Schnelle Fourier-Transformationen bei nicht äquidistanten Daten

Diplomarbeit

vorgelegt von

Karsten Fourmont

Thema gestellt von

Prof. Dr. F. Natterer

27. Juni 1997

Zusammenfassung

Die schnelle Fourier-Transformation(FFT) nach Cooley-Tukey oder Winograd ermöglicht es, die Berechnung der Fourier-Transformation statt in $\mathcal{O}(N^2)$ in $\mathcal{O}(N \log N)$ Operationen durchzuführen, was bereits für moderate N signifikante Beschleunigungen ergibt. Diese Algorithmen beruhen jedoch auf der starken Symmetrie der Transformation, die sich aus der Äquidistanz der Stützstellen ergibt.
In diesem Papier werden Algorithmen vorgestellt, die es ermöglichen, auch bei nicht äquidistanten Daten die Fourier-Transformation in $\mathcal{O}(N \cdot \log N + N \cdot \log(1/\epsilon))$ Operationen zu berechnen; ϵ gibt hierbei die gewünschte Genauigkeit an.
Die Effizienz und Genauigkeit dieser Algorithmen wird anhand einiger numerischer Untersuchungen dargestellt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Problemstellung

Die diskrete Fourier-Transformation wird gewöhnlich in der folgenden Form angewendet:

$$\mathcal{F}(\alpha)_j = \sum_{k=0}^{N-1} \alpha_k \cdot e^{ikj \cdot 2\pi/N} \tag{1.1}$$

für $j = 1, \ldots, N$. Da die Stützstellen hierbei gleichverteilt sind, kann dies per FFT bekanntermaßen in $\mathcal{O}(N \log N)$ Operationen ausgewertet werden.
Wir wollen in dieser Arbeit den etwas allgemeineren Fall betrachten, in dem die Stützstellen nicht äquidistant sind:

$$F(\alpha)_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \tag{1.2}$$

für $j = 1, \ldots, N$. Die $x_1, \ldots, x_N$ sind die frei in $[-\pi, \pi]$ verteilten Stützstellen; sie sollen allerdings aufsteigend sortiert sein, also $x_k < x_l$ für $k < l$.
Der Umstand, daß die Summe hierbei von $-N/2$ bis $N/2 - 1$ läuft, ist für das Problem unwesentlich und stellt lediglich eine Umbenennung der Indizes dar. Wir werden darauf im letzten Abschnitt dieses Kapitels eingehen.
Unsere Ziel ist es, einen Algorithmus zur schnellen Berechnung dieser Fourier-Transformation anzugeben.

Neben nicht äquidistanten Stützstellen x_k kann man auch den Fall nicht äquidistanter

Frequenzen ω_k betrachten:

$$G(\alpha)_j = \sum_{k=1}^{N} \alpha_k \cdot e^{ij\omega_k \cdot 2\pi/N}. \tag{1.3}$$

für $j = -N/2, \ldots, N/2-1$. Die $\omega_1, \ldots, \omega_N$ sind jetzt die frei in $[-N/2, N/2]$ verteilten Frequenzen.
Setzt man in (1.3) $x_k = \omega_k \cdot 2\pi/N$, so erhält man:

$$G(\alpha)_j = \sum_{k=1}^{N} \alpha_k \cdot e^{ijx_k}. \tag{1.4}$$

Hieraus sieht man, daß $G = F^T$ ist, G also den zu F transponierten Operator darstellt. Auch hierzu werden wir im folgenden geeignete Algorithmen angeben, die eine schnelle Berechnung ermöglichen.

Bedauerlicherweise ist die diskrete Fourier-Transformation bei nicht äquidistanten Daten nicht mehr unitär, sodaß man die inverse Transformation nicht mehr direkt angeben kann. Deshalb werden wir versuchen, Algorithmen anzugeben, die die effiziente Berechnung der Inversen von (1.2) und (1.4) zulassen.
Zusammenfassend sind wir an der schnellen Berechnung der folgenden Transformationen interessiert:

Problem 1: Sei α gegeben, berechne $f = F(\alpha)$.

Problem 2: Sei β gegeben, berechne $g = F^T(\beta)$.

Problem 3: Sei f gegeben, berechne $\alpha = F^{-1}(f)$.

Problem 4: Sei g gegeben, berechne $\beta = (F^T)^{-1}(g)$.

Hierbei sind $\alpha = \alpha_{-N/2}, \ldots, \alpha_{N/2-1}$, $\beta = \beta_1, \ldots, \beta_N$, $f = f_1, \ldots, f_N$ und $g = g_{-N/2}, \ldots, g_{N/2-1}$ komplexe Vektoren.
Aus naheliegenden Gründen werden wir Problem 1 das direkte Problem, Problem 2 das transponierte Problem, Problem 3 das inverse Problem und Problem 4 das invers transponierte Problem nennen.

1.2 Die Idee

Die schnellen Fourier-Transformationen nach Cooley-Tukey oder nach Winograd beschleunigen die Berechnung von (1.1), indem Sie aus der durch die Gleichverteilung der Stützstellen bedingten Struktur Nutzen ziehen und durch algebraische Umformungen die Anzahl der benötigten Rechenoperationen verringern.

In dieser Arbeit nehmen wir uns des Problems von nicht gleichverteilen Stützstellen an. Wir wollen davon ausgehen, daß die Stützstellen beliebig verteilt sind und somit keinerlei Symmetrien mehr unterliegen. Somit kann man nicht hoffen, durch algebraische Umformungen eine Verringerung der Anzahl von Rechenoperationen zu erreichen.
Damit bleibt eigentlich nur ein Weg zur schnellen Berechnung: man muß das Problem der nicht äquidistanten Stützstellen auf ein Problem für äquidistante Stützstellen zurückführen.
Hierzu werden wir zwei Lösungsansätze betrachten:

1. Aufteilung der Werte an den nicht äquidistanten Stützstellen auf benachbarte äquidistante Stützstellen.

2. Berechnung einer Formel, die den Zusammenhang zwischen einer Fourier-Transformation an äquidistanten und einer an nicht äquidistanten Punkten herstellt.

Im nächsten Kapitel wenden wir uns der ersten Variante zu; die Transformation einer äquidistanten FT in eine nicht äquidistante FT wird uns in Kapitel 3 beschäftigen.

1.3 Bezeichnungen

FFT bezeichne im folgenden immer die bekannten Algorithmen zur schnellen Berechnung der diskreten Fourier-Transformation bei äquidistanten Stützstellen, zuvorderst also den von J. W. Cooley und J. W. Tukey 1965 angegebenen Algorithmus.
DFT bezeichne eines der vier angesprochenen Probleme, also eine diskrete Fourier-Transformation bei nicht (notwendig) äquidistanten Daten. Zur Unterscheidung von der bekannten FFT werden wir die in dieser Arbeit vorgestellten Algorithmen zur schnellen Berechnung einer diskreten Fourier-Transformtaion als *schnelle DFT* bezeichnen.

1.4 Symmetrische FFT

Wir werden für die FFT bei äquidistanten Daten zumeist die folgende symmetrische Formulierung verwenden:

$$f_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{\pm 2\pi i jk/N} \tag{1.5}$$

für $j = -N/2, \ldots, N/2-1$. Das Vorzeichen im Exponenten bewirkt eine Fourier-Transformation bzw. eine inverse Fourier-Transformation. Gebräuchlicher, und in den meisten FFT-Implementierungen anzutreffen, ist jedoch die folgende Formulierung:

$$\tilde{f}_j = \sum_{k=0}^{N-1} \tilde{\alpha}_k \cdot e^{\pm 2\pi ijk/N} \tag{1.6}$$

für $j = 0, \ldots, N-1$. Dieses beiden Formulierungen sind jedoch völlig äquivalent. Um dies zu zeigen, führen wir in (1.6) die folgende Variablentransformation durch:

$$\tilde{\alpha}_k = \begin{cases} \alpha_k & : \quad k = 0, \ldots, N/2-1 \\ \alpha_{-N+k} & : \quad k = N/2, \ldots, N-1 \end{cases} \tag{1.7}$$

Damit verändert sich (1.6) wie folgt:

$$\begin{aligned}
\tilde{f}_j &= \sum_{k=0}^{N-1} \tilde{\alpha}_k \cdot e^{\pm 2\pi ijk/N} && (1.8) \\
&= \sum_{k=0}^{N/2-1} \tilde{\alpha}_k \cdot e^{\pm 2\pi ijk/N} + \sum_{k=N/2}^{N-1} \tilde{\alpha}_k \cdot e^{\pm 2\pi ijk/N} && (1.9) \\
&= \sum_{k=0}^{N/2-1} \alpha_k \cdot e^{\pm 2\pi ijk/N} + \sum_{k=N/2}^{N-1} \alpha_{-N+k} \cdot e^{\pm 2\pi ijk/N} && (1.10) \\
&= \sum_{k=0}^{N/2-1} \alpha_k \cdot e^{\pm 2\pi ijk/N} + \sum_{k=-N/2}^{-1} \alpha_k \cdot e^{\pm 2\pi i(k+N)j/N} && (1.11) \\
&= \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{\pm 2\pi ijk/N} && (1.12)
\end{aligned}$$

für $j = 0, \ldots, N-1$. Setzen wir jetzt noch

$$\tilde{f}_k = \begin{cases} f_k & : \quad k = 0, \ldots, N/2-1 \\ f_{-N+k} & : \quad k = N/2, \ldots, N-1 \end{cases}, \tag{1.13}$$

so geht wegen $f_j = f_{-N+j}$ Formulierung (1.6) in die symmetrische Version (1.5) über.
Sind α bzw. f auf einem Rechner in einem Array gespeichert, so bewirken die Variablen-Transformationen (1.7) und (1.13) einfach, daß die erste Hälfte des Arrays mit der zweiten Hälfte des Arrays vertauscht wird.
Mit der Routine `swap_ranges` der C++ Standard Template Libraray sieht eine Implementierung der symmetrischen FFT unter Verwendung einer null-basierten FFT aus einer Programmbibliothek dann wie folgt aus:

```
void sym_fft(complex *array,int N) {
 swap_ranges(array,array+N/2,array+N/2); // convert 'array'
                                         // to zero-based form
 fft(array,N,1); // standard fft routine
 swap_ranges(array,array+N/2,array+N/2);  // convert result back
                                          //to symmetric form
}
```

Die Routine `swap_ranges` ist Bestandteil des derzeitigen C++ Sprachstandards. Weitere Informationen über diese Funktion und den C++ Sprachstandard befinden sich im Anhang.

Kapitel 2

schnelle DFT durch Interpolation

Die diskrete Fouriertransformation (DFT) bei nicht äquidistanten Stützstellen wird durch die folgende Formel beschrieben:

$$f_j = F(\alpha)_j = \sum_{k=-N/2}^{k=N/2-1} \alpha_k \cdot e^{ikx_j} \tag{2.1}$$

wobei $x_j \in [-\pi, \pi]$ für $j = 1, \ldots, N$ die Stützstellen festlegt.
Im äquidistanten Fall tritt statt e^{ikx_j} der Term $e^{2\pi ikj/N}$ auf. Wir werden nun in (2.1) den Exponentialteil durch die Summe einiger (weniger) Exponentialfunktionen bzgl. äquidistanter Stützstellen approximieren.

$$e^{ikx_j} \tag{2.2}$$

wird also durch einen Term der Form

$$\sum_q \rho_q \cdot e^{2\pi ikq/N} \tag{2.3}$$

ersetzt, wobei die Summe über q nur einige wenige Summanden umfassen sollte. Wenn man die so entstehenden Terme dann geeignet zusammenfaßt, erhält man eine Fouriertransformation bezüglich äquidistanter Stützstellen, die man per FFT schnell berechnen kann.
Wir wollen also einen Term der Form e^{ikx_j} annähern. Wichtigstes Hilfsmittel hierzu wird die Fourier-Reihe sein.

2.1 Mathematische Grundlagen

Hauptergebnis dieses Abschnittes ist Satz 2.1.7, der eine Approximation für e^{icx} beschreibt. Für das Verständnis des Algorithmus ist die Betrachtung der Herleitung in diesem Abschnitt nicht zwingend erforderlich. Der nur an der Implementierung interessierte Leser kann also sofort zu Satz 2.1.7 weiterblättern.

Lemma 2.1.1 *Für jede reelle $b > 0$ und $a >$ gilt:*

$$\int_a^\infty e^{-bx^2}\,dx < \frac{e^{-ba^2}}{2ba}. \tag{2.4}$$

Beweis:

$$\begin{aligned}
\int_a^\infty e^{-bx^2}\,dx &= \int_0^\infty e^{-b(x+a)^2}\,dx \\
&= \int_0^\infty e^{-bx^2-2bax-ba^2}\,dx \\
&< e^{-ba^2}\int_0^\infty e^{-2bax}\,dx \\
&= \frac{e^{-ba^2}}{2ba}
\end{aligned}$$

□

Die grundlegende Idee für unser Verfahren bildet die Betrachtung der Fourier-Entwicklung von Funktionen $\phi : [-\pi,\pi] \to \mathbf{C}$, die von der folgenden Form sind:

$$\phi(x) = e^{-bx^2}\cdot e^{icx} \tag{2.5}$$

wobei $b > \frac{1}{2}$ und c reelle Zahlen sind.
Für die Fehlerabschätzung benötigen wir die folgenden Lemmata:

Lemma 2.1.2 *Für $b > \frac{1}{2}$ reell, c reell und k ganzzahlig gilt:*

$$\left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx + e^{-b\pi^2}\cdot\int_{-\pi}^{\pi} e^{i(c-k)x}\,dx\right| < 2\pi e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi^2}\right). \tag{2.6}$$

Beweis:
Für $c = k$ ergibt sich die Aussage unmittelbar aus Lemma 2.1.1. Für $c \neq k$ erhält man:

$$\left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx + e^{-b\pi^2}\cdot\int_{-\pi}^\pi e^{i(c-k)x}\,dx\right|$$

$$\begin{aligned}
&\leq \left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx\right| + \left|e^{-b\pi^2}\cdot\int_{-\pi}^\pi e^{i(c-k)x}\,dx\right| && \text{(Dreiecksungleichung)}\\
&\leq 2\int_\pi^\infty e^{-bx^2}\left|\cos((c-k)x)\right|\,dx \; + e^{-b\pi^2}\cdot\int_{-\pi}^\pi \left|e^{i(c-k)x}\right|\,dx\\
&\leq 2\int_\pi^\infty e^{-bx^2}\,dx \; + 2\pi e^{-b\pi^2}\\
&< 2\frac{e^{-b\pi^2}}{2b\pi} + 2\pi e^{-b\pi^2} && \text{(gemäß Lemma 2.1.1 für } a=\pi)\\
&= 2\pi e^{-b\pi^2}\cdot\left(\frac{1}{2b\pi^2}+1\right)\\
&< 2\pi e^{-b\pi^2}\cdot\left(\frac{1}{\pi^2}+1\right) && \text{(da } 2b>1)
\end{aligned}$$

□

Mit etwas mehr Aufwand kann man für (2.6) auch eine andere Abschätzung beweisen:

Lemma 2.1.3 *Für $b > \frac{1}{2}$ reell, c reell und k ganzzahlig gilt:*

$$\left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx + e^{-b\pi^2}\cdot\int_{-\pi}^\pi e^{i(c-k)x}\,dx\right| < \frac{8b\pi e^{-b\pi^2}}{(c-k)^2}\cdot\left(1+\frac{1}{\pi^2}\right). \tag{2.7}$$

Beweis:
Durch partielle Integration des ersten Summanden erhalten wir:

$$2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx \tag{2.8}$$

$$= \frac{2}{c-k}\left[e^{-bx^2}\sin((c-k)x)\right]_\pi^\infty + \frac{4b}{c-k}\int_\pi^\infty xe^{-bx^2}\sin((c-k)x)\,dx \tag{2.9}$$

$$= -\frac{2}{c-k}e^{-b\pi^2}\sin((c-k)\pi) + \frac{4b}{c-k}\int_\pi^\infty xe^{-bx^2}\sin((c-k)x)\,dx. \tag{2.10}$$

Jetzt bringen wir den ersten Summanden in (2.10) auf die andere Seite und integrieren dann erneut partiell:

$$\begin{aligned}
&\left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx + \frac{2}{c-k}e^{-b\pi^2}\sin((c-k)\pi)\right| && (2.11)\\
&= \left|\frac{4b}{c-k}\int_\pi^\infty xe^{-bx^2}\sin((c-k)x)\,dx\right| && (2.12)\\
&= \left|-\frac{4b}{(c-k)^2}\left(\left[xe^{-bx^2}cos((c-k)x)\right]_\pi^\infty - \int_\pi^\infty (1-2bx^2)e^{-bx^2}\cos((c-k)x)\,dx\right)\right|\\
&\le \frac{4b}{(c-k)^2}\left(\left|\pi e^{-b\pi^2}cos((c-k)\pi)\right| + \left|\int_\pi^\infty (1-2bx^2)e^{-bx^2}\cos((c-k)x)\,dx\right|\right) && (2.13)\\
&\le \frac{4b}{(c-k)^2}\left(\left|\pi e^{-b\pi^2}\right| + \int_\pi^\infty \left|(1-2bx^2)e^{-bx^2}\right|\,dx\right) && (2.14)\\
&\le \frac{4b}{(c-k)^2}\left(\pi e^{-b\pi^2} + \left|\int_\pi^\infty e^{-bx^2}\,dx\right| + \left|\int_\pi^\infty 2bx^2e^{-bx^2}\,dx\right|\right) && (2.15)\\
&\le \frac{4b}{(c-k)^2}\left(\pi e^{-b\pi^2} + \int_\pi^\infty e^{-bx^2}\,dx + \int_\pi^\infty x\cdot 2bxe^{-bx^2}\,dx\right) && (2.16)\\
&= \frac{4b}{(c-k)^2}\left(\pi e^{-b\pi^2} + \int_\pi^\infty e^{-bx^2}\,dx + \left[-xe^{-bx^2}\right]_\pi^\infty + \int_\pi^\infty e^{-bx^2}\,dx\right) && (2.17)\\
&< \frac{4b}{(c-k)^2}\left(\pi e^{-b\pi^2} + \frac{e^{-b\pi^2}}{2b\pi} + \pi e^{-b\pi^2} + \frac{e^{-b\pi^2}}{2b\pi}\right) \quad \text{(gemäß Lemma 2.1.1)} && (2.18)\\
&= \frac{8be^{-b\pi^2}}{(c-k)^2}\left(\pi + \frac{1}{2b\pi}\right) && (2.19)
\end{aligned}$$

Insgesamt erhält man so:

$$\begin{aligned}
&\left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx + e^{-b\pi^2}\cdot\int_{-\pi}^\pi e^{i(c-k)x}\,dx\right| && (2.20)\\
= &\left|2\int_\pi^\infty e^{-bx^2}\cos((c-k)x)\,dx + e^{-b\pi^2}\cdot\frac{2\sin((c-k)\pi)}{c-k}\right| && (2.21)\\
< &\frac{8be^{-b\pi^2}}{(c-k)^2}\left(\pi + \frac{1}{2b\pi}\right) \quad \text{(gemäß (2.19))} && (2.22)\\
< &\frac{8b\pi e^{-b\pi^2}}{(c-k)^2}\left(1 + \frac{1}{\pi^2}\right) && (2.23)
\end{aligned}$$

□

Der folgende Satz beschreibt die Approximation der Funktion ϕ durch eine Fourierreihe:

Satz 2.1.4 *Sei* $\phi(x) = e^{-bx^2} \cdot e^{icx}$ *für reelles* $b > \frac{1}{2}$ *und reelles* c. *Dann gilt für jedes* $x \in (-\pi, \pi)$:

$$\left|\phi(x) - \sum_{k=-\infty}^{\infty} \rho_k e^{ikx}\right| < e^{-b\pi^2} \cdot \left(4b + \frac{80}{9}\right), \tag{2.24}$$

wobei die ρ_k *für* $k = -\infty, \ldots, \infty$ *wie folgt definiert sind:*

$$\rho_k = \frac{1}{2\sqrt{b\pi}} e^{-(c-k)^2/4b}. \tag{2.25}$$

Beweis:
σ_k bezeichne den k-ten Fourierkoeffizienten von ϕ, dann kann man $\phi(x)$ durch die folgende Fourierreihe approximieren:

$$\phi(x) = \sum_{k=-\infty}^{\infty} \sigma_k e^{ikx} \tag{2.26}$$

für $x \in (-\pi, \pi)$.
Die Fourierkoeffizienten σ_k lassen sich wie folgt beschreiben:

$$\begin{aligned}
\sigma_k &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \phi(x) e^{-ikx}\, dx && (2.27)\\
&= \frac{1}{2\pi} \left(\int_{-\infty}^{\infty} e^{-bx^2} e^{icx} e^{-ikx}\, dx - \int_{-\infty}^{-\pi} e^{-bx^2} e^{icx} e^{-ikx}\, dx - \int_{\pi}^{\infty} e^{-bx^2} e^{icx} e^{-ikx}\, dx \right) \\
&= \frac{1}{2\pi} \left(\sqrt{\frac{\pi}{b}} \cdot e^{-(c-k)^2/4b} + \int_{\infty}^{\pi} e^{-bx^2-icx+ikx}\, dx - \int_{\pi}^{\infty} e^{-bx^2+icx-ikx}\, dx \right) && (2.28)\\
&= \frac{1}{2\sqrt{b\pi}} \cdot e^{-(c-k)^2/4b} - \frac{1}{2\pi} \left(\int_{\pi}^{\infty} e^{-bx^2} e^{-i(c-k)x}\, dx + \int_{\pi}^{\infty} e^{-bx^2} e^{i(c-k)xx}\, dx \right) && (2.29)\\
&= \rho_k - \frac{1}{\pi} \int_{\pi}^{\infty} e^{-bx^2} cos((c-k)x)\, dx && (2.30)
\end{aligned}$$

Insgesamt erhält man also:

$$2 \int_{\pi}^{\infty} e^{-bx^2} cos((c-k)x)\, dx = 2\pi\, (\sigma_k - \rho_k) \tag{2.31}$$

Die linke Seite kann man in die Aussagen von Lemma 2.1.2 bzw. 2.1.3 einsetzen. Damit ergibt sich:

$$\left| \sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi} \int_{-\pi}^{\pi} e^{i(c-k)x}\, dx \right| \quad < \quad e^{-b\pi^2} \cdot \left(1 + \frac{1}{\pi^2}\right) \tag{2.32}$$

$$\left|\sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{i(c-k)x}\,dx\right| \quad < \quad \frac{4b}{(c-k)^2}\cdot e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi^2}\right) \tag{2.33}$$

Die zweite Abschätzung ist offensichtlich schärfer für $(c-k)^2 > 4b$.
Mit diesen Abschätzungen ermittelt man:

$$\left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - e^{-b\pi^2}\cdot e^{icx}\right| \tag{2.34}$$

$$= \left|\sum_{k=-\infty}^{\infty}\sigma_k e^{ikx} - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - \sum_{k=-\infty}^{\infty}\left(\frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{icx'}e^{-ikx'}\,dx'\right)e^{ikx}\right| \tag{2.35}$$

$$= \left|\sum_{k=-\infty}^{\infty} e^{ikx}\left(\sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{icx'}e^{-ikx'}\,dx'\right)\right| \tag{2.36}$$

$$\leq \sum_{k=-\infty}^{\infty}\left|\sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{icx'}e^{-ikx'}\,dx'\right| \tag{2.37}$$

$$< \sum_{k,|c-k|\geq 3}\frac{4b}{(c-k)^2}\cdot e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi^2}\right) + \sum_{k,|c-k|<3} e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi^2}\right) \tag{2.38}$$

$$< 4be^{-b\pi^2}\cdot\frac{9}{8}\cdot 2\cdot\sum_{k=3}^{\infty}\frac{1}{k^2} + 6e^{-b\pi^2}\cdot\frac{10}{9} \tag{2.39}$$

Da $\sum_{k=1}^{\infty}\frac{1}{k^2} = \frac{\pi^2}{6}$ kann man $\sum_{k=3}^{\infty}\frac{1}{k^2}$ gegen $\frac{4}{9} > \frac{\pi^2}{6} - 1 - \frac{1}{4}$ abschätzen und erhält so:

$$\left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - e^{-b\pi^2}\cdot e^{icx}\right| < e^{-b\pi^2}\cdot\left(4b + \frac{60}{9}\right) \tag{2.40}$$

Wenden wir auf diese Relation nochmals die Dreiecksungleichung an, so erhält man das endgültige Ergebnis:

$$\left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx}\right| \leq \left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - e^{-b\pi^2}\cdot e^{icx}\right| + \left|e^{-b\pi^2}\cdot e^{icx}\right| \tag{2.41}$$

$$< e^{-b\pi^2}\cdot\left(4b + \frac{70}{9}\right) \tag{2.42}$$

□

Gemäß Satz 2.1.4 können Funktionen der Form $e^{-bx^2}e^{icx}$ also durch Fourierreihen mit analytisch berechenbaren Koeffizienten angenähert werden. Der Fehler der Approximation fällt mit zunehmendem b exponentiell ab.

Die Koeffizienten ρ_k haben einen peak bei $k = [c]$ und fallen für $k \to \pm\infty$ exponentiell ab. Deshalb verwenden wir nur die $q+1$ größten Koeffizienten. q ist hierbei so gewählt, daß $q \geq 4b\pi$ und somit $e^{-(q/2)^2/4b} \leq e^{-b\pi^2}$. Der folgende Satz schätzt den dann entstehenden Fehler ab:

Satz 2.1.5 *Seien $b > \frac{1}{2}$, c reell und sei q eine ganze gerade Zahl mit $q \geq 4b\pi$. Dann gilt für jedes $x \in (-\pi, \pi)$*

$$\left| e^{-bx^2} \cdot e^{icx} - \sum_{k=[c]-q/2}^{k=[c]+q/2} \rho_k e^{ikx} \right| < e^{-b\pi^2} \cdot (4b+9), \tag{2.43}$$

wobei die ρ_k wie in (2.25) definiert sind.

