Grad n ist also

$$P(x) = x \left(\sum_{i=0}^{n-1} c_{i+1} x^i \right) + c_0 = x \left(\sum_{i=0}^{n-2} d'_{i+1} x^i (x - x_0) + d'_0 \right) + d_0 \tag{3.3}$$

wobei sich die $d'_i = c_{i+1} + x_0(c_{i+2} + x_0(\ldots(c_{n-1} + x_o c_n))) = d_{i+1}$ bei der Polynomdivision von $\sum_{i=0}^{n-1} c_{i+1} x^i$ durch $x - x_0$ ergeben. Daher ist

$$P(x) = \left(\sum_{i=0}^{n-2} d_{i+2} x^{i+1} + d_1 \right) (x - x_0) + d_0 + x_0 d_1, \tag{3.4}$$

was zu zeigen war.

3.2 Taylorreihen

Nachdem wir nun wissen, wie Polynome effizient ausgewertet werden können, stellt sich die Frage, wie man ein gutes Näherungspolynom für eine Funktion bekommt. Dazu gibt es viele verschiedene Ansätze, deren Vor- und Nachteile im Folgenden kurz besprochen werden. Der älteste Ansatz, der auch in der Analytik weiten Einsatz findet, ist die Taylorentwicklung. Ist eine Funktion f um einen Punkt x_o hinreichend gut differenzierbar, lässt sie sich als bekannterweise lokal als Taylorreihe darstellen:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i, \tag{3.5}$$

wobei $f^{(i)}(x)$ die i-te Ableitung von f an der Stelle x bezeichnet. Falls die Ableitungen existieren und $x - x_0$ klein genug ist, so konvergiert diese Darstellung schnell, und einige Terme genügen, um zufriedenstellende Genauigkeit zu erreichen. Lokal um den Entwicklungspunkt x_0 is eine abgeschnittene Taylorreihe also eine gute polynomielle Näherung. Leider gibt es für die meisten Funktionen einen Konvergenzradius, außerhalb dessen die Reihe nicht einmal konvergiert. Daher eignen sich Taylorreihen vor allem gut für kleine Umgebungen. Auch ist eine abgeschnittene Taylorreihe nur im Entwicklungspunkt x_0 exakt; dort stimmen allerdings gleich die ersten i Ableitungen.

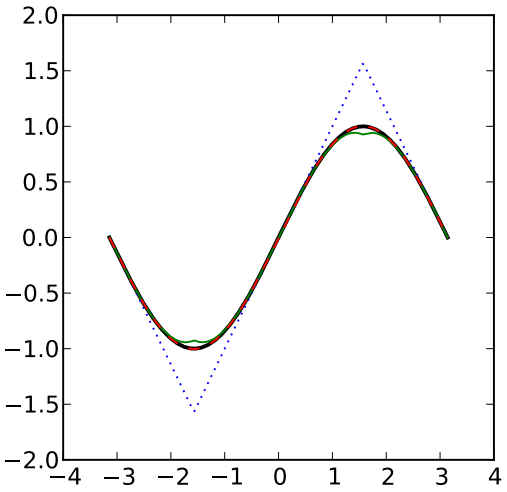


Abbildung 3.1: Näherung der Sinusfunktion durch die abgeschnittene Taylorreihe. Als schwarze durchgezogene Linie ist die tatsächliche Sinusfunktion dargestellt, blau gepunktet ist die Näherung erster Ordnung um Null, x, grün durchgezogen ist die kubische Näherung $x - x^3/6$, und rot gestrichelt $x - x^3/6 + x^5/120$. Die Kurven nutzen die Symmetrie der Sinuskurve, sind also an $\pm\pi/2$ gespiegelt.

Um zum Beispiel die oben angeführte Sinusfunktion mit 7 Stellen Genauigkeit im Interval $[0 : \pi/2]$ auszuwerten, genügen die ersten 7 Terme der Taylorreihe. Mit Hilfe der Symmetrien der Funktion lässt sie sich damit bereits für alle Argumente auswerten. Da

$$\sin'(x) = \cos(x) \quad \text{und} \quad \cos'(x) = -\sin(x),$$

ergibt sich die bekannte Reihe

$$\sin(x) = \sum_{i=0}^{\infty} \frac{\sin^{(i)}(0)}{i!} x^i = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)!} x^{2i+1}. \tag{3.6}$$

Wie gut diese Darstellung mit entsprechender Rückfaltung funktioniert, zeigt Abbildung 3.1. Für viele andere komplexe Funktionen ist es ebenfalls möglich, Taylorreihen analytisch oder numerisch zu bestimmen, die dann zur Auswertung auf dem Computer genutzt werden können.

3.3 Polynom- oder Lagrangeinterpolation

Wie besprochen ist eine abgeschnittene Taylorreihe nur im Entwicklungspunkt exakt (dann allerdings auch die Ableitungen), innerhalb des Konvergenzradius nur eine Annäherung, und außerhalb des Konvergenzradius sogar divergent. Oft möchte man aber eher

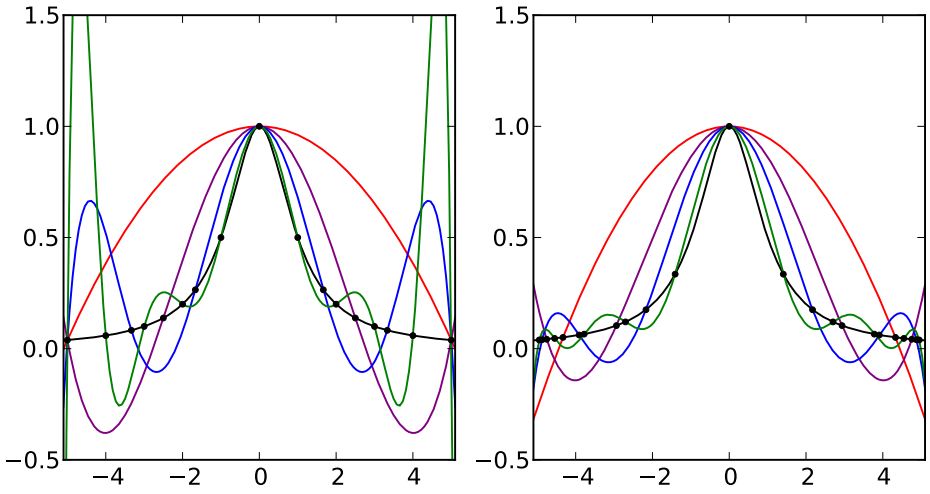


Abbildung 3.2: Lagrange-Interpolation der Rungefunktion $1/(1+x^2)$ (schwarze Linie). Im linken Graph sind die Stützstellen äquidistant gewählt (markierte Punkte), die farbigen Linien sind die interpolierenden Polynome durch 3 (rot), 5 (lila), 7 (blau) und 11 (grün) Stützstellen. Mit steigender Ordnung wird die Interpolation am Rand immer schlechter, das Polynom 10. Ordnung erreicht Werte bis nahe an zwei. Im rechten Graph sind für die gleichen Ordnungen Chebyshev-Stützstellen gewählt worden, die den Interpolationsfehler minimieren.

für einen größeren Wertebereich eine gute (oder wenn möglich exakte) Darstellung der Funktion haben.

Eine Möglichkeit dazu bietet die Polynom- oder Lagrangeinterpolation. Dazu legt man eine Anzahl von Punkten im gewünschten Wertebereich fest (die sogenannten *Stützstellen*). Wie sich zeigt, gibt es dann genau ein Polynom, dass die Funktion an diesen Punkten exakt interpoliert. Genauer: seien Punkte (x_i, y_i), $i = 0(1)n-1$ gegeben mit x_i paarweise verschieden. Dann gibt es genau ein Polynom $P(x) = \sum_{k=0}^{n-1} a_k x^k$ vom Grad $n-1$, so dass $P(x_i) = y_i$, da die Gleichung

$$\begin{aligned} y_1 &= P(x_0) = a_0 + a_1 x_0 + \cdots + a_{n-1} x_0^{n-1} \\ &\vdots \\ y_n &= P(x_{n-1}) = a_0 + a_1 x_{n-1} + \cdots + a_{n-1} x_{n-1}^{n-1} \end{aligned} \tag{3.7}$$

genau eine Lösung hat.

Leider ist aber nicht gewährleistet, dass mit steigender Anzahl von Punkten die Funktion auch zwischen den Stützstellen immer besser angenähert wird. Tatsächlich hat Runge ein einfaches Beispiel angegeben, nämlich die Rungefunktion $1/(1+x^2)$, für die die Näherung mit steigender Anzahl an äquidistanten Punkten immer schlechter wird, siehe Abbildung 3.2.

Bei der etwas allgemeineren Hermite-Interpolation können an den Stützstellen neben den Funktionswerten auch Ableitungen vorgegeben werden. Das eindeutige interpolierende Polynom hat dann einen Grad, der der Gesamtanzahl an vorgegebenen Funktionswerten und Ableitungen entspricht. Ist zum Beispiel nur eine Stützstelle x_0 gegeben und neben dem Funktionswert n Ableitungen, so entspricht das Hermite-Polynom genau den ersten $n+1$ Termen der Taylorreihe.

Das interpolierende Polynom kann nicht nur zur Interpolation verwendet werden, also der Bestimmung an Punkten zwischen den Stützstellen, sondern — mit Vorsicht — auch zur Extrapolation, also um Werte außerhalb des Bereichs zu bestimmen. Da bei der Hermite-Interpolation auch die Ableitungen insbesondere am Rand kontrolliert werden können, ist diese hier tendenziell vorteilhafter. Extrapolation spielt eine wichtige Rolle, wenn eine direkte Auswertung der Zielfunktion numerisch zu teuer oder unmöglich wird. Bei Computersimulationen tritt dies insbesondere in der Nähe von kritischen Punkten auf.

In SciPy liefert die Funktion **`scipy.interpolate.lagrange(x, y)`** das interpolierende Polynom durch die Stützstellen (`x[i]`, `y[i]`).

3.3.1 Lagrangepolynome

Die Koeffiziente a_i können im Prinzip als Lösung von Gleichung (3.7) mit geeigneten Lösern für lineare Gleichungssysteme gefunden werden, was im allgemeinen allerdings recht langsam ist. Daher benutzt man besser eine direkte Darstellung mit Hilfe der *Lagrangepolynome*, die wie folgt definiert sind:

$$L_i(x) = \prod_{k \neq i} \frac{x - x_k}{x_i - x_k}. \tag{3.8}$$

Die Polynominterpolation wird daher auch Lagrange-Interpolation genannt. Wie man leicht sieht, gilt $L_i(x_k) = \delta_{ik}$, so dass das Polynom

$$P(x) = \sum_{i=1}^{n} y_i \, L_i(x) \tag{3.9}$$

das eindeutige interpolierende Polynom durch (x_i, y_i) ist.

Diese Darstellung ist allerdings für praktische Zwecke nur sinnvoll, wenn sich die Stützstellen x_i nicht ändern, da die Bestimmung der Lagrangepolynome $L_i(x)$ zeitaufwändig ist. Geeigneter ist die *baryzentrische Darstellung*

$$P(x) = \sum_{i=0}^{n-1} y_i \, \mu_i \Big/ \sum_{i=0}^{n-1} \mu_i \quad \text{mit} \quad \mu_i := \frac{1}{x - x_i} \prod_{k \neq i} \frac{1}{x_i - x_k}, \tag{3.10}$$

bei der lediglich der Quotient zweier rationaler Funktionen gebildet werden muss.

3.3.2 Neville-Aitken-Schema

Das rekursive Neville-Schema ist eine effiziente Möglichkeit, das interpolierende Polynom auszuwerten ohne es tatsächlich zu berechnen. Das ist nützlich, wenn nur wenige Auswertungen nötig sind, wie zum Beispiel beim Romberg-Integrationsverfahren, bei dem zur Schrittweite 0 extrapoliert wird.

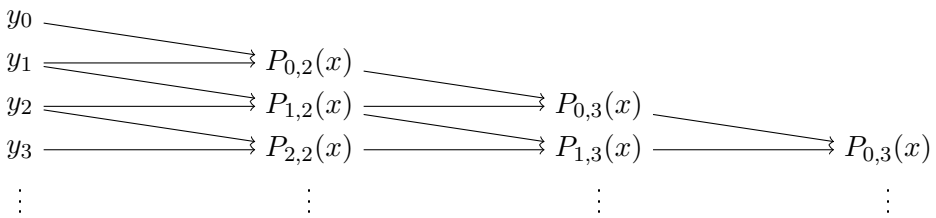
Wir definieren $P_{i,k}$ als das interpolierende Polynom der Ordnung $k-1$ durch die Stützstellen $x_j, j = i(1)i+k-1$. Gesucht ist der Wert $P(x) = P_{0,n}(x)$ des interpolierenden Polynoms an der Stelle x. Dann ist

$$P_{i,1}(x) = y_i \quad \text{für } i = 0(1)n-1 \tag{3.11}$$

und

$$P_{i,k}(x) = \frac{P_{i,k-1}(x)(x_{i+k-1} - x) + P_{i+1,k-1}(x)(x - x_i)}{x_{i+k-1} - x_i} \quad \text{für } k = 2(1)n, i = 0(1)n-k, \tag{3.12}$$

da ja an den inneren Stützstellen x_l, $l = i+1(1)i+k-2$, $P_{i,k-1}(x_l) = P_{i+1,k-1}(x_l) = y_l$ gilt, und per Konstruktion $P_{i,k}(x_i) = y_i$ und $P_{i,k}(x_{i+k-1}) = y_{i+k-1}$. Durch sukzessives Berechnen zunächst der $P_{i,2}(x)$, dann der $P_{i,3}(x)$, usw. lässt sich das interpolierende Polynom bequem an einer fixen Stelle auswerten. Als (Neville-)Schema sieht das so aus:



wobei die Pfeilpaare dividierte Differenzen gemäß (3.12) bedeuten.

3.3.3 Newtonsche Darstellung

Wir betrachten nun die Polynome $P_{0,k}$ des Nevilleschemas. Es gilt offenbar

$$P_{0,k}(x) - P_{0,k-1}(x) = \gamma_k(x - x_0)\cdots(x - x_{k-2}), \tag{3.13}$$

da die beiden Polynome in den Stützstellen $x_0, \ldots, x_{k-2}$ übereinstimmen und die Differenz ein Polynom vom Grad $k-1$ ist, also höchstens $k-1$ Nullstellen hat. Weiter ist γ_k der führende Koeffizient des Polynoms $P_{0,k}(t)$, da $P_{0,k-1}(t)$ ja einen niedrigeren Grad hat. Daraus ergibt sich die folgende *Newtonsche Darstellung* des interpolierenden Polynoms:

$$\begin{aligned} P_{0,n}(x) &= y_0 + \sum_{k=2}^{n} P_{0,k}(x) - P_{0,k-1}(x) \\ &= y_0 + \gamma_2(x - x_0) + \gamma_3(x - x_0)(x - x_1) + \cdots + \gamma_n(x - x_0)\cdots(x - x_{n-2}) \\ &= y_0 + (x - x_0)\Big(\gamma_2 + (x - x_1)\Big(\gamma_3 + \cdots\big(\gamma_{n-1} + (x - x_{n-2})\gamma_n\big)\cdots\Big)\Big). \end{aligned} \tag{3.14}$$

Die letztere Umformung zeigt, dass sich die Newtonsche Darstellung effizient mit einem leicht abgewandelten Hornerschema auswerten lässt:

```
def horner(x0, x, gamma):
    r = 0
    for k in range(len(x)-1, -1, -1):
        r = r*(x0-x[k]) + gamma[k];
    return r
```

Die Koeffizienten γ_i, $i = 2(1)n$ lassen sich dabei bequem mit dem Nevilleschema bestimmen. γ_k ist ja der höchste Koeffizient von $P_{0,k}$ ist, der sich leicht aus (3.12) berechnen lässt. Wenn $\gamma_{i,k}$ den führenden Koeffizienten des Polynoms $P_{i,k}$ bezeichnet, so erhalten wir das Nevilleschema

$$\gamma_{i,1} = y_i \quad \text{für } i = 0(1)n-1 \quad \text{und} \tag{3.15}$$

$$\gamma_{i,k} = \frac{\gamma_{i+1,k-1} - \gamma_{i,k-1}}{x_{i+k-1} - x_i} \quad \text{für } k = 2(1)n, i = 0(1)n-k. \tag{3.16}$$

Da letztlich nur die $\gamma_{0,k}$ interessant sind, also die obere Diagonale des Nevilleschemas, benötigt man für die Berechnung nur einen Vektor

$$\gamma' = (\gamma_{0,1},\gamma_{0,2},\ldots,\gamma_{0,k-1},\gamma_{0,k},\gamma_{1,k},\ldots,\gamma_{n-k,k}), \tag{3.17}$$

der wie folgt berechnet wird:

```
def neville(x, y):
    n = len(x)
    gamma = y.copy()
    for k in range(1, n):
        for i in range(n-k-1, -1, -1):
            gamma[i+k] = (gamma[i+k] - gamma[i+k-1])/(x[i+k] - x[i])
    return gamma
```

Man beachte, dass die Schleife über i herunterläuft, um benötigte Werte nicht zu überschreiben.

3.3.4 Chebyshev-Stützstellen

Bis jetzt haben wir wenig zur Wahl der Stützstellen gesagt. Oft liegt es auch nahe, äquidistante Stützstellen zu verwenden wie im Fadenpendel-Beispiel. Man kann allerdings zeigen, dass die Chebychev-Stützstellen den Fehler der Polynominterpolation minimieren. Diese sind definiert als die Nullstellen der Polynome (!)

$$T_n(\cos\phi) = \cos(n\phi), \tag{3.18}$$

die offensichtlich zwischen -1 und 1 liegen und daher geeignet skaliert werden müssen für die Interpolation in einem allgemeinen Interval. Die Chebychev-Polynome T_n, $n \geq 0$, bilden eine orthogonale Basis der Funktionen über $[-1,1]$ bezüglich des mit $1/\sqrt{1-x^2}$

gewichteten Skalarprodukts. Daher kann jede genügend glatte Funktion auf $[-1:1]$ als eine Reihe

$$f(x) = \sum_{n=0}^{\infty} a_n T_n(x) \tag{3.19}$$

dargestellt werden, die sogenannte Chebyshev-Reihe (siehe auch z.B. Abramowitz und Stegun [AS70]).

Explizit sind diese Nullstellen gegeben durch

$$x_{k,n} = \cos\left(\frac{2k+1}{2n}\pi\right), \quad k = 0(1)n-1. \tag{3.20}$$

Wird die Rungefunktion mit Chebyshevstützstellen interpoliert, so konvergiert das interpolierende Polynom, im Gegensatz zu äquidistanten Stützstellen.

