TUM NumProg, WiSe 2022/2023

Mitschriften basierend auf der Vorlesung von Dr. Hans-Joachim Bungartz

Zuletzt aktualisiert: 27. November 2022

Introduction

About

Hier sind die wichtigsten Konzepte der NumProg Vorlesung von Dr. Hans-Joachim Bungartz im Wintersemester 2022/2023 zusammengefasst.

Die Mitschriften selbst sind in Markdown geschrieben und werden mithilfe einer GitHub-Action nach jedem Push mithilfe von Pandoc zu einem PDF konvertiert.

Eine stets aktuelle Version der PDFs kann über https://github.com/ManuelLerchner/numprog/releases/download/Release/merge.pdf heruntergeladen werden.

How to Contribute

1. Fork this Repository
2. Commit and push your changes to **your** forked repository
3. Open a Pull Request to this repository
4. Wait until the changes are merged

Contributors

Inhaltsverzeichnis

Floating-Point

Fixed-Point

Representation

Bei Fixed-Point wird die Zahl in eine ganze Zahl und eine Bruchzahl aufgeteilt. Diese werden jeweils "normal" kodiert.

- Nachteile:
  - Schneller Overflow, da nur kleine Zahlen dargestellt werden können
  - Konstanter Abstand zwischen zwei Zahlen. Oft nicht benötigt.

Floating-Point

Representation

Eine Floating-Point Zahl wird in Mantisse und Exponent aufgeteilt. Zusammen mit einem Vorzeichenbit, lässt sich so ein sehr großer Wertebereich darstellen.

- Definition normalisierte, t-stellige Float-Zahl
- $\mathbb{F}_{B,t} = \{M \cdot B^E \mid M, E \in \mathbb{Z} \land \ M \text{ ohne führende Nullen bzw: } B^{t-1} \leq M < B^t\}$
- $\mathbb{F}_{B,t,\alpha,\beta} = \{M \cdot B^E \mid M, E \in \mathbb{Z} \ \& \quad M \text{ ohne führende Nullen} \land \alpha \leq E < \beta\}$
- Wobei gilt:
- B Basis
  - t Anzahl der signifikanten Stellen
  - α kleinster möglicher Exponent
  - β größter möglicher Exponent
- Vorteile:
  - Großer Wertebereich, da variable Abstände zwischen zwei Zahlen

Kleinste bzw. größte Zahl

In einem solchen System ist:

- $\sigma = B^{t-1} \cdot B^{\alpha}$ die kleinste positive Zahl, die dargestellt werden kann.
- $\lambda = (B^t - 1) \cdot B^{\beta}$ die größte positive Zahl, die dargestellt werden kann.

Beispiel:

- Mit $B = 10$ und $t = 4$ und $\alpha = -2$ und $\beta = 1$ ergibt sich:
  - $\sigma = 10^{4-1} \cdot 10^{-2} = 10$
  - $\lambda = (10^4 - 1) \cdot 10^1 = 99990$

Maximalen relativen Abstand zweier Float-Zahlen

Die Auflösung einer Float-Zahl ist der maximale relative Abstand zu einer anderen Float-Zahl. Sie berechnet sich wie folgt:

- $\varrho = \frac{1}{M} \leq B^{1-t}$

Beispiel:

- Mit einer Basis von $B = 2$ und $t = 4$ Stellen ergibt sich; $\varrho \leq 2^{-3} = 0.125$. Damit ist der maximale relative Abstand zwischen zwei Float-Zahlen 0.125.

Rundung

Da nur eine endliche Anzahl von Float-Zahlen existieren, muss grundsätzlich nach jeder Operation gerundet werden.

Diese Rundungsfunktion wird als `rd(x)` bezeichnet: Es gilt:

- $rd : \mathbb{R} \to \mathbb{F}$
- surjektiv: $\forall f \in \mathbb{F}\ \exists x \in \mathbb{R} \Rightarrow rd(x) = f$
- idempotent: $rd(rd(x)) = rd(x)$
- monoton: $x \leq y \Rightarrow rd(x) \leq rd(y)$

Rundungsmodi

1. Abrunden:
   - $rd_-(x) = f_l(x)$ wobei $f_l(x)$ die nächstkleinere Float-Zahl ist.
2. Aufrunden:
   - $rd_+(x) = f_r(x)$ wobei $f_r(x)$ die nächstgrößere Float-Zahl ist.
3. Abschneiden (Runden in Richtung 0):
   - $rd_0(x) = f_-(x)$ wenn $x \geq 0$ und $f_+(x)$ wenn $x \leq 0$.
4. Korrektes Runden:
   - Rundet immer zur nächstgelegenen Float-Zahl.
   - Falls die nächstgelegene Zahl gleich weit entfernt ist, wird die Zahl mit gerader Mantisse gewählt.