Beweis:
Für jedes $x \in (-\pi, \pi)$ gilt:

$$\left| \phi(x) - \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx} \right| \tag{2.44}$$

$$= \left| \phi(x) - \left(\sum_{k=-\infty}^{\infty} \rho_k e^{ikx} - \sum_{k>[c]-q/2} \rho_k e^{ikx} - \sum_{k<[c]-q/2} \rho_k e^{ikx} \right) \right| \tag{2.45}$$

$$\leq \left| \phi(x) - \sum_{k=-\infty}^{\infty} \rho_k e^{ikx} \right| + \left| \sum_{k>[c]+q/2} \rho_k e^{ikx} \right| + \left| \sum_{k<[c]-q/2} \rho_k e^{ikx} \right|. \tag{2.46}$$

Mit der Definition der ρ_k und der Dreiecksungleichung kann man die letzten beiden Summen wie folgt abschätzen:

$$\begin{aligned} \left| \sum_{k>[c]+q/2} \rho_k e^{ikx} \right| &\leq \frac{1}{2\sqrt{b\pi}} \sum_{k=[c]+q/2+1} e^{-(c-k)^2/4b} \\ &= \frac{1}{2\sqrt{b\pi}} \sum_{k=q/2}^{\infty} e^{-(c-k-[c]-1)^2/4b} \\ &< \frac{1}{2\sqrt{b\pi}} \sum_{k=q/2}^{\infty} e^{-k^2/4b} \end{aligned} \tag{2.47}$$

sowie

$$\begin{aligned}\left|\sum_{k<[c]-q/2}\rho_k e^{ikx}\right| &\leq \frac{1}{2\sqrt{b\pi}}\sum_{k=-\infty}^{[c]-q/2-1} e^{-(c-k)^2/4b}\\ &= \frac{1}{2\sqrt{b\pi}}\sum_{k=-\infty}^{-q/2} e^{-(c-k-[c]+1)^2/4b} \qquad (2.48)\\ &< \frac{1}{2\sqrt{b\pi}}\sum_{k=q/2}^{\infty} e^{-k^2/4b}\end{aligned}$$

Hierfür wiederum kann man die folgende Abschätzung verwenden:

$$\sum_{k=q/2}^{\infty} e^{-k^2/4b} < e^{-(q/2)^2/4b} + \int_{q/2}^{\infty} e^{-x^2/4b}\,dx \qquad (2.49)$$

$$< e^{-(q/2)^2/4b} + \int_{q/2}^{\infty}\left(1+\frac{4b}{2x^2}\right) e^{-x^2/4b}dx \qquad (2.50)$$

$$= e^{-(q/2)^2/4b} + \left[\frac{-4b}{2x}e^{-x^2/4b}\right]_{q/2}^{\infty} \qquad (2.51)$$

$$= e^{-(q/2)^2/4b}\left(1+\frac{4b}{2(q/2)}\right) \qquad (2.52)$$

Da nach Voraussetzung $q \geq 4b\pi$ gilt, vereinfacht sich dies weiter zu

$$\sum_{k=q/2}^{\infty} e^{-k^2/4b} < e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi}\right) \qquad (2.53)$$

Setzen wir dies in (2.47) und (2.48) ein, so erhalten wir

$$\left|\sum_{k>[c]+q/2}\rho_k e^{ikx}\right| + \left|\sum_{k<[c]-q/2}\rho_k e^{ikx}\right| < \frac{2e^{-b\pi^2}}{\sqrt{2\pi}}\cdot\left(1+\frac{1}{\pi}\right) < e^{-b\pi^2}\cdot\frac{10}{9} \qquad (2.54)$$

Setzen wir dieses Ergebnis in unsere Ausgangsgleichung (2.46) ein, so ergibt sich:

$$\left|\phi(x) - \sum_{k=[c]-q/2}^{[c]+q/2}\rho_k e^{ikx}\right| \qquad (2.55)$$

$$< \left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx}\right| + e^{-b\pi^2}\cdot\frac{10}{9} \qquad (2.56)$$

Wenden wir schließlich noch Lemma 2.1.4 an, so erhalten wir das Ergebnis:

$$\left|\phi(x) - \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx}\right| < e^{-b\pi^2} \cdot \left(4b + \frac{70}{9} + \frac{10}{9}\right) < e^{-b\pi^2} \cdot (4b+9). \tag{2.57}$$

□

Da wir die Ungleichung ja zur Approximation eines Fourier-Terms e^{ikx_j} verwenden wollen, multiplizieren wir die Aussage von Satz 2.1.5 mit e^{bx^2} und erhalten so:

Corollar 2.1.6 *Sei $m \geq 2$ eine ganze Zahl sowie $b > \frac{1}{2}, c$ reell und q eine ganze gerade Zahl mit $q \geq 4b\pi$. Dann gilt für jedes $x \in [-\frac{\pi}{m}, \frac{\pi}{m}]$:*

$$\begin{aligned} \left|e^{icx} - e^{bx^2} \cdot \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx}\right| &< e^{bx^2} \cdot e^{-b\pi^2} \cdot (4b+9) && (2.58) \\ &\leq e^{b\pi^2/m^2} \cdot e^{-b\pi^2} \cdot (4b+9). && (2.59) \end{aligned}$$

Durch eine einfache Skalierung erhalten wir schließlich das endgültige Ergebnis dieses Abschnittes.

Satz 2.1.7 *Seien $b > \frac{1}{2}, c, d > 0$ reelle Zahlen, und seien $m \geq 2, q \geq 4b\pi$ ganze Zahlen. Dann gilt für jedes $x \in [-d, d]$*

$$\left|e^{icx} - e^{b(x\pi/md)^2} \cdot \sum_{k=[cmd/\pi]-q/2}^{k=[cmd/\pi]+q/2} \rho_k e^{ikx\pi/md}\right| < e^{-b\pi^2(1-1/m^2)} \cdot (4b+9) \tag{2.60}$$

wobei die ρ_k für $k = -\infty, \ldots, \infty$ wie folgt definiert sind:

$$\rho_k = \frac{1}{2\sqrt{b\pi}} e^{-(\frac{cmd}{\pi}-k)^2/4b}. \tag{2.61}$$

Beweis:
Ersetzt man in Korollar 2.1.6 $c \to \frac{md}{\pi}c$ und $x \to \frac{\pi}{md}x$, so erhält man:

$$\left|e^{i\frac{md}{\pi}c\frac{\pi}{md}x} - e^{b\left(\frac{\pi}{md}x\right)^2} \cdot \sum_{k=[\frac{md}{\pi}c]-q/2}^{[\frac{md}{\pi}c]+q/2} \frac{1}{2\sqrt{b\pi}} e^{-(\frac{cmd}{\pi}-k)^2/4b} e^{ik\frac{\pi}{md}x}\right| \tag{2.62}$$

$$
\begin{aligned}
&= \left| e^{icx} - e^{b(x\pi/md)^2} \cdot \sum_{k=[cmd/\pi]-q/2}^{[cmd/\pi]+q/2} \frac{1}{2\sqrt{b\pi}} e^{-(\frac{cmd}{\pi}-k)^2/4b} e^{ikx\pi/md} \right| && (2.63)\\
&< e^{b\pi^2/m^2} \cdot e^{-b\pi^2} \cdot (4b+9) && (2.64)\\
&= e^{-b\pi^2(1-1/m^2)} \cdot (4b+9) && (2.65)
\end{aligned}
$$

für jedes $x \in [-d, d]$. □

Diese Fehlerabschätzungen sind sehr pessimistisch, wie die noch folgenden numerischen Untersuchungen zeigen werden. Der Algorithmus liefert nämlich bereits für kleine q (< 50) sehr gute Werte, und da q über $q \geq 4b\pi\ b$ nach oben beschränkt, bleibt b viel zu klein, als daß $e^{-b\pi^2}$ eine scharfe Abschätzung liefern könnte.

2.2 Beschreibung der Algorithmen

Ein Algorithmus für das direkte Problem

In Satz 2.1.7 haben wir eine Approximation für e^{icx} erhalten. Dies wenden wir jetzt auf e^{ikx_j} im folgenden direkten Problem an:

$$f_j = F(\alpha)_j = \sum_{k=-N/2}^{k=N/2-1} \alpha_k \cdot e^{ikx_j} \tag{2.66}$$

Sei $m \geq 2$ eine ganze Zahl,$b > 0$ reell und q gerade mit $q \geq 4b\pi$. Wir definierten $\epsilon > 0$ durch

$$\epsilon = e^{-b\pi^2(1-1/m^2)} \cdot (4b+9). \tag{2.67}$$

ν_j sei die zu $x_j mN/2\pi$ nächstgelegene ganze Zahl für $j = 1, \ldots, N$ und $\{Q_{jk}\}$ sei eine Matrix reeller Zahlen, definiert durch

$$Q_{jl} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(x_j mN/2\pi-(\nu_j+l))^2/4b}. \tag{2.68}$$

für $j = 1, \ldots, N$ und $l = -q/2, \ldots, q/2$. Wenden wir Satz 2.1.7 mit $x = k$, $c = x_j$ und $d = N/2$ an, so ergibt sich:

$$\left| e^{ikx_j} - e^{b(2\pi k/mN)^2} \cdot \sum_{l=-q/2}^{q/2} Q_{jl} \cdot e^{i(\nu_j+l)2\pi k/mN} \right| < \epsilon \tag{2.69}$$

Setzen wir diese Approximation für e^{ikx_j} in die DFT (2.66) ein, so erhalten wir eine Annäherung $\tilde{f}_j$:

$$\tilde{f}_j = \sum_{k=-N/2}^{k=N/2-1} \underbrace{\alpha_k \cdot e^{b(2\pi k/mN)^2}}_{:=u_k} \cdot \sum_{l=-q/2}^{q/2} Q_{jl} \cdot e^{i(\nu_j+l)2\pi k/mN} \tag{2.70}$$

$$= \sum_{l=-q/2}^{q/2} Q_{jl} \underbrace{\sum_{k=-N/2}^{k=N/2-1} u_k \cdot e^{i(\nu_j+l)2\pi k/mN}}_{:=U_{\nu_j+l}} \tag{2.71}$$

$$= \sum_{l=-q/2}^{q/2} Q_{jl} U_{\nu_j+l} \tag{2.72}$$

Hierbei hängt Q_{jl} nur von der Wahl der Stützstellen ab und läßt sich somit vorausberechnen. u_k stellt lediglich eine skalierte Form des Ausgangsvektors α_k dar.

$$U_l = \sum_{k=-N/2}^{k=N/2-1} u_k \cdot e^{il2\pi k/mN} \tag{2.73}$$

für $l = -N/2, \ldots, N/2-1$ läßt sich mittels inverser FFT in $\mathcal{O}(mN \log N)$ Operationen berechnen, wenn man u_k für $k < -N/2$ und $k \geq N/2$ gleich Null setzt.
Ferner ist zu beachten, daß U_l nur definiert ist für $l = -mN/2, \ldots, mN/2-1$. Da aber für ν_j gilt: $-mN/2 \leq \nu_j \leq mN/2$, sind die in (2.72) auftretenden Indizes $\nu_j + l$ aus $[-\frac{mN}{2} - \frac{q}{2}, \frac{mN}{2} + \frac{q}{2}]$. Dies ist aber kein Problem: Aus der Definition von U in (2.71) kann man entnehmen, daß U_l periodisch ist mit der Periode mN, d.h. es gilt $U_{\nu_j+l} = U_{\nu_j+l+mN}$. Der Index $\nu_j + l$ ist also modulo mN zu nehmen.
Die abschließende Berechnung der $\tilde{f}_j$ mittels Q erfordert dann noch $\mathcal{O}(Nq) = \mathcal{O}(N \cdot \log(1/\epsilon))$ Rechenoperationen.
Unser Algorithmus besteht nun aus zwei Stufen: einer Initialisierungsroutine und einer Routine, die die eigentliche Berechnung durchführt.
Wir nennen diesen Algorithmus zur Berechnung der schnellen DFT bei nicht äquidistanten Stützstellen und äquidistanten Frequenzen *Algorithmus A1*.

Die Initialisierungsroutine

Der Initialisierungsroutine werden die Stützstellen x_j für $j = 1, \ldots, N$ sowie die den Grad der Approximation bestimmenden Konstanten m, q und b übergeben. Sie führt dann die folgenden drei Schritte aus:

1. Berechne ν_j als die zu $x_j mN/2\pi$ nächstgelegene ganze Zahl für $j = 1, \ldots, N$.

2. Initialisiere die reelle Matrix $Q_j l$ für $j = 1, \ldots, N$ und $l = -q/2, \ldots, q/2$

$$Q_{jl} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(x_j mN/2\pi - [\nu_j + l))^2/4b} \tag{2.74}$$

   für $j = 1, \ldots, N$ und $l = -q/2, \ldots, q/2$.

3. Berechne die Skalierungsfaktoren $e^{b(2\pi k/mN)^2}$ für $k = -N/2, \ldots, N/2 - 1$ vor, und speichere sie in einem Array zum schnellen Abruf während der eigentlichen Berechnung.

Die Routine legt also ein reellwertiges Array der Dimension N für die Skalierungsfaktoren, ein ebenfalls N-dimensionales Integer-Array ν für die angenäherten Stützstellen sowie eine $N \times q$-Matrix Q an.

Der Algorithmus A1

Dem Algorithmus wird der zu transformierende Vektor α_k für $k = -N/2, \ldots, k = N/2-1$ übergeben. Die Routine benötigt ferner Zugriff auf die von der Initialisierungsroutine vorberechneten Daten sowie auf die dort verwandten Konstanten m, q und b.
Die schnelle Berechnung der DFT erfolgt in drei Schritten:

1. Skaliere den Eingansvektor $[\alpha_{-N/2}, \ldots, \alpha_{N/2}]$ durch

$$u_k = \alpha_k \cdot e^{b(2\pi k/mN)^2}. \tag{2.75}$$

   Die Skalierungsfaktoren $e^{b(2\pi k/mN)^2}$ werden hierbei dem in der Initalisierungsroutine erstellten Array entnommen.

2. Berechne U mittels inverser FFT der Länge mN:

$$U_l = \sum_{k=-N/2}^{k=N/2-1} u_k \cdot e^{il2\pi k/mN} \tag{2.76}$$

   Hierzu muß der Vektor u geeignet mit Nullen erweitert werden, damit er die Dimension mN besitzt. Außerdem muß, wie in der Einleitung beschrieben, auf die symmetrische Formulierung ($k = -mN/2, \ldots, mN/2 - 1$) Rücksicht genommen werden.

3. Berechne das Ergebnis $\tilde{f}_j$ aus U:

$$\tilde{f}_j = \sum_{l=-q/2}^{q/2} Q_{jl} U_{\nu_j+l} \tag{2.77}$$

für $j = 1, \ldots, N$. Der Index $\nu_j + l$ ist hierbei modulo mN zu nehmen.

Ein Algorithmus für das transponierte Problem

Neben dem Problem des vorherigen Abschnitts, kann man auch das dazu transponierte Problem betrachten:

$$f_j = \sum_{k=-N/2}^{k=N/2-1} \alpha_k \cdot e^{ij\omega_k} \tag{2.78}$$

für $j = -N/2, \ldots, N/2-1$.
$\omega_k \in [-\pi, \pi]$ für $k = 1, \ldots, N$ bezeichnet hierbei die Potenzen (oder Frequenzen), nach denen entwickelt wird. Jetzt sind also die Stützstellen äquidistant, während die Frequenzen nicht mehr äquidistant sind.
Sei $m \geq 2$ wieder eine ganze Zahl, $b > 0$ reell und q gerade mit $q \geq 4b\pi$. Wir definierten $\epsilon > 0$ durch

$$\epsilon = e^{-b\pi^2(1-1/m^2)} \cdot (4b+9). \tag{2.79}$$

μ_k sei die zu $m\omega_k \cdot N/2\pi$ nächstgelegene ganze Zahl für $k = 1, \ldots, N$ und $\{P_{lk}\}$ sei eine Matrix reeller Zahlen, definiert durch

$$P_{lk} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(m\omega_k \cdot N/2\pi - (\mu_k + l))^2/4b}. \tag{2.80}$$

für $l = -q/2, \ldots, q/2$ und $k = 1, \ldots, N$. Gemäß Satz 2.1.7 für $d = N/2$, $x = j$ und $c = \omega_k$, erhalten wir

$$\left| e^{i\omega_k j} - e^{b(2\pi j/mN)^2} \cdot \sum_{l=-q/2}^{q/2} P_{lk} \cdot e^{i(\mu_k+l)2\pi j/mN} \right| < \epsilon \tag{2.81}$$

Setzen wir diese Approximation für $e^{i\omega_k x}$ mit $x = 2\pi j/N$ in die DFT (2.78) ein, so erhalten wir eine Annäherung $\tilde{f}_j$:

$$\tilde{f}_j = \sum_{k=-N/2}^{k=N/2-1} \alpha_k \cdot e^{b(2\pi j/mN)^2} \cdot \sum_{l=-q/2}^{q/2} P_{lk} \cdot e^{i(\mu_k+l)2\pi j/mN} \tag{2.82}$$

$$= \quad e^{b(2\pi j/mN)^2} \cdot \sum_{k=-N/2}^{k=N/2-1} \sum_{l=-q/2}^{q/2} \alpha_k \cdot P_{lk} \cdot e^{i(\mu_k+l)2\pi j/mN} \tag{2.83}$$

Jetzt wollen wir $\mu_k + l$ durch einen einzigen Index k ersetzen. Vereinfacht wird dies durch den Umstand, daß die Exponentialfunktion in (2.83) mN-periodisch ist, $\mu_k + l$ und $\mu_k + l + mN$ also identische Werte liefern. Die von l und k abhängenden Koeffizienten $\alpha_k \cdot P_{lk}$ werden gemäß der folgenden Definition zusammengefaßt:

$$\tau_l = \sum_{j,k:\mu_k+j=l} \alpha_k \cdot P_{lk} \tag{2.84}$$

für $l = -mN/2, \ldots, mN/2 - 1$. (2.83) wird somit zu:

$$\tilde{f}_j = e^{b(2\pi j/mN)^2} \cdot \sum_{k=-mN/2}^{k=mN/2-1} \tau_k \cdot e^{2\pi ikj/mN} \tag{2.85}$$

für $j = -N/2, \ldots, N/2 - 1$. Die Summe läßt sich hierbei wieder schnell per inverser FFT der Länge mN berechnen.
Der Algorithmus läuft also im wesentlichen wie der zum vorherigen Problem ab, allerdings mit vertauschter Reihenfolge der drei Schritte: zuerst werden die Koeffizienten auf das äquidistante Gitter verteilt, dann wird eine FFT der Länge mN durchgeführt und schließlich wird das Ergebnis skaliert.
Da das Problem ja wie gesehen als zum vorherigen transponiert aufgefaßt werden kann, liegt diese Vertauschung der Reihenfolge natürlich auf der Hand.
Wir nennen diesen Algorithmus zur Berechnung der schnellen DFT bei äquidistanten Stützstellen und nicht äquidistanten Frequenzen *Algorithmus A2.*

Die Initialisierungsroutine

Der Initialisierungsroutine werden die Frequenzen ω_k für $k = 1, \ldots, N$, an denen die DFT ausgewertet wird, sowie die den Grad der Approximation bestimmenden Konstanten m, q und b übergeben. Sie führt dann die folgenden drei Schritte aus:

1. Berechne μ_k als die zu $m\omega_k \cdot N/2\pi$ nächstgelegene ganze Zahl für $k = 1, \ldots, N$.

2. Initialisiere die reelle Matrix P_{lk} für $l = -q/2, \ldots, q/2$ und $k = 1, \ldots, N$

$$P_{lk} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(m\omega_k \cdot N/2\pi - (\mu_k+l))^2/4b}. \tag{2.86}$$

3. Berechne die Skalierungsfaktoren $e^{b(2\pi j/mN)^2}$ für $j = -N/2, \ldots, N/2 - 1$ vor, und speichere sie in einem Array zum schnellen Abruf während der eigentlichen Berechnung.

Die Routine legt also ein reellwertiges Array der Dimension N für die Skalierungsfaktoren, ein ebenfalls N-dimensionales Integer-Array ν für die angenäherten Stützstellen sowie eine $N \times q$-Matrix P an.

Der Algorithmus

Dem Algorithmus wird der zu transformierende Vektor α_k für $k = 1, \ldots, N$ übergeben. Die Routine benötigt ferner Zugriff auf die von der Initialisierungsroutine vorberechneten Daten sowie auf die dort verwandten Konstanten m, q und b.
Die schnelle Berechnung der DFT erfolgt in drei Schritten:

1. Berechne die neuen Koeffizienten τ_l aus α für $l = -mN/2, \ldots, mN/2$:

$$\tau_l = \sum_{j,k:\mu_k+j=l} \alpha_k \cdot P_{lk} \tag{2.87}$$

   für $k = 1, \ldots, N$ und $j = -q/2, \ldots, q/2$. Der Index $\mu_k + j$ ist hierbei modulo mN zu nehmen.

2. Berechne T mittels inverser FFT der Länge mN:

$$T_j = \sum_{k=-mN/2}^{k=mN/2-1} \tau_k \cdot e^{2\pi ijk/mN} \tag{2.88}$$

   für $j = -mN/2, \ldots, mN/2 - 1$.

3. Skaliere die notwendigen Elemente von T_j, um den Ergebnisvektor zu erhalten.

$$\tilde{f}_j = T_j \cdot e^{b(2\pi j/mN)^2} \tag{2.89}$$

   für $j = -N/2, \ldots, N/2 - 1$. Die Skalierungsfaktoren $e^{b(2\pi j/mN)^2}$ werden hierbei dem in der Initalisierungsroutine erstellten Array entnommen.

2.3 Numerische Untersuchungen

Die Algorithmen für die beiden diskreten Fourier-Transformationen wurden von mir in C++ implementiert. Zu der Implementierung sind folgende Anmerkungen zu machen:

1. Zur Speicherung der Vektoren und Matrizen wurden die beiden von mir implementieren Klassen TMatrix und TVector verwendet. Diese enthalten Standardfunktionalitäten für Matrizen und Vektoren (Addition, Multiplikation etc.) und ermöglichen so einen einfachen und dennoch effizienten Umgang mit diesen Datenstrukturen. Außerdem ist es damit möglich, Indizes nicht nur wie in normalen C-Arrays mit 0 beginnen zu lassen, sondern mit beliebigen integer-Werten. Somit lassen sich auch Matrizen wie beispielsweise das in (2.74) definierte Q, für das der Spaltenindex von $-q/2$ bis $q/2$ läuft, mit gut lesbarem Quellcode implementieren, ohne daß sich das Laufzeitverhalten verschlechtert.

2. Ansonsten lassen sich die formalen Algorithmus-Beschreibungen des vorherigen Abschnittes beinahe direkt nach C++ übertragen. Die Algorithmen wurden als Template-Klassen implementiert, somit können sie sowohl mit einfacher als auch mit doppelter oder gar noch höherer Genauigkeit arbeiten. Dies entspricht den C++ Datentypen `<complex<float> >`, `<complex<double> >` und `<complex<long double> >`. Weitere Information zur Behandlung komplexer Zahlen befinden sich im Anhang.

3. Für die Berechnung der inversen FFT wurde die FFT-Routine der Numerical Recipes [10] verwendet. Diese besitzt ein relativ gutes Laufzeitverhalten und ist im Quellcode verfügbar, somit also auf allen Plattformen einsetzbar. Bei den Laufzeitmessungen wird vor allem die Laufzeit unserer Algorithmen im Verhältnis zur Laufzeit einer FFT gleicher Länge als Maß zur Bewertung der Geschwindigkeit herangezogen; deshalb spielt die Wahl der FFT-Routine keine so entscheidende Rolle.

4. In der Beschreibung der Algorithmen wurden die DFT der Länge N auf eine FFT der Länge mN zurückgeführt. Damit sich dies schnell berechnen läßt, muß mN dafür einen Wert annehmen, für den eine FFT zur Verfügung steht, also z.B. $mN = 2^k$. Die Algorithmen funktionieren aber auch, wenn man für die FFT nicht mehr die Länge mN, sondern allgemein eine Zweierpotenz 2^k mit $2^k \geq mN$ wählt. Auf diese Weise kann man die Algorithmen ohne weitere Vorkehrungen für Fourier-Transformationen beliebiger Länge verwenden. Allerdings erreichen sie natürlich nur

für Zweierpotenzen N die optimale Effizienz. Die entsprechenden Modifikationen wurden bei der Implementierung der Algorithmen vorgenommen, sodaß diese DFTs beliebiger Länge berechnen können.

5. Für die numerischen Untersuchungen wurde der Eingangsvektor α mit Zufallswerten aus dem komplexen Einheitsquadrat gefüllt:

$$0 \leq \mathrm{Re}(\alpha_j) \leq 1, \qquad 0 \leq \mathrm{Im}(\alpha_j) \leq 1. \tag{2.90}$$

Für die Stützstellen $x_0, \ldots, x_N$ der DFT wurden gleichverteilte Zufallswerte aus dem Intervall $[-\pi, \pi]$ verwendet.

6. Neben dem Laufzeitverhalten der Algorithmen sind wir auch an deren Genauigkeit interessiert, ein schneller Algorithmus, der ungenaue Ergebnisse liefert ist uninteressant. Zur Messung der Approximationsfehler verwenden wir die beiden folgenden relativen Fehlernormen:

$$E_\infty = \frac{\left\| \tilde{f}_j - f_j^{Soll} \right\|_\infty}{\|\alpha\|_1} \tag{2.91}$$

sowie

$$E_2 = \frac{\left\| \tilde{f}_j - f_j^{Soll} \right\|_2}{\|f\|_2} \tag{2.92}$$

Die Normen $\|\cdot\|_1$, $\|\cdot\|_2$ und $\|\cdot\|_\infty$ sind hierbei natürlich wie folgt definiert:

$$\|x\|_1 = \sum_{j=1}^{N} |x_j| \qquad \|x\|_2 = \left(\sum_{j=1}^{N} |x_j| \right)^{1/2} \qquad \|x\|_\infty = \max_{1 \leq j \leq N} |x_j|. \tag{2.93}$$

α ist der Eingangsvektor, f^{Soll} die direkt und mit `long double`-Genauigkeit berechnete DFT und $\tilde{f}$ schließlich die mit unseren Algorithmen schnell berechnete DFT.

Wahl der Approximationskonstanten m, q und b

Zuerst ermitteln wir den Fehler, der bei direkter Berechnung mit einfacher und doppelter Genauigkeit auftritt. Diese Werte sind natürlich mit einiger Vorsicht zu betrachten, da auch der Vergleichswert f^{Soll} lediglich mit `long double`-Genauigkeit berechnet wurde und somit ebenfalls mit einem Maschinenfehler behaftet ist. Als Vergleichswerte zum Bewerten des Approximationsfehler unseres Algorithmus sind sie jedoch vollständig ausreichend.

Tabelle 1 zeigt die Rechenfehler für die DFT sowie für die transponierte DFT. Man erkennt, daß die Fehler größenordnungsmäßig mit einfacher Genauigkeit bei 10^{-5}, mit doppelter Genauigkeit bei 10^{-14} liegen.

TABELLE 1A
Maschinenfehler bei direkter Berechnung mit einfacher Genauigkeit

N	DFT		transponierte DFT	
	E_∞	E_2	E_∞	E_2
64	8.5792e-07	4.2883e-07	4.1903e-07	1.4840e-07
128	1.7662e-06	1.3375e-06	6.6483e-07	1.9187e-07
256	2.3618e-06	3.2349e-06	1.0363e-06	2.6266e-07
512	7.4043e-06	5.4340e-06	1.5305e-06	3.5867e-07
1024	1.2313e-05	6.6167e-06	2.6166e-06	5.4829e-07
2048	2.8964e-05	1.9946e-05	3.8820e-06	7.5288e-07
4096	5.1660e-05	4.6070e-05	5.6853e-06	1.0557e-06

TABELLE 1B
Maschinenfehler bei direkter Berechnung mit doppelter Genauigkeit

N	DFT		transponierte DFT	
	E_∞	E_2	E_∞	E_2
64	1.7455e-15	7.1093e-16	7.2167e-16	2.7924e-16
128	4.3711e-15	2.9222e-15	1.4492e-15	4.0965e-16
256	5.2433e-15	5.8508e-15	2.1962e-15	4.7910e-16
512	1.5375e-14	8.4330e-15	3.2681e-15	7.2372e-16
1024	2.8038e-14	1.4416e-14	5.1812e-15	1.0352e-15
2048	3.6559e-14	2.2058e-14	8.0416e-15	1.4139e-15
4096	5.8742e-14	6.7282e-14	1.2233e-14	1.9720e-15

Die Güte und damit natürlich leider ebenso die Laufzeit der Algorithmen wird von den drei Konstanten m, q und b bestimmt. m stellt hierbei gewissermaßen den Oversampling-Faktor dar: bei der Berechnung wird ja eine Fourier-Transformation der Länge mN ausgeführt. Es ist somit ratsam, sich mit einem möglichst kleinen m zu begnügen: denn während eine FFT der Länge $2 \cdot 1024$ nur ungefähr $2,2$ mal langsamer ist als eine FFT der Länge 1024, liegt der Faktor für $4 \cdot 1024$, also $m = 4$ bereits bei $4,8$. Wir werden deshalb im folgenden immer von der kleinstmöglichen Wahl, nämlich $m = 2$ ausgehen. Es wird sich zeigen, daß sich hiermit bereits ausreichende Genauigkeiten erzielen lassen, sodaß ein größeres m nicht sinnvoll ist.

Bleiben also noch die Konstanten q und b. b ist durch $q \geq 4b\pi$ nach oben beschränkt. Die Wahl von b hat keine Auswirkungen auf das Laufzeitverhalten, sehr wohl aber Einfluß auf die Qualität der Approximation. Wir werden deshalb zu jedem q ein adäquates b wählen, um möglichst geringe Approximationsfehler zu erhalten.