3.4 Splines

Wie wir gesehen haben, kann unter ungünstigen Umständen die Güte der Polynominterpolation mit steigender Anzahl an Stützstellen sinken, vor allem, wenn diese äquidistant verteilt sind. Oft ist das aber nicht zu vermeiden, zum Beispiel, wenn die Daten in einem Experiment regelmäßig gemessen werden. Das Problem ist, das die Koeffizienten des Polynoms global gelten, sich glatte Funktionen aber nur lokal wie ein Polynom verhalten (Taylorentwicklung!). Daher ist es günstiger, statt der gesamten Funktion nur kleine Abschnitte durch Polynome zu nähern.

Der einfachste Fall einer solchen Näherung ist die *lineare Interpolation*, bei der die Stützstellen durch Geraden, also Polynome vom Grad 1, verbunden werden. Sind die Stützstellen (x_i, y_i), $i = 1(1)n$ gegeben, so ist der lineare interpolierende Spline

$$P_1(x) = \frac{(x_{i+1}-x)y_i + (x-x_i)y_{i+1}}{x_{i+1}-x_i} \quad \text{für } x_i \le x < x_{i+1}. \tag{3.21}$$

Diese Funktionen sind aber an den Stützstellen im allgemeinen nicht differenzierbar. Soll die Interpolierende differenzierbar sein, müssen Polynome höherer Ordnung genutzt werden. Solche stückweise definierten Polynome heißen *Splines* — das englische Wort Splines bezeichnete dünne Latten, die vor dem Computerzeitalter benutzt wurden, um glatte, gebogene Oberflächen vor allem für Schiffsrümpfe zu entwickeln. Der wichtigste Spline ist der *kubische* oder *natürliche* Spline, der aus Polynomen dritten Grades zusammengesetzt und zweifach stetig differenzierbar ist. Seine allgemeine Form ist

$$P_3(x) = y_i + m_i(x-x_i) + \frac{1}{2}M_i(x-x_i)^2 + \frac{1}{6}\alpha_i(x-x_i)^3 \quad \text{für } x_i \le x < x_{i+1}. \tag{3.22}$$

Da die zwei rechten und linken zweiten Ableitungen an den Stützstellen übereinstimmen müssen, gilt

$$\alpha_i = \frac{M_{i+1}-M_i}{x_{i+1}-x_i}. \tag{3.23}$$

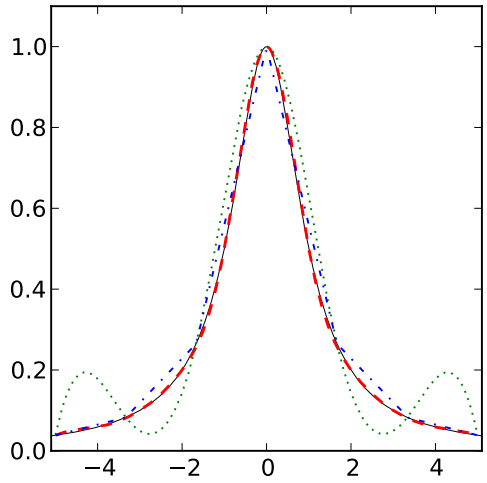


Abbildung 3.3: Spline-Interpolation der Rungefunktion (durchgezogene schwarze Linie). Die gestrichelte blaue Linie ist die lineare Spline-Interpolierende mit 7 Stützstellen, die anderen Kurven sind kubische Splines mit 7 (grün gepunktet) und 11 Stützstellen (rot gestrichelt). Mit 11 Stützstellen ist der Spline von der Rungefunktion praktisch nicht mehr zu unterscheiden.

Aus der Gleichheit der Funktionswerte an den Stützstellen ergibt sich

$$m_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{1}{6}(x_{i+1} - x_i)(2M_i + M_{i+1}). \tag{3.24}$$

Aus der Gleichheit der ersten Ableitungen ergibt sich schliesslich ein Gleichungssystem für die M_i. Hier kommen in den Gleichungen gleichzeitig M_{i-1}, M_i und M_{i+1} vor, daher müssen für die Randwerte weitere Vorgaben gemacht werden. Sollen die Splines am Rand festgelegte 2. Ableitungen M_0 und M_n haben, so hat das Gleichungssystem die Form

$$\begin{pmatrix} 2 & \lambda_1 & & & & 0 \\ \mu_2 & 2 & \lambda_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} & \\ 0 & & & \mu_{n-1} & 2 & \end{pmatrix} \begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{pmatrix} = \begin{pmatrix} 6S_1 - \mu_1 M_0 \\ 6S_2 \\ \vdots \\ 6S_{n-2} \\ 6S_{n-1} - \lambda_{n-1} M_n \end{pmatrix}, \tag{3.25}$$

mit

$$\lambda_i = \frac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}}, \quad \mu_i = \frac{x_{i+1} - x_i}{x_{i+1} + x_{i-1}} \quad \text{und} \quad S_i = \frac{\frac{y_{i+1}-y_i}{x_{i+1}-x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}}{x_{i+1} - x_{i-1}}. \tag{3.26}$$

Auch periodische Funktionen können kubisch interpoliert werden, wobei dann die zusätzliche Bedingungen durch die Kontinuität über die periodische Grenze hinweg gegeben

sind. Die Gleichungen für α_i und m_i sind dabei unverändert, nur das Gleichungssystem wird

$$\begin{pmatrix} 2 & \lambda_1 & & & & \mu_1 \\ \mu_2 & 2 & \lambda_2 & & 0 & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \mu_{n-2} & 2 & \lambda_{n-2} \\ \lambda_{n-1} & & & & \mu_{n-1} & 2 \end{pmatrix} \begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{pmatrix} = \begin{pmatrix} 6S_1 \\ 6S_2 \\ \vdots \\ 6S_{n-2} \\ 6S_{n-1} \end{pmatrix}, \tag{3.27}$$

wobei die Funktion als $x_n - x_1$-periodisch mit $y_1 = y_n$ vorausgesetzt wird. Abbildung 3.3 zeigt die Spline-Interpolation der Rungefunktion, die für diesen Interpolationstyp keine Probleme zeigt.

Die Gleichungssysteme (3.25) und (3.27) sind sehr gut konditioniert und mit einem einfachen Gleichungslöser zu behandeln. Zum Beispiel ist die Gauß-Elimination ist für die hier auftretenden einfachen Bandstrukturen sehr effizient. In SciPy gibt es selbstverständlich bereits eine fertige Routine für die Spline-Interpolation, nämlich **`scipy.interpolate.interp1d(x, y, kind)`**. (`x`,`y`) sind dabei die Stützstellen, und `kind` eine Zeichenkette, die den Typ des Splines bestimmt. Mögliche Werte sind zum Beispiel „linear“ und „cubic“ für lineare bzw. kubische interpolierende Splines.

3.5 Ausgleichsrechnung, Methode der kleinsten Quadrate

Interpolierende Polynome, Taylorreihen und Splines haben gemeinsam, das diese exakt durch die gegebenen Stützstellen verlaufen. Oftmals ist das aber gar nicht gewünscht, da die Daten selbst nicht exakt sind, zum Beispiel wenn diese aus einem Experiment oder einer Simulation stammen. In diesem Fall hat man üblicherweise eine Vorstellung, welche funktionelle Form die Daten annehmen, und möchte nun wissen, mit welchen Parametern diese Funktion am besten mit den Daten verträglich ist. Dazu muss man den Parametersatz bestimmen, so dass der Abstand der Daten von der Funktion minimiert wird.

Seien also wieder Daten (x_i, y_i), $i = 1(1)n$ und eine Funktion $f_v(x)$ gegeben. Gesucht ist dann derjenige Parametervektor v, der die Abweichung

$$\Delta(v) = \sum_i (f_v(x_i) - y_i)^2 \tag{3.28}$$

minimiert. Dieses Verfahren wird auch Methode der kleinste Quadrate genannt, da ja die quadrierten Abweichungen minimiert werden sollen. Ist $f_{a,b}(x) = ax + b$ eine Gerade, spricht man auch von *linearer Regression*. In diesem Fall lässt sich das Optimum einfach bestimmen, da

$$0 = \frac{d}{da}\Delta(a,b) = \sum_i 2(ax_i + b - y_i)x_i = 2N\left(a\left\langle x_i^2\right\rangle + b\left\langle x_i\right\rangle - \left\langle y_i x_i\right\rangle\right) \tag{3.29}$$

und

$$0 = \frac{d}{db}\Delta(a,b) = \sum_i 2(ax_i + b - y_i) = 2N\left(a\left\langle x_i\right\rangle + b - \left\langle y_i\right\rangle\right), \tag{3.30}$$

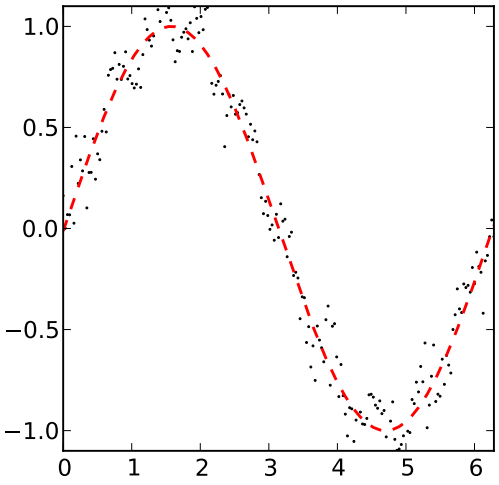


Abbildung 3.4: Methode der kleinsten Quadrate zum Fitten der Sinusfunktion $y = \sin(x)$. 200 Datenpunkte zwischen 0 und 2π wurden als $\sin(x) + 0{,}1\,\sin(10x) + \xi$ erzeugt, wobei ξ eine Gauß-verteilte Pseudozufallsvariable mit Varianz 0,01 war. Die resultierende Sinusfunktion (rot gestrichelt) hat die Form $a\sin(bx+c)$, wobei die Koeffizienten auf gut 2% Genauigkeit $a = b = 1$ und $c = 0$ entsprechen. Die kleine höherfrequente Schwingung kann durch einen Fit allerdings nicht zuverlässig erkannt werden.

wobei $\langle\cdot\rangle$ den Mittelwert über alle Datenpunkte bedeutet. Daraus ergibt sich

$$a = \frac{\langle y_i x_i\rangle - \langle y_i\rangle\,\langle x_i\rangle}{\langle x_i^2\rangle - \langle x_i\rangle^2} \quad \text{und } b = \langle y_i\rangle - a\,\langle x_i\rangle\,, \tag{3.31}$$

was sich einfach auf dem Computer berechnen lässt.

Auch für quadratische und andere einfache Funktionen lassen sich die Koeffizienten geschlossen darstellen, aber bei allgemeinen Funktionen ist dies nicht immer der Fall. Dann muss die nichtlineare Optimierungsaufgabe (3.28) numerisch gelöst werden, was wir später behandeln werden. Für den Moment genügt uns, dass SciPy die Funktion **`scipy.optimize.leastsq(delta, v0, (x, y))`** dafür bereitstellt. `(x, y)` sind dabei die Ausgangsdaten, die hier zu einem Tupel zusammengefasst sind. `v0` ist der Startwert für die Berechnung, der nicht zu weit vom (unbekannten) Optimum entfernt liegen darf. `delta` ist eine Python-Funktion, die als Argumente v, x_i und y_i nimmt und $f_v(x_i) - y_i$ zurückliefert. Da $f_v(x)$ eine beliebig komplizierte Form annehmen kann, ist diese Aufgaben im allgemeinen nicht lösbar, allerdings funktioniert ein solcher *Fit* für einfache Funktionen meistens recht gut. Abbildung 3.4 zeigt einen solchen Funktionsfit an eine verrauschte Sinusfunktion, die mit 200 Datenpunkten auf etwa 2% genau gefittet werden kann. Man beachte, das der Ausgangswert für den Fit mit Hilfe der SciPy-Funktion `leastsq` $a = 0$, $b = 1$, $c = 0$ war; beim Startwert $a = 0$, $b = 0$, $c = 0$ bricht das Verfahren

ab. Das zeigt, dass man tatsächlich nicht zu weit vom Optimum starten kann, was ein gewisses Verständnis der Zielfunktion voraussetzt.

Ist die Funktionsform, die den Daten zugrundeliegt, unbekannt, ist es normalerweise keine gute Idee, die Form zu raten. Generell sollte auch die Anzahl der Parameter sehr klein sein, da sich sonst fast alles „gut“ fitten lässt („With four parameters I can fit an elephant and with five I can make him wiggle his trunk.“ — J. von Neumann).

Soll aber zum Beispiel für Visualisierungszwecke eine ansprechende Kurve entlang der Daten gelegt werden, deren tatsächliche Abhängigkeit unbekannt ist, dann sind *Padé-Funktionen* oft eine gute Wahl. Diese haben die Gestalt $P(x)/Q(x)$, wobei P und Q zwei Polynome mit paarweise verschiedenen Nullstellen sind. Üblicherweise lassen sich schon niedrigen Polynomgraden ansprechende Fits finden, sofern die Grade der beiden Polynome in etwa gleich gewählt werden.

3.6 Fourierreihen

Bis jetzt waren unsere Näherungsfunktionen auf Polynomen basierend, da diese einerseits vom Computer verarbeitet werden können und andererseits aufgrund der Taylorentwicklung glatte Funktionen meist gut approximieren. Für periodische Funktionen sind Polynome aber an sich erst einmal wenig geeignet, da sie selbst nicht periodisch sind. Splines können zwar auch periodisch gemacht werden, aber trotzdem sind trigonometrische Funktionen besser geeignet, um periodische Funktionen darzustellen. Fourierreihen und -transformationen stellen Funktionen als trigonometrische Reihen dar, die meist gut konvergieren und darüberhinaus einige nützliche Eigenschaften haben.

Es gibt zwei Hauptanwendungen der Fourierdarstellung: die Analyse und Aufbereitung periodischer Signale und die Lösung von Differentialgleichungen. Bei periodischen Signalen dient die Fourierdarstellung zur Analyse des *Spektrums* des Signals. Diese gibt nützliche Informationen über die charakteristischen Zeitskalen von Strukturen im Signal, zum Beispiel die Tonhöhe und die Obertonreihe eines Instruments. In dieser Frequenzdarstellung lassen sich auch gezielt einzelne Frequenzen dämpfen, was Rauschen unterdrücken kann und im ursprünglichen Funktionsraum teure Faltungen erfordert. Bei Differentialgleichungen nutzt man aus, dass die Ableitung im Frequenzraum eine algebraische Operation ist, und die Differentialgleichung somit in eine gewöhnliche algebraische (und oft sogar lineare) übergeht.

3.6.1 Komplexe Fourierreihen

Wir betrachten eine periodische Funktion $f(t)$ mit $f(t+T) = f(t)$ für alle $t \in \mathbb{R}$, d.h. f hat Periode T. Dann ist die Fourierdarstellung von f gegeben durch

$$f(t) = \sum_{n\in\mathbb{Z}} \hat{f}_n e^{in\omega t} \tag{3.32}$$

mit $\omega = 2\pi/T$. Die Koeffizienten $\hat{f}_n$ lassen sich berechnen als

$$\hat{f}_n = \frac{1}{T}\int_0^T f(t)e^{-in\omega t}\,dt$$

und sind im allgemeinen komplex, auch wenn f reellwertig ist. Die Beiträge $\hat{f}_{\pm n}$ haben dieselbe Frequenz $\pm n/T$, unterscheiden sich aber in ihrer Phase. Die *Leistung* zu dieser Frequenz ist $\hat{f}_n \hat{f}_{-n}$.

(3.32) lässt sich auch so lesen, dass

$$e^{-in\omega t} = \cos(n\omega t) + i \sin(n\omega t) \tag{3.33}$$

eine orthonormale Basis bezüglich des Skalarprodukts

$$(f, g) = \frac{1}{T} \int_0^T f(t)\overline{g(t)}\, dt \tag{3.34}$$

bilden. Ähnlich wie ein Vektor im $\mathbb{R}^n$ wird die Funktion f also durch die Fouriertransformation in ihre Schwingungskomponenten zerlegt. Insbesondere sind die Fourierkoeffizienten linear in der Funktion, d.h.

$$\widehat{f + \lambda g}_n = \hat{f}_n + \lambda \hat{g}_n. \tag{3.35}$$

Die Voraussetzungen für die Konvergenz der Fourierreihe sind sehr schwach - solange die Funktion wenigstens quadratintegrabel ist, konvergiert die Fourierreihe fast überall, d.h.

$$\left\| f(t) - \sum_{n=-N}^{N} \hat{f}_n e^{in\omega t} \right\| \xrightarrow{N\to\infty} 0. \tag{3.36}$$

Daneben ist die Transformation $f \to \hat{f}$ eine *Isometrie*, genauer gilt das *Parsevaltheorem*

$$\sum_{n\in\mathbb{Z}} |\hat{f}_n|^2 = \frac{1}{\omega} \int_0^t |f(t)|^2\, dt. \tag{3.37}$$

Das Parsevaltheorem besagt auch, dass die Restbeiträge von großen n immer kleiner werden, so dass also eine abgeschnittene Fourierreihe eine Approximation an die gesuchte Funktion darstellt. Anders als abgeschnittene Taylorreihen, die nur in einer schmalen Umgebung um den Aufpunkt exakt sind, konvergiert die Fourierreihe gleichmäßig. Allerdings muss die abgeschnittene Fourierreihe im allgemeinen keinen einzigen Punkt mit der Zielfunktion gemeinsam haben, anders als Taylorreihen oder Splines.

Weiter gilt:

- Die Fourierreihe über einem Interval $[0,T)$ kann aus der Fourierreihe für das Interval $[0,2\pi)$ durch Streckung mit ω berechnet werden:

$$\widehat{f(t)}_n = \frac{1}{T} \int_0^T f(t) e^{-in\omega t}\, dt = \frac{1}{2\pi} \int_0^{2\pi} f(t'/\omega) e^{-int'}\, dt', \tag{3.38}$$

- Es gilt

$$\widehat{f(t + t_0)}_n = e^{in\omega t_0} \widehat{f(t)}_n \tag{3.39}$$

  die Phase kann also nach Belieben verschoben werden. Die Leistung $\hat{f}_n \hat{f}_{-n}$ bleibt dabei natürlich erhalten.

- Für die komplexe Konjugation gilt stets $\hat{\overline{f}}_n = \overline{\hat{f}_n}$, da die Fouriertransformation ja linear ist.
- Ist Funktion f symmetrisch, also $f(t) = f(-t) = f(T-t)$, so ist $\hat{f}_{-n} = \hat{f}_n$, also $\hat{f}$ symmetrisch.
- Ist Funktion f ungerade, also $f(t) = -f(-t) = -f(T-t)$, so ist $\hat{f}_{-n} = -\hat{f}_n$, also $\hat{f}$ ungerade.
- Ist Funktion f reellwertig, also $f(t) = \overline{f(t)}$, so ist $\hat{f}_{-n} = \overline{\hat{f}_n}$. Allerdings sind die Fourierkoeffizienten im allgemeinen komplex!
- Ist die komplexwertige Funktion $f(t) = g(t) + ih(t)$ mit g, h reellwertig, gilt also

$$\hat{f}_n + \hat{\overline{f}}_n = \widehat{2g}_n \quad \text{und } \hat{f}_n - \hat{\overline{f}}_n = \widehat{2ih}_n. \tag{3.40}$$

  Dies bedeutet, dass sich die Fourierreihen zweier reellwertiger Funktionen zusammen berechnen und anschließend wieder trennen lassen. Da die Berechnung der Fourierkoeffizienten sowieso komplex erfolgen muss, erspart dies bei numerischer Auswertung eine Transformation.