Rundungsfehler

Durch jeden Rundungsschritt entsteht zwangsläufig ein Rundungsfehler.

- Absolute Rundungsfehler:
  - $rd(x) - x$
- Relative Rundungsfehler:
  - $\epsilon = \frac{rd(x)-x}{x}$
  - Dieser Rundungsfehler kann bei direktem Runden mit: $|\epsilon| \leq \varrho$ abgeschätzt werden.
  - Bei korrektem Runden gilt: $|\epsilon| \leq \frac{1}{2} \cdot \varrho$

Durch diese Konstruktion des relativen Rundungsfehlers, gilt:

- $rd(x) = x * (1 + \epsilon)$

Um die Operationen, welche Rundungsfehler verursachen, von den "sauberen" Operationen zu unterscheiden, wird eine neue Notation eingeführt:

- $a * b$ bezeichnet den Wert der Multiplikation ohne Rundung
- $a \dot{*} b$ bzw. $rd(a * b)$ bezeichnet den Wert nach der Rundung

Es gibt zwei Möglichkeiten, die entstehenden Rundungsfehler zu modellieren:

1. Als Funktion des exakten Ergebnisses: (Starke Hypothese)

- $a \dot{*} b = f(a * b) = (a * b) \cdot (1 + \epsilon)$
  - Diese Variante wird von fast allen Systemen unterstützt.

2. Als Funktion der Rundungsfehler der Operanden: (Schwache Hypothese)
   - $a \dot{*} b = f(a, b) = (a \cdot (1 + \epsilon_1)) * (b \cdot (1 + \epsilon_2))$

Wobei alle ϵ-Werte betragsmäßig durch die Maschinengenauigkeit $\bar{\epsilon}$ begrenzt sind. Diese entspricht je nach verwendetem Rundungsmodus entweder ϱ oder $\frac{1}{2} \cdot \varrho$.

Achtung:

Die gerundeten Varianten der Operatoren sind nicht mehr assoziativ!

- $(a \dot{*} b) \dot{*} c \neq a \dot{*} (b \dot{*} c)$

Außerdem findet Absorption statt. Das bedeutet, dass z.B. bei der Subtraktion von ähnlich großen Zahlen, die Anzahl der signifikanten Stellen deutlich abnimmt. Und dadurch ein extrem hoher Rundungsfehler entsteht.

Fehleranalyse

Es gibt die Möglichkeit der Vorwärts- und Rückwärtsfehleranalyse.

Vorwärts Fehleranalyse

Hierbei wird das Ergebnis als Funktion des exakten Ergebnisses modelliert.

- $a \dot{+} b = (a + b) \cdot (1 + \epsilon)$
- $a \dot{*} b = (a * b) \cdot (1 + \epsilon)$

Diese Modellierung ist einfach, jedoch in der Praxis nur schwer berechenbar, da die Fehler korreliert sind.

Rückwärts Fehleranalyse

Hierbei wird das Ergebnis als Funktion der Rundungsfehler der Operanden modelliert.

- $a \dot{+} b = (a \cdot (1 + \epsilon)) + (b \cdot (1 + \epsilon))$
- $a \dot{*} b = (a \cdot \sqrt{1 + \epsilon}) * (b \cdot \sqrt{1 + \epsilon})$

Kondition

Die Kondition eines Problems ist ein Maß für die Sensitivität des Problems gegenüber Änderungen der Eingabedaten. Diese ist unabhänging vom verwendeten Algorithmus.

Ist ein Problem gut konditioniert, so ist es sehr stabil gegenüber kleinen Änderungen der Eingabedaten. Bei solchen Problemen lohnt sich die Verwendung eines guten Algorithmus.