Tabelle 2 zeigt die jeweils optimale Wahl für b bei verschiedenen Werten für q sowie den hierbei auftretenden Fehler. Die Werte wurden für $N = 1024$ ermittelt. Wie Berechnungen mit anderen Längen zeigen, ist das optimale b aber unabhängig von N. Ebenso erhält man auch für den Algorithmus A2, der das transponierte Problem löst, dieselben Werte für b. Die Diagramme 1 und 2 stellen nochmals den relativen Fehler E_∞ in Abhänigkeit von b für verschiedene q-Werte grafisch dar.

Man sieht also, daß der Algorithmus bei einfacher Genauigkeit ab $q = 6$ und bei doppelter Genauigkeit ab $q = 24$ dieselbe Genauigkeit erzielt, wie die direkte Berechnung; bei größeren q-Werten übertrifft er diese sogar. Um auf der sicheren Seite zu liegen verwendet die vorliegende Implementierung deshalb bei einfacher Genauigkeit $q = 10$ und bei doppelter Genauigkeit $q = 28$.

TABELLE 2A *optimales b bei einfacher Genauigkeit*

q	b	E_∞
4	0.27	7.75E-05
6	0.4	8.35E-06
8	0.5	8.61E-07
10	0.59	9.04E-08
12	0.72	9.82E-09
14	0.77	7.64E-09
16	0.76	7.34E-09
18	0.74	7.35E-09
20	0.8	7.67E-09

Die Werte wurden für eine DFT der Länge 1024 ermittelt.
Zum Vergleich: bei der direkten Berechnung entsteht ein Fehler E_∞ von 1.23E-05.

TABELLE 2B

optimales b bei doppelter Genauigkeit

q	b	E_∞	q	b	E_∞
6	0.4	8.35E-06	24	1.34	2.00E-14
8	0.5	8.59E-07	26	1.46	2.19E-15
10	0.59	9.01E-08	28	1.56	1.73E-16
12	0.7	6.80E-09	30	1.69	5.27E-17
14	0.8	8.64E-10	32	1.72	4.84E-17
16	0.92	1.33E-10	34	1.67	5.11E-17
18	1.02	1.32E-11	36	1.7	4.96E-17
20	1.13	9.95E-13	38	1.78	5.13E-17
22	1.24	1.27E-13	40	1.7	5.40E-17

Die Werte wurden für eine DFT der Länge 1024 ermittelt.
Zum Vergleich: bei der direkten Berechnung entsteht ein Fehler E_∞ von 2.8E-14.

Laufzeitverhalten

Tabelle 3 zeigt das Laufzeitverhalten von Algorithmus A1 bei einfacher und doppelter Genauigkeit. Algorithmus A1 berechnet eine DFT von der folgenden Form:

$$F(\alpha)_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \tag{2.94}$$

für $j = 1, \ldots, N$. Die $x_1, \ldots, x_N$ sind die frei in $[-\pi, \pi]$ verteilten Stützstellen. Tabelle 3 illustriert das Laufzeitverhalten und den Fehler bei einfacher und doppelter Genauigkeit. Man sieht, daß der Algorithmus bei doppelter Genauigkeit ungefähr um den Faktor 5 langsamer ist, als eine FFT gleicher Länge.

TABELLE 3A

Laufzeitverhalten von A1 bei einfacher Genauigkeit

N	E_∞	E_2	Init msecs	Direct msecs	Eval msecs	FFT msecs	Faktor
64	1.52e-06	2.06e-06	3.00	0.00	0.27	0.07	3.86
128	4.39e-07	2.81e-06	4.00	6.00	0.55	0.13	4.23
256	5.36e-07	2.76e-06	5.00	24.00	1.10	0.24	4.58
512	1.25e-06	2.65e-06	14.00	82.00	2.44	0.51	4.78
1024	2.18e-06	3.88e-06	31.00	309.00	6.00	1.37	4.39
2048	1.12e-06	1.93e-06	62.00	1241.00	12.69	3.62	3.51

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.

TABELLE 3B

Laufzeitverhalten von A1 bei doppelter Genauigkeit

N	E_∞	E_2	Init msecs	Direct msecs	Eval msecs	FFT msecs	Faktor
64	1.25e-15	9.61e-15	3.00	10.00	0.91	0.18	5.06
128	1.31e-15	4.77e-15	7.00	27.00	1.90	0.34	5.58
256	1.43e-15	6.78e-15	19.00	91.00	4.38	0.78	5.61
512	1.65e-15	5.72e-15	38.00	366.00	9.74	1.75	5.54
1024	2.00e-15	9.24e-15	76.00	1478.00	21.25	4.25	5.00
2048	1.88e-15	2.10e-14	165.00	5840.00	46.05	9.41	4.89

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

Tabellen 4a und 4b zeigen das Laufzeitverhalten von Algorithmus A2 bei einfacher und doppelter Genauigkeit. Algorithmus A2 berechnet eine DFT der folgenden Form:

$$f_j = \sum_{k=1}^{N} \alpha_k \cdot e^{ij\omega_k}. \tag{2.95}$$

für $j = -N/2, \ldots, N/2-1$. Die $\omega_1, \ldots, \omega_N$ sind jetzt die frei in $[-\pi, \pi]$ verteilten Frequenzen.
Man sieht, daß der Algorithmus bei doppelter Genauigkeit ungefähr um den Faktor 5 langsamer ist, als eine FFT gleicher Länge.

Die Approximationskonstanten wurden hierbei so gewählt, daß sich die Approximationsfehler in der Größenordnung der bei direkter Berechnung auftretenden Maschinenfehler bewegen.

TABELLE 4A

Laufzeitverhalten von A2 bei einfacher Genauigkeit

N	E_∞	E_2	Init msecs	Direct msecs	Eval msecs	FFT msecs	Faktor
64	9.11e-07	1.53e-06	0.00	18.00	0.25	0.07	3.68
128	6.06e-07	1.48e-06	3.00	61.00	0.55	0.13	4.23
256	8.13e-07	1.93e-06	4.00	244.00	1.10	0.25	4.37
512	6.28e-07	1.55e-06	12.00	965.00	2.45	0.51	4.79
1024	5.39e-07	1.63e-06	36.00	3835.00	6.45	1.35	4.76
2048	5.99e-07	1.82e-06	58.00	15388.00	13.70	3.62	3.79

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

TABELLE 4B

Laufzeitverhalten von A2 bei doppelter Genauigkeit

N	E_∞	E_2	Init msecs	Direct msecs	Eval msecs	FFT msecs	Faktor
64	3.67e-15	4.74e-15	3	19	1.00	0.18	5.55
128	3.07e-15	3.70e-15	9	82	2.01	0.36	5.58
256	2.86e-15	5.73e-15	20	296	4.50	0.77	5.91
512	4.13e-15	9.79e-15	37	1185	11.75	1.75	6.71
1024	2.85e-15	1.71e-14	78	4731	19.35	4.25	4.55
2048	5.10e-15	3.40e-14	152	18959	41.35	9.41	4.39

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

Schließlich ist noch darauf hinzuweisen, daß die ermittelten Laufzeitwerte und besonders der Faktor, der das Verhältnis zur FFT angibt, nicht nur von der Rechnerarchitektur, sondern auch von der Rechnerauslastung zum Zeitpunkt der Messung abhängen. Der Prozessor-Cache hat nämlich einigen Einfluß auf die Effizienz der Algorithmen. Wenn der betreffende Rechner viele Prozesse verwalten muß, enthält der Cache relativ häufig nicht die passenden Daten (cache miss), was die Laufzeit verschlechtert. Aus diesem Grunde kann die Laufzeit auch auf einem Rechner bei verschiedenen Testreihen um bis zu 50% variieren.

Insgesamt betrachtet, scheinen die angegebenen Algorithmen das gestellte Problem relativ gut zu lösen. Sie sind nur um den Faktor 5 langsamer als eine FFT gleicher Länge. 50% der Laufzeit wird dazu verwendet, eine FFT der doppelten Länge auszuführen, sodaß relativ wenig Möglichkeiten bleiben, das Problem ohne Genauigkeitsverluste schneller zu berechnen.

Allerdings haben wir bisher noch keine Möglichkeit gefunden, auch die inversen diskreten Fourier-Transformationen schnell zu berechnen. Eine Methode, dies zu bewerkstelligen wird Gegenstand des nächsten Kapitels sein.

Kapitel 3

schnelle DFT durch Multipolentwicklung

3.1 Die Idee

In diesem Kaptiel werden wir eine weitere Methode vorstellen, mit der die Fourier-Transformation bei beliebigen Stützstellen in $\mathcal{O}(N \cdot \log N + N \cdot \log(1/\epsilon))$ berechnet werden kann. Das grundlegende Prinzip ist vergleichsweise einfach: Wenn F die gesuchte diskrete Fourier-Transformation ist und $\mathcal{F}$ die FFT bei äquidistanten Stützstellen bezeichnet, dann suchen wir einen Operator $\mathcal{P}$, derart daß $F = \mathcal{P}\mathcal{F}$.
Wir werden sehen, daß die Anwendung des Operators $\mathcal{P}$ im Wesentlichen einer Auswertung einer Summe der Form $\sum_k \alpha_k \frac{1}{\tan(y_l - x_k)}$ entspricht. Diese Summe werden wir dann mit Hilfe der schnellen Multipolentwicklung (FMM - fast multipole method) schnell berechnen.

Den Transformations-Operator $\mathcal{P}$ gewinnt man mit folgender Überlegung:
Seien $\{x_1, \ldots, x_N\}$ und $\{y_1, \ldots, y_N\}$ reelle Zahlen im Intervall $[-\pi, \pi]$ (die Polarwinkel der Stützstellen) und $\{\alpha_1, \ldots, \alpha_N\}$ ein Vektor komplexer Zahlen (der zu transformierende Vektor). Ferner seien w_j,z_j,f_j und g_j definiert durch

$$w_j = e^{ix_j} \qquad \text{Stützstellen bzgl. der } x_j \tag{3.1}$$

$$z_j = e^{iy_j} \qquad \text{Stützstellen bzgl. der } y_j \tag{3.2}$$

$$f_j = \sum_{k=1}^{N} \alpha_k \cdot e^{iky_j} \qquad \text{Fouriertransformierte bzgl. der } y_j \tag{3.3}$$

$$g_j = \sum_{k=1}^{N} \alpha_k \cdot e^{ikx_j} \qquad \text{Fouriertransformierte bzgl. der } x_j \tag{3.4}$$

Betrachtet man

$$f_j = \sum_{k=1}^{N} \alpha_k \cdot \left(e^{iy_j}\right)^k, \tag{3.5}$$

so kann man das Polynom

$$P(z) = \sum_{k=1}^{N} \alpha_k \cdot (z)^k \tag{3.6}$$

definieren und es gilt $f_j = P(z_j)$ sowie $g_j = P(w_j)$
Man erhält also die Fouriertransformierte durch Auswertung eines (trigonometrischen) Polynoms, und die Fouriertransofrmierte bzgl. anderer Stützstellen bedeutet lediglich die Auswertung des Polynoms an anderen Punkten.
Die Vorgehensweise ist daher die folgende:

- Wir ermitteln die f_j durch FFT. Diese sind die Funktionswerte der Funktion P an den Stellen z_j.
- Die durch f_j an den Stellen z_j tabullierte Funktion interpolieren wir nun durch Lagrange-Interpolation, das Interpolationspolynom ist dann natürlich wieder P selbst.
- Auswertung dieses Polynoms an den Stellen w_j liefert die gesuchten g_j bzw. einen formalen Zusammenhang $\mathcal{P}$ zwischen den f_j und den g_j.

Wenn man dieses $\mathcal{P}$ schnell berechnen kann, hat man eine Methode, die DFT bei nicht äquidistanten Daten zu ermitteln.

3.2 Polynominterpolation auf dem Einheitskreis

Für unseren Algorithmus müssen wir eine Polynom-Interpolation nach Lagrange durchführen. Die Langrange-Polynominterpolation wird bekanntlich durch die folgende Formel beschrieben:

$$P_N(z) = \sum_{j=1}^{N} f(z_j) \cdot \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{z - z_k}{z_j - z_k} \tag{3.7}$$

Das Produkt ist offensichtlich 1 für $z = z_j$ und 0 für $z = z_k, k \neq j$. Damit folgt unmittelbar, daß $P_N(z_j) = f(z_j)$, das Polynom stimmt also an den Stützstellen mit der zu interpolierenden Funktion überein.
Die folgenden beiden Lemmata spezialisieren die Lagrange-Interpolation für den Fall, daß die z und z_j auf dem komplexen Einheitskreis liegen:

Lemma 3.2.1 *Seien $\{x_1, \ldots, x_N\}$ und $\{y_1, \ldots, y_N\}$ reelle Zahlen im Intervall $[-\pi, \pi]$ (die Polarwinkel der Stützstellen) Ferner seien w_j und z_j definiert durch*

$$w_j = e^{ix_j} \qquad \textit{Stützstellen bzgl. der } x_j \tag{3.8}$$

$$z_j = e^{iy_j} \qquad \textit{Stützstellen bzgl. der } y_j \tag{3.9}$$

für $j = 1, \ldots, N$. Dann gilt

$$\prod_{k=1,k\neq j}^{N} (w_l - z_k) = w_l^{(N-1)/2} \cdot \prod_{k=1,k\neq j}^{N} z_k^{1/2} \cdot 2i \cdot \sin((x_l - y_k)/2) \tag{3.10}$$

für $l = 1, \ldots, N$. Ferner gilt

$$\prod_{k=1,k\neq j}^{N} (z_j - z_k) = z_j^{(N-1)/2} \cdot \prod_{k=1,k\neq j}^{N} z_k^{1/2} \cdot 2i \cdot \sin((y_j - y_k)/2) \tag{3.11}$$

für $j = 1, \ldots, N$.

Beweis:
Der Beweis erfolgt durch einfache algebraische Umformungen und trigonometrische Identitäten:

$$\begin{aligned}
\prod_{\substack{k=1\\k\neq j}}^{N} (w_l - z_k) &= \prod_{\substack{k=1\\k\neq j}}^{N} \left(e^{ix_l} - e^{iy_k}\right) && (3.12)\\
&= \prod_{\substack{k=1\\k\neq j}}^{N} \left(e^{ix_l + y_k/2 - y_k/2} - e^{iy_k - x_l/2 + x_l/2}\right) && (3.13)\\
&= \prod_{\substack{k=1\\k\neq j}}^{N} e^{i(x_l+y_k)/2} \cdot \left(e^{i(x_l-y_k)/2} - e^{-i(x_l-y_k)/2}\right) && (3.14)
\end{aligned}$$

$$= e^{i(N-1)x_l/2} \cdot \prod_{\substack{k=1\\k\neq j}}^{N} e^{iy_k/2} \cdot 2i \cdot \sin\left((x_l - y_k)/2\right) \tag{3.15}$$

$$= w_l^{(N-1)/2} \cdot \prod_{\substack{k=1\\k\neq j}}^{N} z_k^{1/2} \cdot 2i \cdot \sin\left((x_l - y_k)/2\right) \tag{3.16}$$

$$\tag{3.17}$$

Die zweite Aussage folgt natürlich aus der ersten durch Substitution von w_l durch z_j und x_l durch y_j. □

Lemma 3.2.2 *Seien $\{x_1, \ldots, x_N\}$ und $\{y_1, \ldots, y_N\}$ reelle jeweils paarweise verschiedene Zahlen im Intervall $[-\pi, \pi]$ (die Polarwinkel der Stützstellen) und $\{f_1, \ldots, f_N\}$ ein Vektor komplexer Zahlen. Ferner seien w_j und z_j definiert durch*

$$w_j = e^{ix_j} \qquad \textit{Stützstellen bzgl. der } x_j \tag{3.18}$$

$$z_j = e^{iy_j} \qquad \textit{Stützstellen bzgl. der } y_j \tag{3.19}$$

Dann gilt:

$$\sum_{j=1}^{N} f_j \cdot \prod_{\substack{k=1\\k\neq j}}^{N} \frac{w_l - z_k}{z_j - z_k} = w_l^{N/2} \cdot c_l \cdot \sum_{j=1}^{N} f_j \cdot z_j^{-N/2} \cdot d_j \cdot \left(\frac{1}{tan((x_l - y_j)/2} - i\right) \tag{3.20}$$

wobei c_l und d_j wie folgt definiert sind:

$$c_l = \prod_{k=1}^{N} \sin((x_l - y_k)/2) \tag{3.21}$$

$$d_j = \prod_{\substack{k=1\\k\neq j}}^{N} \frac{1}{\sin((y_j - y_k)/2)} \tag{3.22}$$

Beweis:
Dividieren wir die beiden Aussagen des voherigen Lemmas durcheinander, so erhalten wir:

$$\prod_{\substack{k=1\\k\neq j}}^{N} \frac{w_l - z_k}{z_j - z_k} \quad = \quad \frac{w_l^{(N-1)/2}}{z_j^{(N-1)/2}} \cdot \prod_{\substack{k=1\\k\neq j}}^{N} \frac{z_k^{1/2} \cdot 2i \cdot \sin((x_l - y_k)/2)}{z_k^{1/2} \cdot 2i \cdot \sin((y_j - y_k)/2)} \tag{3.23}$$

$$= \frac{w_l^{N/2} e^{-ix_l/2}}{z_j^{N/2} e^{-iy_j/2}} \cdot \prod_{\substack{k=1 \\ k\neq j}}^{N} \frac{\sin((x_l - y_k)/2)}{\sin((y_j - y_k)/2)} \tag{3.24}$$

$$= \frac{w_l^{N/2} e^{-ix_l/2}}{z_j^{N/2} e^{-iy_j/2}} \cdot \frac{1}{\sin((x_l - y_j)/2)} \prod_{k=1}^{N} \sin((x_l - y_k)/2) \cdot \prod_{\substack{k=1 \\ k\neq j}}^{N} \frac{1}{\sin((y_j - y_k)/2)}$$

$$= \frac{w_l^{N/2}}{z_j^{N/2}} \cdot \frac{e^{-i(x_l - y_j)/2}}{\sin((x_l - y_j)/2)} \cdot c_l \cdot d_j \tag{3.25}$$

Der zweite Bruch läßt sich hierbei noch etwas vereinfachen:

$$\frac{e^{-i(x_l - y_j)/2}}{\sin((x_l - y_j)/2)} = \frac{\cos((x_l - y_j)/2) + i \sin(-(x_l - y_j)/2)}{\sin((x_l - y_j)/2)} \tag{3.26}$$

$$= \frac{1}{\tan((x_l - y_j)/2)} - i \tag{3.27}$$

Aufsummieren von (3.25) über alle j ergibt somit schließlich:

$$\sum_{j=1}^{N} f_j \cdot \prod_{\substack{k=1 \\ k\neq j}}^{N} \frac{w_l - z_k}{z_j - z_k} = w_l^{N/2} \cdot c_l \cdot \sum_{j=1}^{N} f_j \cdot z_j^{-N/2} \cdot d_j \cdot \left(\frac{1}{tan((x_l - y_j)/2} - i \right) \tag{3.28}$$

□

3.3 Transformation von Stützstellen

Wir haben gesehen, daß man die Fouriertransformation als Auswertung eines Polynoms an den jeweiligen Stützstellen betrachten kann. In diesem Abschnitt werden wir die Formel herleiten, die diese Auswertung bezüglich eines Satzes von Stützstellen auf die Auswertung bezüglich eines anderen Satzes transformiert.

Satz 3.3.1 *Seien* $\{x_1, \ldots, x_N\}$ *und* $\{y_1, \ldots, y_N\}$ *reelle jeweils paarweise verschiedene Zahlen im Intervall* $[-\pi, \pi]$ *(die Polarwinkel der Stützstellen) und* $\{\alpha_{-N/2}, \ldots, \alpha_{N/2-1}\}$ *ein Vektor komplexer Zahlen (der zu transformierende Vektor). Ferner seien* w_j,z_j,f_j *und* g_j *definiert durch*

$$w_j = e^{ix_j} \qquad \textit{Stützstellen bzgl. der } x_j \tag{3.29}$$

$$z_j = e^{iy_j} \qquad \textit{Stützstellen bzgl. der } y_j \tag{3.30}$$

$$f_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{iky_j} \qquad \textit{Fouriertransformierte bzgl. der } y_j \tag{3.31}$$

$$g_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \qquad \textit{Fouriertransformierte bzgl. der } x_j \tag{3.32}$$

dann besteht zwischen den f_j und g_j die folgende Beziehung:

$$g_l = c_l \cdot \sum_{j=1}^{N} f_j \cdot d_j \cdot \left(\frac{1}{\tan(x_l - y_j)/2)} - i \right) \tag{3.33}$$

wobei c_l und d_j wie oben definiert sind.

Beweis:
Das Polynom P_α sei definiert durch:

$$P_\alpha(z) = \sum_{k=0}^{N-1} \alpha_{k-N/2} \cdot z^k \tag{3.34}$$

Dieses Polynom erhält man natürlich auch, wenn eine Polynominterpolation nach Lagrange in den Stützstellen z_j durchführt:

$$P_\alpha(w_l) = \sum_{j=1}^{N} P_\alpha(z_j) \cdot \prod_{\substack{k=1 \\ k \neq j}} \frac{w_l - z_k}{z_j - z_k} \tag{3.35}$$

Wenden wir hierauf das vorherige Lemma an, so erhalten wir:

$$P_\alpha(w_l) = w_l^{N/2} \cdot \sum_{j=1}^{N} P_\alpha(z_j) \cdot z_j^{N/2} \cdot d_j \cdot \left(\frac{1}{\tan((x_l - y_j)/2)} - i \right) \tag{3.36}$$

Wenden wir die Definition von P_α auf die in den Voraussetzungen angegebenen z_j bzw. w_l an, so ergibt sich

$$P_\alpha(z_j) = z_j^{N/2} \cdot \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot z_j^k = z_j^{N/2} \cdot f_j \tag{3.37}$$

bzw.

$$P_\alpha(w_l) = w_l^{N/2} \cdot \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot w_l^k = w_l^{N/2} \cdot g_l \tag{3.38}$$

Setzen wir diese beiden Gleichungen in (3.36) ein, so erhalten wir

$$g_l = c_l \cdot \sum_{j=1}^{N} f_j \cdot d_j \cdot \left(\frac{1}{\tan((x_l - y_j)/2)} - i \right) \tag{3.39}$$

für $l = 1, \ldots, N$. □

Da der Algorithmus ja auf der FFT bei äquidistanten Stützstellen aufbaut, gehen wir davon aus, daß die y_j, die Stützstellen für die f_j, äquidistant sind. Unter dieser Voraussetzung läßt sich der vorherige Satz noch etwas vereinfachen:

Satz 3.3.2 *Seien $\{x_1, \ldots, x_N\}$ und $\{y_1, \ldots, y_N\}$ reelle jeweils paarweise verschiedene Zahlen im Intervall $[-\pi, \pi]$ (die Polarwinkel der Stützstellen) und $\{\alpha_{-N/2}, \ldots, \alpha_{N/2-1}\}$ ein Vektor komplexer Zahlen (der zu transformierende Vektor). Die y_j seien äquidistant auf $[-\pi, \pi]$ verteilt, also*

$$y_j = (j - 1 - N/2) \cdot \frac{2\pi}{N}. \tag{3.40}$$

Ferner seien w_j,z_j,f_j und g_j definiert durch

$$w_j = e^{ix_j} \qquad \textit{Stützstellen bzgl. der } x_j \tag{3.41}$$

$$z_j = e^{iy_j} \qquad \textit{Stützstellen bzgl. der } y_j \tag{3.42}$$

$$f_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{iky_j} \qquad \textit{Fouriertransformierte bzgl. der } y_j \tag{3.43}$$

$$g_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \qquad \textit{Fouriertransformierte bzgl. der } x_j \tag{3.44}$$

dann besteht zwischen den f_j und g_j die folgende Beziehung:

$$g_l = \sin\left(\frac{Nx_l}{2}\right) \cdot \sum_{j=1}^{N} f_j \cdot \frac{(-1)^j}{N} \cdot \left(\frac{1}{\tan(x_l - y_j)/2)} - i \right) \tag{3.45}$$

Die c_l und d_j sind wieder wie oben definierts.

Beweis:
Aus der Definition der y_j und f_j ergibt sich, daß der Vektor $\{\alpha_{-N/2}, \ldots, \alpha_{N/2-1}\}$ gerade die diskrete Fourier-Transformation des Vektors $\{f_1, \ldots, f_N\}$ ist. Anwendung der inversen Fourier-Transformation liefert:

$$\alpha_k = \frac{1}{N} \cdot \sum_{j=1}^{N} f_j \cdot e^{-iky_j} \tag{3.46}$$

für $k = 1, \ldots, N/2 - 1$. Definiert man eine Funktion $f : [-\pi, \pi] \to \mathbf{C}$:

$$f(x) = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx} \tag{3.47}$$

und setzt (3.46) ein, so erhält man:

$$\begin{aligned} f(x) &= \sum_{k=-N/2}^{k=N/2-1} \frac{1}{N} \cdot \sum_{j=1}^{N} f_j \cdot e^{-iky_j} \cdot e^{ikx} && (3.48) \\ &= \sum_{j=1}^{N} f_j \cdot \frac{1}{N} \cdot \sum_{k=-N/2}^{N/2-1} e^{ik(x-y_j)} && (3.49) \end{aligned}$$

Die zweite Summe in dieser Gleichung kann man als geometrische Reihe auffassen. Man erhält dann mit $\gamma = (x - y_j)/2$ zur Entlastung der Notation:

$$\begin{aligned} \sum_{k=-N/2}^{k=N/2-1} e^{ik(x-y_j)} &= \sum_{k=0}^{N-1} e^{i(k-N/2)(x-y_j)} && (3.50) \\ &= e^{-i \cdot N\gamma} \cdot \sum_{k=0}^{N-1} \left(e^{i(x-y_j)}\right)^k && (3.51) \\ &= e^{-i \cdot N\gamma} \cdot \frac{1 - e^{i \cdot N(x-y_j)}}{1 - e^{i(x-y_j)}} && (3.52) \\ &= \frac{e^{-i \cdot N\gamma} - e^{i \cdot N\gamma}}{e^{i\gamma} \cdot (e^{-i\gamma} - e^{i\gamma})} && (3.53) \\ &= \frac{\sin(N\gamma)}{e^{i\gamma} \cdot \sin(\gamma)} && (3.54) \\ &= \frac{\sin(N\gamma)}{(\cos(\gamma) + i\sin(\gamma)) \cdot \sin(\gamma)} && (3.55) \end{aligned}$$

$$\begin{aligned}
&= \sin(N\gamma)\cdot\frac{1}{\sin(\gamma)\cdot\cos(\gamma)+i\sin^2(\gamma)} && (3.56)\\
&= \sin(N\gamma)\cdot\frac{\sin(\gamma)\cos(\gamma)-i\sin^2(\gamma)}{\sin^2(\gamma)\cos^2(\gamma)+sin^4(\gamma)} && (3.57)\\
&= \sin(N\gamma)\cdot\frac{\frac{\cos(\gamma)}{\sin(\gamma)}-i}{\cos^2(\gamma)+sin^2(\gamma)} && (3.58)\\
&= \sin(N(x-y_j)/2)\cdot(\cot((x-y_j)/2)-i) && (3.59)
\end{aligned}$$

Die Definition der y_j ergibt:

$$\begin{aligned}
\sin(N(x-y_j)/2) &= \sin(Nx/2)\cos(Ny_j/2)-\cos(Nx/2)\sin(Ny_j/2) && (3.60)\\
&= \sin(Nx/2)(-1)^j && (3.61)
\end{aligned}$$

Insgesamt erhält man also

$$f(x)=\sum_{j=1}^{N} f_j\cdot\frac{1}{N}\cdot\sum_{k=-N/2}^{N/2-1}\sin(Nx/2)\cdot(-1)^j\cdot(\cot((x-y_j)/2)-i) \tag{3.62}$$

Da $g_l = f(x_l)$ ist, ergibt sich somit die Behauptung:

$$g_l=\sin\left(\frac{Nx_l}{2}\right)\cdot\sum_{j=1}^{N} f_j\cdot\frac{(-1)^j}{N}\cdot\left(\frac{1}{\tan(x_l-y_j)/2)}-i\right) \tag{3.63}$$

□

3.4 Schnelle Mutlipolentwicklung

3.4.1 Tschebyscheff-Approximation

Wir haben die DFT bzgl. der nicht äquidistanten Stützstellen x_j durch die (mittels der FFT schnell berechenbare) DFT bzgl. der äquidistanten Stützstellen y_j dargestellt. Allerdings benötigt das Berechnen aller g_l mit Hilfe von (3.63) auf direktem Wege immer noch $\mathcal{O}(N^2)$ Rechenoperationen, selbst wenn die Teile, die nur von den Stützstellen abhängen, bereits vorberechnet sind. Bisher haben wir also noch nichts gewonnen. Deshalb suchen wir jetzt nach Möglichkeiten, Summen der Art

$$g_l = \sum_{j=1}^{N} \frac{f_j}{\tan((x_l - y_j)/2)} \tag{3.64}$$

schnell zu berechnen. Hierbei hilft uns die folgende Überlegung:
Die Kernel-Funktion $1/\tan(\theta - \theta_k)$ nimmt große Werte nur für $\theta \approx \theta_k$, also in der Nähe der Singularität an. Deshalb werden wir für ein festes θ diejenigen wenigen Summanden, deren θ_k „nahe bei" θ liegt, direkt berechnet. Die restlichen Summanden kann man dann sehr gut durch eine Tschebyscheff-Interpolation annähren.
Diese Tschebyscheff-Approximation muß nur aus wenigen Termen bestehen (typischerweise 16), da die verbleibenden Summanden eine sehr glatte Funktion darstellen. Eine Tschebyscheff-Approximation von geringer Ordnung kann man dann entsprechend schnell auswerten. In diesem Abschnitt werden deshalb zuerst einmal einige grundlegende Dinge über die Tschebyscheff-Interpolation aufgelistet. Eine ausführlichere Betrachtung findet sich beispielsweise in [8].

