Universität Ulm
Fakultät für Mathematik und Wirtschaftswissenschaften

Stable Implementation of Three-Term Recurrence Relations

Bachelorarbeit

in Mathematik

vorgelegt von
Pascal Frederik Heiter
am 14. Juni 2010

Gutachter

Prof. Dr. Stefan A. Funken

Acknowledgement. First of all, I would like to thank my advisor, Prof. Dr. Stefan A. Funken, for his support, instructions and patience during my work. I achieved an interesting insight in mathematical research.
Moreover, I would like to thank Andreas Bantle for his strenuous efforts to support me, whenever I needed help, and for editing my thesis. Special thanks to Markus Bantle and Christoph Erath for answering many questions.
Last but not least I would like to thank my parents. Without their support, it would never have been possible to write this thesis.

Ulm, June 2010.

Contents

Chapter 1
Introduction

1.1 Motivation

Three-term recurrence relations become significant and useful in numerical mathematics, especially for the calculation of orthogonal polynomials such as Tschebyscheff, Jacobi, Laguerre, Hermite and Legendre polynomials. There exist different types of solutions of three-term recurrence relation, for example minimal and dominant solutions. However, the computation of a minimal solution by a given three-term recurrence relation is in general numerically instable, which is demonstrated by the following example.

Example 1.1.1 We consider

$$Q_k(x) := \frac{1}{2}\int_{-1}^{1} \frac{P_k(t)}{x-t}dt \qquad , \ x \in \mathbb{R} \ , \ k \in \mathbb{N}_0,$$

where $P_k(x)$ is the Legendre polynomial of degree k. The main aim is the calculation of these integrals for $k = 0, 1 \ldots n$. One possible solution is to calculate the partial fraction and integrate elementary functions, but the effort is too high. An alternative way to compute these integrals is given by the computation via the following three-term recurrence relations. It is desirable to have a fast, efficient and stable method. The $Q_k(x)$, also called Legendre functions, satisfy the same recurrence relation as the Legendre polynomials. There are no problems to evaluate this three-term recurrence relation for $x \in [-1, 1]$ as depicted in Figure 1.1 for $k = 30$.

As we see in Figure 1.2, the computation for $|x| > 1$ and $k \gg 1$ contrasts the analytical fact, that $\lim_{k\to\infty} Q_k(x) = 0$ $(|x| > 1)$. We can study the beginning of the oscillation, particularly the point, from which the relative error is larger than a given tolerance. Taking a look at the calculation of $Q_k(z)$ with $z \in \mathbb{C}$, we notice that the points, from which the oscillation begins, are located around an ellipse respecting to the real interval $[-1, 1]$, see also Chapter 3.

There are known several algorithms calculating numerically stable a minimal solution for a three-term recurrence relation, e.g. Miller's backward algorithm, Olver's algorithm and Gautschi's continued fraction algorithm, to name a few, but not all. Gautschi discusses three-term recurrence relations intensively in [5] and [6]. In [11]

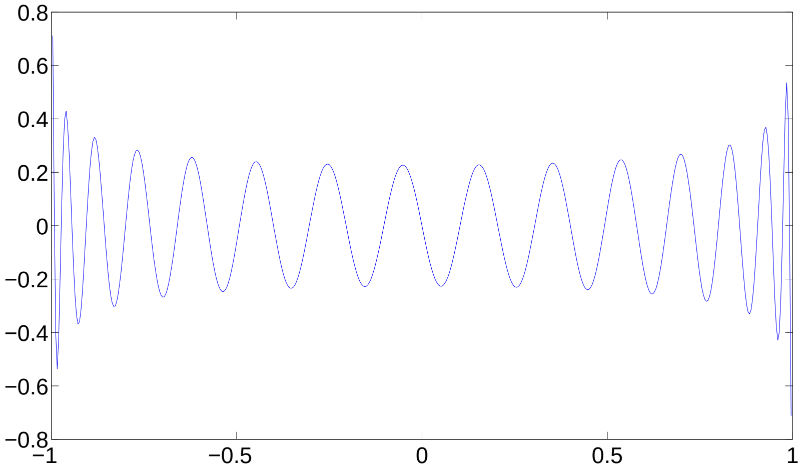

Figure 1.1: $Q_{30}(x)$ computed with a three-term recurrence relation for $x \in [-1, 1]$.

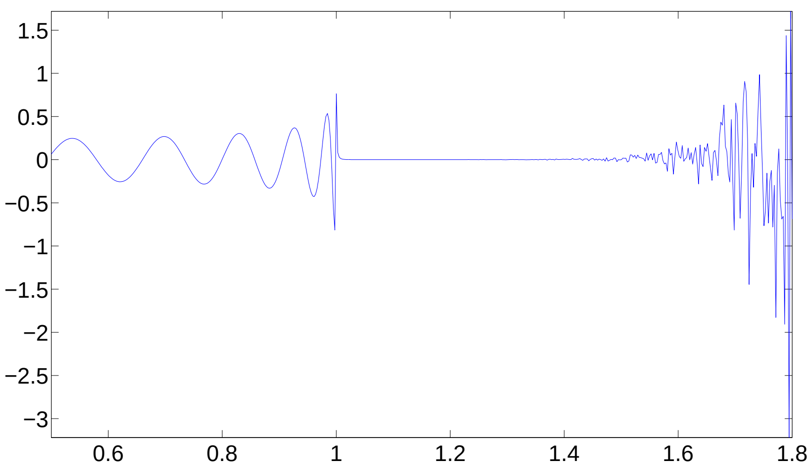

Figure 1.2: $Q_{30}(x)$ computed with a three-term recurrence relation for $x \in [0.5, 1.8]$.

Zhang and Jin present an algorithm to determine $Q_k(z)$. However, this algorithm is in general not stable yet.

An important application of three-term recurrence relations is the numerics of partial differential equations. As against many cases, in which it is just possible to solve the partial differential equation numerically, in a few cases, they can be solved analytically. There are various numerical ways to solve PDEs, e.g. with the finite element method or the boundary element method. Integral operators that occur in the boundary element method can be resolved to elementary integrals, which can be compute by three-term recurrence relations, see [1].

This thesis is embedded in a project, which deals with the efficient p-stablized implementation of the boundary element method. The results of this thesis allow us to create a useful library including many functions to compute integrals of the type

$$\int_{-1}^{1} \log|x-y|P_k(y)dy,$$

where $P_k(y)$ is the Legendre polynomial of degree k. We optimize Gautschi's continued fraction algorithm and use this algorithm to stabilize the calculation. We suggest to calculate these minimal solution with C functions instead of calculation with Matlab, because in C we reach more performance through parallelization e.g. via OpenMP or NVIDIA CUDA. Interfaces between Matlab and C are required, however there are not discuss in this work, see [2].

1.2 Aim of this Thesis

The aim of this work is to gain an efficient, fast and stable method to compute minimal solutions of three-term recurrence relations. The focus is on the investigation of three-term recurrence relations, the analysis of the area of instability, especially the assignment of the specific ellipse parameter and the stable implementation and the detailed description of the implementation.

1.3 Outline

The thesis is structured into five main chapters.
A short overview of the Legendre polynomials and functions is given in Chapter 2 as an example for a three-term recurrence relation. Furthermore, the second chapter contains the theory of three-term recurrence relations and special cases with the related initial values.
Chapter 3 deals with the stable numerical calculation of minimal solutions and introduces two algorithms, namely Miller's backward algorithm and Gautschi's continued fraction algorithm, stabilizing the computation of these solutions. The error analysis and the optimization of the Gautschi's continued fraction algorithm is discussed in the end of Chapter 3.
Chapter 4 includes a detailed description of the created C-Library 'liblegfct.c' and several test methods.