- Die Ableitung der Fourierreihe ist sehr einfach:

$$\frac{d}{dt}f(t) = \sum_{n\in\mathbb{Z}} \hat{f}_n in\omega e^{in\omega t} = \sum_{n\in\mathbb{Z}} \widehat{\left(\frac{df}{dt}\right)}_n e^{in\omega t} \quad \Longrightarrow \quad \widehat{\left(\frac{df}{dt}\right)}_n = in\omega \hat{f}_n. \tag{3.41}$$

3.6.2 Reelle Fourierreihen

Da die Fourieranalyse besonders zur Analyse und Bearbeitung von Messdaten genutzt wird, sind die Fourierreihen reellwertiger Funktionen besonders wichtig. Ist die Funktion f rellwertig, so ist

$$\begin{aligned}\hat{f}_n e^{in\omega t} + \hat{f}_{-n} e^{-in\omega t} = \hat{f}_n e^{in\omega t} + \overline{\hat{f}_{-n} e^{in\omega t}} &= 2\mathrm{Re}(\hat{f}_n e^{in\omega t}) \\ &= 2\mathrm{Re}(\hat{f}_n)\cos(n\omega t) - 2\mathrm{Im}(\hat{f}_n)\sin(n\omega t).\end{aligned} \tag{3.42}$$

Daraus folgt, dass sich die Fourierreihe auch komplett reellwertig schreiben lässt:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(n\omega t) + b_n \sin(n\omega t) \tag{3.43}$$

mit

$$a_n = 2\mathrm{Re}(\hat{f}_n) = \frac{2}{T}\int_0^T f(t)\cos(n\omega t)\,dt \tag{3.44}$$

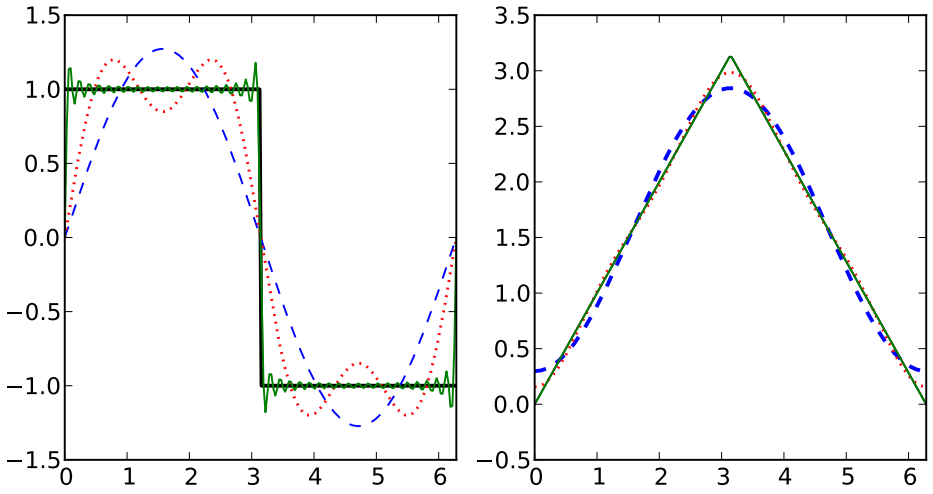


Abbildung 3.5: Abgeschnittene Fourierreihen der Rechteckfunktion (links) und eines Dreieckpulses (rechts). Die Funktionen sind jeweils als schwarze durchgezogene Linien eingezeichnet, die Näherungen mit einem Term blau gestrichelt, mit zwei Termen rot gepunktet, und mit 20 Termen grün durchgezogen. Für den Dreieckpuls ist letztere Näherung nicht mehr von der Funktion zu unterscheiden, während der Rechteckpuls noch deutliche Artefakte an den Unstetigkeiten zeigt.

und

$$b_n = -2\mathrm{Im}(\hat{f}_n) = \frac{2}{T}\int_0^T f(t)\sin(n\omega t)\,dt. \tag{3.45}$$

Für symmetrische Funktionen ist offenbar $b_n = 0$, für ungerade Funktionen $a_n = 0$.

Einige reelle Fourierreihen sind zum Beispiel:

- Konstante $f(t) = f_0$:
$$a_0 = 2f_0, \quad a_n, b_n = 0 \quad \text{sonst} \tag{3.46}$$

- Rechteckfunktion
$$f(t) = \begin{cases} 1 & \text{für } 0 \le t < \frac{T}{2} \\ -1 & \text{für } \frac{T}{2} \le t < T \end{cases} = \frac{4}{\pi}\sum_{n=1}^{\infty}\frac{1}{2n-1}\sin\left((2n-1)\omega t\right) \tag{3.47}$$

- kurzer Rechteckpuls. Wir betrachten nun die auf konstanten Flächeninhalt normierte Funktion
$$f_S(t) = \begin{cases} 1/S & \text{für } 0 \le t < S \\ 0 & \text{für } S \le t < T \end{cases}, \tag{3.48}$$

deren Fourierreihe

$$f_s(t) = \frac{1}{T} + \frac{2}{T}\sum_{n=1}^{\infty} \frac{\sin(n\omega S)}{n\omega S}\cos(n\omega t) + \frac{1-\cos(n\omega S)}{n\omega S}\sin(n\omega t) \tag{3.49}$$

ist. Je kleiner S wird und damit der Träger von f_S, desto langsamer konvergiert ihre Fourierreihe, da die Funktion $\sin(x)/x$ immer dichter an der Null ausgewertet wird. Für jede feste Frequenz n gilt schließlich

$$\widehat{(f_S)}_n \xrightarrow{S\to 0} \frac{1}{T} = \hat{\delta}_n \quad \text{für alle } n \in \mathbb{Z} \tag{3.50}$$

bzw. $a_n \to 2/T$ und $b_n \to 0$. Die δ-Funktion, die ja der formale Grenzwert der f_S ist, und den kleinstmöglichen Träger hat, hat also in gewisser Weise die am schlechtesten (tatsächlich gar nicht!) konvergierende Fourierreihe.

- Dreiecksfunktion

$$f(t) = \begin{cases} t & \text{für } 0 \le t < \frac{T}{2} \\ T - t & \text{für } \frac{T}{2} \le t < T \end{cases} = \frac{\pi}{2} - \frac{4}{\pi}\sum_{n=1}^{\infty} \frac{1}{(2n-1)^2}\cos\left((2n-1)\omega t\right) \tag{3.51}$$

Genau wie die komplexe Fourierreihe lässt sich natürlich auch die reelle Fourierreihe abschneiden, um Näherungen für Funktionen zu bekommen, vergleiche Abbildung 3.5. Es fällt auf, das die Fourierreihe besonders schlecht dort konvergieren, wo die Funktion nicht differenzierbar ist bzw. einen Sprung aufweist.

3.6.3 Diskrete Fouriertransformation

Bei praktischen Anwendungen sind die Integrale zur Bestimmung der Koeffiziente oft nicht analytisch lösbar, oder die Funktion ist von vorneherein nur an diskreten Punkten gegeben, etwa weil es sich um Messdaten handelt. In diesem Fall müssen die Integrale numerisch approximiert werden. Wir betrachten nun also nicht mehr eine kontinuierliche Funktion f, sondern Daten $f_k = f(t_k)$ mit $t_k = k\Delta$, $k = 0(1)N-1$ und Schrittweite $\Delta = \frac{T}{N}$. Dann ist

$$\hat{f}_n = \frac{1}{T}\int_0^T f(t)e^{-in\omega t}\,dt \approx \frac{\Delta}{T}\sum_{k=0}^{N-1} f(k\Delta)e^{-in\omega k\Delta} = \frac{1}{N}\sum_{k=0}^{N-1} f_k e^{-i\frac{2\pi}{N}nk} =: \frac{g_n}{N}. \tag{3.52}$$

Die Koeffizienten

$$\mathrm{DFT}(f_k)_n = g_n = \sum_{k=0}^{N-1} f_k e^{-i\frac{2\pi}{N}nk} \tag{3.53}$$

werden als die *diskrete Fouriertransformierte* bezeichnet, die sehr effizient berechnet werden kann, wie wir im folgenden sehen werden. Analog wird die *inverse diskrete Fouriertransformation*

$$\mathrm{iDFT}(g_n)_k = f(t_k) = \sum_{n=0}^{N-1} \frac{g_n}{N} e^{i\frac{2\pi}{N}nk} \tag{3.54}$$

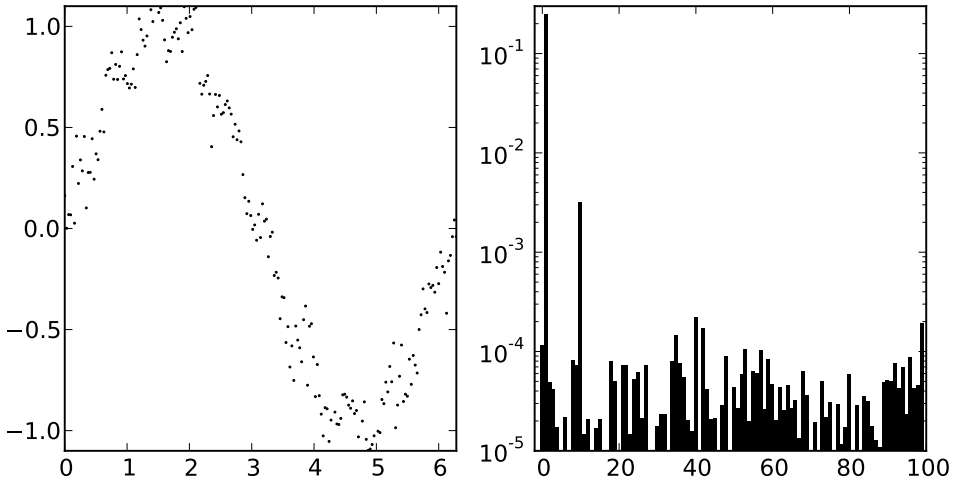


Abbildung 3.6: Diskrete Fouriertransformation von 200 diskreten Datenpunkten, die analog zu Abbildung 3.4 zwischen 0 und 2π als $\sin(x) + 0{,}1\,sin(10x) + \xi$ erzeugt wurden. Anders als der Funktionsfit erlaubt die DFT, auch die kleine zusätzliche Schwingung gut vom Rauschen zu unterscheiden. Im Graphen ist das Leistungsspektrum $|DFT(k)|^2$ gezeigt. Man erkennt die Amplitudenquadrate 0,25 bei 1 und 0,01 bei 10, auch wenn letztere Frequenz durch das Rauschen etwas an Intensität verloren hat.

definiert, die aus den Koeffizienten wieder die Funktion f an den diskreten Eingangspunkten t_k berechnet. Abbildung 3.6 zeigt zum die DFT der Summe zweier verrauschter Sinusfunktionen, aus der die beiden Basisfrequenzen und deren Amplituden klar gegenüber dem Rauschen zu erkennen sind.

Die Koeffizienten sind offenbar periodisch, da

$$g_{n+N} = \sum_{k=0}^{N-1} f_k e^{-i\frac{2\pi}{N}(n+N)k} = \sum_{k=0}^{N-1} f_k e^{-i\frac{2\pi}{N}nk} \underbrace{e^{-2\pi ik}}_{=1} = g_n. \tag{3.55}$$

Insbesondere ist $g_{-k} = g_{N-k}$, und es gibt nur N echt verschiedene Koeffizienten bzw. Frequenzen n/T. DFT-Bibliotheken speichern die Koeffizienten daher meist als Vektor $(g_0, \ldots, g_{N-1})$ bzw. $(g_0, \ldots, g_{N/2-1}, g_{-N/2}, \ldots, g_{-1})$. Ist f reell, so gilt noch dazu $g_{-k} = \overline{g_k}$, sodass lediglich $\lceil N/2 \rceil$ Koeffizienten wirklich verschieden sind. Allerdings sind diese im allgemeinen komplex, so dass die N reellen Freiheitsgrade der Eingangsfunktion erhalten bleiben.

Die endliche Anzahl der diskreten Fourierkoeffizienten bedeutet, dass bei einem reellen Signal mit Schrittweite Δ die maximal darstellbare Frequenz $f_{\text{Nyquist}} = \frac{1}{2\Delta}$ beträgt, die sogenannte *Nyquist-Frequenz*. Signale mit höherer Frequenz f werden zu scheinbaren Signalen niedrigerer Frequenz

$$f_{\text{scheinbar}} = \begin{cases} f \bmod 2f_{\text{Nyquist}} & \text{falls } f \bmod 2f_{\text{Nyquist}} < f_{\text{Nyquist}} \\ 2f_{\text{Nyquist}} - f \bmod 2f_{\text{Nyquist}} & \text{falls } f \bmod 2f_{\text{Nyquist}} \geq f_{\text{Nyquist}}, \end{cases} \tag{3.56}$$

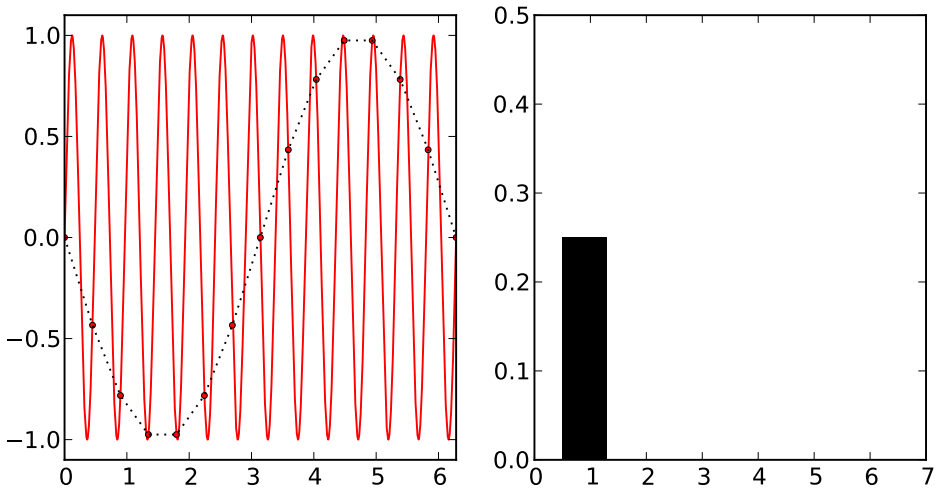


Abbildung 3.7: Diskrete Fouriertransformation von 14 äquidistanten diskreten Datenpunkten (rote Punkte links) der Funktion $\sin(13t)$ (rote Kurve links) im Interval $[0 : 2\pi]$. Die Frequenz der Funktion $13/2\pi$ ist höher als die Nyquist-Frequenz $f_{\text{Nyquist}} = 14/2\pi$, daher kommt es zu Aliasing-Artefakten. Die rekonstruierte Kurve ist links schwarz gepunktet eingezeichnet, ihr Spektrum rechts. Die abgetastete Funktion ist also scheinbar $\sin(t)$, was einer Frequenz von $2f_{\text{Nyquist}} - 13/2\pi$ entspricht.

was auch als *Aliasing* bezeichnet wird. Sollen analoge Signale digital weiter verarbeitet werden, kann es daher notwendig sein, höhere Frequenzen durch analoge Tiefpassfilter zu unterdrücken. Abbildung 3.7 illustriert dieses Problem.

3.6.4 Schnelle Fouriertransformation

Die Berechnung der Fouriertransformierten nach (3.53) ist zwar möglich, aber ziemlich langsam — jeder der N Koeffizienten benötigt offenbar $\mathcal{O}(N)$ Operationen, so dass die DFT insgesamt $\mathcal{O}(N^2)$ Operationen braucht. Das limitiert für praktische Anwendungen N auf einige tausend, was für viele Anwendungen zu wenig ist. Die DFT konnte daher nur durch die *schnelle Fouriertransformation* (FFT) von *Cooley und Tukey* zu breiter Anwendung finden. Diese basiert auf der Beobachtung, dass für $N = 2M$

$$\text{DFT}(f_k)_n = \sum_{k=0}^{M-1} f_{2k} e^{-i\frac{2\pi}{2M}n\,2k} + \sum_{k=0}^{M-1} f_{2k+1} e^{-i\frac{2\pi}{2M}n\,(2k+1)} \tag{3.57}$$

$$= \text{DFT}(f_{2k})_n + e^{-i\frac{2\pi}{2M}n}\text{DFT}(f_{2k+1})_n, \tag{3.58}$$

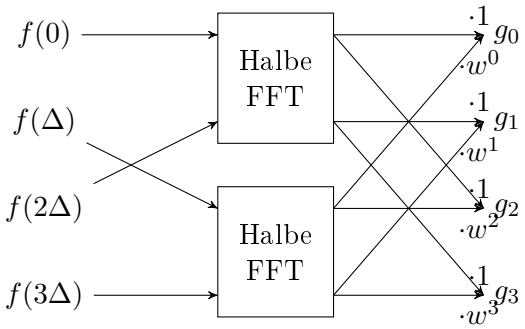
wobei $\text{DFT}(f_{2k})_n$ den n-ten Koeffizienten einer DFT auf den Datenpunkten f_{2k}, $k = 0(1)M-1$, bezeichnet. Gemäß (3.55) ist dabei $\text{DFT}(f_{2k})_n = \text{DFT}(f_{2k})_{n-M}$ für $n > M$.

Die Fouriertransformatierte der N Datenpunkte ergibt sich also als einfache Summe von zwei Fouriertransformierten mit lediglich der halben Menge M an Datenpunkten, wobei die ungerade Hälfte mit der *Einheitswurzel*

$$w^n := e^{-i\frac{2\pi}{2M}n} \tag{3.59}$$

multipliziert wird. Ist nun M wieder durch zwei teilbar, so lassen sich diese Fouriertransformierten ebenfalls als Summe zweier nochmals halb so langer Fouriertransformationen darstellen. Wenn nun N eine Zweierpotenz ist, kann man so fortfahren, bis $M = 1$ erreicht ist, also $\mathrm{DFT}(f_0)_0 = f_0$. Dabei gibt es offenbar $\log_2(N)$ viele Unterteilungsschritte, die jeder $\mathcal{O}(N)$ Operationen kosten. Insgesamt benötigt die FFT also lediglich $\mathcal{O}(N \log N)$ Operationen.

Schematisch funktioniert ein FFT-Schritt wie folgt:



Aufgrund ihres Aussehens wird dieses Datenpfadschema auch als Butterfly-Schema genannt. Damit die beiden Unter-FFTs auf einem zusammenhängenden Satz von Daten operieren können, müssen also auch die Eingabedaten f_k umsortiert werden, ebenso wie auch für die Unter-FFTs. Man kann sich leicht überlegen, dass dabei f_k auf $f_{k'}$ sortiert wird, wobei die Bits k' in Binärdarstellung dieselben wie von k sind, nur in umgedrehter Reihenfolge.