Definition 3.4.1 *Das Tschebyscheff-Polynom p-ter Ordnung $T_p(x)$ wird definiert durch die beiden äquivalenten Formeln:*

$$T_p(x) = \cos(p \arccos x) \tag{3.65}$$

$$T_p(x) = \frac{1}{2} \cdot \left(\left(x + \sqrt{x^2 - 1} \right)^p + \left(x - \sqrt{x^2 - 1} \right)^p \right) \tag{3.66}$$

Definition 3.4.2 *Die Nullstellen $t_1, \ldots, t_p$ des Tschebyscheff-Polynoms p-ten Grades $T_p(x)$ liegen im Intervall $[-1, 1]$:*

$$t_k = -\cos\left(\frac{2k-1}{p} \cdot \frac{\pi}{2} \right) \tag{3.67}$$

für $k = 1, \ldots, p$. Dies ergibt sich unmittelbar aus der Definition (3.65).

Definition 3.4.3 *Die Polynome $u_1, \ldots, u_p$ vom Grade $p-1$ seien definiert durch*

$$u_j(t) = \prod_{\substack{k=1 \\ k \neq j}}^{p} \frac{t - t_k}{t_j - t_k} \tag{3.68}$$

für $j = 1, \ldots, p$. Die t_k sind dabei wie in (3.67) definiert. Offensichtlich gilt: $u_j(t_l) = \delta_{jl}$.

Für eine Funktion $f : [-1, 1] \to \mathbf{C}$ ist die Tschebyscheff-Approximation der Ordnung $p-1$ von f definiert durch das eindeutig bestimmte Polynom der Ordnung $p-1$, das mit f an allen Stützstellen $t_1, \ldots, t_p$ übereinstimmt. Es existieren verschiedene Darstellungen für dieses Polynom, wir verwenden die folgende unmittelbar einleuchtende:

$$\sum_{j=1}^{p} f(t_j) \cdot u_j(t) \tag{3.69}$$

Wegen $u_j(t_l) = \delta_{jl}$ stimmt das Interpolationspolynom für $t = t_j$ mit f überein.
Schließlich benötigen wir noch eine etwas weniger bekannte Eigenschaft der Tschebyscheff-Polynome.

Satz 3.4.4 *t_l seien die in (3.67) definierten Nullstellen des Tschebyscheff-Polynoms T_p vom Grade p. Dann gilt:*

$$\sum_{k=1}^{p-1} T_k(t_j) \cdot T_k(t_l) = \begin{cases} \frac{p-1}{2} & \textit{falls} \quad j = l \\ \frac{-1}{2} & \textit{falls} \quad j \neq l \end{cases} \tag{3.70}$$

Die folgenden Lemmata werden zur Abschätzung des Fehlers der Tschebyscheff-Approximation benötigt.

Lemma 3.4.5 *Sei $T_p(x)$ das Tschebyscheff Polynom vom Grade p. Dann folgt aus Definition (3.65), daß*

$$|T_p(x)| \leq 1 \tag{3.71}$$

für jedes $x \in [-1, 1]$.

Lemma 3.4.6 *Sei $T_p(x)$ das Tschebyscheff Polynom vom Grade p. Dann gilt*

$$|T_p(x)| > \frac{1}{2} \cdot \left|\frac{5x}{3}\right|^p \tag{3.72}$$

für alle x mit $|x| \geq 3$.

Beweis:
Für $|x| \geq 3$ gilt:

$$|T_p(x)| \quad = \quad \frac{1}{2} \cdot \left|\left(x + \sqrt{x^2-1}\right)^p + \left(x - \sqrt{x^2-1}\right)^p\right| \tag{3.73}$$

$$
\begin{aligned}
&> \frac{1}{2} \cdot \left|\left(x + \sqrt{x^2 - 1}\right)^p\right| && (3.74)\\
&\geq \frac{1}{2} \cdot \left|\left(x + \sqrt{x^2 - (x/3)^2}\right)^p\right| && (3.75)\\
&= \frac{1}{2} \cdot \left|x \cdot \left(1 + \sqrt{8/9}\right)^p\right| && (3.76)\\
&> \frac{1}{2} \cdot \left|\frac{5x}{3}\right|^p && (3.77)\\
& && (3.78)
\end{aligned}
$$

□

Lemma 3.4.7 *Sei $u_j(x)$ wie in (3.68) definiert. Dann gilt für alle $x \in [-1, 1]$:*

$$|u_j(x)| \leq 2 - \frac{1}{p}. \tag{3.79}$$

Beweis:
Aus der Definition der u_j ergibt sich, daß $u_j(t_j) = 1$ und $u_j(t_k)$=0 für $k \neq j$. Der Ausdruck

$$\frac{1}{p} + \frac{2}{p} \sum_{k=1}^{p-1} T_k(t_j) \cdot T_k(x) \tag{3.80}$$

ist wegen Satz 3.4.4 ebenfalls 1 für $x = t_j$ und 0 für $x = t_k$, $k \neq j$. Da sowohl u_j als auch (3.80) Polynome vom Grade $p - 1$ sind, sind die beiden Ausdrücke somit identisch. Also gilt für $j = 1, \ldots, p$

$$u_j(x) = \frac{1}{p} + \frac{2}{p} \sum_{k=1}^{p-1} T_k(t_j) \cdot T_k(x) \tag{3.81}$$

Eine einfache Abschätzung mit der Dreiecksungleichung liefert dann für alle $x \in [-1, 1]$

$$
\begin{aligned}
|u_j(x)| &= \left|\frac{1}{p} + \frac{2}{p} \sum_{k=1}^{p-1} T_k(t_j) \cdot T_k(x)\right| && (3.82)\\
&\leq \frac{1}{p} + \frac{2}{p} \sum_{k=1}^{p-1} |T_k(t_j)| \cdot |T_k(x)| && (3.83)\\
&\leq \frac{1}{p} + \frac{2(p-1)}{p} && (3.84)\\
&= 2 - \frac{1}{p}. && (3.85)
\end{aligned}
$$

□

Lemma 3.4.8 *Für jedes $a \in [0, \pi/8]$ gilt:*

$$\tan(3a) \geq 3 \cdot \tan(a) \tag{3.86}$$

Beweis:
Da der Tangens eine größere Steigung hat als die Identität, ist die Aussage eigentlich unmittelbar einleuchtend.
Formal sieht man es jedoch mit einer trigonometrischen Identität für $\tan(3x)$ wie folgt:

$$\begin{aligned}
\tan(3x) - 3\tan(x) &= \frac{3\tan x - \tan^3 x}{1 - 3\tan^2 x} - 3\tan x && (3.87)\\
&= \frac{3\tan x - \tan^3 x - 3\tan x(1 - 3\tan^2 x)}{1 - 3tan^2 x} && (3.88)\\
&= \frac{8\tan^3 x}{1 - 3\tan^2 x} && (3.89)\\
&\geq 0, && (3.90)
\end{aligned}$$

da $1 - 3\tan^2 x$ und $\tan^3 x > 0$ für $x \in [0, \pi/8]$. □

3.4.2 Approximation von $1/\tan(x - x_0)$

Die folgenden beiden Lemmata werden im Beweis von Satz 3.4.11 benötigt.

Lemma 3.4.9 *Sei $p \geq 2$ und $b > 0$ sowie x_0 reelle Zahlen wobei $|x_0| \leq b$. Dann gilt für alle x:*

$$x + 3bx_0 - (3b - xx_0) \cdot \sum_{j=1}^{p} \frac{t_j + 3bx_0}{3b - t_j x_0} \cdot u_j(x) = \left(\frac{3b}{x_0} + 3bx_0\right) \cdot \frac{T_p(x)}{T_p(3b/x_0)} \tag{3.91}$$

Beweis:
Sei $Q(x)$ ein Polynom vom Grade p, definiert durch:

$$Q(x) = x + 3bx_0 - (3b - xx_0) \cdot \sum_{j=1}^{p} \frac{t_j + 3bx_0}{3b - t_j x_0} \cdot u_j(x). \tag{3.92}$$

Setzt man jetzt die Definition der u_j ein und berücksichtigt noch, daß $u_j(t_k) = \delta_{jk}$ ist, so erhält man für alle $k = 1, \ldots, p$:

$$\begin{aligned} Q(t_k) &= t_k + 3bx_0 - (3b - t_k x_0) \cdot \sum_{j=1}^{p} \frac{t_j + 3bx_0}{3b - t_j x_0} \cdot u_j(t_k) && (3.93) \\ &= t_k + 3bx_0 - (3b - t_k x_0) \cdot \frac{t_k + 3bx_0}{3b - t_k x_0} && (3.94) \\ &= 0 && (3.95) \end{aligned}$$

Also erfüllt $Q(x)$ die folgenden Bedingungen:

$$\begin{aligned} Q(3b/x_0) &= \frac{3b}{x_0} + 3bx_0 && (3.96) \\ Q(t_1) &= 0 && (3.97) \\ &\vdots && (3.98) \\ Q(T_p) &= 0. && (3.99) \\ & && (3.100) \end{aligned}$$

Dies gilt aber auch für das folgende Polynom vom Grade p:

$$\left(\frac{3b}{x_0} + 3bx_0\right) \cdot \frac{T_p(x)}{T_p(3b/x_0)} \tag{3.101}$$

Da die beiden Polynome vom Grade p in $p+1$ Punkten übereinstimmen, sind sie identisch. Also gilt:

$$Q(x) = \left(\frac{3b}{x_0} + 3bx_0\right) \cdot \frac{T_p(x)}{T_p(3b/x_0)} \tag{3.102}$$

Berücksichtigt man die Definition von $Q(x)$, so ist dies gerade die Behauptung.

□

Lemma 3.4.10 *Sei $p \geq 2$ und $b > 0$ sowie x_0 reelle Zahlen wobei $|x_0| \leq b$. Dann gilt für alle $x \in [-1, 1]$:*

$$\left| \frac{x + 3bx_0}{3b - xx_0} - \sum_{j=1}^{p} \frac{t_j + 3bx_0}{3b - t_j x_0} \cdot u_j(x) \right| < \frac{3(1 + b^2)}{b \cdot 5^p} \tag{3.103}$$

Beweis:
Division der Aussage des vorherigen Lemmas durch $(3b - xx_0)$ ergibt in Beträgen:

$$\left|\frac{x + 3bx_0}{3b - xx_0} - \sum_{j=1}^{p} \frac{t_j + 3bx_0}{3b - t_j x_0} \cdot u_j(x)\right| = \frac{1}{|3b - xx_0|} \cdot \left|\frac{3b}{x_0} + 3bx_0\right| \cdot \frac{|T_p(x)|}{|T_p(3b/x_0)|} \tag{3.104}$$

Mit Lemma 3.4.6 ergibt sich für $|x_0| \leq b$ die folgende Abschätzung:

$$\begin{aligned} \left|\frac{3b/x_0 + 3bx_0}{T_p(3b/x_0)}\right| &< 3b \cdot (1/x_0 + x_0) \cdot 2 \cdot \left|\frac{3x_0}{5 \cdot 3b}\right|^p & (3.105) \\ &< \frac{6b}{5^p} \cdot (1/b + b) & (3.106) \end{aligned}$$

Lemma 3.4.5 liefert $|T_p(x)| \leq 1$ für $x \in [-1, 1]$. Dies zusammen mit (3.106) auf (3.104) angewendet liefert:

$$\begin{aligned} \left|\frac{x + 3bx_0}{3b - xx_0} - \sum_{j=1}^{p} \frac{t_j + 3bx_0}{3b - t_j x_0} \cdot u_j(x)\right| &< \frac{1}{2b} \cdot \frac{6b}{5^p} \cdot (1/b + b) & (3.107) \\ &= \frac{3(1 + b^2)}{b \cdot 5^p} & (3.108) \end{aligned}$$

für alle $x \in [-1, 1]$. □

Der folgende Satz beschreibt die Approximation von $1/\tan(\theta - \theta_0)$ für den Fall, daß θ_0 aus dem Intervall $[-a, a]$ kommt während θ „weit entfernt" liegt, genauer $|\theta| \geq 3a$.

Satz 3.4.11 *Sei $p \geq 2$ sowie a und θ_0 reelle Zahlen mit $0 < a \leq \pi/8$ sowie $|\theta_0| \leq a$. Dann gilt für alle θ mit $3a \leq |\theta| < \pi/2$:*

$$\left|\frac{1}{\tan(\theta - \theta_0)} - \sum_{j=1}^{p} \frac{t_j + 3 \tan a \tan \theta_0}{3 \tan a - t_j \tan \theta_0} \cdot u_j\left(\frac{3 \tan a}{\tan \theta}\right)\right| < \frac{3 \sec^2 a}{\tan a \cdot 5^p} \tag{3.109}$$

Beweis:
Sei $3a \leq |\theta| < \pi/2$. Die reellen Zahlen b, x und x_0 seien definiert durch $b = \tan a, x = 3 \tan a / \tan \theta$ und $x_0 = \tan \theta_0$. Da $|\theta_0| \leq a$ folgt $|x_0| = |\tan \theta_0| \leq \tan a = b$. Wegen Lemma 3.4.8 gilt ferner $|x| \leq 1$. Eine trigonometrische Identität für den Tangens liefert

$$\frac{1}{\tan(\theta - \theta_0)} = \frac{1 + \tan \theta \tan \theta_0}{\tan \theta - \tan \theta_0} = \frac{1 + 3bx_0/x}{3b/x - x_0} = \frac{x + 3bx_0}{3b - xx_0} \tag{3.110}$$

Die Definitionen für b, x und x_0 sowie Gleichung (3.110) in die Behauptung eingesetzt und mit Lemma 3.4.10 abgeschätzt ergibt:

$$\left|\frac{1}{\tan(\theta-\theta_0)} - \sum_{j=1}^{p} \frac{t_j + 3\tan a \tan\theta_0}{3\tan a - t_j \tan\theta_0} \cdot u_j\left(\frac{3\tan a}{\tan\theta}\right)\right| \tag{3.111}$$

$$= \left|\frac{x+3bx_0}{3b-xx_0} - \sum_{j=1}^{p} \frac{t_j+3bx_0}{3b-t_jx_0} \cdot u_j(x)\right| \tag{3.112}$$

$$< 3 \cdot \frac{1+b^2}{b \cdot 5^p} \tag{3.113}$$

$$= \frac{3\sec^2 a}{\tan a \cdot 5^p} \tag{3.114}$$

$$\tag{3.115}$$

□

Corollar 3.4.12 *Seien $p, N \geq 2$ und a reell mit $0 < a \leq \pi/8$. Die Funktion $f : [-\pi/2, \pi/2] \to \mathbf{C}$ sei definiert durch*

$$f(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta-\theta_k)} \tag{3.116}$$

wobei $\theta_k \in [-a, a]$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist.
$\Phi_1, \ldots, \Phi_p$ seien komplexe Zahlen, definiert durch

$$\Phi_j = f(\arctan(3\tan(a)/t_j)) \tag{3.117}$$

für $j = 1, \ldots, p$. Dann gilt mit $A = \sum_{k=1}^{N} |\alpha_k|$:

$$\left|f(\theta) - \sum_{j=1}^{p} \Phi_j \cdot u_j\left(\frac{3\tan a}{\tan\theta}\right)\right| < A \cdot \frac{3\sec^2 a}{\tan a \cdot 5^p} \tag{3.118}$$

für alle $|\theta| \geq 3a$.

Beweis:
Wenden wir Satz 3.4.11 für $\theta_0 = \theta_k$ an, so erhalten wir:

$$\left|\frac{1}{\tan(\theta-\theta_k)} - \sum_{j=1}^{p} \frac{t_j + 3\tan a \tan\theta_k}{3\tan a - t_j \tan\theta_k} \cdot u_j\left(\frac{3\tan a}{\tan\theta}\right)\right| < \frac{3\sec^2 a}{\tan a \cdot 5^p} \tag{3.119}$$

Mutliplikation mit α_k und anschließende Summation über k liefert

$$\left| f(\theta) - \sum_{k=1}^{N} \sum_{j=1}^{p} \frac{1 + 3\tan(a)/tj \tan\theta_k}{3\tan(a)/t_j - \tan\theta_k} \cdot u_j\left(\frac{3\tan a}{\tan\theta}\right)\right| < A \cdot \frac{3\sec^2 a}{\tan a \cdot 5^p} \tag{3.120}$$

Wenden wir jetzt noch Tangens-Identität (3.149) an, so ergibt sich

$$\left| f(\theta) - \sum_{j=1}^{p} \sum_{k=1}^{N} \frac{1}{\tan(\arctan(3\tan(a)/t_j) - \tan\theta_0)} \cdot u_j\left(\frac{3\tan a}{\tan\theta}\right)\right| < A \cdot \frac{3\sec^2 a}{\tan a \cdot 5^p} \tag{3.121}$$

Dies ist die Behauptung. □

Corollar 3.4.13 *Seien $p, N \geq 2$ und r reell mit $0 < r \leq \pi/8$. Die Funktion $\tilde{f} : [-\pi/2, \pi/2] \to \mathbf{C}$ sei definiert durch*

$$\tilde{f}(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta - x_k)} \tag{3.122}$$

wobei $x_k \in [x_0 - r, x_0 + r]$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. $\Phi_1, \ldots, \Phi_p$ seien komplexe Zahlen, definiert durch

$$\Phi_j = \tilde{f}(\arctan(3\tan(r)/t_j) + x_0) \tag{3.123}$$

für $j = 1, \ldots, p$. Dann gilt

$$\left| \tilde{f}(\theta) - \sum_{j=1}^{p} \Phi_j \cdot u_j\left(\frac{3\tan r}{\tan(\theta - x_0)}\right)\right| < A \cdot \frac{3\sec^2 r}{\tan r \cdot 5^p} \tag{3.124}$$

für alle θ mit $|\theta - x_0| \geq 3r$. Hierbei ist $A = \sum_{k=1}^{N} |\alpha_k|$.

Beweis:
Der Beweis ergibt sich unmittelbar aus dem vorherigen Corollar nach einigen Variablentransformationen: wir setzen in Corollar 3.4.12 $a = r$ und $\theta_k = x_k - x_0$. Damit ist $\theta_k \in [-r, r]$ und die in (3.116) definierte Funktion f wird zu

$$f(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta + x_0 - \theta_k)}. \tag{3.125}$$

Offensichtlich gilt

$$\tilde{f}(\theta + x_0) = f(\theta) \quad \text{bzw.} \quad \tilde{f}(\theta) = f(\theta - x_0). \tag{3.126}$$

Die Definition von Φ_j in (3.117) wir damit zu

$$\Phi_j = \tilde{f}(\arctan(3\tan(a)/t_j) + x_0) \tag{3.127}$$

für $j = 1, \ldots, p$ und nimmt damit die in den Voraussetzungen angegebene Form an. Anwendung von Corollar 3.4.12 liefert

$$\left| f(\theta) - \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan r}{\tan\theta} \right) \right| < A \cdot \frac{3\sec^2 r}{\tan r \cdot 5^p} \tag{3.128}$$

für $\theta \geq 3r$. Ersetzen wir jetzt noch θ durch $\theta - x_0$, so erhalten wir

$$\left| \tilde{f}(\theta) - \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan r}{\tan(\theta - x_0)} \right) \right| < A \cdot \frac{3\sec^2 r}{\tan r \cdot 5^p} \tag{3.129}$$

Das ist aber gerade die Behauptung. □

Die folgenden beiden Lemmata werden in Satz 3.4.16 benötigt.

Lemma 3.4.14 *Sei $p \geq 2$, $b > 0$ und x_0 eine reelle Zahl mit $|x_0| \geq 3b$. Dann gilt für alle x:*

$$1 + xx_0 - (x - x_0) \cdot \sum_{j=1}^{p} \frac{1 + bt_j x_0}{bt_j - x_0} \cdot u_j \left(\frac{x}{b} \right) = (1 + x_0^2) \cdot \frac{T_p(x/b)}{T_p(x_0/b)} \tag{3.130}$$

Beweis:
Sei $Q(x)$ ein Polynom vom Grade p, definiert durch:

$$Q(x) = 1 + xx_0 - (x - x_0) \cdot \sum_{j=1}^{p} \frac{1 + bt_j x_0}{bt_j - x_0} \cdot u_j \left(\frac{x}{b} \right). \tag{3.131}$$

Setzt man jetzt die Definition der u_j ein und berücksichtigt noch, daß $u_j(t_k) = \delta_{jk}$ ist, so erhält man für alle $k = 1, \ldots, p$:

$$\begin{aligned} Q(bt_k) &= 1 + bt_k x_0 - (bt_k - x_0) \cdot \sum_{j=1}^{p} \frac{1 + bt_j x_0}{bt_j - x_0} \cdot u_j(t_k) && (3.132) \\ &= 1 + bt_k x_0 - (bt_k - x_0) \cdot \frac{1 + bt_k x_0}{bt_k - x_0} && (3.133) \\ &= 0 && (3.134) \end{aligned}$$

Also erfüllt $Q(x)$ die folgenden Bedingungen:

$$\begin{aligned} Q(x_0) &= 1 + x_0^2 && (3.135) \\ Q(bt_1) &= 0 && (3.136) \\ &\vdots && (3.137) \\ Q(bt_p) &= 0. && (3.138) \\ & && (3.139) \end{aligned}$$

Dies gilt aber auch für das folgende Polynom vom Grade p:

$$(1 + x_0^2) \cdot \frac{T_p(x/b)}{T_p(x_0/b)} \tag{3.140}$$

Da die beiden Polynome vom Grade p in $p+1$ Punkten übereinstimmen, sind sie identisch. Also gilt:

$$Q(x) = (1 + x_0^2) \cdot \frac{T_p(x/b)}{T_p(x_0/b)} \tag{3.141}$$

Berücksichtigt man die Definition von $Q(x)$, so ist dies gerade die Behauptung. □

Im folgenden Lemma wird ein Approximation für $(1 + xx_0)/(x - x_0)$ ermittelt. Dieser Ausdruck ist wichtig, da eine einfache trigonometrische Identität für den Tangens besagt:

$$\frac{1}{\tan(\theta - \theta_k)} = \frac{1 + \tan\theta \tan\theta_k}{\tan\theta - \tan\theta_0} \tag{3.142}$$

Mit $x = \tan\theta$ und $x_0 = \tan\theta_k$ wird dies zu $(1 + xx_0)/(x - x_0)$.

Lemma 3.4.15 *Sei $p \geq 2$, $1/2 > b > 0$ und x_0 eine reelle Zahl mit $|x_0| \geq 3b$. Dann gilt für alle x:*

$$\left| \frac{1 + xx_0}{x - x_0} - \sum_{j=1}^{p} \frac{1 + bt_j x_0}{bt_j - x_0} \cdot u_j\left(\frac{x}{b}\right) \right| < \frac{1 + 9b^2}{b \cdot 5^p} \tag{3.143}$$

für jedes $x \in [-b, b]$.

Beweis:
Division der Aussage des vorherigen Lemmas durch $(x - x_0)$ ergibt in Beträgen:

$$\left| \frac{1 + xx_0}{x - x_0} - \sum_{j=1}^{p} \frac{1 + bt_j x_0}{bt_j - x0} \cdot u_j\left(\frac{x}{b}\right) \right| = \frac{1 + x_0^2}{|x - x_0|} \cdot \frac{|T_p(x/b)|}{|T_p(x_0/b)|} \tag{3.144}$$

Mit Lemma 3.4.6 ergibt sich also für $|x_0| \geq 3b$ die folgende Abschätzung:

$$\left|\frac{1+x_0^2}{T_p(x_0/b)}\right| < (1+x_0^2)\cdot 2\cdot\left|\frac{3b}{5x_0}\right|^p \tag{3.145}$$

$$< \frac{1+9b^2}{b\cdot 5^p} \tag{3.146}$$

Lemma 3.4.5 liefert $|T_p(x/b)| \leq 1$ für $x \in [-b,b]$. Dies zusammen mit (3.146) auf (3.144) angewendet liefert:

$$\left|\frac{1+xx_0}{x-x_0} - \sum_{j=1}^{p}\frac{1+bt_jx_0}{bt_j-x_0}\cdot u_j\left(\frac{x}{b}\right)\right| < \frac{1+9b^2}{b\cdot 5^p} \tag{3.147}$$

für alle $x \in [-b,b]$. $\square$

Der folgene Satz ähnelt Satz 3.4.11. Allerdings sind hier die Annahmen über θ und θ_0 vertauscht: θ kommt aus dem kleinen Intervall $[-a,a]$ während θ_0 weit entfernt liegt, also $|\theta_0| \geq 3a$.
Die Aussage ist deshalb nicht identisch zu der von Satz 3.4.11, da natürlich weiterhin θ die unabhängige Funktionsvariable darstellt, während θ_0 ein fester Koeffizient ist.

Satz 3.4.16 *Sei $p \geq 2$, a reelle mit $0 < a \leq \pi/8$ und θ_0 reell mit $3a \leq |\theta_0| < \pi/2$. Dann gilt:*

$$\left|\frac{1}{\tan(\theta-\theta_0)} - \sum_{j=1}^{p}\frac{1+t_j\tan a\tan\theta_0}{t_j\tan a-\tan\theta_0}\cdot u_j\left(\frac{\tan\theta}{\tan a}\right)\right| < \frac{1+9\tan^2 a}{\tan a\cdot 5^p} \tag{3.148}$$

für alle $\theta \in [-a,a]$.