Chapter 2

Three-Term Recurrence Relations

In order to define some integrals, we give a short overview of Legendre functions and their properties. Afterwards, we take a look at three-term recurrence relations and prove, that the integrals, we defined before, satisfy three-term recurrence relations. In fact, we gain a method to compute these integrals without being forced to integrate all terms. In the end of this chapter, we calculate the initial values of these three-term recurrence relations.

2.1 Legendre Polynomials and Functions

The differential equation

$$(1-z^2)\frac{d^2}{dz^2}u(z) - 2z\frac{d}{dz}u(z) + k(k+1)u(z) = 0 \ , \quad k \in \mathbb{N}_0, \tag{2.1}$$

has the Legendre polynomials $P_k(z)$ and the Legendre functions $Q_k(z)$ as two linear independent solutions. The Legendre polynomials $P_k(z)$ are also called Legendre functions of 1. kind and the Legendre functions $Q_k(z)$ are called Legendre functions of 2. kind as depicted in Figure 2.1. Every solution of (2.1) can be represented as a linear combination of $P_k(z)$ and $Q_k(z)$, e.g.

$$u(z) = \alpha P_k(z) + \beta Q_k(z) \ , \quad \alpha, \beta \in \mathbb{C}.$$

The Legendre polynomials satisfy the three-term recurrence relation

$$(k+1)P_{k+1}(z) = (2k+1)zP_k(z) - kP_{k-1}(z) \ , \quad k \in \mathbb{N} \tag{2.2}$$

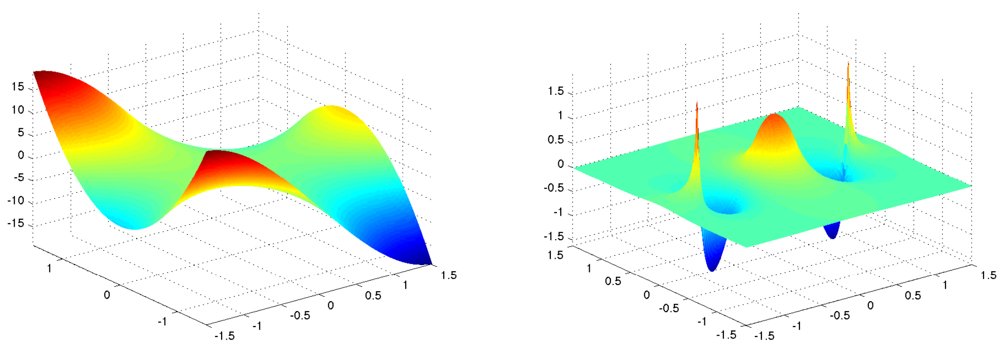

Figure 2.1: Plot of $P_3(z)$ (left) and $Q_3(z)$ (right).

which is well conditioned for $z \in \mathbb{R}$, $|z| < 1$ and with the initial values

$$P_0(z) = 1, \quad P_1(z) = z.$$

The Rodrigues formula

$$P_k(z) = \frac{1}{2^k k!} \frac{d^k}{dz^k}(z^2 - 1) , \quad k \in \mathbb{N}_0$$

is an alternative to represent the Legendre polynomials. Special cases of $P_k(z)$ are

$$\begin{aligned} P_0(z) &= 1, & P_3(z) &= \tfrac{1}{2}(5z^3 - 3z), \\ P_1(z) &= z, & P_4(z) &= \tfrac{1}{8}(35z^4 - 30z^2 + 3), \\ P_2(z) &= \tfrac{1}{2}(3z^2 - 1), & P_5(z) &= \tfrac{1}{8}(63z^5 - 70z^3 + 15z). \end{aligned}$$

The Legendre functions of 2. kind also satisfy the three-term recurrence relation (2.2). Another representation formula is given by

$$Q_k(z) = \frac{1}{2} \log\left(\frac{z+1}{z-1}\right) P_k(z) - W_{k-1}(z),$$

whereas the $W_k(z)$ satisfy the three-term recurrence relation

$$(k+1)W_k(z) = (2k+1)zW_{k-1}(z) - kW_{k-2}(z) , \quad k \in \mathbb{N}$$

with the initial values $W_{-1} = 0$, $W_0 = 1$. The functions $W_k(z)$ can be calculated without the previous three-term recurrence relation as well via

$$W_{k-1}(z) = \sum_{m=1}^{k} \frac{1}{m} P_{m-1}(z) P_{k-m}(z).$$

Special cases of $Q_k(z)$ are

$$\begin{aligned} Q_0(z) &= \tfrac{1}{2}\log\left(\tfrac{z+1}{z-1}\right) = \tfrac{1}{2}P_0(z)\log\left(\tfrac{z+1}{z-1}\right), \\ Q_1(z) &= z \cdot \tfrac{1}{2}\log\left(\tfrac{z+1}{z-1}\right) - 1 = \tfrac{1}{2}P_1(z)\log\left(\tfrac{z+1}{z-1}\right) - W_0(z), \\ Q_2(z) &= \tfrac{1}{2}(3z^2 - 1) \cdot \tfrac{1}{2}\log\left(\tfrac{z+1}{z-1}\right) - \tfrac{3}{2}z = \tfrac{1}{2}P_2(z)\log\left(\tfrac{z+1}{z-1}\right) - W_1(z). \end{aligned}$$

The associated Legendre functions of 2. kind are defined for $x \in [-1, 1]$ as

$$Q_k^m(x) = (-1)^m (1 - x^2)^{\frac{m}{2}} \frac{d^m}{dx^m} Q_k(x) , \quad m \in \mathbb{N}_0,$$

and for the general case $z \in \mathbb{C}$ as

$$Q_k^m(z) = (z^2 - 1)^{\frac{m}{2}} \frac{d^m}{dz^m} Q_k(z) , \quad m \in \mathbb{N}_0.$$

Note, that $Q_k(z) = Q_k^0(z)$ and for further information, see [7]. Some useful properties of the Legendre functions are given in the following lemma.

Lemma 2.1.1 *(i) There holds for the antiderivatives*

$$\int_{-1}^{t} P_k(\xi)d\xi = \frac{1}{2k+1}\left(P_{k+1}(t) - P_{k-1}(t)\right) , \quad k \in \mathbb{N}.$$

(ii) There holds for the derivatives

$$\frac{d}{dz}P_k(z) = \frac{k(k+1)}{2k+1}\frac{P_{k+1}(t) - P_{k-1}(t)}{z^2-1} , \quad k \in \mathbb{N}.$$

(iii) There holds

$$(P_{k+1} - P_{k-1})(\pm 1) = 0 , \quad k \in \mathbb{N}.$$

(iv) Based on the orthogonality of $P_k(t)$, there holds

$$\int_{-1}^{1} P_k(t)P_m(t)dt = \begin{cases} 0, & k \neq m \\ \frac{2}{2k+1}, & k = m \end{cases}$$

(v) For $|\arg(z-1)| < \pi$ we have the following relation between the Legendre functions of 1. and 2. kind

$$Q_k(z) = \frac{1}{2}\int_{-1}^{1} \frac{P_k(z)}{z-t}dt , \quad k \in \mathbb{N}_0.$$

(vi) There holds the functional relations over m and for $x \in [-1, 1]$

$$Q_k^{m+2}(x) = \frac{-2(m+1)x}{\sqrt{1-x^2}}Q_k^{m+1}(x) - (k+m+1)(k-m)Q_k^m(x)$$

and for the general case $z \in \mathbb{C}$, there holds the three-term recurrence relation over m

$$Q_k^{m+2}(z) = \frac{-2(m+1)z}{\sqrt{z^2-1}}Q_k^{m+1}(z) + (k+m+1)(k-m)Q_k^m(z).$$

Proof. See [7] and [8]. □

2.2 Three-Term Recurrence Relation

In the following we want to investigate three-term recurrence relations. A three-term recurrence relation is a specification how to compute a new value with the two previous values. There are two different types of three-term recurrence relation. An inhomogeneous three-term recurrence relation is defined as

$$y_{n+1} = a_n y_n + b_n y_{n-1} + c_n \tag{2.3}$$

and a uniform three-term recurrence relation is defined as

$$y_{n+1} = a_n y_n + b_n y_{n-1} \tag{2.4}$$

whereas $a_n, b_n, c_n \in \mathbb{C}$ are arbitrary numbers. The most popular example for a three-term recurrence relation is the Fibonacci numbers.