Die FFT erlaubt also die effiziente Zerlegung einer Funktion in ihre Schwingungskomponenten, was viele wichtige Anwendungen nicht nur in der Physik hat. Daher gibt es eine Reihe sehr guter Implementierungen der FFT, allen voran die „Fastest Fourier Transform in the West“ (FFTW, http://www.fftw.org). Selbstverständlich bietet auch NumPy eine FFT, **`numpy.fft.fft(f_k)`**, mit der inversen FFT **`numpy.fft.ifft(g_n)`**. Die Routinen sind so implementiert, dass bis auf Maschinengenauigkeit $\mathrm{iFFT}(\mathrm{FFT}(f_k)) = f_k$.

Wichtige Anwendungsbeispiele der diskreten Fouriertransformation sind zum Beispiel die Datenformate JPEG, MPEG und MP3, die alle drei auf einer Abwandlung der DFT beruhen, der *diskreten Cosinustransformation* (DCT) für reelle Daten. Bei dieser wird der Datensatz so in der Zeitdomäne verdoppelt, dass er in jedem Fall eine gerade Funktion repräsentiert, wodurch die Fourierreihe in eine reine Cosinusreihe übergeht mit nur reellen Koeffizienten. Die DCT ist also eine Umwandlung reeller in reelle Zahlen. Wegen Ihrer Wichtigkeit gibt es nicht nur extrem effiziente Implementierungen für die meisten Prozessortypen, sondern auch spezielle Hardware.

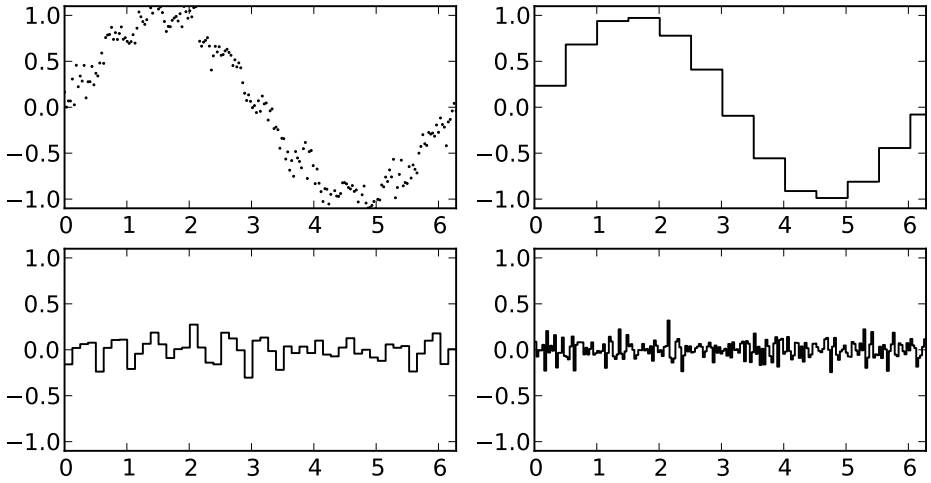


Abbildung 3.8: Diskrete Wavelettransformation von 200 diskreten Datenpunkten, die analog zu Abbildung 3.6 zwischen 0 und 2π als $\sin(x) + 0{,}1\,sin(10x) + \xi$ erzeugt wurden. Für die Transformation wurden die Wavelets von (0,1] auf den Bereich (0,2π] gestreckt. Der linke obere Graph zeigt nochmals das Ausgangssignal, von rechts oben nach rechts unten folgen die Anteile der Stufen 0–3, also $(f,\phi_0)\phi_0 + \sum_{j=0}^{3}\sum_{k=0}^{2^j-1}(f,\psi_{jk})\psi_{jk}$, dann Stufen 4 und 5 ($\sum_{j=4}^{5}\sum_{k=0}^{2^j-1}(f,\psi_{jk})\psi_{jk}$) und schließlich Stufen 6–8, womit die Auflösung der Ausgangsdaten erreicht ist.

3.7 Wavelets

Die Fouriertransformation wird vor allem deshalb für die Kompression von Audio- oder Bilddaten genutzt, weil sie hochfrequente von niederfrequenten Signalen trennt und die menschlichen Sinne die hochfrequenten Anteil meist nicht gut wahrnehmen können. Das ist allerdings nicht ganz korrekt, tatsächlich können wir nur stark lokale Änderungen nicht gut wahrnehmen. Dafür sind Fourierreihen an sich gar nicht so gut geeignet, da ja auch die hochfrequenten Schwingungen alles andere als lokal sind. Als Alternative hat sich die *Multiskalenanalyse* (MSA) oder *diskrete Wavelettransformation* etabliert, die auch im transformierten Raum lokal ist.

Anders als bei der Fouriertransformation, die eine Zerlegung in trigonometrische Funktionen darstellt, gibt es für die MSA verschiedene Sätze von Basisfunktionen mit verschiedenen Eigenschaften wie Differenzierbarkeit und Lokalität. Im folgenden soll die MSA mit Hilfe des Haar-Wavelets dargestellt werden, dass das einfachste und älteste bekannte Wavelet ist. Zunächste betrachten wir die *Skalierungsfunktion*

$$\phi(x) = \chi_{(0,1]} = \begin{cases} 1 & \text{für } 0 < x \leq 1 \\ 0 & \text{sonst} \end{cases} \tag{3.60}$$

sowie das Haar-*Wavelet*

$$\psi(x) = \begin{cases} -1 & \text{für } 0 < x \leq \frac{1}{2} \\ 1 & \text{für } \frac{1}{2} < x \leq 1 \\ 0 & \text{sonst,} \end{cases} \tag{3.61}$$

aus denen wir die Basisfunktionen $\phi_k(x) := \phi(x-k)$ der nullten Stufe und $\psi_{jk}(x) := 2^{j/2}\psi(2^j x - k)$ der j-ten Stufe konstruieren. Durch die Skalierung mit 2^j werden die $\psi_{j,k}$ also immer schmaler, sind aber wegen des Vorfaktors alle normiert, d.h. $\|\psi_{jk}\| = 1$. Ebenso sind auch die ϕ_k normiert. Zusätzlich sind sämtliche Basisfunktionen zu einander orthogonal, wie man sich leicht überlegt. Daher lässt sich jede quadratintegrable Funktion f wie folgt zerlegen:

$$f(x) = \sum_{k\in\mathbb{Z}} (f,\phi_k)\phi_k + \sum_{j\in\mathbb{N}_0}\sum_{k\in\mathbb{Z}} (f,\psi_{jk})\psi_{jk} \tag{3.62}$$

Dies ist die Multiskalenanalyse von f. Die Koeffizienten der Stufe j werden auch Details der Stufe j genannt. In der Praxis ist das Signal durch endlich viele äquidistante Datenpunkte gegeben, analog zur diskreten Fouriertransformation. In diesem Fall sind die Summen endlich, da einerseits der Träger endlich ist und damit nur endlich viele $(f,\phi_k) \neq 0$, und es andererseits keine Details unterhalb der Auflösung des Signals gibt. Man skaliert dann die Wavelets und Skalierungsfunktion so, dass der Abstand der Datenpunkte gerade der halben Breite des Wavelets auf der feinsten Auflösung entspricht, und $\phi = \phi_0$ bereits das gesamte Interval überdeckt. Für eine nur auf [0,1] nichtverschwindende Funktion, deren Werte an 2^N Punkten äquidistanten Punkten bekannt ist, reduziert sich die Multiskalenanalyse zur *diskreten Wavelettransformation*

$$f(x) = (f,\phi)\phi + \sum_{j=0}^{N-1}\sum_{k=0}^{2^j-1} (f,\psi_{jk})\psi_{jk}. \tag{3.63}$$

Die Anzahl der Koeffizienten ist dann $1+1+2+\cdots+2^{N-1} = 2^N$, also genau die Anzahl der Eingabedaten. Genau wie die diskrete Fouriertransformation bildet die Wavelettransformation 2^N Werte $f(k/2^N)$ auf 2^N Werte (f,ϕ) und (f,ψ_{jk}) ab und besitzt eine exakte Rücktransformation, (3.63).

Analog zur schnellen Fouriertransformation gibt es auch eine schnelle Wavelettransformation (FWT), die sogar linearen Aufwand hat, also $\mathcal{O}(N)$ Schritte bei N Datenpunkten benötigt. Eine einfache Implementation der FWT und der inversen FWT für das Haar-Wavelet zeigt Codebeispiel 3.1. Der Kern dieser Transformation liegt darin, die Transformierte von der höchsten Detailauflösung herab aufzubauen, und dadurch die die Integrale approximierenden Summen schrittweise aufzubauen (*Downsampling*). Für genauere Informationen siehe zum Bespiel Daubechies [Dau92].

Abbildung 3.8 zeigt einige Detailstufen der Wavelet-Zerlegung der verrauschten Sinusfunktionen analog Abbildung 3.6. Auch hier lässt sich das Rauschen auf den höheren Detailstufen gut vom Nutzsignal trennen, allerdings kann die Oberschwingung nicht detektiert werden. Das hängt allerdings vor allem daran, dass das Haar-Wavelet nicht

sehr geeignet ist, da es nicht glatt ist, im Gegensatz zum Nutzsignal. Daher sind in den meisten Fällen glatte Wavelets besser geeignet. Das bekannteste Beispiel von glatten Wavelets sind die Daubechies-Wavelets, die daneben auch einen kompakten Träger haben, also stark lokalisiert sind. Mit solchen Wavelets lassen sich sogar reale Musikdaten in Akkorde zurücktransformieren. Auch der JPEG-Nachfolger JPEG2000 basiert auf einer Wavelettransformation statt einer Cosinustransformation, allerdings mit Cohen-Daubechies-Feauveau-Wavelets.

```
# Haar-Wavelet-Transformation
############################
from scipy import *

def haar_trafo(data):
    "Diskrete Wavelettransformation mit Hilfe des Haar-Wavelets."
    # Daten mit kleinstem Integrationsschritt multiplizieren
    c = data.copy() / len(data)
    # Temporaerer Puffer, um benoetigte Werte nicht zu ueberschreiben
    ctmp = zeros(c.shape)
    width = len(c)/2
    while width >= 1:
        for n in range(width):
            tmp1 = c[2*n]
            tmp2 = c[2*n+1]
            # Detail
            ctmp[width + n] = tmp1 - tmp2
            # Downsampling
            ctmp[n]         = tmp1 + tmp2
        # Puffer zurueckschreiben
        c[:2*width] = ctmp[:2*width]
        width = width / 2
    return c

def inverse_haar_trafo(c):
    "Inverse Diskrete Wavelettransformation mit Hilfe des Haar-Wavelets"
    # Rueckgabewerte
    data = zeros(len(c))
    # phi mitnehmen auf der niedrigsten Stufe
    data[0] = c[0]
    width = 1
    cstart = 1
    while width <= len(c)/2:
        for n in range(width-1, -1, -1):
            tmp = data[n]
            data[2*n]     = tmp + width*c[cstart + n]
            data[2*n + 1] = tmp - width*c[cstart + n]
        cstart += width
        width = width * 2
    return data

# Anwendungsbeispiel

x = linspace(0,1,256)
y = cos(x)
coeff = haar_trafo(y)
yrueck = inverse_haar_trafo(coeff)
print max(abs(yrueck - y))
```

Listing 3.1: Diskrete Wavelettransformation und ihre Inverse als Python-Code. Die Länge der Eingabedaten muss eine Zweierpotenz 2^N sein. Die Details sind in einem Vektor c gespeichert, in der Form $c = ((f,\phi_0), (f,\psi_{00}), (f,\psi_{10}), (f,\psi_{11}), (f,\psi_{20}), (f,\psi_{21}), (f,\psi_{22}), \ldots, (f,\psi_{N-1,2^{N-1}-1}))$.

4 Datenanalyse und Signalverarbeitung

In diesem Kapitel geht es darum, was man mit einem gemessenen Signal machen kann und muss. Ein gemessenes Signal kann dabei entweder tatsächlich von einem Messgerät kommen oder aber das Ergebnis einer Computersimulation sein. Zwei Fragen sind dabei vor allem wichtig: welche Eigenschaften hat das Signal, und wie vertrauenswürdig sind die Werte?

Um die Eigenschaften von Signalen zu untersuchen, ist die kontinuierliche Fourieranalyse ein gutes Werkzeug, die das Signal vom Zeit- in den Frequenzraum überträgt. So lassen sich zum Bespiel charakteristische Frequenzen und damit Zeitskalen bestimmen. Außerdem bietet der Übergang in der Frequenzraum analytisch viele Vorteile, die sich auch auf dem Computer nutzen lassen. So werden zum Beispiel langreichweitige Wechselwirkungen in Molekulardynamiksimulationen meist im Frequenzraum berechnet.

Als weiteres Werkzeug werden wir Faltungen kennen lernen, die erlauben, Signale nach bestimmten Frequenzen zu filtern oder aber aus der (gemessenen) Antwort eines linearen Systems auf ein einfaches Eingangssignal die Antwort auf beliebige Signale zu berechnen.

Sollen Signale mit dem Computer weiterverarbeitet werden, müssen diese *digitalisiert* werden, also in eine Reihe von Zahlen übersetzt. Üblicherweise passiert dies dadurch, dass das Signal nur zu äquidistanten Zeitpunkten ausgewertet, *abgetastet* wird. Das wirft die Frage auf, welche Funktionen dadurch überhaupt gut gemessen werden können. Wie wir sehen werden, beschränkt diese Abtastung die Frequenzen, die von einer digitalen Auswertung erfasst werden können.

Die meisten Signale sind außerdem, durch Messungenauigkeiten und prinzipielle stochastische Prozesse, selbst *stochastisch*, d.h. die Verteilung der Ergebnisse vieler Messungen ist vorherbestimmbar, die einzelne Messung hingegen nicht. Trotzdem sind Messungen oft korreliert, zum Beispiel weil eine Observable sich nur kontinuierlich ändert. Durch Korrelationsanalysen lässt sich bestimmen, wann Messungen wirklich unabhängig sind. Dies gibt wiederum Aufschluss über die Zeitskalen wichtiger Prozesse im System, ist aber auch wichtig für eine korrekte Abschätzung des Messfehlers, womit sich der letzte Abschnitt beschäftigt.

4.1 Kontinuierliche Fouriertransformation

Für die Analyse zeitlich veränderlicher Signale besonders nützlich ist die Fouriertransformation, die ein kontinuierliches Signal in den Frequenzraum übersetzt. Dies gilt nicht nur für periodische Signale, sondern zum Beispiel auch dann, wenn die Antwort eines Systems auf ein komplexes Eingangssignal gefragt ist. Der tiefere Grund dafür ist, dass die Fouriertransformation Differential- und Integraloperatoren in einfache algebraische Operationen übersetzt.

Betrachten wir nochmals die Fourierreihe im Interval $[-T/2,T/2)$

$$f(t) = \sum_{n\in\mathbb{Z}} \left(\frac{\Delta\omega}{2\pi} \int_{-T/2}^{T/2} f(t) e^{-in\Delta\omega t}\, dt \right) e^{in\Delta\omega t}$$
$$= \frac{1}{\sqrt{2\pi}} \sum_{n\in\mathbb{Z}} \left(\frac{1}{\sqrt{2\pi}} \int_{-T/2}^{T/2} f(t) e^{-i\omega t}\, dt \right) e^{i\omega t}\, \Delta\omega \quad (4.1)$$

mit der Grundfrequenz $\Delta\omega = 2\pi/T$ und $\omega = n\Delta\omega$. Im Grenzwert $T \to \infty$ ergibt sich

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \mathcal{F}(f)(\omega) e^{i\omega t}\, d\omega \quad (4.2)$$

mit

$$\mathcal{F}(f)(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt. \quad (4.3)$$

Die *kontinuierliche Fouriertransformation* $\mathcal{F}$ ist das Analogon der periodischen Fourierreihe, ist allerdings keine Transformation in eine Reihe mehr, sondern eine Abbildung zwischen Funktionen. Für $\mathcal{F}$ gelten eine Menge sehr starker Aussagen, die wir zum großen Teil in ähnlicher Art schon von der Fourierreihe kennen:

- $\mathcal{F}(f)$ existiert, falls f quadratintegrabel ist, und bildet f auf eine quadratintegrable Funktion ab. Für solche Funktionen mit der zugehörigen Norm $\|f\|_2 = \int_{-\infty}^{\infty} |f(t)|^2\, dt$ gilt dann sogar die Isometrie (*Parsevaltheorem*):
$$\|\mathcal{F}(f)\|_2 = \|f\|_2 \quad (4.4)$$
- $\mathcal{F}$ ist linear, d.h. $\mathcal{F}(f + \lambda g) = \mathcal{F}f + \lambda\mathcal{F}g$.
- $\mathcal{F}$ ist reziprok gegen Streckungen, d.h.
$$\mathcal{F}[f(\alpha t)](\omega) = \frac{1}{|\alpha|}\mathcal{F}(f)\left(\frac{\omega}{\alpha}\right). \quad (4.5)$$
Wird also eine Funktion α immer stärker gestaucht, so wird ihre Transformierte immer weiter gestreckt.
Entsprechend wird aus Zeitumkehr Frequenzumkehr: $\mathcal{F}(f(-t))(\omega) = \mathcal{F}(f)(-\omega)$.
- $\mathcal{F}$ ist invertierbar, die Umkehrfunktion $\mathcal{F}^{-1}$ ist durch (4.2) explizit gegeben. Offenbar ist auch die Umkehrung eine Isometrie, es gilt $\|\mathcal{F}^{-1}(f)\|_2 = \|f\|_2$.
- Weiter gilt $\mathcal{F}(\mathcal{F}(f(t))) = \mathcal{F}(f(-t))$, und damit $\mathcal{F}^4(f) = f$. Insbesondere ist auch $\mathcal{F}^{-1} = \mathcal{F}^3$.
- Eine zeitliche Verschiebung wird zu einer Frequenzmodulation und umgekehrt:
$$\mathcal{F}(f(t - t_0))(\omega) = e^{-i\omega t_0}\mathcal{F}(f(t))(\omega) \quad (4.6)$$
$$\mathcal{F}(e^{i\omega_0 t} f(t))(\omega) = \mathcal{F}(f(t))(\omega - \omega_0). \quad (4.7)$$
Wird also ein niederfrequentes Signal (Radioprogramm) auf ein hochfrequentes Trägersignal aufmoduliert, verschiebt sich nur dessen Spektrum.