Ist ein Problem schlecht konditioniert, so ist es sehr empfindlich gegenüber kleinen Änderungen der Eingabedaten. Somit hat sogar der bestmögliche Algorithmus keinen signifikanten Einfluss auf die Genauigkeit des Ergebnisses. Da dieses ohnehin durch die Fehlerfortpflanzung dominiert wird.

Man betrachtet wiederum den absoluten und den relativen Fehler:

- $err_{abs} = f(x + \delta x) - f(x)$
- $err_{rel} = \frac{f(x+\delta x)-f(x)}{f(x)}$

Die Konditionszahl wird nun wie folgt definiert:

- $kond_{abs} = \frac{err_{abs}}{\delta x}$
- $kond_{rel} = \frac{err_{rel}}{\frac{\delta x}{x}}$

Im Algemeinen ist die Konditionszahl eines Problems $p(x)$ bei der Eingabe x als:

- $kond(p(x)) = \frac{\partial p(x)}{\partial x}$

Laut dieser Definition haben alle Grundrechenarten, außer die Addition / Subtraktion, eine Konditionszahl von ca. 1 und sind somit gut konditioniert.

Die Subtraktion ist schlecht konditioniert, da sie bei ca. gleich großen Zahlen zu einem extrem hohen relativen Fehler führen kann.

Beispiele für gute und schlechte Kondition:

- Gut konditionierte Probleme:
  - Berechnung der Fläche eines Rechtecks
  - Berechnung von Schnittpunkten von fast orthogonalen Geraden
- Schlecht konditionierte Probleme:
  - Berechnung von Nullstellen von Polynomen
  - Berechnung von Schnittpunkten zweier fast paralleler Geraden

Akzeptable Ergebnisse

Ein numerisch akzeptables Ergebnis ist dann gegeben, wenn das berechnete Ergebnis, auch als exaktes Ergebnis von nur leicht gestörten Eingabedaten erklärt werden kann.

- $\tilde{y}$ ist akzeptables Ergebnis für $y = f(x)$, wenn $\tilde{y} \in \{f(\tilde{x}) \mid \tilde{x}$ nahe von x$\}$

Numerische Stabilität

Ein Algorithmus ist numerisch stabil, wenn für alle erlaubten Eingabedaten ein *akzeptables* Ergebnis berechnet wird.

- Die Grundrechenarten sind numerisch stabil (Basierend auf schwacher Hypothese)

- Die Komposition von numerisch stabilen Algorithmen ist nicht zwingend stabil

Beispiel:

- Numerisch instabil:
- Berechnung der Wurzel als: $x = \sqrt{\left(\frac{p}{2}\right)^2 - q} - \frac{p}{2}$
- Numerisch stabil:
  - Berechnung der Wurzel als: $x = \frac{-q}{\sqrt{\left(\frac{p}{2}\right)^2 - q} + \frac{q}{2}}$

Polynom-Interpolation

Problem

Für eine gegebene Funktion $f(x)$, suchen wir eine Funktion $p(x)$ welche einfach zu konstruieren und für weitere Anwendungen nutzbar ist. $p(x)$ soll dabei $f(x)$ annähern und einen *geringen* Fehler aufweisen.

Der resultierende Fehler ist ein Maß für die Qualität der Approximation. Dieser wird als *Fehlerterm* bzw. *Remainder* bezeichnet.

$$f(x) - p(x) = \frac{D^{(n+1)} f(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

Wobei ξ ein Punkt zwischen x_0 und x_n ist.

Lagrange Polynomials

Bei der Lagrange Interpolation werden Basisfunktionen verwendet, welche jeweils an allen Stützstellen, bis auf der Stelle x_i, den Wert 0 annehmen.

$$L_k(x) = \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

Das Resultierende Polynom ergibt sich dann als:

$$p(x) = \sum_{i=0}^{n} y_i \cdot L_i(x)$$

Chebyshev Polynomials

Die Basisfunktionen für die Interpolation sind die Chebyshev Polynome. Diese sind definiert als:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{k+1}(x) &= 2x \cdot T_k(x) - T_{k-1}(x) \end{aligned}$$

Damit lassen sich die dazu gehörigen Koeffizienten berechnen:

$$\begin{aligned} a_0 &= \frac{1}{2\pi} \int_{-1}^{1} \frac{f(x)}{\sqrt{1 - x^2}} dx \\ a_k &= \frac{2}{\pi} \int_{-1}^{1} \frac{f(x) \cdot T_k(x)}{\sqrt{1 - x^2}} dx \end{aligned}$$