Example 2.2.1 The Fibonacci numbers satisfy the three-term recurrence relation

$$F_{n+1} = F_n + F_{n-1}\ , \quad n \in \mathbb{N}$$

with the initial values $F_0 = 0$ and $F_1 = 1$.
The first values are

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	$\ldots$
0	1	1	2	3	5	8	13	21	34	55	$\ldots$.

Note, that this three-term recurrence relation is uniform, see Equation (2.4) with $a_n = b_n = 1$.

Given a three-term recurrence relation

$$(k-n+1)f_{k+1+s}(z) = (2k+1)zf_{k+s}(z) - (k+n)f_{k-1+s}(z)\ , \quad k \in \mathbb{N},\ s, n \in \mathbb{Z}, \tag{2.5}$$

we want to transform these relation to gain a modified formula, such that the leading coefficients of all $f_{k+s}(z)$ are equal to one. This property is required by Gautschi's continued fraction algorithm, which is discussed in Chapter 3.

Lemma 2.2.2 *Let $n, s \in \mathbb{Z}$, $f_{k+s}(z)$ $(k \in \mathbb{N}_0)$ satisfy (2.5) and*

$$\tilde{f}_{k+s}(z) := \prod_{j=2}^{k} \frac{j-n}{2j-1} f_{k+s}(z).$$

Then there holds the three-term recurrence relation

$$\tilde{f}_{k+1+s}(z) = z\tilde{f}_{k+s}(z) - b_k \tilde{f}_{k-1+s}(z)\ , \quad k \in \mathbb{N}, \tag{2.6}$$

with

$$b_{k+s} = \frac{(k-n)(k+n)}{(2k-1)(2k+1)}.$$

Proof. We take a look at the leading coefficients of f_k

$$\begin{aligned}
f_{0+s}(z) &= 1\\
f_{1+s}(z) &= z\\
f_{2+s}(z) &= \frac{3}{2-n} z f_{1+s}(z) + (1+n) f_{0+s}(z) = \frac{3}{2-n} z^2 + \ldots
\end{aligned}$$

$$\begin{aligned}
f_{3+s}(z) &= \frac{5}{3-n} z f_{2+s}(z) + (2+n) f_{1+s}(z) = \frac{3\cdot 5}{(2-n)(3-n)} z^3 + \dots \\
f_{4+s}(z) &= \frac{7}{4-n} z f_{3+s}(z) + (3+n) f_{2+s}(z) = \frac{3\cdot 5\cdot 7}{(2-n)(3-n)(4-n)} z^4 + \dots \\
&\vdots
\end{aligned}$$

and obtain

$$f_{k+s}(z) = \prod_{j=2}^{k} \frac{2j-1}{j-n} z^k - \sum_{m=0}^{k-1} \alpha_m z^m. \tag{2.7}$$

We prove (2.7) by induction on k. For the initial step with $k=0$, there holds

$$f_{0+s}(z) = \prod_{j=2}^{0} \frac{j-n}{2j-1} f_{0+s}(z) = 1.$$

Let (2.7) be true for an arbitrary $k \in \mathbb{N}$, but fixed. Then there holds by using the induction hypothesis (2.7)

$$\begin{aligned}
(k+1-n) f_{k+1+s}(z) &= (2k+1) z f_{k+s}(z) - (k+n) f_{k-1+s}(z) \\
&= (2k+1) z \left(\prod_{j=2}^{k} \frac{2j-1}{j-n} z^k - \sum_{m=0}^{k-1} \alpha_m z^m \right) - (k+n) f_{k-1+s}(z) \\
&= (2k+1) \prod_{j=2}^{k} \frac{2j-1}{j-n} z^{k+1} - (2k+1) \sum_{m=0}^{k-1} \alpha_m z^{m+1} - (k+n) f_{k-1+s}(z)
\end{aligned}$$

This is equivalent to

$$\begin{aligned}
f_{k+1+s}(z) &= \frac{2k+1}{k+1-n} \prod_{j=2}^{k} \frac{2j-1}{j-n} z^{k+1} \underbrace{- \frac{2k+1}{k+1-n} \sum_{m=0}^{k-1} \alpha_m z^{m+1} - \frac{k+n}{k+1-n} f_{k-1+s}(z)}_{=:\sum_{m=0}^{k} \beta_m z^m} \\
&= \prod_{j=2}^{k+1} \frac{2j-1}{j-n} z^{k+1} - \sum_{m=0}^{k} \beta_m z^m.
\end{aligned}$$

To get a leading coefficient one, we have to multiply $f_{k+1+s}(z)$ by $\prod_{j=2}^{k+1} \frac{j-n}{2j-1}$. This leads to

$$\tilde{f}_{k+s}(z) = \prod_{j=2}^{k} \frac{j-n}{2j-1} f_{k+s}(z).$$

Now, we prove the three-term recurrence relation (2.6). Let

$$d_{k+s} = \frac{\Pi_{j=1}^{k}(2j-1)}{\Pi_{i=2}^{k}(i-n)}.$$

Then there holds

$$\begin{aligned}
d_{k+1+s}\tilde{f}_{k+1+s}(z) &= f_{k+1+s}(z) = \frac{2k+1}{k+1-n} z f_{k+s}(z) + \frac{k+n}{k+1-n} f_{k-1+s}(z) \\
&= \frac{2k+1}{k+1-n} d_k z \tilde{f}_{k+s}(z) + \frac{k+n}{k+1-n} d_{k-1}\tilde{f}_{k-1+s}(z) \\
&= d_{k+1+s} z \tilde{f}_{k+s}(z) + \frac{k+n}{k+1-n} d_{k-1+s}\tilde{f}_{k-1+s}(z).
\end{aligned}$$

By dividing the last equation by d_{k+1+s}, we get

$$\begin{aligned}
\tilde{f}_{k+1+s}(z) &= z\tilde{f}_{k+s}(z) + \frac{k+n}{k+1-n}\frac{d_{k-1+s}}{d_{k+1+s}}\tilde{f}_{k-1+s}(z) \\
&= z\tilde{f}_{k+s}(z) + \frac{(k+n)(k-n)}{(2k-1)(2k+1)}\tilde{f}_{k-1+s}(z),
\end{aligned}$$

which proves the assumption. □

Remark 2.2.3 *Let $k \in \mathbb{N}_0$, $s \in \mathbb{Z}$, $f_{0+s}(z) = 1$ and $f_{1+s}(z) = z$. Then $f_{k+s}(z)$ is a polynomial and the leading coefficient is equal to one.*

We will prove several three-term recurrence relation in the next chapter. To simplify the corresponding proofs, we show the following lemma.