- Aus der Linearität folgt, dass stets gilt: $\mathcal{F}(\overline{f})(\omega) = \overline{\mathcal{F}(f)(\omega)}$.
- Ist Funktion f symmetrisch, also $f(t) = f(-t)$, so ist $\mathcal{F}(f)(-\omega) = \mathcal{F}(f)(\omega)$, also symmetrisch
- Ist Funktion f ungerade, also $f(t) = -f(-t)$, so ist $\mathcal{F}(f)(-\omega) = -\mathcal{F}(f)(\omega)$, also ungerade.
- Ist Funktion f reellwertig, also $f(t) = \overline{f(t)}$, so ist $\mathcal{F}(f)(-\omega) = \overline{\mathcal{F}(f)(\omega)}$, aber im allgemeinen komplexwertig!
- Für die Fouriertransformierte der Ableitung gilt

$$\begin{aligned}\mathcal{F}\left(\frac{d}{dt}f(t)\right)(\omega) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\frac{df}{dt}(t)e^{-i\omega t}\,dt \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(t)\frac{d}{dt}e^{-i\omega t}\,dt = i\omega\mathcal{F}(f(t))(\omega).\end{aligned} \tag{4.8}$$

  Dies spielt eine wichtige Rolle beim Lösen von Differenzialgleichungen, weil diese in gewöhnliche algebraische Gleichungen übergehen.

- Es gilt die Poissonsche Summenformel

$$\sum_{k\in\mathbb{Z}} f(t_0 + k\,\delta) = \frac{\sqrt{2\pi}}{|\delta|}\sum_{n\in\mathbb{Z}}\mathcal{F}(f)\left(\frac{2\pi n}{\delta}\right)\exp\left(i\frac{2\pi n}{\delta}t_0\right). \tag{4.9}$$

  Diese Gleichung beruht darauf, dass $\sum_{t\in\mathbb{Z}} f(\cdot + t\,\delta)$ eine δ-periodische Funktion ist, die durch eine Fourierreihe dargestellt werden kann. Wegen (4.6) reicht es dabei o.B.d.A. $\delta = 1$ zu betrachten:

$$\begin{aligned}\sum_{k\in\mathbb{Z}} f(t_0 + k) = \sum_{n\in\mathbb{Z}}\int_0^1\sum_{k\in\mathbb{Z}} f(\tau + k)e^{-2\pi in\tau}\,d\tau\, e^{2\pi int_0} = \\ \sum_{n\in\mathbb{Z}}\int_{-\infty}^{\infty} f(\tau)e^{-2\pi in\tau}\,d\tau\, e^{2\pi int_0} = \sqrt{2\pi}\sum_{n\in\mathbb{Z}}\mathcal{F}(f)(2\pi n)\,e^{2\pi int_0}.\end{aligned} \tag{4.10}$$

  Eine wichtige Anwendung der Poissonschen Summenformel ist die Summation schlecht konvergenter Reihen. Fällt die Funktion f sehr langsam gegen unendlich ab, so fällt ihre Fouriertransformierte wegen der Reziprozität (4.5) im allgemeinen schneller, so das aus einer langsam eine rasch konvergierende Reihe wird.

4.1.1 Spezielle Fouriertransformierte

Die Fouriertransformierte einer Gaußglocke ist

$$\mathcal{F}\left(\frac{1}{\sqrt{2\pi}}e^{-t^2/2}\right) = \frac{1}{2\pi}e^{-\omega^2/2}\int_{-\infty}^{\infty} e^{-(t-i\omega)^2/2}\,dt = \frac{1}{\sqrt{2\pi}}e^{-\omega^2/2} \tag{4.11}$$

Die Gaussglocke ist also eine Eigenfunktion der Fouriertransformation zum Eigenwert 1. Die Fouriertransformation hat tatsächlich sehr viel mehr echt verschiedene Eigenfunktionen, die Familie der Hermitefunktionen [Pin02]. Wegen der Isometrieeigenschaft kann die Fouriertransformation aber nur Eigenwerte vom Betrag eins haben; tatsächlich hat sie nur die vier Eigenwerte ± 1 und $\pm i$, da $\mathcal{F}^4 = 1$. Daher ist jeder Eigenwert stark degeneriert, und der Eigenraum zu jedem Eigenwert unendlichdimensional.

Die (formale) Fouriertransformierte der δ-Funktion ist

$$\mathcal{F}(\delta(t))(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \delta(t) e^{-i\omega t}\, dt = \frac{1}{\sqrt{2\pi}}, \tag{4.12}$$

also einfach die konstante Funktion $1/\sqrt{2\pi}$, die ebensowenig wie die δ-Funktion eine quadratintegrable Funktion ist. Alternativ lässt sich die Beziehung aus der Fouriertransformierten der Gaußglocke mit sinkender Varianz herleiten.

4.1.2 Numerische kontinuierliche Fouriertransformation

Um die Eigenschaften der kontinuierlichen Fouriertransformation numerisch zu nutzen, machen wir den Grenzübergang $T \to \infty$ rückgängig und ziehen uns auf ein für die betrachtetete Funktion hinreichend großes T zurück, so dass $\int_{|t|>T} |f(t)|^2\, dt$ hinreichend klein ist. Ist das Signal wie in der Praxis endlich, so könnte T zum Beispiel so groß sein, dass das Signal vollständig abgedeckt ist. In jedem Fall ist das Signal eine Funktion f, die wir nur an auf einem äquidistanten Gitter $t_k = T\left(-\frac{1}{2} + \frac{k}{N}\right)$, $k = 0(1)N-1$ kennen. Dann ist mit $\omega_0 = \frac{2\pi}{T}$

$$\begin{aligned}\mathcal{F}(f)(\omega_0 n) &\approx \frac{1}{\sqrt{2\pi}} \int_{-T/2}^{T/2} f(t) e^{-in\omega_0 t}\, dt \approx \frac{1}{\sqrt{2\pi}} \sum_{k=0}^{N-1} f(t_k) e^{-in\omega_0 T\left(-\frac{1}{2}+\frac{k}{N}\right)} \frac{T}{N} \\ &= \frac{T}{N} \frac{e^{-2\pi i n/2}}{\sqrt{2\pi}} \sum_{k=0}^{N-1} f(t_k) e^{-2\pi i n k/N} = \frac{T}{N} \frac{(-1)^n}{\sqrt{2\pi}} \mathrm{DFT}(f(t_k))_n, \end{aligned} \tag{4.13}$$

wobei DFT die diskrete Fouriertransformation aus (3.53) bezeichnet. Die Koeffizienten der DFT sind periodisch, daher auch die eben Näherung $\mathcal{F}(f)$. Tatsächlich sollten die Koeffizienten aber nur als Frequenzen im Interval $[-\omega_0 N/2, \omega_0 N/2]$ interpretiert werden, alle Frequenzen außerhalb dieses Intervals sollten als Null angesehen werden. Der Grund dafür ist das Abtasttheorem, dass im folgenden Abschnitt besprochen wird. Dieses besagt, dass bei einem Zeitschritt $\Delta = T/N$ nur Kreisfrequenzen bis $\omega_0 N/2 = \pi/\Delta$ eindeutig gemessen werden können. Daher ist die Beschränkung auf das innerste Interval die natürlich Interpretation.

Die bekannten schnellen FFT-Routinen lassen sich also auch für die numerische Bearbeitung der kontinuierlichen Fouriertransformation nutzen. Auf diese Weise ist z.B. Abbildung 4.1 entstanden.

4.1.3 Abtasttheorem

In der Praxis sind Signale meist als diskrete Werte an (endlich vielen) äquidistanten Stellen beziehungsweise Zeitpunkten gegeben, zum Beispiel, weil ein Messgerät Daten in regelmäßigen Abständen liefert. Eine wichtige Frage ist, wie gut man das reale Signal und sein Frequenzspektrum aus den diskreten Datenpunkten rekonstruieren kann.

Hierzu betrachten wir zunächst ein *bandbreitenbeschränktes Signal* f, d.h. ein Signal, dessen Fouriertransformierte einen kompakten Träger $[-\omega_0,\omega_0]$ hat. Dann besagt die Poissonsche Summenformel (4.9), dass für $\omega \in [-\omega_0,\omega_0]$

$$\mathcal{F}(f)(\omega) = \sum_{k\in\mathbb{Z}} \mathcal{F}(e^{-i\omega\cdot} f)\,(2k\omega_0) = \frac{1}{\sqrt{2\pi}} \sum_{n\in\mathbb{Z}} e^{-i\omega n\Delta} f\,(n\Delta) \tag{4.14}$$

mit $\Delta = \pi/\omega_0$. Die Fouriertransformierte eines bandbreitenbeschränkten Signals lässt sich also *exakt* nur aus den Funktionswerten an den äquidistanten diskreten Stellen mit Abtastrate Δ berechnen. Dadurch kann natürlich auch die Funktion exakt aus ihrer Fouriertransformierten rekonstruiert werden:

$$f(t) = \frac{\Delta}{2\pi} \sum_{n\in\mathbb{Z}} f\,(n\Delta) \int_{-\omega_0}^{\omega_0} e^{-i\omega(t-n\Delta)}\,d\omega = \frac{\Delta}{\pi} \sum_{n\in\mathbb{Z}} f\,(n\Delta)\,\frac{\sin(\omega_0(t-n\Delta))}{t-n\Delta}. \tag{4.15}$$

Ist nun umgekehrt eine Signal $f(n\Delta)$ an äquidistanten diskreten Stellen gegeben, so lässt sich diesem umgekehrt gemäß (4.15) ein kontinuierliches Signal zuordnen, dessen Fouriertransformierte nur in $[-\omega_0,\omega_0]$ nicht verschwindet, wobei $\omega_0 = \pi/\Delta$. Das bedeutet, dass bei Abtastrate Δ nur Frequenzen bis zur Nyquist-Frequenz $f_{\text{Nyquist}} = \frac{1}{2\Delta}$ eindeutig abgetastet werden können, ähnlich wie im periodischen Fall.

4.2 Faltungen

Die *Faltung* der quadratintegrablen Funktionen f und g ist definiert als

$$(f \star g)(t) := \int_{-\infty}^{\infty} f(t')g(t-t')\,dt'. \tag{4.16}$$

Das negative Vorzeichen von t' in der zweiten Funktion sorgt dafür, dass die Faltung kommutativ ist. Weitere Eigenschaften der Faltung sind Linearität in den Komponenten und sogar Assoziativität. Die Faltung verhält sich also so ähnlich wie die klassische Multiplikation und wird daher mit dem Zeichen $\star$ bezeichnet. Vereinfacht gesagt, deponiert die Faltung an jedem Punkt t die Funktion f, skaliert mit $g(\cdot - t)$. Daher ist z.B.

$$(\delta(\cdot - t_0) \star g)(t) = g(t - t_0). \tag{4.17}$$

Wird nun statt der unendlich dünnen δ-Funktion zum Beispiel eine Gaußglocke gewählt, so wird die Funktion g verschmiert bzw. geglättet, siehe Abbildung 4.1.

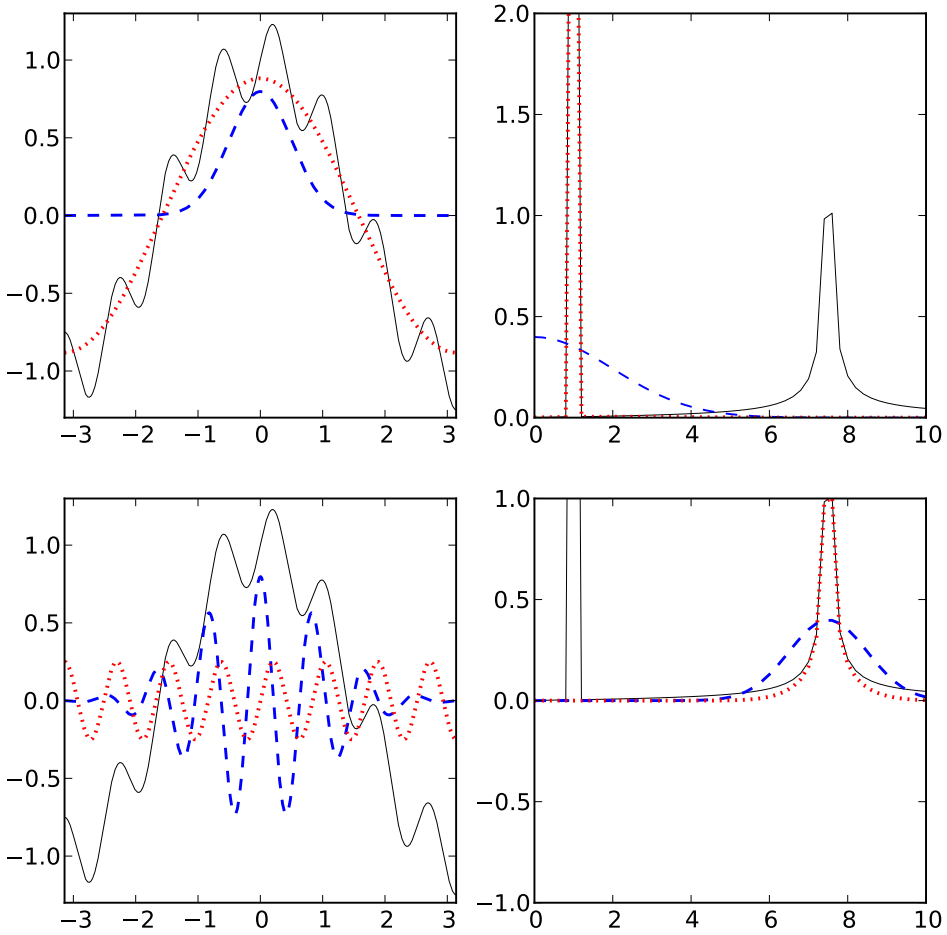


Abbildung 4.1: Oben links: Summe zweier Schwingungen $\cos(x)+0{,}25\sin(7{,}5x)$ (schwarze Linie), die mit einer Gaußglocke (blau gepunktet) gefaltet wird. Das Ergebnis (rote dicke Linie) ist die quasi ungestörte langsame Schwingung ohne die höherfrequente Schwingung, die weggeglättet wurde. Oben rechts: Fouriertransformierte der Funktionen. Klar sichtbar ist die hochfrequente Störung mit einer Frequenz von 7,5, die im gefilterten Signal fehlt. In der unteren Reihe wurde eine frequenzverschobene Gaußglocke benutzt, um statt der langsamen die schnelle Schwingung zu filtern; die Farbcodierung ist wie oben. Gemäß (4.7) bewirkt die Frequenzverschiebung eine Modulation der Gaußfunktion.

Die Fouriertransformierte der Faltung zweier Funktionen ist

$$\begin{aligned}\mathcal{F}(f \star g)(\omega) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t')g(t-t')\,dt' e^{-i\omega t}\,dt \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t')g(t) e^{-i\omega(t+t')}\,dt\,dt' \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t') e^{-i\omega t'}\,dt' \int_{-\infty}^{\infty} g(t) e^{-i\omega t}\,dt = \sqrt{2\pi}\mathcal{F}(f)(\omega)\mathcal{F}(g)(\omega).\end{aligned} \tag{4.18}$$

Die Faltung geht also in eine punktweise Multiplikation über. Im Fourierraum lässt sich also sehr viel schneller falten, als im Realraum, wo ja für jeden Punkt ein Integral zu lösen ist.

4.2.1 Filter

Außerdem lässt (4.18) noch eine weitere Interpretation der Faltung der Funktion g mit der Funktion f zu: ist f bzw. $\mathcal{F}(f)$ reellwertig und symmetrisch, so werden die einzelnen Frequenzkomponenten der Funktion g mit den Frequenzanteilen von f gestreckt bzw. gestaucht, g also frequenzgefiltert. Durch Wahl einer symmetrischen und reellwertigen Fouriertransformierten und Rücktransformation lassen sich also beliebige Frequenzfilter realisieren; in der Praxis wird natürlich direkt im Fourierraum gefiltert. Später lernen wir die numerische diskrete Fouriertransformation kennen, die nicht zuletzt wegen dieser Filtereigenschaften so wichtig ist. Abbildung 4.1 illustriert einen (gaußschen) Tiefpassfilter und einen Bandfilter, die nur bestimmte Frequenzen passieren lassen. Daher wird beim Tiefpassfilter die aufgeprägte hochfrequente Schwingung unterdrückt, beim Hochpassfilter hingegen die an sich dominante langsame Schwingung.

4.2.2 Antwort zeitinvarianter linearer Systeme

Eine weitere wichtige Anwendung der Faltung ist die Bestimmung der Antwort eines zeitinvarianten linearen Systems auf ein beliebiges Eingangssignal. Einfach zu messen ist typischerweise die Antwort $A_\theta(t)$ des Systems auf einen Einschaltvorgang, also ein Eingangssignal der Form

$$\theta(t) = \begin{cases} 0 & \text{für } t < 0 \\ 1 & \text{für } t \geq 0. \end{cases} \tag{4.19}$$

Um daraus die Antwort auf ein beliebiges Eingangssignal zu bestimmen, schreiben wir das Eingangssignal f als $f = f \star \delta$. Wegen der Linearität der Faltung und der Systemantwort ist die Antwort auf das Signal f gegeben durch die Faltung $A_f(t) = f \star A_\delta$ mit der Antwort auf einen δ-Impuls. Diese Antwort wiederum lässt sich aus der Sprungantwort durch einfach Ableitung erhalten, was mit Hilfe der Fouriertransformierten sehr bequem zu berechnen ist:

$$A_f(t) = f \star A_\delta = f \star \frac{d}{dt} A_\theta = \sqrt{2\pi}\mathcal{F}^{-1}\big(i\omega\mathcal{F}(f)\mathcal{F}(A_\theta)\big). \tag{4.20}$$

4.3 Kreuz- und Autokorrelation

Bis jetzt haben wir uns mit der Verarbeitung idealer Signale beschäftigt, die zu einem gegebenen Zeitpunkt einen prinzipiell eindeutig vorherbestimmten Wert haben. Reale Signale sind aber oft verrauscht, entweder durch Messungenauigkeiten, Bauteiltoleranzen oder prinzipielle stochastische Prozesse. Trotzdem möchte man oft wissen, ob zwei gemessene Signale von einander abhängig, *korreliert*, sind. Zum Beispiel könnte man die Position eines Elektrons und seinen Spin, die Menge der verkauften Eis- und Sonnencreme, oder auch die Position eines Pendels zu zwei verschiedenen Zeitpunkten betrachten. In den beiden letzteren Fällen werden diese im allgemeinen korreliert, also abhängig sein. Allerdings gibt dies keinen Aufschluss über den dahinterstehenden kausalen Mechanismus. Im Fall des Pendels rührt die Korrelation daher, dass es sich in kurzer Zeit nicht beliebig weit von seiner Ausgangsposition bewegen kann. Bei der Eis- und Sonnencreme wird es vermutlich auch eine Korrelation geben, aber weder erzeugt Eiscreme Sonnenbrand, noch macht Sonnencreme Lust auf Eis. Allerdings haben wir Menschen nunmal bei strahlendem Sonnenschein mehr Lust auf Eis, aber brauchen auch Sonnencreme.