Die Interpolation ist dann gegeben durch:

$$p(x) = \sum_{k=0}^{n} a_k T_k(x)$$

Bernstein Polynomials

Mithilfe der Bernstein Polynome lässt sich die Interpolation durch Bezier Kurven realisieren. Diese sind definiert als:

$$B_i^n = \binom{n}{i}(1-t)^{n-i} \cdot t^i$$

Die Interpolation ist dann gegeben durch:

$$p(t) = \sum_{i=0}^{n} b_i B_i^n(t)$$

Hierbei stellen b_i die Kontrollpunkte dar. (Diese können auch höherdimensional sein)

Variationen der Problemstellung

Es gibt zwei verschidenen Anwendungen der Interpolation welche auftreten können:

1. Simple Nodes
   - Man hat eine Menge von Punkten $P = \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, welche durch ein Polynom interpoliert werden sollen.
   - Diese Variante wird auch als *Lagrange Interpolation* bezeichnet.
2. Multiple Nodes
   - Man hat heine Menge von Knoten $P = \{(x_0, y_0, y_0'), (x_1, y_1, y_1'), \ldots, (x_n, y_n, y_n')\}$ welche durch ein Polynom interpoliert werden sollen.
   - Hierbei ist y_i' die Ableitung von y_i.
   - Diese Variante wird als *Hermit Interpolation* bezeichnet.

Algorithmen

Schema von Aitken und Neville

Wenn man nicht an der expliziten Representation des Interpolationspolynoms interessiert ist, sondern nur den Funktionswert an einem festen x Wert bestimmen möchte eignet sich das Schema von Aitken und Neville.

Algorithmus:

1. Initialisiere konstante Polynome welche den Funktionswert an den Stützstellen annehmen.
   - $p_{0,0} = y_0, p_{1,0} = y_1, \ldots, p_{n,0} = y_n$
2. Verfeinere rekursiv die Polynome durch die Kombibation mehrerer Polynome.
   - $p_{i,j} = \frac{x_{i+k} - x}{x_{i+k} - x_i} \cdot p_{i,k-1}(x) + \frac{x - x_i}{x_{i+k} - x_i} \cdot p_{i+1,k-1}(x)$

Zur Berechnung mit Hand kann folgendes Schema herangezogen werden:

Als Pseudo-Code:

Diese Form der Interpolation eignet sich nur, wenn nur relative wenige Werte ausgewertet werden müssen. Ansonsten lohnt sich die Bestimmung des expliziten Polynoms.

x_i	$i \setminus k$	0		1		2		$\cdots$
x_0	0	$p[0,0] = y_0$	$\nearrow$	$p[0,1]$	$\nearrow$	$p[0,2]$	$\rightarrow$	$\cdots$
x_1	1	$p[1,0] = y_1$	$\nearrow$	$p[1,1]$	$\rightarrow$	$\vdots$		
x_2	2	$p[2,0] = y_2$	$\rightarrow$	$\vdots$				
$\vdots$	$\vdots$	$\vdots$						

Abbildung 1: Dreiecks-Schema für Aitken-Neville

```
for i=0:n; p[i,0]:=f_x[i]; end
for k=1:n
    for i=0:n-k
        p[i,k] := p[i,k-1] + (x-x[i])/(x[i+k]-x[i])*(p[i+1,k-1] - p[i,k-1]);
    end
end
```

Abbildung 2: aitken_neville_code

Newton Interpolation

Die Newton Interpolation ist eine spezielle Form der Lagrange Interpolation. Hierbei werden die Koeffizienten des Polynoms durch die Differenzenquotienten der Stützstellen bestimmt.

Algorithmus:

1. Initialisiere die Differenzenquotienten
   - $[x_i]f = f(x_i) = y_i$
2. Rekursiv berechne die Differenzenquotienten
   - $[x_i, x_{i+1}, \ldots, x_{i+k}]f = \frac{[x_{i+1},\ldots,x_{i+k}]f - [x_i,\ldots,x_{i+k-1}]f}{x_{i+k} - x_i}$

Die Interpolation ist dann gegeben durch:

$$p(x) = \sum_{i=0}^{n} [x_0, x_1, \ldots, x_i]f \cdot \prod_{j=0}^{i-1} (x - x_j)$$

Diese Methode eignet sich gut, um die explizite Form des Polynoms zu erhalten. Außerdem ist es leicht möglich, weitere Stützstellen hinzuzufügen.