Lemma 2.2.4 *Let $f_k(z)$ satisfy (2.5) and*

$$g_{k+s}(z) := \frac{f_{k+1}(z) - f_{k-1}(z)}{2k+1}.$$

Then there holds the three-term recurrence relation

$$(k-n+2)g_{k+1+s}(z) = (2k+1)zg_{k+s}(z) - (k+n-1)g_{k-1+s}(z)\ , \quad k \in \mathbb{N},\ s,n \in \mathbb{Z}.$$

Proof. By using the definition of $g_{k+s}(z)$ and the three-term recurrence relation of $f_k(z)$, we obtain

$$\begin{aligned}
g_{k+1+s}(z) &= \frac{f_{k+2}(z) - f_k(z)}{2k+3} \\
&= \frac{1}{2k+3}\left(\frac{2k+3}{k-n+2} z f_{k+1} - \frac{k+1+n}{k-n+2} f_k(z) - f_k(z)\right) \\
&= \frac{1}{2k+3}\left(\frac{2k+3}{k-n+2} z f_{k+1} - \frac{2k+3}{k-n+2} f_k(z)\right) \\
&= \frac{1}{k-n+2}\left(z f_{k+1} - f_k(z)\right).
\end{aligned}$$

We expand the last term with $\pm z f_{k-1}(z)$, use the three-term recurrence relation of $f_k(z)$ again, and obtain

$$\begin{aligned}
g_{k+1+s}(z) &= \frac{1}{k-n+2}\left(z f_{k+1} - z f_{k-1}(z) + z f_{k-1}(z) - f_k(z)\right) \\
&= \frac{1}{k-n+2}\Bigg(z(f_{k+1} - f_{k-1}(z)) + \frac{k-n}{2k-1} f_k(z) - f_k(z) \\
&\quad + \frac{k+n-1}{2k-1} f_{k-2}(z)\Bigg) \\
&= \frac{1}{k-n+2}\left(z(f_{k+1} - f_{k-1}(z)) - \frac{k+n-1}{2k-1} f_k(z) - \frac{k+n-1}{2k-1} f_{k-2}(z)\right) \\
&= \frac{1}{k-n+2}\left(z(f_{k+1} - f_{k-1}(z)) - \frac{k+n-1}{2k-1}(f_k(z) - f_{k-2}(z))\right).
\end{aligned}$$

In the end, we have to use the definition of $g_{k+s}(z)$ and get

$$g_{k+1+s}(z) = \frac{1}{k-n+2}\left((2k+1) z g_{k+s}(z) - (k+n-1) g_{k-1+s}(z)\right),$$

which is equivalent to

$$(k-n+2) g_{k+1+s}(z) := (2k+1) z g_{k+s}(z) - (k+n-1) g_{k-1+s}(z).$$

□

2.3 Special Cases of Three-Term Recurrence Relations

Let $P_k(x)$ be the Legendre polynomial with degree k. We use the notations

$$\tilde{Q}_k^{-1}(z) := \int_{-1}^{1} P_k(t) \log(t-z)\, dt\,, \quad k \in \mathbb{N}_0 \tag{2.8}$$

$$\tilde{Q}_k^{m}(z) := \int_{-1}^{1} \frac{P_k(t)}{(z-t)^{m+1}}\, dt\,, \quad k, m \in \mathbb{N}_0 \tag{2.9}$$

$$N_k(t) := \begin{cases} \frac{1}{2}(P_0(t) - P_1(t)) & k = 1 \\ \frac{1}{2}(P_0(t) + P_1(t)) & k = 2 \\ \int_{-1}^{t} P_k(\xi)\, d\xi & k > 2 \end{cases} \tag{2.10}$$

$$R_k^{-1}(z) := \int_{-1}^{1} N_k(t) \log(t-z)\, dt\,, \quad k \in \mathbb{N} \tag{2.11}$$

$$R_k^{m}(z) := \int_{-1}^{1} \frac{N_k(t)}{(z-t)^{m+1}}\, dt\,, \quad k \in \mathbb{N}, m \in \mathbb{N}_0. \tag{2.12}$$

In the following, we prove, that the functions, we defined above, satisfy three-term recurrence relations. In fact, we gain a fast method evaluating integrals with a complexity of $\mathcal{O}(n)$.

2.3.1 Three-Term Recurrence Relation for $\tilde{Q}_k^m(z)$

Remember the definitions

$$\begin{aligned}
\tilde{Q}_k^{-1}(z) &:= \int_{-1}^{1} P_k(t)\log(t-z)dt\ , \quad k \in \mathbb{N}_0,\\
\tilde{Q}_k^{m}(z) &:= \int_{-1}^{1} \frac{P_k(t)}{(z-t)^{m+1}}dt\ , \quad k,m \in \mathbb{N}_0.
\end{aligned}$$

In the following lemma, we give a useful property of $\tilde{Q}_k^m(z)$ and describe the relation between $Q_k^m(z)$ an $\tilde{Q}_k^m(z)$.

Lemma 2.3.1 *Let $z \in \mathbb{C}\backslash\{\pm 1\}$.*

(i) There holds the relation between Q_k^m and $\tilde{Q}_k^m$

$$Q_k^m(z) = \frac{(z^2-1)^{\frac{m}{2}}}{2} \cdot (-1)^m m! \tilde{Q}_k^m\ , \quad k \in \mathbb{N}_0, m \in \mathbb{N}.$$

(ii) There holds the relation between $\tilde{Q}_k^m$ and $\tilde{Q}_k^{m+1}$

$$\tilde{Q}_k^m(z) = \frac{-(m+1)}{2k+1}(\tilde{Q}_{k+1}^{m+1}(z) - \tilde{Q}_{k-1}^{m+1}(z))\ , \quad k \in \mathbb{N}, m \in \mathbb{N}_0 \cup \{-1\}.$$

Proof. ad (i) The definition of $Q_k^m(z)$ and property (v) of Lemma 2.1.1 leads to

$$Q_k^m(z) = (z^2-1)^{\frac{m}{2}}\frac{d^m}{dz^m}Q_k(z) = \frac{(z^2-1)^{\frac{m}{2}}}{2}\frac{d^m}{dz^m}\int_{-1}^{1}\frac{P_k(t)}{z-t}dt.$$

By derivating m times, we get

$$\begin{aligned}
Q_k^m(z) &= \frac{(z^2-1)^{\frac{m}{2}}}{2}\frac{d^m}{dz^m}\int_{-1}^{1}\frac{P_k(t)}{z-t}dt\\
&= \frac{(z^2-1)^{\frac{m}{2}}}{2} \cdot (-1)^m m! \int_{-1}^{1}\frac{P_k(t)}{(z-t)^{m+1}}dt\\
&= \frac{(z^2-1)^{\frac{m}{2}}}{2} \cdot (-1)^m m! \tilde{Q}_k^m.
\end{aligned}$$

ad (ii) Let $m > -1$. We use integration by parts and get

$$
\begin{aligned}
\tilde{Q}_k^m(z) &= \int_{-1}^{1} \frac{P_k(t)}{(z-t)^{m+1}} dt \\
&= \underbrace{\int_{-1}^{t} P_k(\xi) d\xi \cdot \frac{1}{(z-t)^{m+1}} \Bigg|_{-1}^{1}}_{=0} - \int_{-1}^{1} \left(\int_{-1}^{t} P_k(\xi) d\xi \right) \cdot \frac{m+1}{(z-t)^{m+2}} dt.
\end{aligned}
$$

Using property (i) of Lemma 2.1.1, we obtain

$$
\begin{aligned}
\tilde{Q}_k^m(z) &= -\int_{-1}^{1} \frac{1}{2k+1} \left(P_{k+1}(t) - P_{k-1}(t) \right) \frac{m+1}{(z-t)^{m+2}} dt \\
&= -\frac{m+1}{2k+1} \int_{-1}^{1} \frac{P_{k+1}(t) - P_{k-1}(t)}{(z-t)^{m+2}} dt \\
&= \frac{-(m+1)}{2k+1} \left(\tilde{Q}_{k+1}^{m+1}(z) - \tilde{Q}_{k-1}^{m+1}(z) \right).
\end{aligned}
$$

The proof for $m = -1$ can be done similarly. □

The three-term recurrence relation for $\tilde{Q}_k^m(z)$ is considered in the next theorem. The initial values are given in Chapter 2.4.