Formal betrachten wir zunächst zwei Observablen A und B. Diese beiden heißen genau dann unkorreliert, falls die Mittelwerte $\langle A \cdot B\rangle = \langle A\rangle \langle B\rangle$ bzw. $\langle (A - \langle A\rangle)(B - \langle B\rangle)\rangle = 0$ erfüllen. Im allgemeinen wird man aber vielleich nicht erwarten, dass sich die Änderung einer Observablen unmittelbar in einer anderen niederschlägt, sondern erst nach einer Zeit τ. Man betrachtet daher die Korrelation zwischen A und B mit einem zeitlichen Versatz von τ, die *Kreuzkorrelationsfunktion*:

$$C(A,B)(\tau) := \langle A(0)B(\tau)\rangle\,, \tag{4.21}$$

wobei die Signale A und B zeitinvariant sein sollen, so dass der Zeitpunkt $t = 0$ beliebig gewählt sein kann. Für große τ dekorrelieren die Signale, daher gilt $C(A,B) \to \langle A\rangle \langle B\rangle$ für $\tau \to \infty$. Wird stattdessen die normierte Kreuzkorrelation $C(A-\langle A\rangle\,, B-\langle B\rangle)$ betrachtet, verschwindet dieses also im Limit $\tau \to \infty$.

$\langle\cdot\rangle$ bezeichnet in der Physik üblicherweise den Ensemblemittelwert, also den Mittelwert über alle möglichen Realisationen des Experiments. Sind nun $A = A(t)$ und $B = B(t)$ zeitliche Messreihen eines zeitinvarianten Systems, so ermittelt man die Mittelwerte üblicherweise als Zeitmittelwerte, also Integrale über die Zeit:

$$C(A,B)(\tau) = \langle A(0) \cdot B(\tau)\rangle \stackrel{!}{=} \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} A(t)B(t+\tau)\,dt. \tag{4.22}$$

Das Ausrufezeichen soll andeuteten, dass dies eine Annahme ist, denn die Gleichheit gilt nur genau dann, wenn das System *ergodisch* ist, d.h., dass der Prozess bei einer unendlich langen zeitlichen Messung alle möglichen Realisationen einmal besuchen wird. Diese Annahme wird meist gemacht, obwohl die Ergodizität für die meisten Systeme nicht bewiesen werden kann. Hinzu kommt, dass ja in der Praxis niemals über beliebig lange Zeiträume gemittelt werden kann. Daher können auch endliche, aber hohe Energiebarrieren zu einem systematischen Fehler führen.

$\langle A\rangle$ bzw. $\langle B\rangle$ sind oft gar nicht genau bekannt, und müssen numerisch durch (Zeit-)Mittelung der Daten bestimmt werden. Da hier aber normalerweise dieselben Daten

zugrunde gelegt werden, die auch zu Berechnung der Kreuzkorrelation selber genutzt werden, sind diese notwendigerweise korreliert, was zu kleinen Abweichungen in der Kreuzkorrelationsfunktion führt. Im häufigen Fall, dass eine Observable aus Symmetriegründen einen Mittelwert von Null haben muss, sollte daher auf keinen Fall der numerische gemessene Mittelwert abgezogen, „um das Ergebnis zu verbessern“. Diese übliche Praxis ist falsch, da sie ja erzwingt, dass der letzte Datenpunkt der gemessenen Kreuzkorrelationsfunktion notwendigerweise auf $\langle A\rangle\langle B\rangle$ abfällt, selbst wenn einfach nur das Messinterval zu kurz gewählt wurde. Daher führt diese Methode zu einer Unterschätzung der Langzeitkorrelationen!

Analog zu (4.22) kann man die Kreuzkorrelation auch für *endliche* Signale definieren. Für zwei quadratintegrable Signale f und g ist in Analogie die Kreuzkorrelationsfunktion definiert als

$$C(f,g)(\tau) = \int_{-\infty}^{\infty} f(t)g(t+\tau)\,dt, \tag{4.23}$$

wobei wegen der Quadratintegrabilität auf die Normierung verzichtet werden kann. Diese Kreuzkorrelation misst keine Korrelationen im stochastischen Sinne mehr, weil das Integral nun keine Zeitmittelung mehr darstellt. Stattdessen ist $C(f,g)(\tau)$ in diesem Fall ein Maß dafür, wie sehr sich die Signale f und g mit einem Zeitversatz τ im Verlauf ähneln. Diese Form der Kreuzkorrelation ähnelt einer Faltung sehr. Tatsächlich ist

$$C(f,g)(\tau) = \big(f \star g(-\cdot)\big)(-\tau) = \big(f(-\cdot) \star g\big)(\tau) \tag{4.24}$$

und kann damit effizient im Frequenzraum bestimmt werden:

$$C(f,g) = \sqrt{2\pi}\mathcal{F}^{-1}\big(\mathcal{F}(f)(-\omega)\mathcal{F}(g)(\omega)\big). \tag{4.25}$$

Kommen wir nun zu unserem Ausgangsproblem mit zwei stochastischen, zeitinvarianten Variablen A und B und dem Zeitmittel zurück. Wie üblich nehmen wir an, dass A und B an endlich vielen, diskreten Zeitpunkten $tk\Delta$, $k = 0(1)n-1$, gemessen wurden. Die Messung erstreckt sich also über den Zeitraum $[0,T]$ mit $T = n\Delta$. Dann ist eine Näherung für die Kreuzkorrelation von A und B gegeben durch

$$C(A,B)(k\Delta) = \langle A(0)\cdot B(k\Delta)\rangle \approx \frac{1}{N-k}\sum_{l=0}^{N-k} A(l\Delta)B\left((l+k)\Delta\right) \quad \text{für } k = 0(1)K-1. \tag{4.26}$$

Die unterschiedliche Gewichtung $1/(N-k)$ ergibt sich durch die unterschiedliche Anzahl an Messungen für den Versatz k. Für große Versätze ist die Anzahl der Messungen sehr klein (für $k = N-1$ nur noch eine), daher muss $k \ll N$ sein. Form (4.26) ist numerisch allerdings nicht sehr effizient auszuwerten, da im allgemeinen $2NK$ viele Operationen benötigt werden, und K meist einige hundert Datenpunkte beträgt. Daher liegt es nahe, auch hier die FFT zur Beschleunigung einzusetzen. (4.25) und (4.13) zusammen ergeben für gleich lange Messreihen A und B:

$$\begin{aligned} C(A,B)(k\Delta) &\approx \sqrt{2\pi}\mathcal{F}^{-1}\big(\mathcal{F}(f)(-\omega)\mathcal{F}(g)(\omega)\big)(k\Delta) \\ &\approx \frac{1}{N}\,\mathrm{iDFT}\big(\overline{\mathrm{DFT}(A)(n)}\,\mathrm{DFT}(B)(n)\big)(k). \end{aligned} \tag{4.27}$$

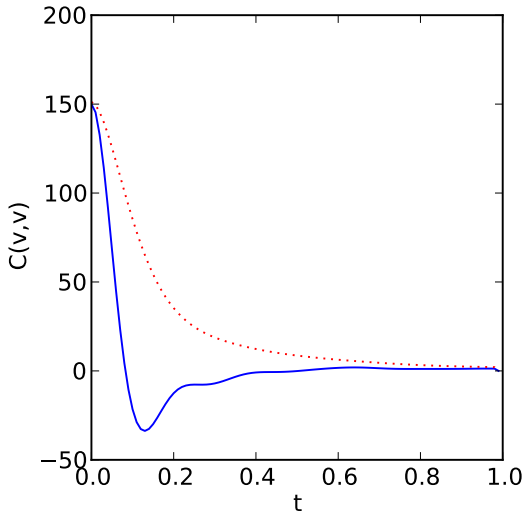


Abbildung 4.2: Geschwindigkeitsautokorrelationsfunktion einer temperierten Lennard-Jones-Flüssigkeit mit niedriger Dichte (rot gepunktet) und hoher Dichte (blau durchgezogen). Für die niedrige Dichte ist die Autokorrelationsfunktion im wesentlichen exponentiell abfallend, mit einer Korrelationszeit $\tau_c = 0.17$, die durch die Kopplungszeit des Thermostaten bestimmt ist.

Man sollte dabei beachten, dass durch die Benutzung der DFT das Signal implizit mit Periode $N\Delta$ periodisiert wird. Daher sollte $C(A,B)(k\Delta)$ nur für $k \ll N/2$ interpretiert werden. Die Berechnung der 2 DFTs sowie der inversen DFT braucht etwa $6N \log N$ Operationen, was normalerweise wesentlich weniger als die $2NK$ der direkten Berechnung ist.

In Python sieht die Berechnung der Kreuzkorrelation so aus:

```
import numpy
import numpy.fft as fft

def kreuzkorrelation(A, B):
    ftA = fft.fft(A).conj()
    ftB = fft.fft(B)
    return numpy.real(fft.ifft(ftA*ftB))/A.shape[0]
```

4.3.1 Autokorrelationsfunktion

Im Spezialfall $A = B$ spricht man von der *Autokorrelationsfunktion*. Diese ist offenbar symmetrisch, daher sind nur Zeitversätze $\tau \geq 0$ von Interesse. Die Autokorrelationsfunktion misst gewissermaßen, wie lange es dauert, bis die Observable nicht mehr von seinem vorherigen Wert abhängt, wann es diesen sozusagen „vergisst“. In dem häufigen Fall, dass die Autokorrelationsfunktion zunächst exponentiell abfällt, lässt sich dem Gedächtnis

eine Zeitkonstante τ_c, die *Korrelationszeit*, zuordnen. Diese kann man entweder durch einen geeigneten Funktionsfit bestimmen, oder aber durch Integration aus

$$\int_0^\infty C(A,A)(\tau)\,d\tau = \int_0^\infty C(A,A)(0)e^{-\tau/\tau_c}\,d\tau = \tau_c\,C(A,A)(0). \tag{4.28}$$

Abbildung 4.2 zeigt die Geschwindigkeitsautokorrelation $C(v,v)$ einer temperierten Lennard-Jones-Flüssigkeit bei zwei verschiedenen Dichten. Die Temperierung wird dabei mit Hilfe eines Thermostaten erreicht, der die Teilchen stochastisch an ein Wärmebad koppelt. Dadurch dekorreliert die Geschwindigkeit eines Teilchens in einer Korrelationszeit von etwa 1/6.

$C(v,v)$ wird häufig dazu benutzt, um die Diffusionskonstante $D = \int_0^\infty C(v,v)(\tau)\,d\tau$ des Systems zu bestimmen, die also eng mit der Korrelationszeit des Thermostaten verwandt ist. Bei niedriger Dichte ist das System annähernd ein ideales Gas, und die Teilchen dekorrelieren im wesentlichen nur durch den Einfluss des Thermostaten, der durch Zufallskräfte wirkt, daher ist $C(v,v)$ tatsächlich gut exponentiell abfallend. Formel (4.28) bestimmt die Korrelationszeit mit guter Genauigkeit zu etwa $\tau_c = 0.17$, wie durch den Thermostaten zu erwarten. Im Falle der dichteren Flüssigkeit hingegen kann diese Formel nicht angewendet werden, da die Autokorrelation kein einfacher exponentieller Abfall mehr ist, da auch Stoßprozesse eine wichtige Rolle spielen. Diese führen zum Durchschwingen der Autokorrelationsfunktion.

4.4 Messfehlerabschätzung

In diesem letzten Abschnitt zur Datenanalyse geht es darum, wie der der Messfehler bei der Messung des Erwartungswerts einer stochastischen Observable abgeschätzt werden kann. Wir betrachten also eine Messreihe x_i, $i = 1(1)N$, die verschiedene Messungen einer stochastischen Observablen x darstellen. Der Erwartungswert dieser Observablen lässt sich dann bekanntermaßen als

$$\langle x\rangle \approx \bar{x} := \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.29}$$

abschätzen. Doch was ist nun der Fehler, den wir mit dieser Schätzung machen? Dieser ist die zu erwartende quadratische Abweichung

$$\left\langle (\overline{x} - \langle x\rangle)^2 \right\rangle = \frac{1}{N^2}\sum_{i,j=1}^{N}\langle x_i x_j\rangle - \frac{2}{N}\sum_{i=1}^{N}\langle x_i\rangle\langle x\rangle + \langle x\rangle^2 = \frac{2}{N^2}\sum_{i>j}\langle x_i x_j\rangle + \frac{1}{N}\left\langle x^2\right\rangle - \langle x\rangle^2 \tag{4.30}$$

An dieser Stelle nimmt man nun an, dass die Messungen paarweise unabhängig sind, also, dass $\langle x_i x_j\rangle = \langle x_i\rangle\langle x_j\rangle$ für $i \neq j$. In der Praxis lässt sich das zum Beispiel durch Betrachten der Autokorrelationsfunktion sicherstellen, in dem nur Messwerte mit einem zeitlichen Abstand berücksichtigt werden, der sehr viel größer als die Korrelationszeit

ist. Unter der Annahme, dass die Messungen x_i alle paarweise unabhängig sind gilt also weiter

$$\left\langle (\overline{x} - \langle x \rangle)^2 \right\rangle = \frac{N(N-1)}{N^2} \langle x \rangle^2 + \frac{1}{N} \left\langle x^2 \right\rangle - \langle x \rangle^2 = \frac{1}{N} \left(\left\langle x^2 \right\rangle - \langle x \rangle^2 \right) = \frac{1}{N} \sigma^2(x). \quad (4.31)$$

Sind die Messungen unabhängig voneinander, ist der quadratische Fehler von N Messungen also gerade ein N-tel der Varianz

$$\sigma^2(x) = \left\langle \left(x - \langle x \rangle \right)^2 \right\rangle = \left\langle x^2 \right\rangle - 2 \left\langle \langle x \rangle x \right\rangle + \langle x \rangle^2 = \left\langle x^2 \right\rangle - \langle x \rangle^2 , \quad (4.32)$$

die die erwartete quadratische Abweichung einer Messung vom Mittelwert angibt. Sie ist eine Eigenschaft der zu messenden Observablen und daher nicht von der Anzahl der Messungen abhängig.

Als Fehlerbalken wird üblicherweise die *Standardabweichung* der Messung $\overline{x}$, angegeben. Diese ist durch

$$\sqrt{\langle (\overline{x} - \langle x \rangle)^2 \rangle} = \frac{1}{\sqrt{N}} \sigma(x) \quad (4.33)$$

gegeben. Dies bedeutet, dass für eine Halbierung des Fehlerbalken bereits viermal soviele Messungen durchgeführt werden müssen, und für eine Größenordnung an Genauigkeit hundert Mal soviele. Besonders für Computersimulationen ist das ein Problem, da die Rechenzeit im allgemeinen proportional zur Anzahl der Messungen ist. Dauert also eine Simulation eine Woche, was nicht ungewöhnlich ist, so würde eine Messung mit einer Größenordnung mehr Genauigkeit fast zwei Jahre in Anspruch nehmen!

Zur Berechnung des Fehlers benötigen wir noch eine Schätzung der Varianz, die im allgemeinen ebensowenig wie der Mittelwert bekannt ist und geschätzt werden muss. Dazu ersetzt man die Mittelwerte in $\left\langle x^2 \right\rangle - \langle x \rangle^2$ durch den Schätzer (4.29). Für diesen Ausdruck gilt:

$$\left\langle \overline{x^2} - \bar{x}^2 \right\rangle = \left\langle x^2 \right\rangle - \frac{1}{N^2} \sum_{i=1}^{n} \left\langle x_i^2 \right\rangle - \frac{1}{N^2} \sum_{i \neq j} \langle x_i x_j \rangle = \frac{N-1}{N} \left(\left\langle x^2 \right\rangle - \langle x \rangle^2 \right), \quad (4.34)$$

wobei wir wieder annehmen, dass die Messungen unabhängig voneinander sind. Das der Ausdruck auf der linken Seite selber kein guter Schätzer ist, liegt also daran, dass für $\bar{x}$ zweimal derselbe Datensatz benutzt wurde. Ein guter Schätzer für $\sigma^2(x)$ ergibt sich aus der Umkehrung von (4.34):

$$\sigma^2(x) \approx \frac{N}{N-1} \left(\overline{x^2} - \bar{x}^2 \right) = \frac{1}{N-1} \left(\sum_{i=1}^{N} x_i^2 - N \bar{x}^2 \right). \quad (4.35)$$

Auf dem Computer lassen sich also $\langle x \rangle$ und $\sigma^2(x)$ bequem in einem Durchlauf der Daten abschätzen:

```
sum  = 0
sum2 = 0
for v in x:
```

```
    sum  += v
    sum2 += v*v
mittel = sum/len(x)
sigma2 = (sum2 - len(x)*mittel**2)/(len(x)-1)
fehler = sqrt(sigma2/N)
```

In der Praxis wird oft einfach angenommen, dass die Messungen unabhängig sind. Was passiert nun, wenn dies nicht der Fall ist? Betrachten wir also eine Observable die o.B.d.A. Mittelwert $\langle x\rangle = 0$ habe. Dann ist wie oben gezeigt

$$\left\langle (\overline{x} - \langle x\rangle)^2 \right\rangle = \frac{2}{N^2} \sum_{i>j} \langle x_i x_j\rangle + \frac{1}{N}\sigma^2(x). \tag{4.36}$$

Der tatsächliche Fehler ist also um $\frac{2}{N^2}\sum_{i>j}\langle x_i x_j\rangle$ größer als man aufgrund der Varianz erwarten würde. In die Abschätzung für die Varianz ging ebenfalls die Unabhängigkeit ein, für diese gilt nun

$$\frac{N}{N-1}\left\langle \overline{x^2} - \bar{x}^2 \right\rangle = \frac{1}{N-1}\left(N\left\langle x^2\right\rangle - \left\langle x^2\right\rangle - \frac{1}{N}\sum_{i\neq j}\langle x_i x_j\rangle \right) \tag{4.37}$$

$$= \sigma^2(x) - \frac{2}{N(N-1)}\sum_{i>j}\langle x_i x_j\rangle\,, \tag{4.38}$$

d.h., die Varianz wird zusätzlich noch um $\frac{1}{(N-1)N^2}\sum_{i\neq j}\langle x_i x_j\rangle$ *unterschätzt*. Insgesamt wird der quadratischen Fehler also um ebenfalls

$$\begin{aligned} F &= \left\langle (\overline{x} - \langle x\rangle)^2 \right\rangle - \frac{1}{N}\frac{N}{N-1}\left\langle \overline{x^2} - \bar{x}^2 \right\rangle \\ &= \left(\frac{2}{N^2} + \frac{2}{(N-1)N^2}\right)\sum_{i>j}\langle x_i x_j\rangle = \frac{2}{N(N-1)}\sum_{j=1}^{N}\sum_{n=1}^{N-j}\langle x_j x_{j+n}\rangle \end{aligned} \tag{4.39}$$

unterschätzt. Für eine Observable x, die in gleichmäßigen Zeitabständen Δ gemessen wird, lässt sich dieser Ausdruck mit Hilfe der Autokorrelationsfunktion $C(\tau) = C(x,x)(\tau)$ abschätzen:

$$F \approx \frac{2}{N(N-1)}\sum_{j=1}^{N}\int_{n=0}^{N-j} C(n\Delta) \approx \frac{2}{N}\int_{n=0}^{\infty} C(n\Delta) = \frac{2}{N}\frac{\tau_c}{\Delta}C(0) = \frac{2\tau_c}{\Delta}\frac{\sigma^2(x)}{N}, \tag{4.40}$$

wobei τ_c die Korrelationszeit gemäß (4.28) ist und angenommen wurde, dass $\tau_c \ll N\Delta$, so dass die Erweiterung des Integrals bis unendlich keinen nennenswerten Beitrag mehr liefert. Verglichen mit (4.31) erhält man mit korrelierten Daten eine Abschätzung für den Fehler, die um einen Faktor $1 - 2\tau_C/\Delta$ zu klein ist, was sich durch noch so gute Statistik nicht ausgleichen lässt! Daher lassen sich ohne Kenntnis der Korrelationszeit der Daten keine verlässlichen Aussagen über die Güte der Daten machen. Nur, wenn $\Delta \gg \tau_c$, ist der Fehler durch korrelierte Daten vernachlässigbar.