Beweis:
Sei $\theta \in [-a,a]$. Die reellen Zahlen b, x und x_0 seien definiert durch $b = \tan a$, $x = \tan\theta$ und $x_0 = \tan\theta_0$. Da $|\theta| \leq a$ folgt $|x| = |\tan\theta| \leq \tan a = b$. Wegen Lemma 3.4.8 gilt ferner $|x_0| \geq \tan 3a \geq 3b$. Für den Tangens gilt die folgende trigonometrische Identität:

$$\frac{1}{\tan(\alpha-\beta)} = \frac{1+\tan\alpha\tan\beta}{\tan\alpha-\tan\beta} \tag{3.149}$$

Somit hat man

$$\frac{1}{\tan(\theta-\theta_0)} = \frac{1+\tan\theta\tan\theta_0}{\tan\theta-\tan\theta_0} = \frac{1+xx_0}{x-x_0} \tag{3.150}$$

Die Definitionen für b, x und x_0 sowie Gleichung (3.150) in die Behauptung eingesetzt und mit Lemma 3.4.15 abgeschätzt ergibt:

$$\left|\frac{1}{\tan(\theta-\theta_0)} - \sum_{j=1}^{p} \frac{1+t_j \tan a \tan\theta_0}{t_j \tan a - \tan\theta_0} \cdot u_j\left(\frac{\tan\theta}{\tan a}\right)\right| \tag{3.151}$$

$$= \left|\frac{1+xx_0}{x-x_0} - \sum_{j=1}^{p} \frac{1+t_j b x_0}{t_j b - x_0} \cdot u_j\left(\frac{x}{b}\right)\right| \tag{3.152}$$

$$< \frac{1+9b^2}{b\cdot 5^p} \tag{3.153}$$

$$= \frac{1+9\tan^2 a}{\tan a \cdot 5^p} \tag{3.154}$$

für $\theta \in [-a, a]$. □

Corollar 3.4.17 *Seien $p, N \geq 2$ und a reell mit $0 < a \leq \pi/8$ Die Funktion $f : [-\pi/2, \pi/2] \to \mathbf{C}$ sei definiert durch*

$$f(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta-\theta_k)} \tag{3.155}$$

wobei $|\theta_k| \geq 3a$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. Ψ_j sei definiert durch

$$\Psi_j = f(\arctan(t_j \tan a)) \tag{3.156}$$

für $j = 1, \ldots, p$. Dann gilt:

$$\left|f(\theta) - \sum_{j=1}^{p} \Psi_j \cdot u_j\left(\frac{\tan\theta}{\tan a}\right)\right| < A\frac{1+9\tan^2 a}{\tan a\cdot 5^p} \tag{3.157}$$

Hierbei ist $A = \sum_{k=1}^{N} |\alpha_k|$.

Beweis:
Wenden wir Satz 3.4.16 für $\theta_0 = \theta_k$ an, so erhalten wir:

$$\left|\frac{1}{\tan(\theta-\theta_k)} - \sum_{j=1}^{p} \frac{1+t_j \tan a \tan\theta_0}{t_j \tan a - \tan\theta_0} \cdot u_j\left(\frac{\tan\theta}{\tan a}\right)\right| < \frac{1+9\tan^2 a}{\tan a\cdot 5^p} \tag{3.158}$$

Mutliplizieren wir dies mit α_k und summieren dann über k, so erhalten wir

$$\left| f(\theta) - \sum_{k=1}^{N} \sum_{j=1}^{p} \frac{1 + t_j \tan a \tan \theta_0}{t_j \tan a - \tan \theta_0} \cdot u_j \left(\frac{\tan \theta}{\tan a} \right) \right| < A \cdot \frac{1 + 9 \tan^2 a}{\tan a \cdot 5^p} \tag{3.159}$$

Wenden wir jetzt noch Tangens-Identität (3.149) an, so ergibt sich

$$\left| f(\theta) - \sum_{j=1}^{p} \sum_{k=1}^{N} \frac{1}{\tan(\arctan(t_j \tan a) - \tan \theta_0)} \cdot u_j \left(\frac{\tan \theta}{\tan a} \right) \right| < A \cdot \frac{1 + 9 \tan^2 a}{\tan a \cdot 5^p} \tag{3.160}$$

Dies ist die Behauptung. □

Corollar 3.4.18 *Seien $p, N \geq 2$. Sei r reell mit $0 < r \leq \pi/8$ Die Funktion $\tilde{f} : [-\pi/2, \pi/2] \to \mathbf{C}$ sei definiert durch*

$$\tilde{f}(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta - x_k)} \tag{3.161}$$

wobei $x_k, y_0 \in [-\pi/2, \pi/2]$, $|x_k - y_0| \geq 3r$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. Ψ_j sei definiert durch

$$\Psi_j = \tilde{f}(\arctan(t_j \tan a) + y_0) \tag{3.162}$$

für $j = 1, \ldots, p$. Dann gilt mit $A = \sum_{k=1}^{N} |\alpha_k|$:

$$\left| \tilde{f}(\theta) - \sum_{j=1}^{p} \Psi_j \cdot u_j \left(\frac{\tan(\theta - y_0)}{\tan r} \right) \right| < A \frac{1 + 9 \tan^2 r}{\tan r \cdot 5^p} \tag{3.163}$$

für $\theta \in [y_0 - r, y_0 + r]$.

Beweis:
Die Aussage ergibt sich durch einfache Variablentransformationen unmittelbar aus dem vorherigen Corollar, der Beweis verläuft hierbei analog wie der zu Corollar 3.4.13. □

3.4.3 Rekursive Berechnung der FMM

Veranschaulichen wir uns die Aussagen von Corollar 3.4.18 einmal anhand eines Beispiels. Sei $\tilde{f} : [-\pi/2, \pi/2] \to \mathbf{C}$ definiert durch

$$f(y) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(y - x_k)} \tag{3.164}$$

wobei $x_k, y_0 \in [-\pi/2, \pi/2]$ und $|x_k - y_0| \geq 3r$ für $k = 1, \ldots, N$ ist. Wir wollen f im Intervall $[y_0 - r, y_0 + r]$ schnell auswerten, also $f(y_1), \ldots, f(y_N)$ berechnen für $y_k \in [y_0 - r, y_0 + r]$.

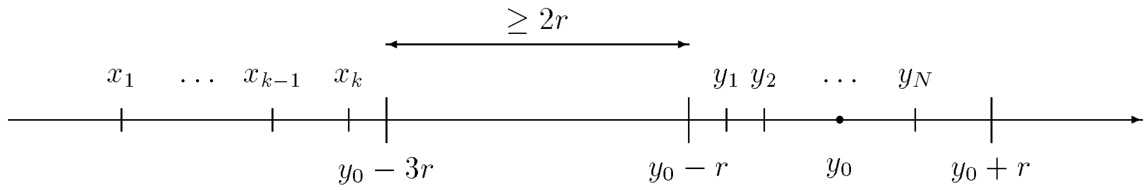

Abbildung 3.1: Interpolation für $y_k \in [y_0 - r, y_0 + r]$

Abbildung 3.1 veranschaulicht dies. Man erkennt, daß die Punkte y_j aus einem Intervall der Breite $2r$ stammen und dieses Intervall von den Knoten x_l einen Abstand von mindestens ebenfalls $2r$ besitzen muß, um den Voraussetzungen von Corollar 3.4.18 zu genügen. Dann kann man mit Hilfe des Corollars $f(y_l)$ für schnell berechnen.
In diesem Beispiel haben wir spezielle Annahmen über die Verteilung der Stützstellen x_k gemacht: sie müssten einen Mindestabstand von $3r$ zu y_0 haben. Der allgemeine Fall funktioniert aber ähnlich: wir teilen f in zwei Summen auf. Die eine enthält die Summanden, für die die x_k nahe bei y_0 liegen; diese Summe wird direkt berechnet. Die Summe über die restlichen Summanden genügt den Voraussetzungen von Corollar 3.4.18 und kann somit schnell berechnet werden.
Zu Beachten ist, das die Aufteilung von f natürlich vom jeweiligen Argument y_l abhängt. Dies berücksichtigen wir wie folgt:
Wir teilen den Definitionsbereich von f in n_i Subintervalle der Länge $2r$ auf. I_i bezeichne hierbei das i-te Subintervall. y_{0i} sei die Intervallmitte des i-ten Intervalls. Wir wollen jetzt

$$f(y) = \sum_{k=1}^{p} \frac{\alpha_k}{\tan(y - x_k)} \tag{3.165}$$

für ein $y = y_k$ aus dem Intervall I_{i_0} mit Hilfe von Corollar 3.4.18 auswerten; y_0 bezeichne deshalb die Intervallmitte des Intervalls I_{i_0}.
Wir haben gesehen, daß das Corollar nur für Knoten x_k in Intervallen I_i greift, die von I_{i_0} durch mindestens ein weiteres Intervall getrennt sind. Wir nennen diese Intervalle die von I_{i_0} vollständig getrennten (*englisch: well seperated*) Intervalle. Abbildung 3.2 veranschaulicht die Aufteilung in Intervalle: die von I_{i_0} vollständig getrennten Intervalle sind schraffiert markiert.

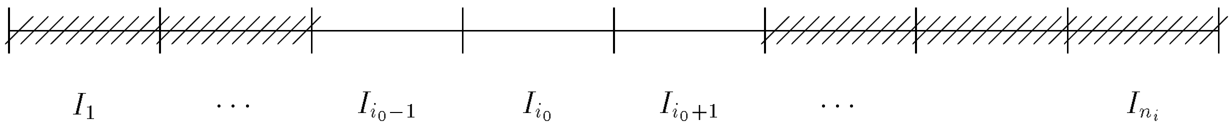

Abbildung 3.2: vollständig getrennte Intervalle

Zwei Intervalle I_i und I_{i_0} sind also dann vollständig getrennt, wenn gilt:

$$|i - i_0| > 1. \tag{3.166}$$

Nicht vollständig getrennt von I_{i_0} sind also die drei Intervalle I_{i_0-1}, I_{i_0+1} und natürlich I_{i_0} selbst. Da die Funktion f π-periodisch ist, sind die Indizes modulo n_i zu nehmen, das heißt konkret, daß das erste Intervall nicht vollständig getrennt vom letzten ist und umgekehrt.
Damit kann man jetzt die Menge der von einem Intervall vollständig getrennten Stützstellen definieren:

$$WS_i = \{k : x_k \text{ liegt in einem von } I_i \text{ vollständig getrennten Intervall}\} \tag{3.167}$$

Jetzt definieren wir noch f^i, eine Einschränkung von f auf das Intervall I_i:

$$f^i(y) = \sum_{k \in I_i} \frac{\alpha_k}{\tan(y - x_k)} \tag{3.168}$$

Die Summe läuft also lediglich über diejenigen Summanden k, für die x_k in I_i liegt. Insgesamt ist also

$$f(y) = \sum_{i=1}^{n_i} f^i(y) \tag{3.169}$$

Mit dieser Terminologie kann man f auch wie folgt aufteilen: f^{WS} sei die Summe aller derjenigen f^i für die i ein von i_0 vollständig getrenntes Intervall darstellt und $f^P := f^{i-1} + f^i + f^{i+1}$ sei die Summe über die nicht vollständig getrennten Intervalle. Insgesamt ergibt sich also für ein y aus dem Intervall i_0:

$$f(y) = f^{WS}(y) + f^P(y) \tag{3.170}$$

WS steht hierbei für *well separated* und P für *Pol*.
Zu beachten ist, daß WS_i und damit auch f^{WS} und f^P davon abhängen, in welchem Intervall das Argument y liegt.
Für jedes Intervall I_i können wir jetzt Corollar 3.4.18 anwenden, mit $\tilde{f} = f^{WS_i}$. Man erhält dann n_i Vektoren Ψ^i:

$$\Psi^i_j = f^{WS_i}(\arctan(t_j \tan a) + y_0) \tag{3.171}$$

für $j = 1, \ldots, p$; y_0 ist die Intervallmitte des Intervalls I_i.
Wenn wir jetzt f für ein y_l aus dem Intervall I_i auswerten wollen, so ergibt sich dies mit Corollar 3.4.18 und (3.170) wie folgt:

$$\begin{aligned} f(y_l) &= f^P(y_l) + f^{WS}(y_l) && (3.172)\\ &\approx f^P(y_l) + \sum_{k=1}^{p} \Psi^i_k \cdot u_j\left(\frac{\tan(y_l - y_0)}{\tan r}\right). && (3.173) \end{aligned}$$

Dies kann man schnell berechnen, da einerseits f^P nur wenige Summanden enthält (sofern n_i ausreichend groß ist, man also ausreichend viele Intervalle bemüht) und andererseits die Tschebyscheff-Interpolation nur von geringer Ordnung (typischerweise 16) sein muß, um eine ausreichende Genauigkeit (d.h. eine Genauigkeit in der Größenordnung der Maschinengenauigkeit) zu erreichen. Der nur von den Stützstellen abhängende Ausdruck $u_j(\tan(y_l - y_0)/\tan r)$ wird natürlich vorberechnet.
Auf diesem Wege werden wir unser f auswerten. Jetzt stehen wir allerdings vor dem Problem, die n_i Vektoren Ψ^i schnell zu berechenen. Diese hängen bedauerlicherweise massiv von den Koeffizienten α_k ab, können also nicht vorberechnet werden. Weiterhin besteht die Summe f^{WS}, die für jedes Ψ^i wegen (3.171) p-mal berechnet werden muß, aus beinahe ebensovielen Summanden wie f selbst, sodaß man f^{WS} nicht schnell direkt berechnen kann.
Wir müssen also jetzt nach anderen Möglichkeiten suchen, die Ψ^i zu ermitteln.

Abbildung 3.3 verdeutlicht die Bedeutung von Ψ. Der Bereich mit den Knoten x_k, aus denen die Summe f^{WS_i} gebildet wird, ist schraffiert dargestellt und das Intervall, in dem f^{WS_i} mit Hilfe von Ψ^i approximiert werden kann, ist dick markiert. Dieses Intervall stellt also quasi den „Definitionsbereich" von Ψ^i dar; Ψ^i dient dazu, f^{WS_i} im Intervall I_i zu approximieren. Aus diesem Grund nennen wir Ψ^i im folgenden die *lokale Erweiterung* für das Intervall I_i.

Genau andersherum verhält es sich mit dem Vektor Φ, der in Corollar 3.4.11 eingeführt wurde. Für ein Intervall i definieren wir Φ^i durch:

$$\Phi_j^i = f^i(\arctan(3\tan(r)/t_j) + x_0) \tag{3.174}$$

für $j = 1, \ldots, p$. x_0 bezeichne die Intervallmitte des Intervalls i.
Dies wird in Abbildung 3.4 skizziert: Die Knoten x_k der Summe f^i stammen jetzt nur noch aus dem einen schraffiert dargestllten Intervall i selbst. Deshalb kann Φ^i schnell berechnet werden. Mit Hilfe von Corollar 3.4.13 kann f^i dann auf dem gesamten dick markierten Bereich ausgewertet werden. Wir nennen Φ^i die *far-field Erweiterung* von f für das Intervall i. Der Terminus „far-field" ist deshalb entstanden, weil die schnelle Multipolentwicklung zuerst (z.B. in [6]) bei der Berechnung elektrostatischer Potentiale angewendet wurde. Diese werden durch Gleichungen der Form

$$\varphi(y) = \sum_{k=1}^{N} q_k \cdot \frac{1}{|y - x_k|} \tag{3.175}$$

beschrieben, wenn man von N Punktladungen in den Punkten x_k mit der jeweiligen Ladung q_k ausgeht. (Vergleiche z.B. [9].) Die hier auftretende Kernel-Funktion $1/|y - x_k|$ verhält sich bezüglich der Tschebyscheff-Approximation fast genauso wie der von uns betrachtete Kernel $1/\tan(y - x_k)$. Deshalb kann man die für dieses Problem entwickelten Algorithmen auch auf unseren Fall übertragen.
Zurück zu unserem Problem: wir haben gesehen, daß wir für jedes Intervall I_i die lokale Erweiterung Ψ^i berechnen müssen, um die Funktion f schnell auswerten zu können. Leider kann man Ψ_i nicht vorberechnen, da es von den α_k abhängt und man kann es auch nicht schnell direkt berechnen, da es „lange" Summen, nämlich Auswertungen von f^{WS} enthält. Was hingegen schnell berechnet werden kann, sind die far-field Erweiterungen Φ^i. Hier läuft die Summe jeweils nur über die Knoten x_k aus einem Intervall I_i. Ein erster Ansatz wäre es also, zu versuchen, die Ψ^i aus den Φ^i zu berechnen. Es ist plausibel, daß man für die lokale Erweiterung Ψ^i alle Knoten im in Abbildung 3.3 schraffiert dargestellten Bereich berücksichtigen muß. Da dies $n_i - 3$ Intervalle sind, muß man die far-field Erweiterungen

Φ^i dieser Intervalle, die ja gerade auf diesen Knoten aufbauen, nehmen und geeignet aufaddieren.
Dies ist tatsächlich möglich, im folgenden Abschnitt wird dazu ein Satz hergeleitet, mit dem man die lokale Erweiterung Ψ bezüglich eines Intervalls aus den far-field Erweiterungen anderer Intervalle gewinnen kann.
Dies bewirkt aber leider immer noch keine ausreichende Beschleunigung, da jetzt für jedes Intervall $n_i - 3$ far-field Erweiterungen transformiert werden müssen. Dies dauert immer noch zu lange.
Spätestens seit dem Artikel von Cooley und Tukey 1965 [1] weiß man aber, was man in solchen Fällen zu tun hat: „divide and conquer", oder um es mit den Worten des Informatikers L. Peter Deutsch zu sagen: „*to iterate is human, to recurse divine*".
Dazu führen wir jetzt *nlevs* Stufen ein und unterteilen den Definitionsbereich $[-\pi/2, \pi/2]$ der Funktion f auf der Stufe l in 2^l Intervalle. Für jede Stufe kann man jetzt die Einschränkungen der Funktion f auf einzelne Intervalle $I_{l,i}$ definieren. Diese nennen wir im folgenden $f^{l,i}$. Die Summe läuft hierbei natürlich über alle Knoten x_k, die im Intervall i auf Stufe l liegen. Analog werden $f^{WS_{l,i}}$ sowie die beiden Erweiterungen $\Phi^{l,i}$ und $\Psi^{l,i}$ definiert.
Die Aufteilung in Stufen wird in Abbildung 3.5 dargestellt. Man beachte, daß wie bei Graphendarstellungen üblich, die „größte" oder „höchste" Stufe (im Bild $l = 4$) am unteren Rand dargestellt ist, die Abbildung als gleichsam auf dem Kopf steht.
Insgesamt ist es unser Ziel, die lokalen Erweiterungen Ψ auf der höchsten Stufe berechnen; dort ist das Intervall in die meisten Subintervalle aufgeteilt. Entsprechend wenige Knoten x_k sind also nicht von einem festen Intervall vollständig getrennt und müssen somit direkt berechnet werden, da sie nicht von dem jeweiligen Ψ erfaßt werden. Wenn man also die lokalen Erweiterungen Ψ auf der höchsten Stufe berechnet hat, kann man damit schnell die Funktion f auswerten.
Wir versuchen jetzt, die Ψ auf Stufe l durch die Ψ der darunterliegenden Stufe $l-1$ zu berechnen. Abbildung 3.6 zeigt die $\Psi^{l,i}$ auf zwei Stufen $l-1$ und l.
Man erkennt, daß der Definitionsbereich von $\Psi^{l,2i-1}$ eine Teilmenge des Definitionsbereichs von $\Psi^{l-1,i}$ ist. Deshalb kann man aus einem Ψ einer niedrigeren Stufe im Wesentlichen das Ψ auf einer höheren Stufe errechnen. Wie die schraffierten Intervalle zeigen, haben wir damit aber noch nicht alle Knoten x_k erfaßt: die Anteile die durch Knoten in den Intervallen $I_{l,2i-3}$, $I_{l,2i+1}$ und $I_{l,2i+2}$ erzeugt werden, müssen noch hinzuaddiert werden. Weiter oben wurde allerdings schon bemerkt, daß man die far-field Erweiterung eines Intervalls in die lokale Erweiterung eines anderen Intervalls transformieren kann. Wenn wir also die far-field Erweiterungen der Intervalle $I_{l,2i-3}$, $I_{l,2i+1}$ und $I_{l,2i+2}$ kennen, so können wir daraus die entsprechenden lokalen Erweiterungen für das Intervall $I_{l,2i-1}$ berechnen

und zu unserem aus der Transformation von $\Psi^{l-1,i}$ gewonnenen Vektor hinzuaddieren. Das Resultat ist dann eine Approximation für $\Psi^{l,2i-1}$.
Man kann also die Ψ auf einer Stufe berechnen, sofern man die Ψ auf der darunterliegenden Stufe und die Φ der aktuellen Stufe kennt. Etwas formaler formuliert:

$$\Psi^{l,2i-1} = S_L\Psi^{l-1,i} + T_2\Phi^{l,2i-3} + T_3\Phi^{l,2i+1} + T_4\Phi^{l,2i+2} \tag{3.176}$$

mit geeigneten Transformationen S_L, T_1, T_3 und T_4. Dies gilt für $\Psi^{l,k}$ in ungeraden Intervallen ($k = 2i-1$), in geraden Intervallen $k = 2i$ erhält man Ähnliches, nur mit leicht geänderten Indizes. Dort werden dann die Transformationen S_R (R für rechts), T_1, T_2 und T_3 auftauchen:

$$\Psi^{l,2i} = S_R\Psi^{l-1,i} + T_1\Phi^{l,2i-3} + T_2\Phi^{l,2i-2} + T_3\Phi^{l,2i+2} \tag{3.177}$$

Der Rekursionsanfang ist ziemlich einfach: wie in Abbildung 3.5 ersichtlich, ist auf Stufe 1 kein Intervall von den anderen vollständig getrennt. Deshalb sind $\Psi^{1,1}$ und $\Psi^{1,2}$ jeweils gleich Null.
Jetzt müssen wir allerdings noch einen Weg finden, die far-field Erweiterungen Φ auf einer Stufe schnell zu berechnen. Hierbei gehen wir genau umgekehrt vor: auf der höchsten Stufe *nlevs* kann man die $\Phi^{nlevs,i}$ schnell berechnen, da dort die Summen $f^{nlevs,i}$ nur über wenige Summanden laufen. Wir wollen also die Φ einer Stufe durch die einer höheren Stufe beschreiben. Abbildung 3.7 verdeutlicht dies.
Die Abbildung legt nahe, daß man $\Phi^{l,i}$ aus den zwei Erweiterungen der nächsthöherer Stufe, nämlich aus $\Phi^{l+1,2i-1}$ sowie $\Phi^{l+1,2i}$ gewinnen kann. Gesucht sind also Transformationen M_L und M_R, sodaß

$$\Phi^{l,i} = M_L\Phi^{l+1,2i-1} + M_R\Phi^{l+1,2i} \tag{3.178}$$

Verbleiben noch die Φ auf der höchsten Stufe. Dort berechnen wir sie direkt aus der Definition: Für $j = 1,\ldots,p$ ist $\Phi_j^{nlevs,i}$ definiert durch

$$\begin{aligned}\Phi_j^{nlevs,i} &= f^{nlevs,i}(\arctan(3\tan(r)/t_j) + x_0) &(3.179)\\ &= \sum_k \alpha_k \frac{1}{\tan(\arctan(3\tan(r)/t_j) + x_0 - x_k)}, &(3.180)\end{aligned}$$

wobei die Summe über alle k läuft, für die x_k in $I_{nlevs,i}$ liegt. Nimmt man an, daß die Stützstellen im wesentlichen gleichverteilt sind, in jedem Intervall also gleichviele x_k liegen, so besteht die Summe aus $N/2^l$ Summanden. Wenn man den Tangensausruck, der nur von den Stützstellen abhängt, vorberechnet, so erfordert die Berechnung der 2^l $\Phi^{nlevs,i}$

lediglich Np Rechenoperationen, denn jeder Vektor $\Phi^{nlevs,i}$ besteht aus p Komponenten. Da p vergleichsweise klein ist (typischerweise 16), kann man dies schnell berechnen.
Damit ist die grundlegende Vorgehensweise geklärt. Allerdings benötigen wir jetzt noch drei Transformationen:

1. Berechne die far-field Erweiterung Φ einer Stufe mit Hilfe der far-field Erweiterung der nächsthöheren Stufe (Operatoren M_L und M_R).
2. Berechne die lokale Erweiterung Ψ mit Hilfe der lokalen Erweiterung einer niedrigeren Stufe (Operatoren S_L und S_R).
3. Transformiere die far-field Erweiterung Φ eines Intervalls in eine lokale Erweiterung Ψ eines anderen Intervalls innerhalb einer Stufe (Operatoren T_1, T_2, T_3 und T_4).

Genau zu diesem Zweck werden im nächsten Abschnitt drei Sätze hergeleitet.

3.4.4 Transformationen für die Erweiterungen Φ und Ψ

Der erste Satz dieses Abschnittes ermöglicht es, eine lokale ErweiterungΨ aus einer far-field Erweiterung Φ zu gewinnen.

Satz 3.4.19 *Seien $p, N \geq 2$ und a, c, d reelle Zahlen mit $0 < a \leq \pi/8$ und $|c| - d > 3a$. Sei die Funktion $f : [-\pi/2, \pi/2] \to \mathbf{C}$ definiert durch*

$$f(\theta) = \sum_{k=1}^{p} \frac{\alpha_k}{\tan(\theta - \theta_k)} \tag{3.181}$$

wobei $\theta_k \in [-a, a]$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. $\Phi_1, \ldots, \Phi_p$ seien komplexe Zahlen, definiert durch

$$\Phi_j = f(\arctan(3\tan(a)/t_j)) \tag{3.182}$$

für $k = 1, \ldots, p$. Schließlich sei $\Psi_1, \ldots, \Psi_p$ ein komplexer Vektor, definiert durch

$$\Psi_k = \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan a}{\tan(c + \arctan(t_k \tan d))} \right) \tag{3.183}$$

für $k = 1, \ldots, p$. Dann gilt für alle $\theta \in [c - d, c + d]$:

$$\left| f(\theta) - \sum_{k=1}^{p} \Psi_k \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| < A \cdot \frac{6p\sec^2 a + 3}{\tan a \cdot 5^p} \tag{3.184}$$

Hierbei ist $A = \sum_{k=1}^{N} |\alpha_k|$.

Beweis:
Fügt man in (3.184) eine „günstige 0" ein und wendet die Dreiecksungleichung an, so ergibt sich:

$$\left| f(\theta) - \sum_{k=1}^{p} \Psi_k \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| \leq S_1 + S_2 \tag{3.185}$$

mit

$$S_1 = \left| f(\theta) - \sum_{k=1}^{p} f(c + \arctan(t_k \tan d)) \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| \tag{3.186}$$

und

$$S_2 = \left| \sum_{k=1}^{p} (f(c + \arctan(t_k \tan d)) - \Psi_k) \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| \tag{3.187}$$

Mit Corollar 3.4.17 und der Dreiecksungleichung lässt sich S_1 abschätzen:

$$S_1 < A \cdot \frac{1 + 9\tan^2 a}{\tan a \cdot 5^p} \tag{3.188}$$

Wendet man Corollar 3.4.12 auf S_2 an, so erhält man zusammen mit Lemma 3.4.7 zur Abschätzung von u_j:

$$\begin{aligned} S_2 &\leq \sum_{k=1}^{p} |f(c + \arctan(t_k \tan d)) - \Psi_k| \cdot \left(2 - \frac{1}{p}\right) && (3.189) \\ &= \sum_{k=1}^{p} \left| f(c + \arctan(t_k \tan d)) - \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan a}{\tan(c + \arctan(t_k \tan d))} \right) \right| \cdot \left(2 - \frac{1}{p}\right) \\ &< A \cdot \frac{6p \cdot \sec^2 a}{\tan a \cdot 5^p} && (3.190) \end{aligned}$$

Setzen wir diese Abschätzungen für S_1 und S_2 in (3.185) ein, so erhalten wir:

$$\left| f(\theta) - \sum_{k=1}^{p} \Psi_k \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| < A \cdot \frac{6p\sec^2 a + 3}{\tan a \cdot 5^p} \tag{3.191}$$

$\square$

Das folgende Corollar entsteht aus Satz 3.4.19 durch einfache Variablentransformationen.