Lemma 2.3.2 *Let $m \in \mathbb{N}_0 \cup \{-1\}$ and $\tilde{Q}_k^m(z)$ as defined above. Then there holds the three-term recurrence relation over k*

$$
(k-m+1)\tilde{Q}_{k+1}^m(z) = (2k+1)z\tilde{Q}_k^m(z) - (k+m)\tilde{Q}_{k-1}^m(z)\ , \quad k \geq 1
$$

and another three-term recurrence relation over m

$$
\tilde{Q}_k^{m+1}(z) = \frac{2mz}{(z^2-1)(m+1)}\tilde{Q}_k^m(z) + \frac{(k+m)(k-m+1)}{(z^2-1)(m+1)m}\tilde{Q}_k^{m-1}(z)\ , \quad m \in \mathbb{N}, k \in \mathbb{N}_0
$$

with the initial values as in Lemma 2.4.3. Note, that for $m = -1$, the first three-term recurrence relation only holds for $k > 1$.

Proof. Let $k \geq 1$. Then there holds by using the three-term recurrence relation for Legendre polynomials

$$
\begin{aligned}
\tilde{Q}_{k+1}^m(z) &= \int_{-1}^{1} \frac{P_{k+1}(t)}{(z-t)^{m+1}} dt \\
&= \frac{1}{k+1} \int_{-1}^{1} \frac{(2k+1)tP_k(t) - kP_{k-1}(t)}{(z-t)^{m+1}} dt.
\end{aligned}
$$

We expand $tP_k(t)$ with $\pm zP_k(t)$ and obtain for the last equation

$$\begin{aligned}
\tilde{Q}^m_{k+1}(z) &= \frac{2k+1}{k+1}\left(\int_{-1}^{1}\frac{(t-z+z)P_k(t)}{(z-t)^{m+1}}dt\right) - \frac{k}{k+1}\int_{-1}^{1}\frac{P_{k-1}(t)}{(z-t)^{m+1}}dt \\
&= \frac{2k+1}{k+1}\left(-\int_{-1}^{1}\frac{P_k(t)}{(z-t)^{m}}dt + z\int_{-1}^{1}\frac{P_k(t)}{(z-t)^{m+1}}dt\right) - \frac{k}{k+1}\int_{-1}^{1}\frac{P_{k-1}(t)}{(z-t)^{m+1}}dt \\
&= \frac{2k+1}{k+1}\left(-\int_{-1}^{1}\frac{P_k(t)}{(z-t)^{m}}dt + z\tilde{Q}^m_k(z)\right) - \frac{k}{k+1}\tilde{Q}^m_{k-1}(z).
\end{aligned}$$

To complete the proof, we need the result of the previous Lemma 2.3.1 and obtain

$$\begin{aligned}
\tilde{Q}^m_{k+1}(z) &= \frac{2k+1}{k+1}\left(-\frac{-m}{2k+1}\left(\tilde{Q}^m_{k+1}(z)-\tilde{Q}^m_{k-1}(z)\right) + z\tilde{Q}^m_k(z)\right) - \frac{k}{k+1}\tilde{Q}^m_{k-1}(z) \\
&= \frac{m}{k+1}\tilde{Q}^m_{k+1}(z) - \frac{m}{k+1}\tilde{Q}^m_{k-1}(z) + \frac{2k+1}{k+1}z\tilde{Q}^m_k(z) - \frac{k}{k+1}\tilde{Q}^m_{k-1}(z).
\end{aligned}$$

Thus we have

$$(k+1)\tilde{Q}^m_{k+1}(z) - m\tilde{Q}^m_{k+1}(z) = (2k+1)z\tilde{Q}^m_k(z) - m\tilde{Q}^m_{k-1}(z) - k\tilde{Q}^m_{k-1}(z)$$

and

$$(k+1-m)\tilde{Q}^m_{k+1} = (2k+1)z\tilde{Q}^m_k(z) - (k+m)\tilde{Q}^m_{k-1}(z).$$

The proof for $\tilde{Q}^{-1}_{k+1}(z)$ can be done similarly. The three-term recurrence relation for $\tilde{Q}^m_k(z)$ over m can be proved by using Lemma 2.3.1 and Lemma 2.1.1 (vi). □

Remark 2.3.3 *The three-term recurrence relation for $\tilde{Q}^m_k(z)$ is a special case of (2.5) with $s=0$ and $n=m$.*

2.3.2 Three-Term Recurrence Relation for $N_k(t)$

Remember the definition

$$N_k(t) = \begin{cases} \frac{1}{2}(P_0(t)-P_1(t)), & k=1, \\ \frac{1}{2}(P_0(t)+P_1(t)), & k=2, \\ \int_{-1}^{t} P_k(\xi)d\xi, & k>2. \end{cases}$$

Lemma 2.3.4 *Let $N_k(t)$ as defined above. Then there holds*

(i) $N_1(t) = \frac{1}{2}(1-t)$

(ii) $N_2(t) = \frac{1}{2}(1+t)$

(iii) The $N_k(t)$ satisfies the three-term recurrence relation

$$(k+2)N_{k+3}(t) = (2k+1)tN_{k+2}(t) - (k-1)N_{k+1}(t)\ , \quad k \geq 2$$

with $N_3(t) = \frac{1}{2}(t^2-1)$ and $N_4(t) = \frac{1}{2}(t^3-t)$.

Note, that the $N_k(t)$ are called Lobatto shape functions.

Proof. We notice the relation

$$N_k(t) = \frac{1}{2k-3}(P_{k-1}(t) - P_{k-3}(t))\ , \quad k \geq 3$$

based on property (i) of Lemma 2.1.1. The Legendre polynomials satisfy (2.5). We define

$$g_{k+2}(z) := N_k(t)\ , \quad k \in \mathbb{N}$$

use Lemma 2.2.4 with $s = 2$, $n = 0$ and complete the proof. The initial values are treaded in Lemma 2.4.4. □

Remark 2.3.5 *The three-term recurrence relation for $N_k(t)$ is a special case of (2.5) with $s = 2$ and $n = -1$.*

2.3.3 Three-Term Recurrence Relation for $R_k^m(z)$

Remember the definitions

$$\begin{aligned} R_k^{-1}(z) &:= \int_{-1}^{1} N_k(t)\log(t-z)dt\ , \quad k \in \mathbb{N}, \\ R_k^m(z) &:= \int_{-1}^{1} \frac{N_k(t)}{(z-t)^{m+1}}dt\ , \quad k \in \mathbb{N}, m \in \mathbb{N}_0. \end{aligned}$$

We investigate the relation between $R_k^m(z)$ and $\tilde{Q}_k^m(z)$.