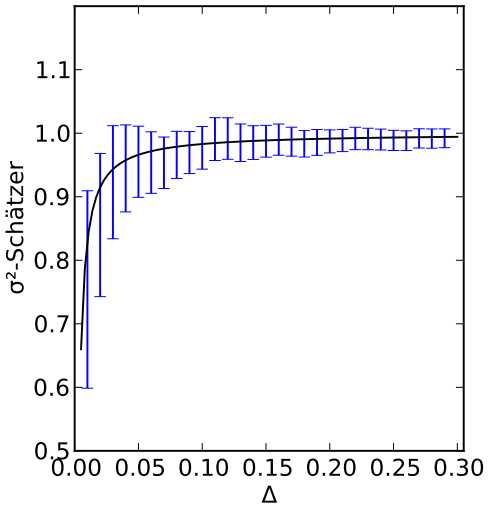


Abbildung 4.3: Geschätzte, scheinbare Varianz eines dünnen temperierten Lennard-Jones-Systems als Funktion der gewählten Schrittweite Δ. Die schwarze Linie zeigt die Abschätzung der scheinbaren Varianz gemäß (4.40) als $(1 - 2\tau_C/(\Delta N))\sigma^2(x)$, was gut zu den gemessenen Daten passt.

Abbildung 4.3 zeigt die scheinbare Varianz des weniger dichten thermalisierten Lennard-Jones-Systems als Funktion der gewählten Schrittweite. Wie gezeigt ist die Korrelationszeit dieses Systems $\approx 0{,}17$, so dass bei rascheren Messungen eine Unterschätzung der Varianz und auch des Fehlers zu erwarten ist, was von der Abbildung bestätigt wird.

5 Nichtlineare Gleichungssysteme

Im zweiten Kapitel hatten wir uns mit der Lösung linearer Gleichungssysteme beschäftigt, die ja eine wesentliche Grundlage der numerischen Mathematik darstellen. Allerdings tauchen in der Praxis, besonders in der Physik, leicht auch nichtlineare Gleichungssysteme auf. In diesem Fall kann man meist keine allgemeine Aussage über Existenz und Anzahl der Lösungen machen, und kann auch keine exakten Verfahren zur Lösung angeben.

Nichtlineare Gleichungssysteme werden typischerweise in zwei Formen betrachtet. Sei eine Funktion $f : M \in \mathbb{R}^n \to \mathbb{R}^n$ gegeben. Dann suchen wir die *Nullstellen*

$$x, \quad \text{so dass } f(x) = 0, \tag{5.1}$$

also die Lösungen zur Gleichung $f(x) = 0$. Man beachte, dass anders als im Fall der linearen Gleichungssysteme gefordert wird, dass der Bildraum wie auch der Ursprungsraum M zu einem Vektorraum derselben Dimension n gehören. Ohne diese Voraussetzung ist eine eindeutige Lösung im allgemeinen unmöglich. Anders als im Falle der linearen Gleichungssysteme ist es hier auch nicht ohne weiteres möglich, den Lösungsraum anzugeben, falls die Lösung nicht eindeutig ist. Tatsächlich kann der Lösungsraum ja einen beliebig komplexe Mannigfaltigkeit innerhalb M darstellen, die dann gar nicht geschlossen parametrisiert werden kann. Das macht die numerische Bestimmung dieser Lösungsmannigfaltigkeit sehr schwierig.

Alternativ können wir für eine Funktion $g : M \in \mathbb{R}^n \to \mathbb{R}^n$ die *Fixpunkte* suchen. Diese sind

$$x, \quad \text{so dass } g(x) = x, \tag{5.2}$$

also die Lösungen der Gleichung $g(x) = x$. Eine Fixpunktgleichung lässt sich natürlich stets auch als Nullstellenproblem mit $f(x) = g(x) - x$ formulieren; anders herum geht dies im allgemeinen nicht.

Die beiden Formulierungen unterscheiden sich allerdings im natürlichen Lösungsansatz. Die Nullstellengleichung (5.1) ähnelt dem linearen Gleichungssystem (2.1). Das *Newtonverfahren* beruht auf einer lokalen Linearisierung und dem Lösen dieses linearen Gleichungssystems. Die Fixpunktgleichung hingegen legt nahe, den Fixpunkt durch *sukzessive Substitution* zu suchen: $x_0 \to g(x_0) \to g(g(x_0)) \to \ldots$.

5.1 Sukzessive Substitution

Eine Abbildung $g : M \to M$ mit $M \subset \mathbb{R}^n$ heißt Lipschitz-stetig (L-stetig), falls es ein $L \in \mathbb{R}$ gibt, so dass

$$\|g(x) - g(y)\| \leq L \|x - y\| \quad \forall x,y \in M. \tag{5.3}$$

Alle auf M differenzierbaren Funktionen mit beschränkter Ableitung sind L-stetig, wenn ihre Ableitung beschränkt ist. Die Lipschitzkonstante ergibt sich aus dem Mittelwertsatz zu $L = \max_{x\in M} \|g'(x)\|$. Es gibt allerdings noch mehr L-stetige Funktionen, zum Beispiel die in 0 nicht differenzierbare Betragsfunktion, die auf ganz $\mathbb{R}$ L-stetig mit $L = 1$ ist. Auf der anderen Seite ist offenbar jede L-stetige Funktion auch stetig, d.h., die L-stetigen Funktionen sind eine eigene Klasse zwischen den stetigen und differenzierbaren Funktionen.

Hat eine Funktion $g : M \to M$ eine Lipschitz-Konstante $L < 1$, so heißt g *kontrahierend*, weil zwei verschiedene Punkte durch die Abbildung stets näher aneinander geschoben werden. Wir betrachten nun einen beliebigen Startpunkt $x_0 \in M$ und definieren damit die Folge der *sukzessive Substitution*:

$$x_n := g(x_{n-1}) \quad \text{für } n \geq 1. \tag{5.4}$$

Dann gilt für alle n,$m \in \mathbb{N}$ der *Banachsche Fixpunktsatz*

$$\begin{aligned}
\|x_{n+m} - x_n\| &= \left\| \sum_{k=0}^{m-1} x_{n+k+1} - x_{n+k} \right\| \leq \sum_{k=0}^{m-1} \|x_{n+k+1} - x_{n+k}\| \\
&= \|g(g(...g(x_{n+1}))) - g(g(...g(x_n)))\| + \cdots \\
&\quad + \|g(g(x_{n+1})) - g(g(x_n))\| + \|g(x_{n+1}) - g(x_n)\| + \|x_{n+1} - x_n\| \\
&\leq \sum_{k=0}^{m-1} L^k \|x_{n+1} - x_n\| \leq \frac{1}{1-L} \|x_{n+1} - x_n\| \leq \frac{L^n}{1-L} \|g(x_0) - x_0\| .
\end{aligned} \tag{5.5}$$

Die sukzessive Substitution definiert also eine Cauchyfolge, die in M konvergiert, sofern M abgeschlossen ist (z.B. $M = \mathbb{R}^n$ oder M Einheitskugel). Für den Grenzwert $\overline{x}$ dieser Folge gilt

$$\overline{x} = \lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} g(x_n) = g(\overline{x}), \tag{5.6}$$

er ist also ein Fixpunkt. Das Verfahren wird abgebrochen, wenn $|x_n - x_{n-1}| = |g(x_{n-1}) - x_{n-1}|$ hinreichend klein ist.

Wir betrachten nun zwei Fixpunkte $\overline{x}$ und $\overline{y}$. Dann gilt

$$\|\overline{x} - \overline{y}\| = \|g(\overline{x}) - g(\overline{y})\| \leq L \|\overline{x} - \overline{y}\| \implies \overline{x} = \overline{y}. \tag{5.7}$$

Das bedeutet, dass es nur genau einen Fixpunkt $\overline{x}$ von g in M gibt, und dass die sukzessive Substitution für jeden Startwert *global* gegen $\overline{x}$ konvergiert. (5.5) gibt auche eine *a priori*-Abschätzung des Fehlers:

$$\|\overline{x} - x_n\| \leq \frac{L^n}{1-L} \|g(x_0) - x_0\| \tag{5.8}$$

sowie eine Abschätzung der Konvergenzrate:

$$\frac{\|x_{n+1} - \overline{x}\|}{\|x_n - \overline{x}\|} = \frac{\|G(x_n) - G(\overline{x})\|}{\|x_n - \overline{x}\|} \leq L < 1. \tag{5.9}$$

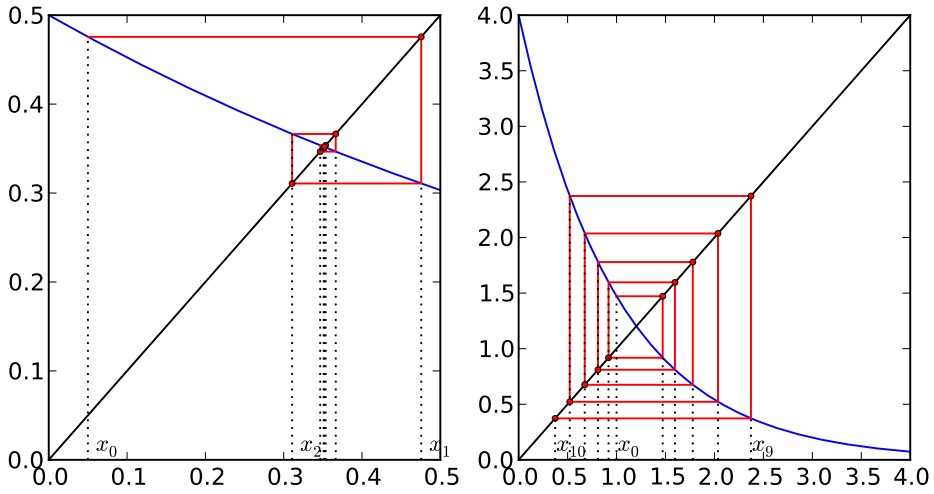


Abbildung 5.1: Sukzessive Substitution mit Funktion $g(r) = e^{-r}/\phi_0$ mit $\phi_0 = 2$ (links) und $\phi_0 = 1/4$ (rechts). Blau durchgezogen ist die Funktion g, die Winkelhalbierende ist schwarz dargestellt. Die Punkte auf der Winkelhalbierenden markieren die Punkte (x_1,x_1), (x_2,x_2) usw., durch die das Lot auf g gefällt wird, um den nächsten Punkt der sukzessiven Substitution zu erhalten. Im linken Graph sind die ersten sieben Glieder dargestellt, die exponentielle Konvergenz ist gut zu sehen. Im rechten Graph konvergiert das Verfahren nicht mehr.

Die sukzessive Substitution konvergiert also linear.

Neben dieser globalen Konvergenzeigenschaft konvergiert die sukzessive Substituion auch lokal: ist $g : \mathbb{R}^n \to \mathbb{R}^n$ eine differenzierbare Funktion und hat einen Fixpunkt $\overline{x}$ mit $\|g'(x)\| < 1$, so gibt es eine Umgebung des Fixpunktes, in dem die sukzessive Substitution gegen diesen Fixpunkt konvergiert.

5.1.1 Beispiel

Als Beispiel für eine Anwendung des Banachschen Fixpunktsatzes betrachten wir die dimensionslose Form des Yukawa- oder Debye-Hückel-Potentials $\phi(r) = e^{-r}/r$. Wir fragen uns, wann für welches r dieses Potential einen gegeben Wert ϕ_0 annimmt. Das führt zu der Fixpunktgleichung

$$g(r) = \frac{e^{-r}}{\phi_0} = r \tag{5.10}$$

Die linke Seite ist eine auf $[0,\infty)$ L-stetige Funktion mit $L = 1/\phi_0$, wie man durch Ableiten leicht sieht.

Abbildung 5.1 zeigt die sukzessive Substitution für $g(r)$. Graphisch lässt sich das Verfahren visualisieren, in dem in jeder Iteration der Funktionswert $x_{n+1} = y = g(x_n)$ an der Winkelhalbierenden $y = x$ auf die x-Achse zurückgespiegelt wird. Im linken Graph ist

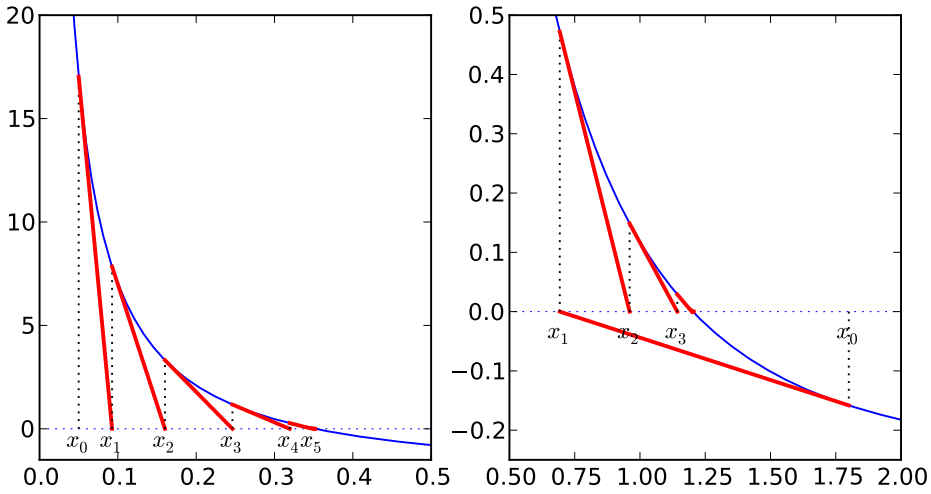


Abbildung 5.2: Newtonverfahren für die Funktion $f(r) = e^{-r}/r - \phi_0$. Wie schon beim Graphen zur sukzessiven Substitution ist links $\phi_0 = 2$, rechts $\phi_0 = 1/4$, allerdings konvergiert das Newtonverfahren für beide Werte. Blau dargestellt ist f, die roten, dicken Linien stellen die Tangenten dar, deren Nullstellen die neuen Näherungen für die gesuchte Nullstelle von f sind.

$\phi_0 = 2$ und damit $L = 1/2$, so dass die sukzessive Substitution exponentiell konvergiert. Für das letzte abgebildete Glied, x_7, gilt $|x_7 - \overline{x}| \leq 1/2^8 = 1/256$. Im rechten Graph ist $\phi_0 = 1/4$ und damit $L = 4$. Insbesondere ist auch im Fixpunkt $g'((\overline{x})) > 1$. Abgebildet sind die ersten zehn Glieder der sukzessiven Substitution, die hier nicht mehr konvergiert. Wird hingegen ϕ_0 so gewählt, dass $g'((\overline{x})) < 1$, aber $L > 1$, so konvergiert das Verfahren zwar noch, aber nicht mehr exponentiell.

5.2 Newtonverfahren in einer Dimension

Nachdem wir bis jetzt die suzessive Substitution zur Bestimmung von Fixpunkten betrachten haben, geht es nun um die Nullstellensuche. Sei also zunächst eine stetig differenzierbare Funktion $f : [a,b] \to \mathbb{R}$ gegeben und deren Nullstellen x, $f(x) = 0$, gesucht. Ähnlich wie bei der sukzessiven Iteration starten wir mit einem Startwert x_0. Um uns nun der Nullstelle der Funktion zu nähern, linearisieren wir in der aktuellen Näherung x_n und lösen nach der Nullstelle x_{n+1} auf:

$$x_{n+1} = g(x_n) := x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{für } n \geq 0, \tag{5.11}$$

wobei wir annehmen, dass $f'(x) \neq 0$ auf $[a,b]$. Für eine Nullstelle $\overline{x}$ von f gilt offenbar $g(\overline{x}) = \overline{x}$, d.h. wir suchen einen Fixpunkt von g, den wir wieder durch sukzessive Substi-

tution annähern können. Man bricht das Verfahren wie auch die sukzessive Substitution ab, wenn $|x_n - x_{n-1}|$ bzw. $|f(x_n)|$ hinreichend klein sind.

Ist nun f sogar zweifach stetig differenzierbar, so gilt

$$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2} \implies g'(\overline{x}) = 0. \tag{5.12}$$

Das Newtonverfahren konvergiert also zumindest lokal gegen einen Fixpunkt $\overline{x}$ von g beziehungsweise eine Nullstelle von f. Tatsächlich konvergiert das Verfahren wenigstens quadratisch, wenn f zweifach differenzierbar ist, da

$$\begin{aligned} x_{n+1} - \overline{x} = \frac{(x_n - \overline{x})f'(x_n) - f(x_n)}{f'(x_n)} &= \frac{(x_n - \overline{x})}{f'(x_n)} \left(f'(x_n) - \frac{f(x_n) - f(\overline{x})}{x_n - \overline{x}} \right) \\ &= \frac{(x_n - \overline{x})}{f'(x_n)} \left(f'(x_n) - f'(\xi') \right) = \frac{(x_n - \overline{x})^2}{f'(x_n)} f''(\xi) \end{aligned} \tag{5.13}$$

und somit

$$\frac{|x_{n+1} - \overline{x}|}{|x_n - \overline{x}|^2} \leq \frac{\max_{\xi \in [a,b]} |f''(\xi)|}{\max_{\xi \in [a,b]} |f'(\xi)|} \tag{5.14}$$

Ist f nur differenzierbar, so lässt sich ähnlich zeigen, dass das Newtonverfahren superlinear konvergiert. Das Newtonverfahren konvergiert also in jedem Fall schneller als die sukzessive Substitution, erfordert allerdings eine mindestens stetig differenzierbare Funktion.

Bis jetzt haben wir nur die lokale Konvergenz des Newton-Verfahrens. Ist die Zielfunktion $f \in C^1([a,b])$ allerdings konvex bzw. konkav, also f' monoton wachsend bzw. fallend, und hat eine Nullstelle, so kann man zeigen, dass das Newtonverfahren global gegen eine Nullstelle $\overline{x}$ von f konvergiert. Dabei ist das Verfahren nach dem ersten Schritt monoton, d.h. entweder $x_1 \leq x_2 \leq \ldots \leq \overline{x}$ oder $x_1 \geq x_2 \geq \ldots \geq \overline{x}$.