Corollar 3.4.20 *Seien $p, N \geq 2$ und seien r, x_0 und y_0 reelle Zahlen mit $0 < r \leq \pi/8$ bzw. $|x_0 - y_0| > 4r$. Die Funktion $\tilde{f} : [-\pi/2, \pi/2] \to \mathbf{C}$ sei definiert durch*

$$\tilde{f}(y) = \sum_{k=1}^{p} \frac{\alpha_k}{\tan(y - x_k)} \tag{3.192}$$

wobei $x_k \in [x_0 - r, x_0 + r]$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. $\Phi_1, \ldots, \Phi_p$ seien komplexe Zahlen, definiert durch

$$\Phi_j = \tilde{f}(\arctan(3\tan(a)/t_j) + x_0). \tag{3.193}$$

Schließlich sei $\Psi_1, \ldots, \Psi_p$ ein komplexer Vektor, definiert durch

$$\Psi_k = \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan r}{\tan(y_0 - x_0 + \arctan(t_k \tan r))} \right). \tag{3.194}$$

Dann gilt für alle $y \in [y_0 - r, y_0 + r]$:

$$\left| \tilde{f}(y) - \sum_{k=1}^{p} \Psi_k \cdot u_j \left(\frac{\tan(y - y_0)}{\tan r} \right) \right| < A \cdot \frac{6p\sec^2 r + 3}{\tan r \cdot 5^p} \tag{3.195}$$

Hierbei ist $A = \sum_{k=1}^{N} |\alpha_k|$. Außerdem gilt

$$\left| \Psi_k - \tilde{f}(y_0 + \arctan(t_k \tan r)) \right| < A \cdot \frac{1 + 9\tan^2 r}{\tan r \cdot 5^p} \tag{3.196}$$

Beweis:
Wir wenden Satz 3.4.19 an und setzen $a = d = r$,$\theta_k = x_k - x_0$ und $c = y_0 - x_0$. $|c| - d > 3a$ wird somit zu $|x_0 - y_0| > 4r$ und $\theta_k \in [-a, a]$ zu $x_k \in [x_0 - r, x_0 + r]$.
(3.181) wird zu

$$f(\theta) = \sum_{k=1}^{p} \frac{\alpha_k}{\tan(\theta + x_0 - x_k)} \tag{3.197}$$

Offensichtlich gilt

$$f(\theta - x_0) = \tilde{f}(\theta), \tag{3.198}$$

weshalb (3.182) dann wie folgt lautet:

$$\Phi_k = \tilde{f}(\arctan(3\tan a)/t_k) + x_0). \tag{3.199}$$

Auch die Definition von Ψ in (3.183) geht in die in (3.193) angegebene Form über. In (3.184) setzen wir $\theta = y - x_0$. Die Forderung $\theta \in [c-d, c+d]$ wird dann zu $y \in [y_0 - r, y_0 + r]$ und unter Berücksichtigung von $\tilde{f}(\theta) = f(\theta - x_0)$ wird (3.184) zur ersten Behauptung des Corollars. Die Behauptung über Ψ ergibt sich sofort, wenn man die Definition von Ψ mit (3.124) in Corollar 3.4.13 vergleicht. $\square$

Dieser Corollar ermöglicht es durch (3.194), eine far-field Erweiterung Φ in eine lokale Erweiterung Ψ zu transformieren. Sei r der Intervallradius auf Stufe l. Wenn wir jetzt die far-field Erweiterung $\Phi^{l,i-3}$ des Intervalls $I_{l,i-3}$ in die lokale Erweiterung $\Psi^{l,i}$ umwandeln wollen, so geschieht dies durch Anwendung von Corollar 3.4.20 mit $c = y_0 - x_0 = 6r$, da beide Intervalle einen Abstand vom dreifachen der Intervallbreite, also $6r$, haben. (3.194) wird dann zu:

$$\Psi_k^{l,i} = \sum_{j=1}^{p} \Phi_j^{l,i-3} \cdot u_j \left(\frac{3 \tan r}{\tan(6r + \arctan(t_k \tan r))} \right). \tag{3.200}$$

Definiert man jetzt die $p \times p$-Matrix T_1 durch:

$$T_1(k,j) = u_j \left(\frac{3 \tan r}{\tan(6r + \arctan(t_k \tan r))} \right), \tag{3.201}$$

so schreibt sich dies kürzer als:

$$\Psi^{l,i} = T_1 \cdot \Phi^{l,i-3}. \tag{3.202}$$

Analog definiert man noch die Matrizen $T2$, $T3$ und $T4$ durch Anwendung von Corollar 3.4.20 mit $c = 4r$, $c = -4r$ und $c = -6r$:

$$\begin{aligned}
T_2(k,j) &= u_j \left(\frac{3 \tan r}{\tan(4r + \arctan(t_k \tan r))} \right) && (3.203)\\
T_3(k,j) &= u_j \left(\frac{3 \tan r}{\tan(-4r + \arctan(t_k \tan r))} \right) && (3.204)\\
T_4(k,j) &= u_j \left(\frac{3 \tan r}{\tan(-6r + \arctan(t_k \tan r))} \right). && (3.205)
\end{aligned}$$

Diese Matrizen hängen von r ab; man benötigt also jeweils $nlevs - 1$ Stück für die Stufen $2, \ldots, nlevs$; auf Stufe 1 gibt es nur zwei Intervalle, die nicht vollständig voneinander getrennt sind. Damit hat man die gesuchten Transformationen T_i gefunden.

Der folgende Satz ermöglicht es, far-field Erweiterungen auf unterschiedlichen Stufen zu transformieren.

Satz 3.4.21 *Seien $p, N \geq 2$ und a, c, d reelle Zahlen sodaß $0 < a \leq \pi/8$und $[c-d, c+d] \supset [-a, a]$. Sei die Funktion $f : [-\pi/2, \pi/2] \to \mathbf{C}$ definiert durch die Formel*

$$f(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta - \theta_k)}, \tag{3.206}$$

wobei $\theta_k \in [-a, a]$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ eine Menge komplexer Zahlen ist. Weiterhin sei $\Phi_1, \ldots, \Phi_p$ eine Menge komplexer Zahlen, definiert durch

$$\Phi_k = f(\arctan(3\tan(a)/t_k)) \tag{3.207}$$

für $k = 1, \ldots, p$ und seien $\tilde{\Phi}_1, \ldots, \tilde{\Phi}_p$ komplexe Zahlen, definiert durch

$$\tilde{\Phi}_k = \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan(a)}{\tan(c + \arctan(3\tan(d)/t_k))} \right) \tag{3.208}$$

für $k = 1, \ldots, p$.
Dann gilt für jedes θ mit $|\theta - c| \geq 3d$

$$\left| f(\theta) - \sum_{k=1}^{p} \tilde{\Phi}_k \cdot u_j \left(\frac{3\tan(d)}{\tan(\theta - c)} \right) \right| < A \cdot \frac{6p\sec^2(r)}{\tan(r) \cdot 5^p} \tag{3.209}$$

wobei $A = \sum_{k=1}^{N} |a_k|$.

Beweis:
Anwendung der Dreiecksungleichung auf die Aussage des Satzes liefert:

$$\left| f(\theta) - \sum_{k=1}^{p} \tilde{\Psi}_k \cdot u_j \left(\frac{3\tan d}{\tan(\theta - c)} \right) \right| \leq S_1 + S_2 \tag{3.210}$$

mit

$$S_1 = \left| f(\theta) - \sum_{k=1}^{p} f(c + \arctan(3\tan(d)/t_k)) \cdot u_j \left(\frac{3\tan d}{\tan(\theta - c)} \right) \right| \tag{3.211}$$

und

$$S_2 = \left| \sum_{k=1}^{p} (f(c + \arctan(3\tan(d)/t_k)) - \tilde{\Psi}_k) \cdot u_j \left(\frac{3\tan d}{\tan(\theta - c)} \right) \right| \tag{3.212}$$

Mit Corollar 3.4.12 und der Dreiecksungleichung lässt sich S_1 abschätzen:

$$S_1 < A \cdot \frac{3\sec^2 a}{\tan a \cdot 5^p} \tag{3.213}$$

Wendet man Corollar 3.4.12 hingegen auf S_2 an, so erhält man zusammen mit Lemma 3.4.7 zur Abschätzung von u_j:

$$\begin{aligned} S_2 &\leq \sum_{k=1}^{p} \left| f(c+\arctan(3\tan(d)/t_k)) - \tilde{\Psi}_k \right| \cdot \left(2 - \frac{1}{p}\right) \\ &= \sum_{k=1}^{p} \left| f(c+\arctan(3\tan(d)/t_k)) - \sum_{j=1}^{p} \Psi_j \cdot u_j \left(\frac{3\tan a}{\tan(c+\arctan(3\tan(d)/t_k))} \right) \right| \cdot \left(2 - \frac{1}{p}\right) \\ &< A \cdot (2p-1) \cdot \frac{3 \cdot \sec^2 a}{\tan a \cdot 5^p} \end{aligned} \tag{3.214}$$

Setzen wir diese Abschätzungen für S_1 und S_2 in (3.210) ein, so erhalten wir:

$$\left| f(\theta) - \sum_{k=1}^{p} \tilde{\Psi}_k \cdot u_j \left(\frac{3\tan d}{\tan(\theta - c)} \right) \right| < A \cdot \frac{6p\sec^2(a)}{\tan(a) \cdot 5^p} \tag{3.215}$$

□

Das folgende Corollar stellt wieder eine einfache Variablentransformation des vorherigen Satzes dar:

Corollar 3.4.22 *Seien $p, N \geq 2$ und x_0, r, c, d reelle Zahlen sodaß $0 < a \leq \pi/8$ und $[x_0 + c - d, x_0 + c + d] \supset [x_0 - a, x_0 + a]$.*
Die Funktion $\tilde{f} : [-\pi/2, \pi/2] \to \mathbf{C}$ sei definiert durch

$$\tilde{f}(y) = \sum_{k=1}^{p} \frac{\alpha_k}{\tan(y - x_k)} \tag{3.216}$$

wobei $x_k \in [x_0 - r, x_0 + r]$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. $\Phi_1, \ldots, \Phi_p$ seien komplexe Zahlen, definiert durch

$$\Phi_j = \tilde{f}(\arctan(3\tan(r)/t_j) + x_0). \tag{3.217}$$

für $j = 1, \ldots, p$ und $\tilde{\Phi}_1, \ldots, \tilde{\Phi}_p$ komplexe Zahlen, definiert durch

$$\tilde{\Phi}_k = \sum_{j=1}^{p} \Phi_j \cdot u_j \left(\frac{3\tan(r)}{\tan(c+\arctan(3\tan(d)/t_k))} \right) \tag{3.218}$$

für $k = 1, \ldots, p$. Dann gilt für jedes θ mit $|\theta - x_0 - c| \geq 3r$

$$\left|\tilde{f}(\theta) - \sum_{k=1}^{p} \tilde{\Phi}_k \cdot u_j\left(\frac{3\tan(d)}{\tan(\theta - x_0 - c)}\right)\right| < A \cdot \frac{6p\sec^2(a)}{\tan(a)\cdot 5^p} \tag{3.219}$$

Hierbei ist $A = \sum_{k=1}^{N} |a_k|$. *Außerdem gilt:*

$$\left|\tilde{\Phi}_k - \tilde{f}(c + x_0 + \arctan(3\tan(d)/t_k))\right| < A \cdot \frac{3\sec^2(r)}{\tan(r)\cdot 5^p} \tag{3.220}$$

Beweis:
Die Aussage ergibt sich sofort aus dem vorherigen Satz, wenn man $a = r$ und $\theta_k = x_k - x_0$ setzt. Den Zusatz für $\tilde{\Phi}$ erhält, wenn man (3.124) aus Corollar 3.4.13 mit der Definition von $\tilde{\Phi}$ vergleicht. □

Die folgenden Ausführungen sind wegen der vielen auftretenden Indizies ein wenig unübersichtlich. Sie beschreiben lediglich formal, wie man eine far-field Erweiterung als Summe der beiden darüberliegenden kleineren far-field Erweiterungen erhalten kann. Abbildung 3.7 illustriert dies.
Wir wenden Corollar 3.4.22 zweimal an: einmal mit $c = r$, $d = 2r$ und $\tilde{f} = f^{l+1,2i-1}$ auf das Intervall $I_{l+1,2i-1}$ und einmal mit $c = -r$, $d = 2r$ und $\tilde{f} = f^{l+1,2i}$ auf das Intervall $I_{l+1,2i}$. r sei hierbei der Intervallradius auf Stufe $l+1$. Zu beachten ist, daß x_0 die Intervallemitte des jeweils betrachteten Intervalls auf Stufe $l+1$ darstellt. Die Intervallmitte y_0 auf Stufe l ist daher in beiden Fällen $x_0 + c$: einmal wegen $c = r$ und einmal wegen $c = -r$. Da $f^{l,i} = f^{l+1,2i-1} + f^{l+1,2i}$ ist, liefert die Summe der beiden so entstandenen $\tilde{\Phi}$ eine Approximation für $\Phi^{l,i} = f^{l,i}(\arctan(3tan(2r)/t_j + y_0)$. Aus (3.218) ergibt sich nämlich:

$$\begin{aligned}
\Phi_k^{l,i} &= \sum_{j=1}^{p} \Phi_j^{l+1,2i-1} \cdot u_j\left(\frac{3\tan(r)}{\tan(r + \arctan(3\tan(2r)/t_k))}\right) && (3.221)\\
&+ \sum_{j=1}^{p} \Phi_j^{l+1,2i} \cdot u_j\left(\frac{3\tan(r)}{\tan(-r + \arctan(3\tan(2r)/t_k))}\right) && (3.222)
\end{aligned}$$

für $k = 1, \ldots, p$.
Somit kann man die $p \times p$-Matrizen M_L und M_R defnieren:

$$\begin{aligned}
M_L(k,j) &= u_j\left(\frac{3\tan(r)}{\tan(r + \arctan(3\tan(2r)/t_k))}\right) && (3.223)\\
M_R(k,j) &= u_j\left(\frac{3\tan(r)}{\tan(-r + \arctan(3\tan(2r)/t_k))}\right) && (3.224)
\end{aligned}$$

Die Matrizen hängen nur von der Intervallgröße r, nicht aber von der Intervallposition x_0 ab. Man benötigt also nur $nlevs-1$-Versionen der Matrizen für die Stufen $1, \ldots, nlevs-1$.

Der letzte Satz schließlich beschreibt die Transformation von lokalen Erweiterungen Ψ:

Satz 3.4.23 *Seien $p, N \geq 2$ und a, c, d reelle Zahlen mit $0 < a \leq \pi/8$ und $[c-d, c+d] \subset [-a, a]$. Sei die Funktion $f : [-\pi/2, \pi/2] \to \mathbf{C}$ definiert durch*

$$f(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta - \theta_k)} \tag{3.225}$$

wobei $3a \leq |\theta_k| \leq \pi/2$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. $\Psi_1, \ldots, \Psi_p$ seien komplexe Zahlen, definiert durch

$$\Psi_k = f(\arctan(t_k \tan a)). \tag{3.226}$$

Schließlich sei $\tilde{\Psi}_1, \ldots, \tilde{\Psi}_p$ ein komplexer Vektor, definiert durch

$$\tilde{\Psi}_k = \sum_{j=1}^{p} \Psi_j \cdot u_j\left(\frac{\tan(c + \arctan(t_k \tan d))}{\tan a}\right) \tag{3.227}$$

für $k = 1, \ldots, p$. Dann gilt für alle $\theta \in [c-d, c+d]$:

$$\left| f(\theta) - \sum_{k=1}^{p} \tilde{\Psi}_k \cdot u_j\left(\frac{\tan(\theta - c)}{\tan d}\right)\right| < 2 \cdot A \cdot \frac{p(1 + 9\tan^2 a)}{\tan a \cdot 5^p} \tag{3.228}$$

Hierbei ist $A = \sum_{k=1}^{N} |\alpha_k|$.

Beweis:
Addiert man $\pm \sum_{k=1}^{p}(f(c + \arctan(t_k \tan d)) \cdot u_j(\tan(\theta - c)/\tan d)$ als günstige 0 zum abzuschätzenden Ausdruck, so erhält man nach Anwendung der Dreiecksungleichung

$$\left| f(\theta) - \sum_{k=1}^{p} \tilde{\Psi}_k \cdot u_j\left(\frac{\tan(\theta - c)}{\tan d}\right)\right| \tag{3.229}$$

$$\begin{aligned}
&\leq \left| f(\theta) - \sum_{k=1}^{p} f(c + \arctan(t_k \tan d)) \cdot u_j\left(\frac{\tan(\theta - c)}{\tan d}\right)\right| && (3.230)\\
&\quad + \left| \sum_{k=1}^{p} (f(c + \arctan(t_k \tan d)) - \tilde{\Psi}_k) \cdot u_j\left(\frac{\tan(\theta - c)}{\tan d}\right)\right| && (3.231)\\
&= S_1 + S_2 && (3.232)
\end{aligned}$$

mit

$$S_1 = \left| f(\theta) - \sum_{k=1}^{p} f(c + \arctan(t_k \tan d)) \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| \tag{3.233}$$

und

$$S_2 = \left| \sum_{k=1}^{p} (f(c + \arctan(t_k \tan d)) - \tilde{\Psi}_k) \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| \tag{3.234}$$

Mit Corollar 3.4.17 und der Dreiecksungleichung lässt sich S_1 abschätzen:

$$S_1 < A \cdot \frac{1 + 9 \tan^2 a}{\tan a \cdot 5^p} \tag{3.235}$$

Wendet man Corollar 3.4.17 hingegen auf S_2 an, so erhält man zusammen mit Lemma 3.4.7 zur Abschätzung von u_j:

$$\begin{aligned} S_2 &\leq \sum_{k=1}^{p} \left| f(c + \arctan(t_k \tan d)) - \tilde{\Psi}_k \right| \cdot \left(2 - \frac{1}{p} \right) \\ &= \sum_{k=1}^{p} \left| f(c + \arctan(t_k \tan d)) - \sum_{j=1}^{p} \Psi_j \cdot u_j \left(\frac{\tan(c + \arctan(t_k \tan d))}{\tan a} \right) \right| \cdot \left(2 - \frac{1}{p} \right) \\ &< A \cdot (2p - 1) \cdot \frac{1 + 9 \tan^2 a}{\tan a \cdot 5^p} \end{aligned} \tag{3.236}$$

Setzen wir diese Abschätzungen für S_1 und S_2 in (3.229) ein, so erhalten wir:

$$\left| f(\theta) - \sum_{k=1}^{p} \tilde{\Psi}_k \cdot u_j \left(\frac{\tan(\theta - c)}{\tan d} \right) \right| < 2 \cdot A \cdot \frac{p(1 + 9 \tan^2 a)}{\tan a \cdot 5^p}. \tag{3.237}$$

□

Corollar 3.4.24 *Seien $p, N \geq 2$ und r, c reelle Zahlen mit $0 < r \leq \pi/8$ und $[y_0 + c - d, y_0 + c + d] \subset [y_0 - r, y_0 + r]$. Sei die Funktion $f : [-\pi/2, \pi/2] \to \mathbf{C}$ definiert durch*

$$\tilde{f}(\theta) = \sum_{k=1}^{N} \frac{\alpha_k}{\tan(\theta - x_k)} \tag{3.238}$$

wobei $3r \leq |x_k - y_0| \leq \pi/2$ für $k = 1, \ldots, N$ und $\alpha_1, \ldots, \alpha_N$ ein komplexer Vektor ist. $\Psi_1, \ldots, \Psi_p$ seien komplexe Zahlen, definiert durch

$$\Psi_k = \tilde{f}(\arctan(t_k \tan a) + y_0). \tag{3.239}$$

Schließlich sei $\tilde{\Psi}_1, \ldots, \tilde{\Psi}_p$ ein komplexer Vektor, definiert durch

$$\tilde{\Psi}_k = \sum_{j=1}^{p} \Psi_j \cdot u_j \left(\frac{\tan(c + \arctan(t_k \tan d))}{\tan r} \right) \tag{3.240}$$

für $k = 1, \ldots, p$. Dann gilt für alle $\theta \in [y_0 + c - d, y_0 + c + d]$:

$$\left| \tilde{f}(\theta) - \sum_{k=1}^{p} \tilde{\Psi}_k \cdot u_j \left(\frac{\tan(\theta - y_0 - c)}{\tan d} \right) \right| < 2 \cdot A \cdot \frac{p(1 + 9\tan^2 r)}{\tan r \cdot 5^p} \tag{3.241}$$

Hierbei ist $A = \sum_{k=1}^{N} |\alpha_k|$. Außerdem gilt:

$$\left| \tilde{\Psi}_k - \tilde{f}(y_0 + c + arctan(t_k \tan d)) \right| < A \cdot \frac{1 + 9\tan^2 d}{\tan d \cdot 5^p} \tag{3.242}$$

Beweis:
Die Aussage ergibt sich wieder sofort aus dem vorherigen Satz, wenn man $a = r$ und $\theta_k = x_k - y_0$ setzt. Die Zusatzaussage über $\tilde{\Phi}$ erhält man, wenn man die Definition von $\tilde{\Psi}$ mit (3.163) in Corollar 3.4.18 vergleicht. □

Mit diesem Satz kann man Teile der lokalen Erweiterung einer Stufe durch die lokale Erweiterung der darunterliegenen Stufe gewinnen.
Aus Abbildung 3.6 erkennt man, daß man das Intervall $I_{l-1,i}$ auf seine linke Hälfte einschränken muß. Wir wenden deshalb das vorherige Corollar an und wählen $c = -\frac{1}{2}r$, $d = \frac{1}{2}r$. r ist hierbei der Intervallradius auf Stufe $l-1$ und y_0 die Intervallmitte von $I_{l-1,i}$. Aufgrund der Wahl von c ist $y_0 + c$ genau die Intervallmitte des Intervalls $I_{l,2i-1}$ und das im Corollar angegeben $\tilde{\Psi}_k \approx \tilde{f}(y_0 + c + arctan(t_k \tan d))$ hat genau die für $\Psi^{l,2i-1}$ passende Form.
Um das gesamte $\Psi^{l,2i-1}$ zu erhalten, müssen wir aber noch die bisher unberücksichtigten Knoten in den Intervallen $I_{l,2i-3}$, $I_{l,2i+1}$ und $I_{l,2i+2}$ berücksichtigen. Dies bewerkstelligen die eben hergeleiteten Transformationn T_2,T_3 und T_4. Insgesamt hat mal also:

$$\Psi^{l,2i-1} = S_L \Psi^{l-1,i} + T_2 \Phi^{l,2i-3} + T_3 \Phi^{l,2i+1} + T_4 \Phi^{l,2i+2} \tag{3.243}$$

S_L ergibt sich hierbei aus (3.240) wegen $c = -\frac{1}{2}r$ und $d = \frac{1}{2}r$ zu:

$$S_L(k,j) = u_j \left(\frac{\tan(-\frac{1}{2}r + \arctan(t_k \tan \frac{1}{2}r))}{\tan r} \right) \tag{3.244}$$

für $j,k=1,\ldots,p$. Dies gilt für ungeradee Intervallen ($k=2i-1$), in geraden Intervallen $k=2i$ erhält man analog:

$$\Psi^{l,2i} = S_R\Psi^{l-1,i} + T_1\Phi^{l,2i-3} + T_2\Phi^{l,2i-2} + T_3\Phi^{l,2i+2} \tag{3.245}$$

Hierbei wurde wieder Corollar 3.4.24 angewendet, diesmal mit $c=+\frac{1}{2}r$, $d=\frac{1}{2}r$. S_R ergibt sich deshalb zu:

$$S_R(k,j) = u_j\left(\frac{\tan(\frac{1}{2}r + \arctan(t_k\tan\frac{1}{2}r))}{\tan r}\right). \tag{3.246}$$

S_L,S_R hängen wieder nur vom Intervallradius r, nicht aber von der Intervallposition y_0 ab. Man benötigt deshalb auch hier jeweils $nlevs-1$ Matrizen für die Stufen $2,\ldots,nlevs$.

3.4.5 Beschreibung der FMM

Damit haben wir alle Hilfsmittel zusammen. Wir können jetzt einen Algorithmus angeben, der Summen der Art

$$f(x) = \sum_{k=1}^{N}\alpha_k\frac{1}{\tan(x-x_k)} \tag{3.247}$$

schnell an den N Punkten $y_1,\ldots,y_N$ auswertet. Er besteht aus fünf Schritten:

1. Berechne die 2^{nlevs} far-field Erweiterungen $\Phi^{nlevs,i}$ auf der größten Stufe $nlevs$ direkt. Es handelt sich hierbei um Vektoren der Länge p, definiert durch

$$\Phi_j^{nlevs,i} = f^{nlevs,i}(\arctan(3\tan(r)/t_j) + x_0^i) \tag{3.248}$$

   für $j=1,\ldots,p$ und $i=1,\ldots,2^{nlevs}$. x_0^i ist hierbei die Intervallmitte des Intervalls I_i

2. Berechne die far-field Erweiterungen auf den anderen Stufen aus den far-field Erweiterungen der jeweils nächstgrößeren Stufe:

for l=$nlevs-1,\ldots,1$
  for $i=1,\ldots,2^l$
    $\Phi^{l,i} = M_L\cdot\Phi^{l+1,2i-1} + M_R\cdot\Phi^{l+1,2i}$

3. Berechne die lokalen Erweiterungen. Ein $\Psi^{l,i}$ setzt sich zusammen aus der lokalen Erweiterung der darunterliegenden (kleineren) Stufe und den far-field Erweiterungen der drei Subintervalle, die vollständig vom betrachteten Intervall getrennt sind, aber nicht durch die far-field Erweiterung der darunterliegenden Stufe berücksichtigt wurden:

   for l=1, ..., $nlevs - 1$
       for $i = 1, \ldots, 2^l$
           $\Psi^{l,2i-1} = S_L \cdot \Psi^{l-1,i} + T_2 \cdot \Phi^{l,2i-3} + T_3 \Phi^{l,2i+1} + T_4 \Phi^{l,2i+2}$
           $\Psi^{l,2i} = S_R \cdot \Psi^{l-1,i} + T_1 \cdot \Phi^{l,2i-3} + T_2 \cdot \Phi^{l,2i-2} + T_3 \cdot \Phi^{l,2i+2}$

   Auf Stufe 1 gilt $\Psi^{1,1} = \Psi^{1,2} = 0$. Wegen der Periodizität der Funktion f sind die Intervall-Indizes außerdem modulo 2^l zu nehmen.

4. Mit den lokalen Erweiterungen der größten Stufe $\Psi^{nlevs,i}$ kann man jetzt $f_l = f(y_l)$ berechnen.

   for $i = 1, \ldots, 2^{nlevs}$
       $f_l = \sum_{k=1}^{p} \Psi_k^{nlevs,i} \cdot u_k \left(\frac{\tan(y_l - y_0)}{\tan r} \right)$

   y_0 ist hierbei die Intervallmitte des Intervalls, in dem y_l auf Stufe $nlevs$ liegt.