Lemma 2.3.6 *Let $m \in \mathbb{N}_0 \cup \{-1\}$. There holds the following relation between $R_k^m(z)$ and $\tilde{Q}_k^m(z)$*

$$R_{k+2}^m(z) = \frac{1}{2k+1}(\tilde{Q}_{k+1}^m(z) - \tilde{Q}_{k-1}^m(z))\ , \quad k \in \mathbb{N}.$$

Proof. Let $m \in \mathbb{N}_0$. Based on the definitions of $R_k^m(z)$ and $N_k(t)$ and property (i)

of Lemma 2.1.1, we get

$$\begin{aligned} R_{k+2}^m(z) &= \int_{-1}^{1} \frac{N_{k+2}(t)}{(z-t)^{m+1}} dt \\ &= \int_{-1}^{1} \frac{P_{k+1}(t) - P_{k-1}(t)}{2k+1} \frac{1}{(z-t)^{m+1}} dt \\ &= \frac{1}{2k+1} \left(\int_{-1}^{1} \frac{P_{k+1}(t)}{(z-t)^{m+1}} dt - \int_{-1}^{1} \frac{P_{k-1}(t)}{(z-t)^{m+1}} dt \right) \\ &= \frac{1}{2k+1} \left(\tilde{Q}_{k+1}^m(z) - \tilde{Q}_{k-1}^m(z) \right) . \end{aligned}$$

The proof for $m = -1$ can be done similarly. □

The last lemma allows us to simplify the proof of the three-term recurrence relation for $R_k^m(z)$.

Lemma 2.3.7 *Let $m \in \mathbb{N}_0 \cup \{-1\}$ and $R_k^m(z)$ as defined above. Then there holds the three-term recurrence relation*

$$(k - m + 2) R_{k+3}^m(z) = (2k + 1) z R_{k+2}^m(z) - (k + m - 1) R_{k+1}^m(z) \, , \quad k > 1.$$

with the initial values as in Lemma 2.4.5.

Proof. We already showed that $\tilde{Q}_k^m(z)$ satisfies (2.5). We define

$$g_{k+2}(z) = R_{k+2}^m(z).$$

Now, we can use Lemma 2.2.4 and proved the three-term recurrence relation

$$(k - m + 2) R_{k+3}^m(z) = (2k + 1) z R_{k+2}^m(z) - (k + m - 1) R_{k+1}^m(z) \, , \quad k > 1.$$

□

Remark 2.3.8 *The three-term recurrence relation for $R_k^m(z)$ is s special case of (2.5) with $s = 2$ and $n = m - 1$.*

2.4 Computation of Initial Values

In order to evaluate a three-term recurrence relation, we have to calculate the initial values. Therefore, we compute the initial values for the three-term recurrence relations, which are introduced before and we use a few identities to simplify the calculation.

Lemma 2.4.1 *There holds for $z \in \mathbb{C}\backslash\{\pm 1\}$*

$$\int_{-1}^{1} \frac{dt}{t-z} = -\log\left(\frac{z+1}{z-1}\right),$$

$$\int_{-1}^{1} \frac{t^k}{t-z}dt = \int_{-1}^{1} t^{k-1}dt + z\int_{-1}^{1} \frac{t^{k-1}}{t-z}dt$$

and

$$\int_{-1}^{1} \log(t-z)dt = (1-z)\log(1-z) + (1+z)\log(-1-z) - 2,$$

$$\int_{-1}^{1} t^k \log(t-z)dt = \frac{t^{k+1}}{k+1}\log(t-z)\Big|_{-1}^{1} - \frac{1}{k+1}\int_{-1}^{1} \frac{t^{k+1}}{t-z}dt$$

respectively.

Proof. With the antiderivative of $\frac{1}{t-z}$, there holds

$$\int_{-1}^{1} \frac{dt}{t-z} = -\log\left(\frac{z+1}{z-1}\right).$$

We rewrite $\frac{t^k}{t-z}$, use the linearity of the integral and get

$$\begin{aligned}\int_{-1}^{1} \frac{t^k}{t-z}dt &= \int_{-1}^{1} \frac{t^{k-1}(t-z) + zt^{k-1}}{t-z}dt \\ &= \int_{-1}^{1} t^{k-1}dt + z\int_{-1}^{1} \frac{t^{k-1}}{t-z}dt.\end{aligned}$$

With integration by parts, we obtain

$$\int_{-1}^{1} \log(t-z)dt = t\log(t-z)\Big|_{-1}^{1} - \int_{-1}^{1} \frac{t}{t-z}dt.$$

We use the result of the first part of this lemma and summarize the term

$$\begin{aligned}\int_{-1}^{1} \log(t-z)dt &= t\log(t-z)\Big|_{-1}^{1} - (z(\log(1-z) - \log(-1-z)) + 2) \\ &= (1-z)\log(1-z) + (1+z)\log(-1-z)) - 2.\end{aligned}$$

Again, integration by parts leads to

$$\int_{-1}^{1} t^k \log(t-z)dt = \frac{t^{k+1}}{k+1}\log(t-z)\Big|_{-1}^{1} - \frac{1}{k+1}\int_{-1}^{1} \frac{t^{k+1}}{t-z}dt.$$

□

Remark 2.4.2 *There holds for $z \in \mathbb{C}\backslash\{\pm 1\}$*

(i)

$$\begin{aligned}
\int_{-1}^{1} \frac{t}{t-z} dt &= z(\log(1-z) - \log(-1-z)) + 2 \\
\int_{-1}^{1} \frac{t^2}{t-z} dt &= z^2(\log(1-z) - \log(-1-z)) + 2z \\
\int_{-1}^{1} \frac{t^3}{t-z} dt &= z^3(\log(1-z) - \log(-1-z)) + 2z^2 + \frac{2}{3} \\
\int_{-1}^{1} \frac{t^4}{t-z} dt &= z^4(\log(1-z) - \log(-1-z)) + 2z^3 + \frac{2}{3}z \\
\int_{-1}^{1} \frac{t^5}{t-z} dt &= z^5(\log(1-z) - \log(-1-z)) + 2z^4 + \frac{2}{3}z^2 + \frac{2}{5}.
\end{aligned}$$

(ii)

$$\begin{aligned}
\int_{-1}^{1} \log(t-z) dt &= (1-z)\log(1-z) + (1+z)\log(-1-z) - 2 \\
\int_{-1}^{1} t\log(t-z) dt &= \frac{1}{2}(1-z^2)\log\left(\frac{z-1}{z+1}\right) - z \\
\int_{-1}^{1} t^2\log(t-z) dt &= \frac{1}{3}(1-z^3)\log(1-z) + \frac{1}{3}(1+z^3)\log(-1-z) \\
&\quad -\frac{2}{3}z^2 - \frac{2}{9} \\
\int_{-1}^{1} t^3\log(t-z) dt &= \frac{1}{4}(1-z^4)\log\left(\frac{z-1}{z+1}\right) - \frac{1}{2}z^3 - \frac{1}{6}z \\
\int_{-1}^{1} t^4\log(t-z) dt &= \frac{1}{5}(1-z^5)\log(1-z) + \frac{1}{5}(1+z^5)\log(-1-z) \\
&\quad -\frac{2}{5}z^4 - \frac{2}{15}z^2 - \frac{2}{25}.
\end{aligned}$$

2.4.1 Initial Values of $\tilde{Q}_k^m(z)$

The next lemma contains the initial values of $\tilde{Q}_k^m(z)$.