Der entstehende Fehler ist hierbei $\mathcal{O}(h^{n+1})$. Wobei h die Distanz zwischen den Stützstellen ist.

Auch für die Newton Interpolation gibt es ein Schema für die händische Berechnung:

x_i	$i \setminus k$	0		1		2		$\cdots$
x_0	0	$c_{0,0} = y_0$	$\nearrow$	$c_{0,1}$	$\nearrow$	$c_{0,2}$	$\rightarrow$	$\cdots$
x_1	1	$c_{1,0} = y_1$	$\nearrow$	$c_{1,1}$	$\rightarrow$	$\vdots$		
x_2	2	$c_{2,0} = y_2$	$\rightarrow$	$\vdots$				
$\vdots$	$\vdots$	$\vdots$						

Abbildung 3: newton_interpolation_schema

Dementsprechend sind auch die Variablen in den Formeln anders benannt:

$$c_{i,0} = f(x_i) = y_i$$

$$c_{i,k} = \frac{c_{i+1,k-1} - c_{i,k-1}}{x_{i+k} - x_i}.$$

$$p(x) = c_{0,0} + c_{0,1} \cdot (x - x_0) + \ldots + c_{0,n} \cdot \prod_{i=0}^{n-1} (x - x_i) \ .$$

Abbildung 4: newton_interpolation_formeln

Kondition von Interpolationspolynomen

Die Kondition der Polynomialen Interpolation ist besonders bei einer großen Anzahl von Stützstellen ($n > 7$) ein Problem. Da das entstehende Polynom besonders an den Randstellen extrem oszillieren kann.

Polynom - Splines

Definition

Anstatt alle Punkte durch ein gemeinsames Polynom zu interpolieren, wird der Bereich in mehrere Intervalle unterteilt und für jedes Intervall ein eigenes Polynom erstellt, welches dann an den Intervallgrenzen mit den anderen Polynomen "zusammengeklebt" wird.

Ein Spline $s(x)$ von der Ordnung m bzw. mit Grad $m-1$ ist eine Kette von Polynomen mit Grad $m-1$, welche jeweils zwischen zwei Stützstellen die Funktion interpolieren. Außerdem ist $s(x)$ auf dem gesamten Intervall jeweils $m-2$ mal stetig differenzierbar ist.

Beispiel:

- $m = 1 \rightarrow$ Stückweise konstante Funktion, Treppenfunktion
- $m = 2 \rightarrow$ Stückweise lineare Funktion, stetig
- $m = 3 \rightarrow$ Stückweise quadratische Funktion, stetig und einmal stetig differenzierbar

Kubische Splines

Für den Fall $m = 4$ erhält man kubische Splines. Diese eignen sich gut für die Interpolation von Datenpunkten, da sie einfach zu berechnen sind und eine gute Approximation liefern.

Durch geeignete Herleitung, erhält man für jedes Teilintervall folgende Basisfunktionen:

$$\begin{aligned}\alpha_1(t) &= 1 - 3t^2 + 2t^3 \\ \alpha_2(t) &= 3t^2 - 2t^3 \\ \alpha_3(t) &= t - 2t^2 + t^3 \\ \alpha_4(t) &= t^3 - t^2\end{aligned}$$

Damit erhält man für die Funktion $s(x)$ folgende Form:

$$\begin{aligned}s(x) &= p * i\left(\frac{x - x_i}{h_i}\right) := p_i(t) \\ &= y_i \cdot \alpha_1(t) + y * i + 1 \cdot \alpha * 2(t) + h_i \cdot y_1' \cdot \alpha_3(t) + h_i \cdot y * i + 1' \cdot \alpha_4(t)\end{aligned}$$

Hierbei benötigt man allerdings die Ableitung des gewünschten Polynoms an den Stützstellen. Sollten diese aber nicht bekannt sein, können diese unter der Annahme von 2-mal stetiger Differenzierbarkeit folgendermaßen ermittelt werden:

Somit müssen lediglich die Ableitungen in den Randpunkten des Interpolationsintervalls müssen angegeben werden.