5. Jetzt müssen zu f_l noch diejenigen Summanden hinzuaddiert werden, die zu Knoten gehören, die in den drei nicht vollständig von y_l getrennten Intervallen liegen:

   for $i = 1, \ldots, 2^{nlevs}$
       $f_l = f_l + f_P(y_l)$

Bezeichnet man mit s die Anzahl der Knoten, die durchschnittlich in einem Intervall der größten Stufe liegen, also $s = N/2^{nlevs}$, so erfordert die Berechnung der 5 Schritte die folgende Anzahl Rechenoperationen:

Stufe	Operationen
1	$\mathcal{O}(Np)$
2	$\mathcal{O}(2Np^2/s)$
3	$\mathcal{O}(8Np^2/s)$
4	$\mathcal{O}(Np)$
5	$\mathcal{O}(3Ns)$

Operationen die nur von der Verteilung der Stützstellen abhängen werden hierbei natürlich vorberechnet und somit nicht berücksichtigt.
Insgesamt benötigen wir also $\mathcal{O}(N \cdot (2p + 10p^2/s + 3s))$ Rechenoperationen. Diese Anzahl wird minimal für $s = \sqrt{10/3} \cdot p$. Wir wählen deshalb $s \approx 2p$. Der Algorithmus benötigt damit größenordnungsmäßig $13 \cdot N \cdot p$ Rechenoperationen.
Da Matrizen M, T und S sind ebenso wie die Kernel-Funktion $1/tan(x)$ reellwertig. Betrachtet man die auftretenden Operationen, so erkennt man, daß die Additionen immer komplexe Additionen sind, während die Multiplikationen lediglich Multiplikationen von reellen mit komplexen Zahlen darstellen. Aufwendige rein komplexe Multiplikationen treten nicht auf.
Wählen wir $s = 2p$ so ergibt sich wegen $s = N/2^{nlevs}$ die Anzahl der Stufen $nlevs$ zu $nlevs = \log_2(N/2p)$.
Dies ist die schnelle Multipolentwicklung (FMM, fast multipole method). Wir haben hierbei nicht besonders viele Eigenschaften der Kernel-Funktion $1/tan(x-y)$ verwendet. Man kann den Algorithmus daher ohne größere Änderungen auch für andere Kernel vwerden, z.B. für 1/(x-y). Dieser Fall ist sogar etwas einfacher, da 1/(x-y) nicht periodisch ist und deshalb etwas weniger Verwaltungsaufwand erforderlich ist.
Der Algorithmus für die FMM wurde von mir so implementiert, daß er auch unabhängig von der hier betrachteten Fourier-Transformation eingesetzt werden kann. Desweiteren wurden alle Abhängigkeiten von der Kernel-Funktion in einer eigenen Klasse FMM_Kernel gekapselt, damit der eigentliche Algorithmus für möglichst viele Kernel-Funktionen verwendet werden kann. Derzeit existieren Kernel-Klassen für 1/tan(x-y) und 1/(x-y).

Jetzt aber zurück zur Fourier-Transformation:

3.5 Schnelle DFT mittels Multipolentwicklung

Wir betrachten wieder die diskrete Fourier-Transformation $F : \mathbf{C}^N \to \mathbf{C}^N$, definiert durch:

$$F(\alpha)_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \tag{3.249}$$

für $j = 1, \ldots, N$. Die $x_1, \ldots, x_N$ sind die frei in $[-\pi, \pi]$ verteilten Stützstellen; sie sollen allerdings aufsteigend sortiert sein, also $x_k < x_l$ für $k < l$.
Genauer sind wir an der schnellen Berechnung der folgenden Transformationen interessiert:

Problem 1: Sei α gegeben, berechne $f = F(\alpha)$.

Problem 2: Sei β gegeben, berechne $g = F^T(\beta)$.

Problem 3: Sei f gegeben, berechne $\alpha = F^{-1}(f)$.

Problem 4: Sei β gegeben, berechne $\beta = (F^T)^{-1}(g)$.

Hierbei sind $\alpha = \alpha_{-N/2}, \ldots, \alpha_{N/2-1}$, $\beta = \beta_1, \ldots, \beta_N$, $f = f_1, \ldots, f_N$ und $g = g_{-N/2}, \ldots, g_{N/2-1}$ komplexe Vektoren.
Weiterhin benötigen wir noch die diskrete Fourier-Transformation bei äquidistanten Stützstellen: y bezeichne den Vektor der äquidistanten Stützstellen:

$$y_k = \frac{2\pi}{N}(k - 1 - N/2) \tag{3.250}$$

für $k = 1, \ldots, N$. Die zugehörige Fourier-Transformation ist definiert durch:

$$\mathcal{F}(\alpha)_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{iky_j} \tag{3.251}$$

für $k = 1, \ldots, N$. Als Matrix Operator schreibt sich das wie folgt:

$$\mathcal{F}_{jk} = e^{2\pi i(j-N/2-1)\cdot(k-N/2-1)/N} \tag{3.252}$$

Wir haben bereits in der Einleitung gesehen, daß man dies trotz der etwas ungewöhnlichen symmetrischen Formulierung mit einem normalen FFT-Algorithmus, etwa nach Cooley-Tukey, in $\mathcal{O}(N \log N)$ Rechenoperation berechnen kann. Bekanntermaßen gilt $\mathcal{F}^T = \mathcal{F}$ und $\mathcal{F}^{-1} = (1/N)\overline{\mathcal{F}}$, die inverse Fourier-Transformation kann hierbei natürlich ebenfalls in $\mathcal{O}(N \log N)$ Rechenoperationen ausgewertet werden.

Die Sätze 3.3.1 und 3.3.2 geben an, wie man die Fourier-Transformation bezüglich eines Satzes von Stützstellen auf die Fourier-Transformation bezüglich eines anderen Satzes von Stützstellen transformiert.
Wenden wir Satz 3.3.2 mit Bezeichnungen dieses Abschnittes an, so erhält man gerade $f_j = \mathcal{F}(\alpha)_j$ sowie $g_j = F(\alpha)_j$. Der Satz sagt dann aus, das sich das gesuchte g aus dem per FFT schnell berechenbaren f durch folgende Formel ergibt:

$$g_l = \sin\left(\frac{Nx_l}{2}\right) \cdot \sum_{j=1}^{N} f_j \cdot \frac{(-1)^j}{N} \cdot \left(\frac{1}{\tan(x_l - y_j)/2)} - i\right) \tag{3.253}$$

Damit kann man einen Matrix-Operator $\mathcal{P}$ definierten, sodaß $g = \mathcal{P}f$:

$$\mathcal{P}_{jk} = \sin\left(\frac{Nx_j)}{2}\right) \cdot \frac{(-1)^k}{N} \cdot \left(\frac{1}{\tan((x_j - y_k)/2)} - i\right) \tag{3.254}$$

für $j, k = 1, \ldots, N$. Den transponierte Operator kann man direkt angeben:

$$\mathcal{P}^T_{jk} = \frac{(-1)^j}{N} \cdot \sin\left(\frac{Nx_k)}{2}\right) \cdot \left(\frac{1}{\tan((x_k - y_j)/2)} - i\right) \tag{3.255}$$

$$= \frac{(-1)^{j+1}}{N} \cdot \sin\left(\frac{Nx_k)}{2}\right) \cdot \left(\frac{1}{\tan((y_j - x_k)/2)} + i\right) \tag{3.256}$$

$$\tag{3.257}$$

$\mathcal{P}$ transformiert die Fourier-Transformation bezüglich der Stützstellen $y_1, \ldots, y_N$ in eine Fourier-Transformation bezüglich der Stützstellen $x_1, \ldots, x_N$. $\mathcal{P}^{-1}$ ist daher einfach die Abbildung, die die Fourier-Transformation bezüglich der Stützstellen $x_1, \ldots, x_N$ zurück in eine Fourier-Transformation bezüglich der Stützstellen $y_1, \ldots, y_N$ transformiert.
Mit Satz 3.3.1 kann man $\mathcal{P}^{-1}$ daher direkt angeben:

$$\mathcal{P}^{-1}_{jk} = c_j \cdot d_k \cdot \left(\frac{1}{\tan((y_j - x_k)/2)} - i\right) \tag{3.258}$$

c_j und d_k sind dabei wie in (3.2.2) bzw. (3.2.2) definiert. Man muß lediglich beachten, daß jetzt y und x vertauscht sind, also:

$$c_l = \prod_{k=1}^{N} \sin((y_l - x_k)/2) \tag{3.259}$$

$$d_j = \prod_{\substack{k=1 \\ k \neq j}}^{N} \frac{1}{\sin((x_j - x_k)/2)} \tag{3.260}$$

Auch hier läßt sich der transponierte Operator natürlich wieder direkt angeben:

$$(\mathcal{P}^T)^{-1}_{jk} = d_j \cdot c_k \cdot \left(\frac{1}{\tan((y_k - x_j)/2)} - i\right) \tag{3.261}$$

$$= -d_j \cdot c_k \cdot \left(\frac{1}{\tan((x_j - y_k)/2)} + i\right) \tag{3.262}$$

$$\tag{3.263}$$

Es gelten also die folgenden Formulierungen:

$$\begin{aligned} F &= \mathcal{P} \cdot \mathcal{F} \\ F^T &= \mathcal{F} \cdot \mathcal{P}^T \\ F^{-1} &= \mathcal{F}^{-1} \cdot \mathcal{P}^{-1} \\ (F^T)^{-1} &= (\mathcal{P}^T)^{-1} \cdot \mathcal{F}^{-1} \end{aligned} \tag{3.264}$$

Sowohl $\mathcal{P}$ als auch $\mathcal{P}^{-1}$ kann man mit der schnellen Multipolentwicklung in $\mathcal{O}(Nq) = \mathcal{O}(N\log(1/\epsilon)$ Rechenoperationen anwenden. Da die Berechnung der FFT $\mathcal{F}$ $\mathcal{O}(N\log N)$ Operationen erfordert, kann man die vier Probleme (3.264) jeweils in $\mathcal{O}(N\log N) + \mathcal{O}(N\log(1/\epsilon)$ Rechenoperationen lösen.

Bei den Operatoren $\mathcal{P}$ tauchen Tangensausdrücke im Nenner auf. Diese sind daher natürlich nicht ohne weiters für $x_j = y_k$ definiert, da dann der Tangens und somit der Nenner 0 wird. Dieser Fall tritt auf, falls eine der frei verteilten Stützstellen x_j mit eine der äquidistanten Stützstellen y_k übereinstimmt.
Man kann die Formeln aber problemlos stetig fortsetzen und sie behalten dabei auch ihre Gültigkeit. Betrachten wir die Definition von $\mathcal{P}$ in (3.254): Im Beweis von Satz 3.3.2 ergab sich in (3.60), daß

$$\sin(N(x_l - y_j)/2) = sin(Nx_l/2)\cdot(-1)^j. \tag{3.265}$$

Die Definition von $\mathcal{P}$ wird damit zu

$$\mathcal{P}_{jk} = \left(\frac{\sin(N(x_j - y_k)/2)}{N\cdot\tan((x_j - y_k)/2)} - i\sin(N(x_j - y_k)/2).\right) \tag{3.266}$$

Für $x_j \to y_k$ ergibt sich dann mit der Regel von l'Hospital unmittelbar

$$\mathcal{P}_{jk} = 1 \qquad \text{falls } x_j = y_k. \tag{3.267}$$

Gleiches gilt natürlich für $\mathcal{P}^T$. Bei $\mathcal{P}^{-1}$ und $(\mathcal{P}^T)^{-1}$ liegen die Dinge sogar noch etwas einfacher, setzt man die Definition von c_j ein, so erhält man:

$$\begin{aligned}\mathcal{P}^{-1}_{jk} &= \prod_{l=1,l\neq k}^{N}\sin((y_j - x_l)/2)\cdot d_k\cdot\left(\frac{\sin((y_j - x_k)/2)}{\tan((y_j - x_k)/2)} - i\sin((y_j - x_k)/2)\right)\\ &= \prod_{l=1,l\neq k}^{N}\sin((y_j - x_l)/2)\cdot d_k\cdot(\cos((y_j - x_k)/2) - i\sin((y_j - x_k)/2))\end{aligned} \tag{3.268}$$

Also

$$\mathcal{P}^{-1}_{jk} = \prod_{l=1,l\neq k}^{N}\sin((y_j - x_l)/2)\cdot d_k \qquad \text{falls } y_j = x_k. \tag{3.269}$$

Für $(\mathcal{P}^T)^{-1}$ gilt natürlich entsprechendes. Wenn man diese Fälle bei der Implementierung des Algorithmus berücksichtigt, so ist zu beachten, daß man die Summen

$$\sum_{k=1}^{N}\frac{\alpha_k}{\tan((x_j - y_k)/2)} \tag{3.270}$$

natürlich per schneller Multipolentwicklung berechnet und daher nicht einfach einzelne P_{jk} umdefinieren kann. Man geht stattdessen so vor, daß man $1/tan(0)$ als 1 definiert und dann den Rest des Algorithmus entsprechend anpaßt. Mit „Rest" sind hierbei die normalerweise erforderlichen Multiplikationen mit $sin(Nx_l/2)$ oder c_l gemeint.
Die Definition von $1/\tan(0) = 1$ verursacht keine (Laufzeit-)Probleme, weil Berechnungen der Tangensfunktion Berechnungen auf den Stützstellen darstellen und daher ohnehin während der Initialisierung getätigt werden.
In der formalen Beschreibung der Algorithmen im nächsten Abschnitt wird auf diese Problematik nicht weiter eingegangen, um die Beschreibung übersichtlicher zu halten. Die entsprechenden Modifikationen wurden aber bei der Implementierung der Algorithmen durchgeführt.

3.6 Formale Beschreibung der Algorithmen

Die Algorithmen bestehen im wesentlichen aus einer schnellen Fourier-Transformation und einer schnellen Multipolentwicklung, bereichert um einige Skalierungen. Die Skalierungen benötigen lediglich $\mathcal{O}(N)$ Rechenoperationen, ihre Rechenzeit kann daher im Verhältnis zur FFT und zur FMM vernachläßigt werden. Der komplexe Teil des Algorithmus liegt in der schnellen Multipolentwicklung, so wie sie in Abschnitt 3.4 bescrhieben wurde.
Die vier Algorithmen 1-4 dienen zur Berechnung von F, F^T, F^{-1} und $(F^T)^{-1}$. Sie ergeben sich unmittelbar aus den in (3.264) angegebenen Beziehungen.

Algorithmus B1

Eingangsvektor ist der Vektor $\{\alpha_{-N/2}, \ldots, \alpha_{N/2-1}\}$. Es wird die DFT

$$f_j = F(\alpha)_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \tag{3.271}$$

für $j = 1, \ldots, N$ berechnet. Dies ist genau dieselbe Aufgabenstellung wie bei Algorithmus A1 aus Kapitel 2.

1. Berechne
$$g_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{iky_j} \tag{3.272}$$
für $j = 1, \ldots, N$ per FFT.

2. Skaliere das Zwischenresultat:

$$g_j = g_j \cdot (-1)^j / N \qquad \text{für } j = 1, \ldots, N \tag{3.273}$$

3. Berechne

$$f_l = \sum_{j=1}^{N} g_j \cdot \frac{1}{\tan((x_l - y_j)/2)} \tag{3.274}$$

für $j = 1, \ldots, N$ durch schnelle Multipolentwicklung.

4. Berechne das Endergebnis:

$$s = \sum_{j=1}^{N} g_j \tag{3.275}$$

$$f_l = (f_l - i \cdot s) \cdot \sin(N x_l / 2) \tag{3.276}$$

für $l = 1, \ldots, N$. Die Koeffizienten $\sin(N x_l/2)$ sollten hierbei in einem Array vorberechnet werden.

Die weiteren Algorithmen unterscheiden sich prinzipiell nicht wesentlich von Algorithmus 1.

Algorithmus B2

Eingangsvektor ist der Vektor $\{\alpha_1, \ldots, \alpha_N\}$ Es wird die folgende DFT berechnet

$$f_j = F^T(\alpha)_j = \sum_{k=1}^{N} \alpha_k \cdot e^{ijx_k} \tag{3.277}$$

für $j = -N/2, \ldots, N/2 - 1$. Dies ist dieselbe Aufgabenstellung wie bei Algorithmus $A2$ aus Kapitel 2.

1. Skaliere den Eingangsvektor

$$\alpha_l = \alpha_l \cdot \sin(N x_l / 2) \tag{3.278}$$

für $l = 1, \ldots, N$. Die Koeffizienten $\sin(N x_l/2)$ sollten hierbei in einem Array vorberechnet werden.

2. Berechne
$$a_l = \sum_{j=1}^{N} \alpha_j \cdot \frac{1}{\tan((y_l - x_j)/2)} \tag{3.279}$$
für $j = 1, \ldots, N$ durch schnelle Multipolentwicklung.

3. Skaliere das Zwischenresultat:
$$s = \sum_{j=1}^{N} \alpha_j \tag{3.280}$$
$$a_j = (a_j + i \cdot s) \cdot (-1)^{j+1}/N \tag{3.281}$$
für $j = 1, \ldots, N$.

4. Berechne das Endergebnis
$$f_j = \sum_{k=-N/2}^{N/2-1} a_k \cdot e^{iky_j} \tag{3.282}$$
für $j = 1, \ldots, N$ per FFT.

Algorithmus B3

Eingangsvektor ist der Vektor $\{f_1, \ldots, f_N\}$. Es wir die inverse DFT
$$\alpha_j = F^{-1}(f)_j \tag{3.283}$$
für $j = -N/2, \ldots, N/2 - 1$ berechnet. F ist hierbei wie bei der Beschreibung von Algorithmus B1 definiert.

1. Skaliere den Eingangsvektor
$$f_l = f_l \cdot d_l \tag{3.284}$$
für $l = 1, \ldots, N$. d_l ist durch (3.260) definiert und sollte natürlich vorberechnet werden.

2. Berechne
$$a_l = \sum_{k=1}^{N} f_k \cdot \frac{1}{\tan((y_l - x_k)/2)} \tag{3.285}$$
für $l = 1, \ldots, N$ durch schnelle Multipolentwicklung.

3. Skaliere das Zwischenresultat:
$$s = \sum_{j=1}^{N} f_j \tag{3.286}$$
$$a_j = (a_j - i \cdot s) \cdot c_j / N \tag{3.287}$$
für $j = 1, \ldots, N$. c_j ist durch (3.2.2) definiert; der Faktor $1/N$ entstammt der inversen Fourier- Transformation des nächsten Schrittes.

4. Berechne das Endergebnis
$$\alpha_j = \sum_{k=1}^{N} a_k \cdot e^{iky_j} \tag{3.288}$$
für $j = -N/2, \ldots, N/2 - 1$ per inverser FFT.

Algorithmus B4

Eingangsvektor ist der Vektor $\{f_1, \ldots, f_N\}$. Es wir die inverse DFT
$$\alpha_j = (F^T)^{-1}(f)_j \tag{3.289}$$
für $j = 1, \ldots, N$ berechnet. F^T ist hierbei wie bei der Beschreibung von Algorithmus B2 definiert.

1. Berechne
$$a_j = \sum_{k=1}^{N} f_k \cdot e^{-iky_j} \tag{3.290}$$
für $j = 1, \ldots, N$ per inverser FFT.

2. Skaliere das Zwischenergebnis
$$a_j = a_j \cdot c_j / N \tag{3.291}$$
für $j = 1, \ldots, N$. c_j ist durch (3.2.2) definiert; der Faktor $1/N$ entstammt der inversen Fourier- Transformation des vorherigen Schrittes.

3. Berechne
$$\alpha_l = \sum_{k=1}^{N} a_k \cdot \frac{1}{\tan((x_l - y_k)/2)} \tag{3.292}$$
für $l = 1, \ldots, N$ durch schnelle Multipolentwicklung.

4. Berechne das Endergebnis:

$$s = \sum_{j=1}^{N} a_j \tag{3.293}$$

$$\alpha_l = -(\alpha_l + i \cdot s) \cdot d_l \tag{3.294}$$

für $l = 1, \ldots, N$. d_l ist durch (3.260) definiert und sollte natürlich vorberechnet werden.

3.7 Numerische Untersuchungen

Die vier beschreibenen Algorithmen wurden von mir in C++ implementiert.
Es wurden dieselben Rahmenbedingungen wie für die schnellen DFT-Algorithmen aus Kapitel 2 verwendet, um vergleichbare Ergebnisse zu erzielen:

1. Für die FFT wurde die Routine der Numerical Recipes verwendet.

2. die Fehler wurden mit den beiden relativen Fehlernormen E_∞ und E_2 gemessen:

$$E_\infty = \frac{\left\| \tilde{f}_j - f_j^{Soll} \right\|_\infty}{\|\alpha\|_1} \tag{3.295}$$

sowie

$$E_2 = \frac{\left\| \tilde{f}_j - f_j^{Soll} \right\|_2}{\|f\|_2} \tag{3.296}$$

f^{Soll} wurde wieder direkt und mit `<long double>`-Genauigkeit berechnet.

3. Die Eingangsvektoren α wurden mit Zufallswerten aus dem komplexen Einheitsquadrat gefüllt:

$$0 \leq \mathrm{Re}(\alpha_j) \leq 1, \qquad 0 \leq \mathrm{Im}(\alpha_j) \leq 1. \tag{3.297}$$

4. Für die Stützstellen $x_1, \ldots, x_N$ der DFT wurden gleichverteilte Zufallswerte aus dem Intervall $[-\pi, \pi]$ verwendet. Bei den beiden inversen Problemen wurden die Stützstellen wie folgt definiert:

$$x_j = -\pi + 2\pi \cdot \frac{j + 0.5 + \delta_j}{N} \tag{3.298}$$

für $j = 1, \ldots, N$, wobei δ_j eine gleichverteilte Zufallszahl aus dem Intervall $[-0.1, 0.1]$ ist. Durch diese Definition wird vermieden, daß die Stützstellen zu dicht beiananderliegen.

Die Ergebnisse der Laufzeitmessungen und Fehleruntersuchungen für die Algorithmen B1-B4 sind in den Tabellen 1-4 dargestellt.

TABELLE 1
Laufzeitverhalten von Algorithmus B1 bei doppelter Genauigkeit

N	Eval msecs	Init msecs	E_∞	E_2	FFT msecs	Faktor
128	6.40	359	7.82e-13	2.66e-12	0.34	18.60
256	18.60	703	1.18e-12	5.39e-12	0.78	23.79
512	37.40	1235	1.03e-12	8.55e-12	1.75	21.37
1024	84.40	2250	1.21e-12	1.40e-11	4.25	19.86
2048	162.60	4203	1.35e-12	2.49e-11	9.41	17.29

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

TABELLE 2
Laufzeitverhalten von Algorithmus B2 bei doppelter Genauigkeit

N	Eval msecs	Init msecs	E_∞	E_2	FFT msecs	Faktor
128	6.20	453	4.55e-15	1.42e-14	0.34	18.02
256	18.80	891	8.54e-15	2.94e-14	0.78	24.10
512	40.60	1578	1.94e-14	1.73e-13	1.78	22.78
1024	72.00	2796	3.96e-14	2.13e-13	4.25	16.94
2048	187.60	5172	4.19e-13	7.74e-12	9.38	20.01

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

TABELLE 3
Laufzeitverhalten von Algorithmus B3 bei doppelter Genauigkeit

N	Eval msecs	Init msecs	E_∞	E_2	FFT msecs	Faktor
128	6.20	492	8.05e-13	2.73e-12	0.34	18.02
256	18.80	1086	1.03e-12	5.57e-12	0.78	24.10
512	40.60	2423	1.14e-12	9.31e-12	1.78	22.78
1024	72.00	6421	1.21e-12	1.40e-11	4.25	16.94
2048	187.60	19980			9.38	20.01

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

TABELLE 4

Laufzeitverhalten von Algorithmus B4 bei doppelter Genauigkeit

N	Eval msecs	Init msecs	E_∞	E_2	FFT msecs	Faktor
128	6.20	484	7.99e-13	2.71e-12	0.35	17.71
256	18.80	1078	1.09e-12	5.59e-12	0.78	24.10
512	34.40	2453	9.84e-13	8.36e-12	1.79	19.21
1024	90.60	6438	1.03e-12	8.42e-12	4.26	21.26
2048	165.60	20094	6.03e-12	2.56e-11	9.39	17.64

Die Laufzeit der FFT ist nur zu Vergleichszwecken angegeben.
Faktor ist der Quotient aus Eval und FFT

Die Algorithmen sind ungefähr um den Faktor 20 langsamer als eine FFT gleicher Länge. Damit sind für die direkte Berechnung von F und F^T die beiden in Kapitel 2 vorgestellten Algorithmen A1 und A2 klar effizienter, zumal diese auch noch eine höhere Genauigkeit erreichen.
Die Algorithmen B3 und B4 ermöglichen es aber, auch die Inverse einer diskreten Fourier-Transformation bei nicht äquidistanten Stützstellen mit vertretbarem Zeitaufwand zu berechnen.
Die schnelle Multipolentwicklung kann aber auch in anderen Bereichen sinnvoll angewendet werden, wo Summen über geeignete Kernel-Funktionen schnell berechnet werden müssen.

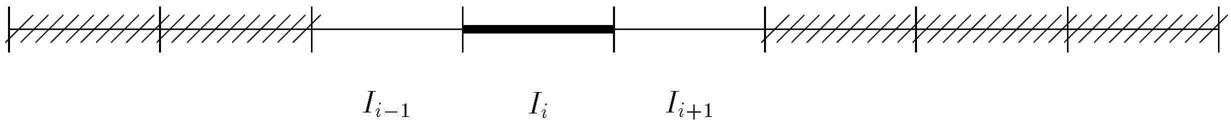

Abbildung 3.3: lokale Erweiterung Ψ^i

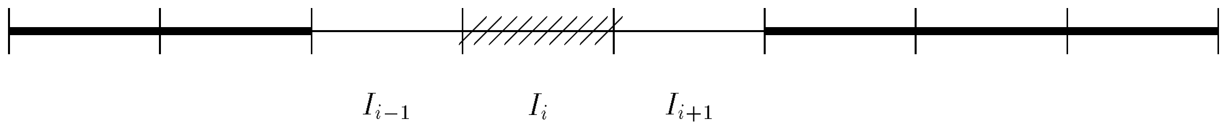

Abbildung 3.4: Far-Field Erweiterung Φ^i

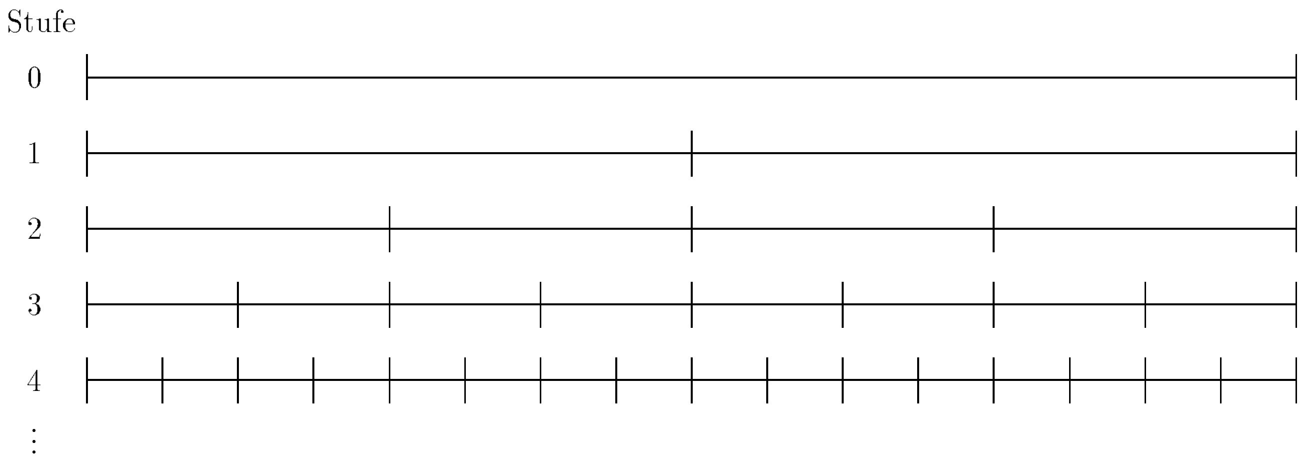

Abbildung 3.5: Aufteilung des Intervalls auf unterschiedlichen Stufen

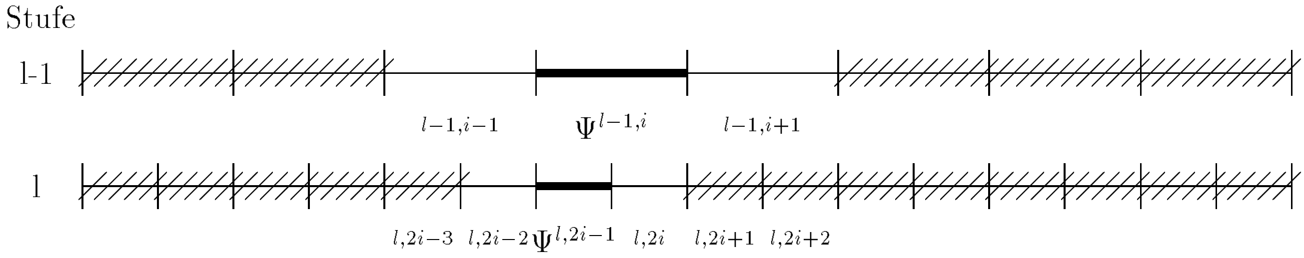

Abbildung 3.6: lokale Erweiterung Ψ auf zwei Stufen

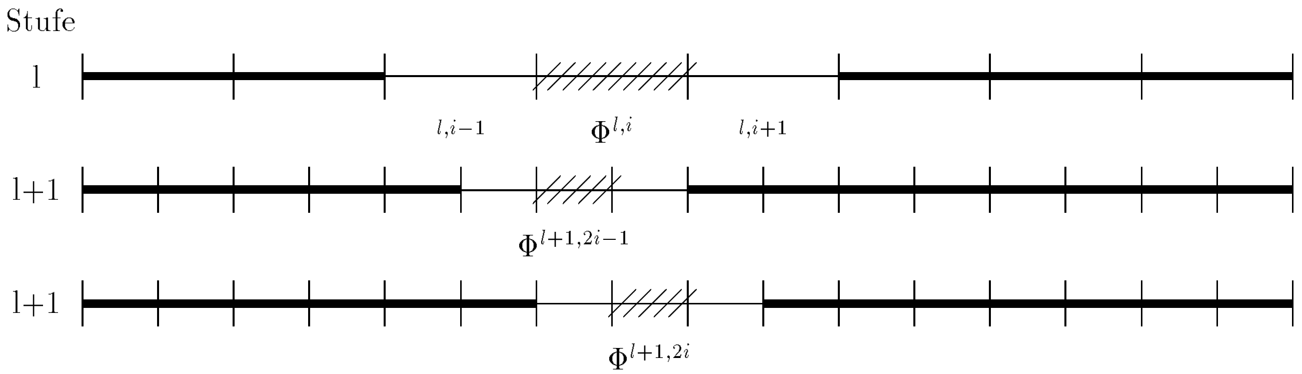

Abbildung 3.7: far-field Erweiterung Φ auf zwei Stufen

Anhang A

C++

Der C++ Sprachstandard steht wohl mittlerweile definitiv fest, obwohl die endgültige Verabschiedung erst Ende 1999/Anfang 1999 erfolgen wird.
Im folgenden wird eine kurze Auswahl an für Numeriker interessanten Eigenschaften von C++ vorgestellt.