Lemma 2.4.3 *There holds for* $z \in \mathbb{C}\backslash[-1,1]$

(i) $\tilde{Q}_0^{-1}(z) = (1-z)\log(1-z) + (1+z)\log(-(1+z)) - 2$

(ii) $\tilde{Q}_1^{-1}(z) = \frac{1}{2}(1-z^2)\log\left(\frac{z-1}{z+1}\right) - z$

(iii) $\tilde{Q}_2^{-1}(z) = z\tilde{Q}_1^{-1}(z) + \frac{2}{3}$

(iv) $\tilde{Q}_0^{0}(z) = \log\left(\frac{z+1}{z-1}\right)$

(v) $\tilde{Q}_1^{0}(z) = z\tilde{Q}_0^{0}(z) - 2$

(vi) $\tilde{Q}_0^{m}(z) = \frac{(z-1)^{-m}-(z+1)^{-m}}{m}\ , \quad m \in \mathbb{N}$

(vii) $\tilde{Q}_1^{m}(z) = \begin{cases} \frac{2z}{z^2-1} - \log\left(\frac{z+1}{z-1}\right) & , \ m=1 \\ z\frac{(z-1)^{-m}-(z+1)^{-m}}{m} - \frac{(z-1)^{-(m-1)}-(z+1)^{-(m-1)}}{m-1} & , \ m \geq 1. \end{cases}$

Otherwise, if $z \in (-1,1)$*, the initial values can be calculated by Cauchy's principal value.*

Proof. With integration by parts, we get

$$\begin{aligned}
\tilde{Q}_0^{-1}(z) &= \int_{-1}^{1} P_0(t)\log(t-z)dt = \int_{-1}^{1} 1\cdot\log(t-z)dt \\
&= t\log(t-z)\Big|_{-1}^{1} - \int_{-1}^{1}\frac{t}{t-z}dt = t\log(t-z)\Big|_{-1}^{1} - \left(\int_{-1}^{1}1dt + z\int_{-1}^{1}\frac{1}{t-z}dt\right) \\
&= t\log(t-z) - t - z\log(t-z)\Big|_{-1}^{1} = (t-z)\log(t-z) - t\Big|_{-1}^{1} \\
&= (1-z)\log(1-z) + (1+z)\log(-(1+z)) - 2,
\end{aligned}$$

$$\begin{aligned}
\tilde{Q}_1^{-1}(z) &= \int_{-1}^{1} P_1(t)\log(t-z)dt = \int_{-1}^{1} t\cdot\log(t-z)dt \\
&= \frac{t^2}{2}\log(t-z)\Big|_{-1}^{1} - \frac{1}{2}\int_{-1}^{1}\frac{t^2}{t-z}dt \\
&= \frac{t^2}{2}\log(t-z)\Big|_{-1}^{1} - \frac{1}{2}\int_{-1}^{1} t + \frac{zt}{t-z}dt \\
&= \frac{t^2}{2}\log(t-z)\Big|_{-1}^{1} - \frac{1}{2}\left(\underbrace{\int_{-1}^{1} t}_{=0} + z\int_{-1}^{1} 1 + \frac{z}{t-z}dt\right)
\end{aligned}$$

$$\begin{aligned}
&= \frac{t^2}{2}\log(t-z) - \frac{zt}{2} - \frac{z^2}{2}\log(t-z)\,\Big|_{-1}^{1} \\
&= \frac{1}{2}(t^2-z^2)\log(t-z) - \frac{zt}{2}\,\Big|_{-1}^{1} \\
&= \frac{1}{2}(1-z^2)\log\left(\frac{z-1}{z+1}\right) - z
\end{aligned}$$

and

$$\begin{aligned}
\tilde{Q}_2^{-1}(z) &= \int_{-1}^{1} P_2(t)\log(t-z)dt = \frac{1}{2}\int_{-1}^{1}(3t^2-1)\log(t-z)dt \\
&= \frac{1}{2}\left(3\int_{-1}^{1} t^2\log(t-z)dt - \int_{-1}^{1}\log(t-z)dt\right) \\
&= \frac{1}{2}\left(3\left(\frac{1}{3}t^3 log(t-z)\,\Big|_{-1}^{1} - \frac{1}{3}\int_{-1}^{1}\frac{t^3}{t-z}dt\right) - \tilde{Q}_0^{-1}(z)\right) \\
&= \frac{1}{2}\left(t^3 log(t-z)\,\Big|_{-1}^{1} - \int_{-1}^{1}\frac{t^3}{t-z}dt - \tilde{Q}_0^{-1}(z)\right).
\end{aligned}$$

We use Lemma 2.4.1 and obtain

$$\begin{aligned}
\tilde{Q}_2^{-1}(z) &= \frac{1}{2}\left((1-z^3)\log(1-z) + (1+z^3)\log(-(1+z)) - 2z^2 - \frac{2}{3} - \tilde{Q}_0^{-1}(z)\right) \\
&= \frac{1}{2}\Big((1-z^3)\log(1-z) + (1+z^3)\log(-(1+z)) - 2z^2 - \frac{2}{3} \\
&\quad -(1-z)\log(1-z) - (1+z)\log(-(1+z)) + 2\Big) \\
&= \frac{1}{2}\left((z-z^3)\log(1-z) + (z^3-z)\log(-(1+z)) - 2z^2 + \frac{4}{3}\right) \\
&= z\left(\frac{1}{2}(1-z^2)\log\left(\frac{z-1}{z+1}\right) - z\right) + \frac{2}{3} \\
&= z\tilde{Q}_1^{-1}(z) + \frac{2}{3}.
\end{aligned}$$

We calculate the initial values of $\tilde{Q}_k^m(z)$. There holds

$$\begin{aligned}
\tilde{Q}_0^0(z) &= \int_{-1}^{1}\frac{1}{z-t}dt = -\log(z-t)\,\Big|_{-1}^{1} \\
&= \log\left(\frac{z+1}{z-1}\right) \\
\tilde{Q}_0^m(z) &= \int_{-1}^{1}\frac{1}{(z-t)^{m+1}}dt = \frac{(z-t)^{-m}}{m}\,\Big|_{-1}^{1}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{(z-1)^{-m}-(z+1)^{-m}}{m}\,, \quad m \in \mathbb{N} \\
\tilde{Q}_1^0(z) &= \int_{-1}^{1} \frac{t}{z-t}dt = -z\log(z-t) - t\Big|_{-1}^{1} \\
&= z\log\left(\frac{z+1}{z-1}\right) - 2 = z\tilde{Q}_0^0(z) - 2 \\
\tilde{Q}_1^m(z) &= \int_{-1}^{1} \frac{t}{(z-t)^{m+1}}dt = \int_{-1}^{1} \frac{z-(z-t)}{(z-t)^{m+1}}dt \\
&= z\int_{-1}^{1} \frac{1}{(z-t)^{m+1}}dt - \int_{-1}^{1} \frac{1}{(z-t)^{m}}dt \\
&= z\tilde{Q}_0^m(z) - \tilde{Q}_0^{m-1}(z)\,, \quad m \in \mathbb{N}. \\
&= \begin{cases} \frac{2z}{z^2-1} - \log\left(\frac{z+1}{z-1}\right) & , \ m = 1 \\ z\frac{(z-1)^{-m}-(z+1)^{-m}}{m} - \frac{(z-1)^{-(m-1)}-(z+1)^{-(m-1)}}{m-1} & , \ m \geq 1 \end{cases}
\end{aligned}
$$

□

2.4.2 Initial Values of $N_k(t)$

The next lemma contains the first values of $N_k(t)$.

Lemma 2.4.4 *There holds*

(i) $N_1(t) = \frac{1}{2}(1-t)$

(ii) $N_2(t) = \frac{1}{2}(1+t)$

(iii) $N_3(t) = \frac{1}{2}(t^2-1)$

(iv) $N_4(t) = \frac{1}{2}(t^3-t)$.