Der Fehler für kubische Splines kann durch $|f(x) - s(x)| = \mathcal{O}(h^4)$ abgeschätzt werden. Dies ist wesentlich besser als bei der Interpolation durch ein einziges Polynom.

Diese Formel garantiert, dass:

$$\begin{pmatrix} 4 & 1 & & \\ 1 & 4 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{pmatrix} \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_{n-2} \\ y'_{n-1} \end{pmatrix} = \frac{3}{h} \begin{pmatrix} y_2 - y_0 - \frac{h}{3} y'_0 \\ y_3 - y_1 \\ \vdots \\ y_{n-1} - y_{n-3} \\ y_n - y_{n-2} - \frac{h}{3} y'_n \end{pmatrix}.$$

Abbildung 5: Berechnung der Ableitungen

$$\begin{aligned} s(x_i) &= y_i & \forall i \\ s(x_{i+1}) &= y_{i+1} & \forall i \\ s'(x_i) &= y'_i & \forall i \\ s'(x_{i+1}) &= y'_{i+1} & \forall i \end{aligned}$$

Trigonometrische Interpolation

Definition

Bei dieser Form von Interpolation werden die Basisfunktionen durch Sinus- und Kosinus-Funktionen ersetzt. Diese Form der Interpolation ist besonders gut geeignet, wenn die zu interpolierenden Datenpunkte periodisch sind.

Um den Rechenaufwand zu minimieren behilft man sich der komplexen Darstellung von Sinus- und Kosinus-Funktionen. $e^{i\theta} = \cos(\theta) + i\sin(\theta)$

Die verwendeten Stützstellen liegen gleichverteilt auf dem Einheitskreis. Es gilt: $z_j = e^{\frac{2\pi i}{n}j}$

Der kontinuierliche Interpolant ist gegeben durch: $z_t = e^{2\pi i t} \quad t \in [0,1]$

Das resultierende Polynom hat die Form:

$$p(t) = \sum_{k=0}^{n} c_k \cdot z^k = \sum_{k=0}^{n} c_k \cdot e^{2\pi i k t}$$

Diskrete Fourier Transformation

Es soll eine Interpolation gefunden werden die die gleichverteilten Punkte $P = [(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)]$ interpoliert. Hierbei ist $\omega = e^{\frac{2\pi i}{n}}$ die n-te Wurzel von 1.

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \bar{\omega} & \bar{\omega}^2 & \ldots & \bar{\omega}^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \bar{\omega}^{n-1} & \bar{\omega}^{2(n-1)} & \ldots & \bar{\omega}^{(n-1)^2} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Man erhält jetzt also die Werte $[(f_0, c_0), (f_1, c_1), \ldots, (f_{n-1}, c_{n-1})]$. Welche jeweils die Frequenz und die Amplitude der jeweiligen Basisfunktionen darstellen.

Damit kann das Polynom $p(t)$ berechnet werden.

Inverse Diskrete Fourier Transformation

Die Inverse Diskrete Fourier Transformation ist die Umkehrung der Diskreten Fourier Transformation. Sie berechnet die Funktionswerte y_i aus den Koeffizienten c_i.

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

Fast Fourier Transformation

Die Fast Fourier Transformation ist eine effiziente Methode zur Berechnung der Diskreten Fourier Transformation. Sie ist eine rekursive Methode, die die Berechnung der Diskreten Fourier Transformation in $\mathcal{O}(n \log n)$ durchführt.

Algorithmus:

- Divide:
  1. Teile die Daten in zwei Teile auf. Einmal die geraden und einmal die ungeraden Indizes.
  2. Wiederhole Schritt 1 für die beiden Teile, solange bis nur noch ein Wert übrig ist.
- Konquer:
  1. Kombiniere jeweils die geraden und ungeraden Werte eines Teils zu einem neuen Wert.
  2. Wende dazu den *Butterfly* Operator an.
     - $[a_j, b_j] \mapsto [a_j + \omega^j b_j, a_j - \omega^j b_j]$

Numerische Quadratur

Problem

Gegeben sei eine Funktion $f(x)$, die auf einem Intervall $[a, b]$ definiert ist. Wir wollen nun die Fläche unter der Kurve numerisch $f(x)$ berechnen. Die Integration kann auch als gewichtete Summe der Funktionswerte auf einem Gitter gesehen werden.

$$I(f) \approx Q(f) := \sum_{i=0}^{n} g_i \cdot y_i$$

Idee: Anstatt eine komplizierte Funktion zu integrieren, und diese womöglich sehr oft ausrechnen zu müssen, um die benötigten Stützpunkte zu erhalten, können wir die Funktion durch Polynome interpolieren und dann die Fläche unter dieser Polynome exakt berechnen.

$$Q(f) = \int_a^b \tilde{f}(x)dx := \int_a^b p(x)dx$$

Hierbei stellt $\tilde{f}(x) = p(x)$ die Interpolationsfunktion dar.