A.1 Ein Überblick über die C++ Standard Template Library (STL)

Die C++ Standard Template Library (im folgenden kurz mit STL bezeichnet) beinhaltet „Container“ zum Speichern von Datenmengen sowie „Algorithmen“, die auf diesen Containern Operationen durchführen. Die STL ist auf ein optimales Laufzeitverhalten abgestimmt. Deshalb wählt Sie einen nicht objektorientierten Ansatz und verzichtet auf Techniken wie Polymorphie (virtuelle Funktionen) oder Vererbungsmechanismen. Daten (in der STL vor allem Container) und Algorithmen sind strikt voneinander getrennt. Dies hat den Vorteil, daß man die vorliegenden Algorithmen auch problemlos auf eigene Datenstrukturen anwenden kann.
Das grundlegende Konzept der STL sind Templates (weshalb der Begriff Template ja auch in den Namen übernommen wurde). Templates ermöglichen es den Containern, beliebige Datensätze zu speichern und den Algorithmen, mit beliebigen Containern zusammenzuarbeiten. Da Templates zur Compile-Zeit (optimierbaren) Quellcode erzeugen, sind auf Templates basierende Algorithmen, anders als ihre auf Polymorphie und virtuellen Funktionen aufbauenden objektorientieren Gegenstücke, genauso effizient wie direkt in C implementierte Routinen.

Das Bindeglied zwischen Containern und Algorithmen stellen die sogenannten Iteratoren dar. Diese ermöglichen das Iterieren über und den Zugriff auf die Elemente eines Containers. Iteratoren sind genauso wie Pointer zu handhaben und tatsächlich sind die Iteratoren beispielsweise eines STL-Vectors nichts anderes als Pointer. Die wesentlichen Operationen auf Iteratoren sind folglich Inkrementierung, Dekrementierung und Derefernzierung, also Ausdrücke der Art: `++Iterator, --Iterator` und `*Iterator`. Bei Iteratoren für „Random-Access-Container" wie beispielsweise Vektoren kommt noch Iterator-Arithmetik wie `Iterator=Iterator+10` hinzu. Für ein erstes Verständnis der STL (besonders wenn man sich auf die für Numeriker interessante Klasse vector beschränkt) reicht es völlig aus, sich Iteratoren als Pointer vorzustellen.
Man kann die Algorithmen der STL sogar auf normale C-Arrays anwenden. In diesem Falle übernehmen normale Pointer die Funktion der Iteratoren. Da die STL auf Templates aufbaut, ist dies ohne Laufzeiteinbußen gegenüber direkt auf Arrays aufbauenden Algorithmen möglich.
Fast alle in der STL vorhandenen Algorithmen erwarten die Angabe eines Bereiches, auf dem der Algorithmus arbeiten soll. Dieser Bereich wird durch einen Iterator für den Beginn des Bereichs und einen für das Ende spezifiziert. Zu beachten ist hierbei, daß der Iterator für das Ende selbst nicht mehr zum angegebenen Bereich gehört, sondern auf das nächste Element dahinter verweist. Der Bereich ist also gleichsam ein halboffenes Intervall, und tatsächlich wird in der STL-Litertaur häufig die Schreibweise [begin,end) verwendet. Ein Code-Beispiel:

```
#include <algorithm>
using namespace std;
int main()
{
   double Temp[1000];
   sort(Temp,Temp+1000); // sortiert das gesamte Array
}
```

Abgesehen davon, daß das Array nicht initialisiert wurde, ist dieses Programm so lauffähig. Zu beachten ist, daß Temp+1000 nicht auf das letzte Array-Element verweist, sondern auf das Element danach. Dereferenzierung hiervon, etwa mit `*(Temp+1000)` ist nicht zulässig. Nebenbei sei noch bemerkt, daß der C++ Standard nicht mehr zwingend einen Rückgabewert für die *int main*-Prozedur erwartet. Es wird ggf. automatisch ein `return 0;` eingefügt, das deshalb nicht mehr zwingend angegeben werden muß.
Weitere Algorithmen der STL:

```
sort(source_begin,source_end);
        // sortiert den Angegebenen Bereich
copy(source_begin,source_end,dest_begin);
        // kopiert den source-Bereich nach dest
fill(begin,end,Wert);
        // fuellt den angegebenen Bereich mit dem Wert ,,Wert''
swap_ranges(range1_begin, range1_end,range2_begin);
        // vertauscht die Inhalte der Bereiche
```

Die für Numeriker wohl mit Abstand wichtigste Datenstruktur ist der Vektor: Der Template-Typ `vector` stellt ein Array mit Elementen eines Typs dar. Die Anwendung eines Vektor wird im folgenden Code-Beispiel verdeutlicht:

```
#include <vector>
#include <algorithm> // fuer sort
int main()
 { vector<double> vector(10); // vector containing 10 doubles

   for(int i=0;i<vector.size();++i)
      vector[i]=-2.2*i; // element access: same as for c arrays: v[i] = x
   swap(vector.begin(),vector.begin()+5;vector.begin()+5);
                     // swaps first and second half of vector

   sort(vector.begin(),vector.end()); // sorts whole vector
   fill(vector.begin(),vector.begin()+3,7.1); // sets first 3 eles to 7.1
   sort(vector.begin(),vector.begin()+4); // sorts first 4 elements

   vector.insert(4.4)     // insert at end
   cout << vector.size(); // size is now 11
   vector<double> v2;   // init empty vector
   v2=v;    // automatically allocates memory, v2 has now 11 elements
  }
```

Die Elementfunktionen `begin()` und `end()` liefern hierbei Iteratoren (lies: Pointer) auf den Anfang und das Ende (genauer: das Element hinter dem letzten) des Vektors zurück. Damit kann man dann die Algorithmen der STL auf einen Vector anwenden. Zu beachten ist, daß Vektoren Datenstrukturen sind, von ihrer Natur her also eher einem C-Array als

einem mathematischen Vektor entsprechen. So existieren beispielsweise keine überladenen Operatoren, um Vektoren zu addieren oder zu subtrahieren.
In unserer Implementierung dienen STL-Vektoren dann auch als Container zur Speicherung von Matrizen oder mathematischen Vektoren, sie stellen aber nicht selbst die mathematischen Vektoren dar. Diese Aufgabe ist eher den im folgenden Kapitel betrachteten valarrays zugedacht.
Man sollte beachten, daß die Algorithmen der STL strikt von den Containern getrennt sind. Wie oben beschrieben, kann man Routinen wie `sort` auch auf ganz normale C-Arrays anwenden und ist deshalb nicht zwingend auf den STL-Datentyp `vector` angewiesen.
Eine vollständigere Beschreibung der STL findet sich z.B. in [3].

A.2 Ein Blick in die Zukunft: valarrays

Einer der wesentlichen Gründe, warum C bzw. C++ Fortran bei der Implementierung numerischer Programme bisher nicht vollständig ersetzt hat, liegt in dem sogenannten *Pointer-Aliasing*-Problem.
Man betrachte das folgende Codefragment einer Matrizenmultiplikation:

```
for k = 1..n
   c[i][j]+=a[i][k]*b[k][j];
```

In Fortran kann der Compiler davon ausgehen, daß a, b und c grundsätzlich verschiedene Arrays darstellen, die sich im Speicher nicht überlappen. Die Berechnung des ersten Schrittes `c[i][j]+=a[i][1]*b[1][j]` ist deshalb völlig unabhänging vom zweiten Schritt `c[i][j]+=a[i][2]*b[2][j]`. Deshalb kann der Compiler diverse Optimierungen vornehmen, z.B. die Aufgabe auf mehrere CPUs verteilen oder `c[i][j]` in einem CPU-Register halten.
In C hingegen können sich a,b und c überlappen (zumindest theoretisch, der C-Compiler hat aber zur Compile-Zeit keine Möglichkeit das herauszufinden). Deshalb könnten beispielsweise `c[i][j]` und `b[2][j]` auf dieselbe Speicherstelle verweisen. Deshalb kann `c[i][j]` nicht in einem Register gespeichert werden und die einzelnen Schritte können auch auf einem Vektorrechner nicht parallel ausgeführt werden.
Dieses Problem ist schon länger bekannt, bereits bei der Normierung der Sprache C (nicht C++) durch das Ansi-Komittee wurde ein Vorschlag gemacht, mittels eines `noalias` Bezeichners dem Compiler mitzuteilen, daß zwei Speicherbereiche sich nicht überlappen. Leider kam der Vorschlag erst kurz vor der Verabschiedung des Standards und war noch nicht ausgereift – nur deshalb wurde er nicht angenommen.

Bei der Standardisierung von C++ wurde nun ein erneuter Versuch unternommen, dieses Problem zu beheben: durch Einführung einer template-Klasse `valarray`. Hierzu wird in der vorläufigen Version des Ansi C++ Standards gesagt:

> The template class valarray<T> is a one-dimensional smart array, with elements numbered sequentially from zero. It is a representation of the mathematical concept of an ordered set of values. ...
>
> The intent is to specify an array template that has the minimum functionality necessary to address aliasing ambiguities and the pro- liferation of temporaries. Thus, the valarray template is neither a matrix class nor a field class. However, it is a very useful building block for designing such classes. [2]

Der Funktionsumfang eines valarrays umfaßt die Standardoperationen, die auf einem (mathematischen) Vektor sinnvoll sind:

```
#include<iostream>
#include<valarray>
int main()
 { valarray<double> a(10),b(10),c(10); // three valarrays,
                                       // each containing 10 elements
   a.fill(1.2);  // sets each element in a to 1.2
   b=1.2;        // does the same, but looks nicer
   c=a+b;        // now each element in c is 2.4
   c[0]=5;       // sets first element to 5
   c[9]=3;
   if(a==b)      // a[j] must be == b[j] for all j=0..9
     cout << "equal!";
   if(a<=b)      // dito, compares each element
     cout << "less!";
   c*=a;         // element-wise multiplication
 }
```

Die valarray-Klasse ist auf bestmögliches Laufzeitverhalten optimiert, wie im C++ Standard gesagt, soll sie eher die Basis für Matrix- und Vektorklassen sein, als selbst direkt in Applikationen verwendet zu werden.
So verändert z.B. ein einmal angelegtes Valarray nicht mehr seine Größe, was besonders bei Zuweisungen zu merkwürdigen Effekten führen kann:

```
valarray<double> a(5),b(10);
a=b; // undefined, may crash!
```

Außerdem stellen valarrays zwar theorethisch lediglich eine Bibliotheken-Klasse dar, erfordern also keine zusätzlichen Sprachelemente – trotzdem setzt eine effiziente Implementierung eines valarrays zahlreiche Sprachkonstrukte voraus, die erst in sehr wenigen Compilern realisiert sind (z.B. default template parameter).
Deshalb existiert bisher noch keine vollständige portable valarray-Implementierung; auf darauf aufbauende Matrix- Klassen wird man also noch ein wenig warten müssen.
Ich habe deshalb für die Implementierung der Algorithmen eigene Klassen `TMatrix` und `TVector` verwendet. Diese stellen dieselben Grund-Funktionalitäten wie valarrays bereit, verfügen aber auch über erweiterte Operationen wie Matrix-Vektor Multiplikationen und lassen sich zudem zur Laufzeit in der Größe verändern. Wenn erst einmal vallarray-Implementierungen verfügbar sind, können TMatrix und TVector so modifiziert werden, daß sie intern valarrays verwenden.

A.3 Komplexe Zahlen

Komplexe Zahlen sind im C++ Standard als Template-Typen definiert, der Template-Typ gibt an, welche Darstellung für den Real- und Imaginärteil verwendet wird. Beispiele hierfür sind:

```
complex<double> a(5.2,3.2); // a is 5.3+i*3.2
complex<float>  b(2.1,2.1); // 2.1+i*2.1
complex<int>    c(1,3);    //  1+i*3
```

In Normalfall wird man also `complex<double>` verwenden. Mit diesen Typen kann dann genauso wie mit normalen `double`-Werten gerechnet werden:

```
complex<double> a(1,2);
complex<double>  b(3,4);
complex<double> c;
c=a+b;
c*=(a-b);
c*=3.14; // complex-real multiplication uses only 2flops
```

Das Laufzeitverhalten ist ebensogut wie bei manueller Implementierung durch zwei double-Werte für eine komplexe Zahl. Insbesondere erfordert eine Mutliplikation einer reellen Zahl mit einer komplexen Zahl lediglich zwei reelle Rechenoperationen. Insgesamt besteht also keinerlei Notwendigkeit mehr, komplexe Zahlen „zu Fuß" zu programmieren.

Anhang B

Die Implementierung

B.1 Die Klassen TMatrix und TVector

Die Klasse TVector bildet die Basis für sämtliche Implementierungen. Sie ist so angelegt, daß man sie verwenden kann, ohne sich um Speicherzuweisungen oder ähnliche Dinge kümmern zu müssen: Ein Vektor sollte letztlich so unproblemtatisch zu verwenden sein, wie ein einfacher float-Wert.
Das folgende Code Beispiel verdeutlicht die Verwendung von Vektoren:

```
#include "tvector.h"
{
   TVector<double> a(10); // zero based vector with 10 elements
   TVector<double> b(5);  // 5 elements

   a=2.0 // sets all elements to 2
   b=a;  // b now has 10 elements
   TVector<double> c=b;
   c(0)=4;  // element access is done by (), not []
   c(1)=5.2;
   c(9)=-4.1;
   sort(c.begin(),c.end()); //STL algorithm: sort

   c=a+b;

   cout << "Vector c:" << c; // output to screen is often useful
```

```
   // TVectors may have other base-indices than 0:
   TVector<float> d(10,-5); // Vector with 10 elements, indices: -5..4

   for(int j=d.ib();j<d.ie();++j) // ib() is lowest index(index_begin):-5
                                  // ie() is index end, i.e. maximum index+1: 5
      d(j)=j;
   cout << " Dim:" << d.Dim();     // Dimension of d

   float *dd=d.Data();            // dd is standard c array: d[0]..d[9]

   d.SetBase(5);                 // now d is 5..14, without change of data

   cout << " MaxNorm:" << d.NormMax(); // maximum Norm of d
   cout << " Diff MaxNorm:" << NormMax(a-b); // alternative version
}
```

Die Klasse TMatrix ist ähnlich angelegt:

```
#include "tvector.h"
#include "tmatrix.h"
{
   TMatrix<double> A(10,10); // indices 0..9 x 0..9
   TMatrix<double> B; // creates empty matrix - dimension 0

   A=3.0; // fills matrix with one element

   for(int i=A.RowBase();i<=A.RowMax();++i)
      for(int j=A.ColBase();j<=A.ColMax();++j)
          A(i,j)=i+j;   // element access via A(Row,Col)

   B=A; // auotmatically allocates memory for B
   A.Invert();
   TMatrix<double> C=A*B; // should be identity matrix

   cout << "C:" << C;  // stream output
   C=A+B;
```

```
   TVector<double> v(10); // vector

   c=C*v;  // matrix-vector multiplication


}
```

B.2 Die Verwendung der Algorithmen

Insgesamt sind von mir sieben Algorithmen implementiert worden:

- die beiden Algorithmen aus Kapitel 2: A1 und A2.
- die vier Algorithmen B1-B4 aus Kapitel 3, wobei allerdings gegenüber B1 und B2 immer A1 und A2 der Vorzug gegeben werden sollte.
- die schnelle Multipolentwicklung (FMM) für $1/tan(x-y)$ sowie $1/(x-y)$.

Alle Algorithmen wurden als Template-Klassen implementiert und mit einem identischen Interface versehen.
Alle Initialisierungen werden durch den Klassen-Konstruktor durchgeführt. Die eigentliche Auswertung erfolgt über den überladenen ()-Operator. Schließlich steht bei allen Algorithmen noch eine DirectEval-Methode zur Verfügung, um ggf. das Ergebnis zu überprüfen.
Die Code-Beispiele in den folgenden Unterabschnitten verdeutlichen die Anwendung.

B.2.1 Das direkte Problem

Zuerst wollen wir einen Algorithmus zur Auswertung der folgenden DFT verwenden:

$$F(\alpha)_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{ikx_j} \tag{B.1}$$

für $j = 1, \ldots, N$. Die $x_1, \ldots, x_N$ sind die frei in $[-\pi, \pi]$ verteilten Stützstellen; sie sollen allerdings aufsteigend sortiert sein, also $x_k < x_l$ für $k < l$.
Der Algorithmus wurde in Kapitel 2 als Algorithmus A1 beschrieben. Er ist in der Klasse neFFT implementiert, die in der Datei nefft.h definiert wird.
Eine Anwendung des Algorithmus sieht also wie folgt aus:

```
#include <algorithm>
#include <iostream.h>
#include <math.h>
#include <complex>

#include "test.h" // to create random nodes
#include "nefft.h"

int main()
{  const int N=1024;
   TVector<double> Nodes(N); // nodes for DFT
   RandomSCirclePoints(N,Nodes); // create sorted random nodes

   neFFT<double> DFT(Nodes); // init class neFFT, does pre-calculations

   TVector<complex<double> > alpha(N); // complex input vector;
   FillRandom(Nodes);   // some values

  TVector<complex<double> > Result;

  Result=DFT(alpha); // operator() does the job

  // debugging: test result

  TVector<complex<double> >DirectResult;

  DirectResult=DFT.DirectEval(alpha); // direct calculation

  cout << "Error in MaxNorm: " << NormMax(DirectResult-Result) << endl;

  complex<double> *p=Result.Date();
  // for those who don't want to use TVector: result as c-array
}
```

Ein Wort noch zu den Indizes für die Eingangs- und Ausgangsvektoren: Laut Formulierung der Problemstellung besitzt der Eingangsvektor α die Indizes $j = -N/2, \ldots, N/2$. Die Klasse TVector unterstützt Vektoren, die nicht bei einem Index von 0 beginnen. Allerdings ist die Klasse neFFT roubust genug, auch Vektoren mit anderen Index-Bereichen, also auch die standardmäßigen nullbasierten Vektoren zu akzeptieren. Man braucht sich deshalb um die Index-Basis der Eingangsvektoren keine Sorgen zu machen. Zu beachten ist aber, das der Ausgangsvektor immer mit dem in der Problemstellung angegebenen Indexbereich versehen wird, hier also $1, \dots, 1024$.

B.2.2 Das transponierte Problem

Jetzt wollen wir die folgende DFT berechnen:

$$F^T(\alpha)_j = \sum_{k=1}^{N} \alpha_k \cdot e^{ij\omega_k}. \tag{B.2}$$

für $j = -N/2, \dots, N/2 - 1$. $\omega_1, \dots, \omega_N$ sind jetzt die frei in $[-\pi, \pi]$ verteilten Frequenzen. Der Algorithmus wurde in Kapitel 2 als Algorithmus A2 vorgestellt. Die entsprechende Klasse lautet tneFFT und befindet sich in der Include-Datei nefftt.h. Man verwendet sie genauso wie neFFT:

```
#include <algorithm>
#include <iostream.h>
#include <math.h>
#include <complex>

#include "test.h" // to create random nodes
#include "nefftt.h"

int main()
{  const int N=1024;
   TVector<double> omega(N); // frequencies for DFT
   RandomSCirclePoints(N,omega); // create sorted random nodes

   neFFT<double> DFT(omega); // init class neFFT, does pre-calculations

   TVector<complex<double> > alpha(N); // complex input vector;
   FillRandom(Nodes);   // some values

  TVector<complex<double> > Result;
```

```
  Result=DFT(alpha); // operator() does the job

}
```

Auch hier könnte man das Ergebnis wieder mittels DFT.DirectEval(alpha) überprüfen.

B.2.3 Die inversen Probleme

Schließlich betrachten wir noch die Aufgabe, $F^{-1}(\alpha)$ sowie $(F^T)^{-1}(\alpha)$ zu berechnen. Die entsprechenden Algorithmen B3 und B4 beginden sich als FMM_iFFT und FMM_itFFT in den Dateien fmm_ifft.h und fmm_itfft.h. Man verwendet sie wieder wie folgt:

```
#include <algorithm>
#include <iostream.h>
#include <math.h>
#include <complex>

#include "test.h" //  to create random nodes
#include "fmm_ifft.h"
#include "fmm_itfft.h"

int main()
{  const int N=1024;
   TVector<double> nodes(N); // nodes/frequencies for DFT
   LightRandomSymNodes(N,nodes); // random, but not too close to each other

   FMM_iFFT<double> iDFT(nodes); // init may take some time

   FMM_itFFT<double> itDFT(nodes); // also init itDFT

   TVector<complex<double> > alpha(N); // complex input vector;
   FillRandom(Nodes);   // some values

  TVector<complex<double> > iResult,itResult;

  iResult=iDFT(alpha);
```

```
  itResult=itDFT(alpha);

}
```

B.2.4 Der Algorithmus zur schnellen Multipolentwicklung

Der Algorithmus zur schnellen Multipolentwicklung berechnet Summen der Art

$$f(y_l) = \sum_{k=1}^{N} \alpha_k \cdot \frac{1}{\tan(y_l - x_k)} \tag{B.3}$$

Die entsprechende Template Klasse FMM ist in fmm.h definiert. Template-Parameter ist in diesem Falle nicht bloß der Datentyp, sondern eine Kernel-Klasse, die die Abhängigkeiten von der Kernel-Funktion $1/\tan(x-y)$ kapselt; für den Datentyp double sind die beiden Typen FMM_1tanx und FMM_1x allerdings schon vordefiniert. Einzelheiten entnimmt man den Quellcode in fmm.h.
Eine einfache Beispielanwendung sieht wie folgt aus:

```
#include <algorithm>
#include <iostream.h>
#include <math.h>
#include <complex>

#include "fmm.h"

typedef TVector<double> RVector; // real vector
typedef TVector<complex<double> > CVector; // complex vector

// UnequalNodes creates sortet Nodes with x_j!=y_k for all j,k
void UnequalNodes(RVector &x,RVector &y,double lbound,double rbound)
{
   double width=(rbound-lbound)/(x.Dim()+1)/2.0;
   double w=lbound;
   for(int i=x.Base();i<=x.Max();++i)
    { x(i)=w;
      w+=width;
      y(i)=w;
      w+=width;
```

```
    }
}

int main()
{
   const int N=1024;
   RVector xNodes(N,1);
   RVector yNodes(N,1);


   UnequalNodes(xNodes,yNodes,-M_PI_2,M_PI_2);


   CVector Alpha(N,1);
   FillRandom(Alpha);

   FMM_1tanx FMM(xNodes,yNodes);

   CVector Result(N,1);

   Result=FMM(Alpha);

   CVector DirectResult=FMM.DirectEval(Alpha); // Debugging

}
```

Wenn man statt der Kernel-Funktion $1/\tan(x-y)$ die Funktion $1/(x-y)$ verwenden will, so nimmt man statt der Klasse FMM_1tanx einfach die Klasse FMM_1x. Die Knoten dürfen dann allerdings nur noch aus dem Intervall $[-1,1]$ stammen.

In Wahrheit besteht aber natürlich das Dasein mehr als zur Hälfte nicht aus Handlungen, sondern aus Abhandlungen und aus der aufgestapelten Unpersönlichkeit dessen, was man gehört hat und weiß.

Robert Musil

Literaturverzeichnis

[1] J. W. Cooley and J. W. Tukey: *An algorithm for the machine computation of complex Fourier series,* Math. Comp 19, 1965.

[2] Iso/Ansi: *Commitee Draft (Doc No: X3J16/96-0225 WG21/N1043),* Definition des Vorläufigen C++ Standards.

[3] Nicolai Josuttis: *Die C++ Standardbibliothek.* Enthält eine knappe aber gute Einführung in die C++ Standard Template Library (STL) und berücksichtigt auch Valarrays. Addison-Wesley, Bonn, 1996.

[4] A. Dutt and V. Rokhlin: *Fast Fourier transforms for nonequispaced data,* SIAM J. Sci. Statist. Comput Vol 14, 1368-1393

[5] A. Dutt and V. Rokhlin: *Fast Fourier transforms for nonequispaced data, II,* Applied and Computational Harmonic Analysis 2, 85-100, 1995.

[6] J. Carrier, L. Greengard, and V. Rokhlin: *A fast adaptive multipole algorithm for particle simulations,* SIAM J. Sci. Statist. Comput., 9 (1988), 669-686

[7] A. Dutt, M. Gu, and V. Rokhlin: *Fast Algorithms for Polynomial Interpolation, Integration, And Differentiation* SIAM J. Numer. Anal., Vol 33, No 5, pp 1689-1711, October 1996.

[8] H. R. Schwarz: *Numerische Mathematik* B. G. Teubner, Stuttgart, 1988

[9] W. Nolting: *Grundkurs: Theoretische Physik, Band 3: Elektrodynamik,* Verlag Zimmermann-Neufang, Ulmen, 1992.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and Brian P. Flannery *Numerical Recipes in C* Cambridge University Press, 1992

Hiermit versichere ich, diese Arbeit eigentständig und ohne unzulässige Hilfsmittel angefertigt zu haben.