5.2.1 Beispiel

Wir betrachten wieder die Aufgabe $e^{-r}/r = \phi_0$, bzw. $f(r) = e^{-r}/r - \phi_0 = 0$. Die Ableitung dieser Funktion ist $-e^{-r}(1+r)/r^2$, f fällt also monoton. Daher konvergiert das Newtonverfahren global und monoton, wie in Abbildung 5.2 zu sehen. Im linken Graphen ist $r_0 < \overline{r}$, daher startet das Verfahren sofort monoton. Im rechten Graphen ist $r_0 < \overline{r}$. Hier wird im ersten Schritt $r_1 < \overline{r}$, und erst dann wächst die Näherung wieder monoton. In jedem Fall konvergiert das Newtonverfahren, anders als die sukzessive Substitution, für beide Werte von ϕ_0 innerhalb weniger Schritte zuverlässig gegen die Nullstelle.

5.2.2 Wurzelziehen

Wir betrachten die Gleichung $f(x) = x^k - a = 0$ auf der positiven Halbachse. Dann konvergiert für jeden Startwert $x_0 > 0$ das Newtonverfahren

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = \left(1 - \frac{1}{k}\right) x_n + \frac{1}{k} \frac{a}{x_n^{k-1}} \tag{5.15}$$

gegen die einzige Nullstelle, nämlich die k-te Wurzel aus a. Sinnvollerweise wählt man daher $x_0 = a$ als Startwert. Für $k = 2$ ergibt sich das *Heron-Verfahren* $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$, das bereits im 2. Jhdt. vor Christus zum Wurzelziehen benutzt wurde.

Für die Wurzel aus $a = 2$ sind die ersten 5 Schritte des Heronverfahrens:

Schritt n	x_n	Anzahl korrekter Stellen
0	1.000000000000000	1
1	1.500000000000000	1
2	1.416666666666667	2
3	1.414215686274510	5
4	1.414213562374690	11
5	1.414213562373095	15

Mit der auf Rechnern üblichen doppelten Genauigkeit ist das Verfahren damit konvergiert.

Die Anzahl der Rechenoperationen für n Schritte entspricht der Auswertung eines Polynoms mit $3n/2$ Koeffizienten. Wäre man zum Beispiel nur an der Wurzel im Bereich [0,5] interessiert und würde hierzu eine interpolierendes Polynom mit 7 Chebyshev-Stützstellen nutzen, wäre $\sqrt{2} \approx 1.40966$ mit gerade einmal einerq korrekten Stelle. Mit einer Taylorentwicklung um 1 würde es etwas besser. Bei 7 Termen liefert diese $\sqrt{2} \approx 1.4214$ mit 2 korrekten Stellen.

Für $k = -1$ wird aus der Wurzelaufgabe eine Division, da wir die Nullstelle der Funktion $f(x) = \frac{1}{x} - a$ suchen. Die Lösung kann nur mit Hilfe der Grundrechenarten durch die Iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = 2x_n - ax_n^2 \tag{5.16}$$

bestimmt werden. Allerdings ist die Ableitung von a/x unbeschränkt, daher konvergiert das Verfahren nur für Startwerte, die hinreichend nah an der Lösung sind. Wie man sich in diesem Fall leicht überlegt, konvergiert das Verfahren nur für $x_o \in (0, 2/a)$, was schwierig zu erfüllen ist, ohne a^{-1} bereits zu kennen.

5.2.3 Nullstellen von Polynomen

Ist p ein Polynom, so lassen sich dessen Nullstellen (approximativ) mit Hilfe des Newtonverfahrens bestimmen:

$$x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n)}, \tag{5.17}$$

wobei $p(x_n)$ und $p'(x_n)$ durch ein modifiziertes Hornerschema bestimmt werden können:

```
double newton_step(double *series, int n, double xn)
{
  double p = c[n];
  double dp = n*c[n];
  for(int i = n-1; i >= 0; --i) {
    p  =  p*xn +   c[i];
    dp = dp*xn + i*c[i];
```

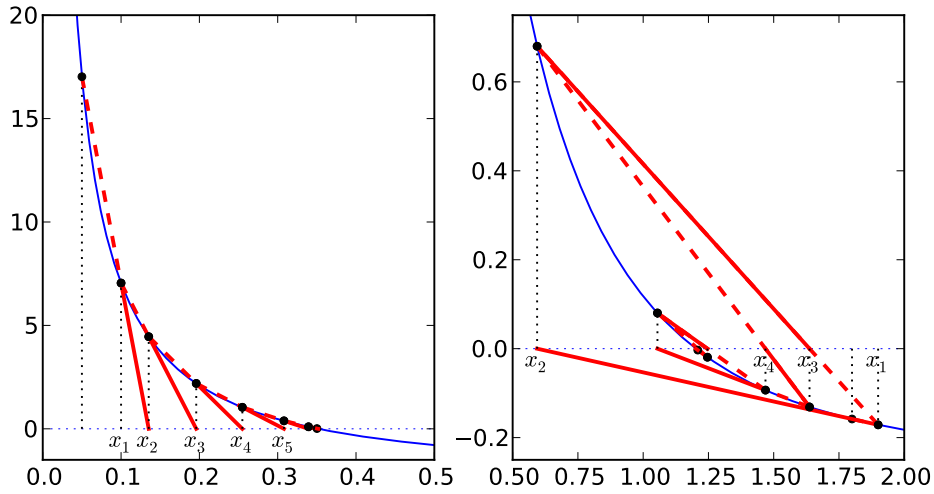


Abbildung 5.3: Regula falsi für die Funktion $f(r) = e^{-r}/r - \phi_0$. Wie zuvor ist links $\phi_0 = 2$, rechts $\phi_0 = 1/4$. Blau dargestellt ist wieder f, die roten, gestrichelten dicken Linien stellen die Sekanten dar, deren Nullstellen die neuen Näherungen für die gesuchte Nullstelle von f sind. Durchgezogen ist der Abschnitt der Sekante durch x_{n-1} und x_n, der von x_n zu x_{n+1} führt.

```
  }
  return xn - p/dp;
}
```

Das Newtonverfahren liefert natürlich nur eine Nullstelle des Polynoms. Durch die Polynomdivision, wieder mit Hilfe des Hornerschemas wie in Abschnitt 3.1, lässt sich diese aber abspalten und das Newtonverfahren erneut starten, bis alle Nullstellen gefunden sind.

5.3 Regula falsi

Regula falsi

In vielen Fällen ist es nicht einfach oder unmöglich, die Ableitung einer Funktion zu bestimmen. In diesem Fall kann man die Ableitung durch die dividierte Differenz annähern, wobei nun zwei Startpunkte x_0 und x_1 gebraucht werden. Daraus ergibt sich die *Regula falsi*

$$x_{n+1} = x_n - f(x_n)\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}, \tag{5.18}$$

die nicht mehr quadratisch, aber wenigstens superlinear konvergiert.

Das Bestimmen der Ableitung von f ist immer dann unmöglich, wenn f sehr komplex ist. Ein Extremfall wäre eine Molekulardynamik-Computersimulation, bei der zum

Beispiel in Abhängigkeit vom aktuellen Volumen V der mittleren Druck $P(V)$ in einem gegebenen System bestimmt wird. Den mittleren Druch nach V abzuleiten, ist vollkommen aussichtslos, wenn die Wechselwirkungen zwischen den Teilchen hinreichend komplex sind. Dabei ist es sehr wohl von großem Interesse, dasjenige Volumen zu bestimmen, für das $P(V)$ gleich einem vorgegebenen Außendruck P_0 ist, denn dies ist die natürliche Bedingung im isobarischen Ensemble. In diesem Fall ist zusätzlich noch die Funktion $P(V)$ mit einer oftmals nicht kleinen statistischen Unsicherheit belegt, weil die Druckschwankungen, also die Varianz des Drucks, sehr groß sind. Trotzdem konvergiert die Regula falsi im allgemeinen zufriedenstellend, solange keine zu hohen Ansprüche an die Genauigkeit gestellt werden. Schliesslich kann die Nullstelle nicht genauer als die vorhandenen Daten bestimmt werden.

Als Beispiel für die Regula falsi soll ein weiteres Mal die Funktion $f(r) = e^{-r}/r - \phi_0$ dienen. Für diese zeigt Abbildung 5.3 die erste paar Schritte der Regula falsi. Unter geeigneten Umständen, nämlich, wenn die angenäherte Tangente hinreichend gut mit der tatsächlichen über einstimmt, konvergiert die Regula falsi praktisch genauso gut wie das normale Newtonverfahren. Sind die Startpunkte allerdings ungünstig gewählt, wie im rechten Beispiel, so kann es passieren, dass diese abwechselnd um die Nullstelle liegen, und damit nicht mehr monoton konvergieren.

5.4 Bisektion

Wie wir gesehen haben, konvergiert das Newtonverfahren und seine Varianten sehr schnell, allerdings oft nur unter der Voraussetzung, dass der Startwert hinreichend nah an einer Nullstelle liegt. Wie aber kann man einen solchen Startwert finden? Hierfür wird ein langsameres, robusteres Verfahren gebraucht, zum Beispiel das *Bisektionsverfahren* in einer Dimension.

Sei also wieder $f \in C([a,b])$ eine stetige Funktion, und $f(a)f(b) < 0$. Dann hat f gemäß Mittelwertsatz wenigstens eine Nullstelle im Interval $[a,b]$, die wir suchen. Dazu setzen wir zunächst $a_0 = a$ und $b_0 = b$. Dann betrachten wir den Intervallmittelpunkt

$$m_n = \frac{a_n + b_n}{2}. \tag{5.19}$$

Ist $f(m_n)f(a_n) < 0$, haben also unterschiedliche Vorzeichen, so muss eine Nullstelle im halb so großen Interval $a_1 = a_0$, $b_1 = m_0$ liegen, mit dem wir nun weiter verfahren. Anderfalls ist notwendigerweise $f(m_n)f(b_n) \leq 0$, und die Nullstelle ist im neuen Interval $a_1 = m_0$, $b_1 = b_0$. Ist $b_n - a_n$ kleiner als die gewünschte Genauigkeit für die Nullstelle, bricht man einfach ab. m_n ist dann die endgültige Näherung für die Nullstelle.

Das Verfahren halbiert in jedem Schritt die Intervallgröße und damit auch den maximalen Abstand der Näherung zur tatsächlichen Nullstelle $\overline{x}$. Es gilt als

$$\frac{|m_{n+1} - \overline{x}|}{|m_n - \overline{x}|} \leq \frac{1}{2}, \tag{5.20}$$

das Verfahren konvergiert also nur linear, dafür aber global.

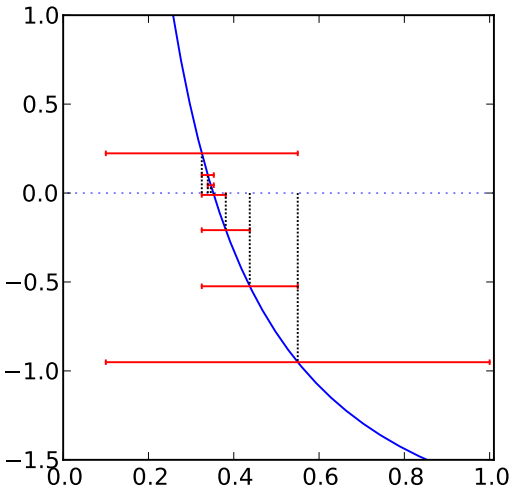


Abbildung 5.4: Bisektion für die Funktion $f(r) = e^{-r}/r - \phi_0$, hier nur mit $\phi_0 = 2$, da die genau Form der Funktion für diese Methode unerheblich ist. Blau dargestellt ist wieder f, die roten Balken markieren die nacheinander generierten, kleiner werdenden Intervalle $[a_n, b_n]$, die der Sichtbarkeit wegen auf Höhe von $f(m_n)$ eingezeichnet sind. Die gestrichelten, schwarzen Linien markieren noch einmal ausdrücklichen die Intervalmitten m_n, die im nächsten Schritt eine der beiden Intervalgrenzen werden.

In der Praxis kombiniert man die Bisektion mit dem Newtonverfahren, in dem zunächst einige Schritte des Bisektionsverfahrens gestartet werden, und dann vom Intervallmittelpunkt aus das Newtonverfahren. Konvergiert dieses, so ist man fertig. Läuft das Newtonverfahren hingegen aus dem Bisektionsinterval heraus, verkleinert man dieses weiter durch Bisektion, und versucht dann erneut, die Nullstelle mit dem Newtonverfahren zu bestimmen.

In Abbildung 5.4 dient uns ein letztes Mal die Funktion $f(x) = e^{-r}/r - 2$ als Beispiel für die Bisektion. Mit sieben Schritten schließt diese bei Startinterval $[0{,}1, 1]$ die Nullstelle bis auf $[0{,}3391, 0{,}3461]$, als etwa 10^{-2} genau ein. Zum Vergleich: das Newtonverfahren mit Startwert 0,05 erreicht in sieben Schritten eine Genauigkeit von etwa 10^{-5}, und selbst die Regula falsi 10^{-3}.

5.5 Newtonverfahren in mehreren Dimensionen

Bis jetzt haben wir das Newtonverfahren nur für eindimensionale Funktionen betrachtet. Im mehrdimensionalen funktioniert das Verfahren aber sehr ähnlich, wobei die Ableitung zur Jacobimatrix wird.

Sei also $f \in C^1(M, \mathbb{R}^n)$ eine stetig differenzierbare Abbildung von $M \in \mathbb{R}^n$ in den $\mathbb{R}^n$. Wir suchen nun eine Nullstelle $\overline{x} \in D$, d.h. eine Lösung des nichtlinearen Gleichungssys-

tems $f(x) = 0$.

Wie schon im Eindimensionalen starten wir mit einer Näherung $x^{(0)} \in M$, und berechnen die nächste Näherung $x^{(1)}$ durch Linearisieren von f in $x^{(0)}$. Die Linearisierung ergibt sich aus der Taylorentwicklung:

$$F(x^{(1)}) \doteq F(x^{(0)}) + F'(x^{(0)}) \left(x^{(1)} - x^{(0)}\right), \tag{5.21}$$

wobei

$$F'(x) = \left(\frac{d}{dx_j} F_k(x)\right)_{k,j} = \begin{pmatrix} \frac{d}{dx_1} F_1(x) & \dots & \frac{d}{dx_n} F_1(x) \\ \vdots & & \vdots \\ \frac{d}{dx_1} F_n(x) & \dots & \frac{d}{dx_n} F_n(x) \end{pmatrix} \tag{5.22}$$

die *Jacobimatrix* von f an der Stelle x bezeichnet. Die neue Näherung $x^{(1)}$ suchen wir als Nullstelle der Linearisierung (5.21), also aus der Bedingung $F(x^{(1)}) \stackrel{!}{=} 0$. Da wir ja damit nur die linearisierte Gleichung gelöst haben, linearisieren wir erneut im neuen Punkt $x^{(1)}$, und so weiter. Ein Schritt des Newtonverfahrens ist dann also

$$x^{(m+1)} = x^{(m)} + d^{(m)} \quad \text{mit } F'(x^{(m)})\, d^{(m)} = -F(x^{(m)}). \tag{5.23}$$

Die *Newtonkorrektur* $d^{(m)}$ wird als aus der Lösung eines linearen Gleichungssystems gewonnen, zum Beispiel mit Hilfe der Gaußelimination. Allerdings ist f' im allgemeinen vollbesetzt, daher verwendetet man normalerweise schnellere, approximative Verfahren, die wir später kennenlernen werden. Ist $F'(x^{(m)})$ in einem Schritt singulär, so bricht das Verfahren ab. Ansonsten wird weiter iteriert, bis $\left\|d^{(m)}\right\|$ hinreichend klein ist.

Auch im mehrdimensionalen konvergiert dieses Verfahren lokal mindestens quadratisch, sofern in einer abgeschlossenen Umgebung der Nullstelle $\left\|F'(x)^{-1}\right\|$ beschränkt ist und f zweifach stetig differenzierbar. Allerdings gibt es kein langsames Verfahren ähnlich der Bisektion, dass man dem Vefahren vorausschicken könnte, um die Nullstelle einzugrenzen. Die globale Suche nach Nullstellen in mehreren Dimensionen ist also eine schwierige Aufgabe. Eine Möglichkeit ist, zunächst mit Hilfe von Optimierungsverfahren ein x_0 zu finden mit möglichst kleiner Norm $\|f(x_0)\|$, und von dort das (gedämpfte) Newtonverfahren zu starten. Die globale Optimierung in vielen Dimensionen ist selber eine sehr schwierige Aufgabe, allerdings existieren hierfür Ansätze wie genetische Algorithmen oder Simulated Annealing, die wir später kennenlernen werden.

5.5.1 Gedämpftes Newtonverfahren

Leider ist im Mehrdimensionalen die Umgebung um die Nullstelle, in der das Verfahren konvergiert, oftmals deutlich kleiner. Das Verfahren springt dann leicht über die Nullstelle hinweg, wie es im eindimensionalen nur am Anfang des Verfahrens vorkommt (z.B. Abbildung 5.2 rechts, im ersten Schritt). Um dies zu verhindern, kann man die Schrittweite reduzieren, als den Schritt $d^{(m)}$ verkürzen. Die Iteration lautet dann

$$x^{(m+1)} = x^{(m)} + \lambda d^{(m)} \quad \text{mit } F'(x^{(m)})\, d^{(m)} = -F(x^{(m)}), \tag{5.24}$$

```
# Gedaempftes Newtonverfahren
#############################

def gedaempfter_newton(f, fprime, x0, epsilon):
    xn = x0
    konvergiert = False
    while not konvergiert:
        # Newton-Korrektur
        dn = solve(fprime(xn), f(xn))
        if norm(dn) < epsilon:
            konvergiert = True
        else:
            # Schrittweitendaempfung
            lambda = 1.0
            abstieg = False
            while not abstieg:
                # neue Naeherung
                xneu = xn + lambda*dn
                if norm(f(xneu)) < norm(f(xn)):
                    abstieg = True
                else:
                    lambda = lambda / 2.0
            xn = xneu
    return xn
```

Listing 5.1: Gedämpftes Newtonverfahren in mehreren Dimensionen. `f(x)` muß eine vektorwertige Funktion sein, `fprime(x)` ihre Ableitung, d.h. eine matrixwertige Funktion.

wobei die Dämpfung $\lambda \in (0,1]$ so gewählt wird, dass $\left\|F(x^{m+1})\right\| \leq \|F(x^m)\|$. Dazu wird zum Bespiel mit $\lambda = 1$ begonnen, und λ solange verringert, bis die Bedingung erreicht ist.

Literatur

[AS70] M. Abramowitz und I. Stegun. *Handbook of mathematical functions*. New York: Dover Publications Inc., 1970.

[Dau92] Ingrid Daubechies. *Ten lectures on wavelets*. Bd. 61. Society for Industrial Mathematics, 1992.

[Pin02] Mark Pinsky. *Introduction to Fourier Analysis and Wavelets*. Brooks/Cole, 2002.

Index