Sonderfall: Wenn man $p(x)$ mittels Lagrange Interpolation berechnet, kann man die Resultierenden Faktoren vorberechnen.

Kondition der numerischen Quadratur

Wenn alle gewichte g_i der Interpolation positiv sind, dann ist die die numerischen Quadratur gut konditioniert. Der Fehler des Ergebnisses ist dann proportional zum Fehler der Eingabedaten.

Sollten jedoch manche Gewichte negativ sein, dann ist die numerische Quadratur schlecht konditioniert.

Algorithmen

Rechteckregel

Die Rechteckregel ist die einfachste Form der numerischen Quadratur. Hierbei wird die Fläche unter der Kurve durch ein Rechtecke abgeschätzt. Im gesamten Intervall $[a, b]$ ergibt sich dann:

$$Q_R(f) := H \cdot f\left(\frac{a+b}{2}\right)$$

Für das Integrationsintervall der Länge H wird also der Mittelpunkt berechnet und eine Konstante Funktion durch diesen Punkt gelegt. Dieses einfache Polynom wird nun exakt integriert.

Der entstandene Fehler bei dieser Variante ist in $O(H^3 \cdot f''(\xi))$.

Trapezregel

Ansatz: Die Fläche unter der Kurve kann im Intervall $[a, b]$ auch durch ein Trapez abgeschätzt werden. Dieses Polynom wird nun exakt integriert.

$$Q_T(f) := H \cdot \frac{(f(a) + f(b))}{2}$$

Der entstandene Fehler bei dieser Variante ist immer noch in $O(H^3 \cdot f''(\xi))$.

Kepler'sche Regel

Anstatt die Funktion $f(x)$ durch ein lineares Polynom abzuschätzen verwenden wir ein quadratisches Polynom. Dieses Polynom wird nun exakt integriert.

Für das Intervall $[a, b]$ ergibt sich:

$$Q_F(f) := H \cdot \frac{\left(f(a) + 4 \cdot f\left(\frac{a+b}{2}\right) + f(b)\right)}{6}$$

Die Fehlerordnung ist hier in: $O(H^5 \cdot f^{(4)}(\xi))$.

Trapezregel mit mehreren Teilintervallen

Die Trapezregel kann auch auf mehrere Teilintervalle der Länge $h = \frac{b-a}{n}$ angewendet werden. Hierbei wird die Fläche unter der Kurve in Jedem Intervall durch ein Trapez abgeschätzt.

Auf dem gesamten Intervall $[a, b]$ ergibt sich:

$$Q_{TS}(f) := h \cdot (\frac{f_0}{2} + f_1 + f_2 + \cdots + f_{n-1} + \frac{f_n}{2})$$

Der Fehler ist hier in $O(H \cdot h^2 \cdot f''(\xi))$.

Simpson'sche Regel

Die Simpson'sche Regel ist eine Erweiterung der Trapezregel. Hierbei wird die Funktionin Jedem Intervall durch ein Parabel-Polynom abgeschätzt.

Auf dem gesamten Intervall $[a, b]$ ergibt sich:

$$Q_{SS}(f) := \frac{h}{3} \cdot (f_0 + 4 \cdot f_1 + 2 \cdot f_2 + 4 \cdot f_3 + \cdots + 2 \cdot f_{n-2} + 4 \cdot f_{n-1} + f_n)$$

Der Fehler ist hier in $O(H \cdot h^4 \cdot f^{(4)}(\xi))$.

Nicht gleichmäßige Gitter

Um mehrere Abtastpunkte am Rand des Intervalls können die Abtastpunkte auch anders gewählt werden.

$x_i = a + H \cdot \frac{1-\cos\left(\frac{i \cdot \pi}{n}\right)}{2}$