Proof. ad (i), ad (ii) Nothing to show.
ad (iii), ad (iv) We use property (i) of Lemma 2.1.1 to rewrite $N_3(t)$ and $N_4(t)$. We obtain

$$
\begin{aligned}
N_3(t) &= \int_{-1}^{t} P_1(\xi)d\xi = \frac{1}{3}(P_2(t) - P_0(t)) \\
&= \frac{1}{3}\left(\frac{3}{2}t^2 - \frac{1}{2} - 1\right) = \frac{1}{2}(t^2-1)
\end{aligned}
$$

and

$$\begin{aligned} N_4(t) &= \int_{-1}^{t} P_2(\xi)d\xi = \frac{1}{5}(P_3(t) - P_1(t)) \\ &= \frac{1}{5}\left(\frac{5}{2}t^3 - \frac{3}{2}t - t\right) = \frac{1}{2}(t^3 - t). \end{aligned}$$

□

2.4.3 Initial Values of $R_k^m(z)$

The next lemma contains the initial values of $R_k^m(z)$.

Lemma 2.4.5 *There holds for $z \in \mathbb{C}\backslash[-1,1]$*

(i) $R_1^{-1}(z) = \left(\frac{3}{4} + \frac{1}{2}z - \frac{1}{4}z^2\right)\log(-1-z) + \left(\frac{1}{4} - \frac{1}{2}z + \frac{1}{4}z^2\right)\log(1-z) + \frac{1}{2}z - 1$

(ii) $R_2^{-1}(z) = \left(\frac{1}{4} + \frac{1}{2}z + \frac{1}{4}z^2\right)\log(-1-z) + \left(\frac{3}{4} - \frac{1}{2}z + \frac{1}{4}z^2\right)\log(1-z) - \frac{1}{2}z - 1$

(iii) $R_3^{-1}(z) = \left(-\frac{1}{3} - \frac{1}{2}z + \frac{1}{6}z^3\right)\log(-1-z) + \left(-\frac{1}{3} + \frac{1}{2}z - \frac{1}{6}z^3\right)\log(1-z) - \frac{1}{3}z^3 + \frac{8}{9}$

(iv) $R_4^{-1}(z) = -\frac{1}{8}(z^4 - 2z^2 + 1)\log\left(\frac{z-1}{z+1}\right) - \frac{1}{4}z^3 + \frac{5}{12}z$

(v) $R_1^m(z) = \frac{1}{3}(\tilde{Q}_0^m(z) - \tilde{Q}_1^m(z))$

(vi) $R_2^m(z) = \frac{1}{3}(\tilde{Q}_0^m(z) + \tilde{Q}_1^m(z))$

(vii) $R_3^m(z) = \frac{1}{3}(\tilde{Q}_2^m(z) - \tilde{Q}_0^m(z))$

(viii) $R_4^m(z) = \frac{1}{5}(\tilde{Q}_3^m(z) - \tilde{Q}_1^m(z))$.

Otherwise, if $z \in (-1,1)$, the initial values can be calculated by Cauchy's principal value.

Proof. We calculate the initial values by using lemma (2.4.1). There holds

$$\begin{aligned} R_1^{-1}(z) &= \int_{-1}^{1} N_1(t)\log(t-z)dt = \frac{1}{2}\int_{-1}^{1}(1-t)\log(t-z)dt \\ &= \frac{1}{2}\left(\int_{-1}^{1}\log(t-z)dt - \int_{-1}^{1} t\log(t-z)dt\right) \\ &= \left(\frac{3}{4} + \frac{1}{2}z - \frac{1}{4}z^2\right)\log(-1-z) + \left(\frac{1}{4} - \frac{1}{2}z + \frac{1}{4}z^2\right)\log(1-z) + \frac{1}{2}z - 1, \end{aligned}$$

$$
\begin{aligned}
R_2^{-1}(z) &= \int_{-1}^{1} N_2(t)\log(t-z)dt = \frac{1}{2}\int_{-1}^{1}(1+t)\log(t-z)dt \\
&= \frac{1}{2}\left(\int_{-1}^{1}\log(t-z)dt + \int_{-1}^{1} t\log(t-z)dt\right) \\
&= \left(\frac{1}{4}+\frac{1}{2}z+\frac{1}{4}z^2\right)\log(-1-z) + \left(\frac{3}{4}-\frac{1}{2}z+\frac{1}{4}z^2\right)\log(1-z) - \frac{1}{2}z - 1,
\end{aligned}
$$

$$
\begin{aligned}
R_3^{-1}(z) &= \int_{-1}^{1} N_3(t)\log(t-z)dt = \frac{1}{2}\int_{-1}^{1}(t^2-1)\log(t-z)dt \\
&= \frac{1}{2}\left(\int_{-1}^{1} t^2\log(t-z)dt - \int_{-1}^{1}\log(t-z)dt\right) \\
&= \left(-\frac{1}{3}-\frac{1}{2}z+\frac{1}{6}z^3\right)\log(-1-z) + \left(-\frac{1}{3}+\frac{1}{2}z-\frac{1}{6}z^3\right)\log(1-z) - \frac{1}{3}z^3 + \frac{8}{9}
\end{aligned}
$$

and

$$
\begin{aligned}
R_4^{-1}(z) &= \int_{-1}^{1} N_4(t)\log(t-z)dt = \frac{1}{2}\int_{-1}^{1}(t^3-t)\log(t-z)dt \\
&= \frac{1}{2}\left(\int_{-1}^{1} t^3\log(t-z)dt - \int_{-1}^{1} t\log(t-z)dt\right) \\
&= -\frac{1}{8}(z^4-2z^2+1)\log\left(\frac{z-1}{z+1}\right) + \frac{5}{12}z - \frac{1}{4}z^3.
\end{aligned}
$$

We use the relation of Lemma 2.3.6 and obtain the remaining values. □

Remark 2.4.6 *There holds for* $z \in \mathbb{C}\backslash[-1,1]$

(i) $R_1^0(z) = \frac{1-z}{2}\log\left(\frac{z+1}{z-1}\right) + 1$

(ii) $R_2^0(z) = \frac{1+z}{2}\log\left(\frac{z+1}{z-1}\right) - 1$

(iii) $R_3^0(z) = \frac{z^2-1}{2}\log\left(\frac{z+1}{z-1}\right) - z$

(iv) $R_4^0(z) = \frac{z^3-z}{2}\log\left(\frac{z+1}{z-1}\right) - z^2 + \frac{2}{3}$

(v) $R_1^1(z) = \frac{1}{2}\log\left(\frac{z+1}{z-1}\right) - \frac{1}{z+1}$

(vi) $R_2^1(z) = -\frac{1}{2}\log\left(\frac{z+1}{z-1}\right) + \frac{1}{z-1}$

(vii) $R_3^1(z) = -z\log\left(\frac{z+1}{z-1}\right) + 2$

(viii) $R_4^1(z) = \frac{1}{2}(1-3z^2)\log\left(\frac{z+1}{z-1}\right) + 3z.$

Otherwise, if $z \in (-1,1)$*, the initial values can be calculated by Cauchy's principal value.*

Chapter 3
Stable Numerical Calculation of Minimal Solutions

We introduced several three-term recurrence relations and their related initial values in the previous chapter, but we don't discuss the stability of evaluating them. Hence, we take a look on computational aspects and notice, that the forward evaluation is not stable outside the real interval $[-1, 1]$, but the area, in which the relative error is smaller than a given tolerance, is an ellipse on the complex plane around the real interval $[-1, 1]$. Two algorithms are introduced, which steady the calculation outside $[-1, 1]$. We give the implementation of all algorithms in MATLAB. First of all, we motivate the idea of a minimal and dominant solution of a three-term recurrence relation with the following example. In fact, there exists other types of solutions, too.

Example 3.0.1 A unified three-term recurrence relation

$$x_{k+1} = ax_k + bx_{k-1} \ , \quad a, b \neq 0, b \neq \frac{-a^2}{4}$$

can be written as

$$\begin{pmatrix} x_{k+1} \\ x_k \end{pmatrix} = A \begin{pmatrix} x_k \\ x_{k-1} \end{pmatrix} \quad \text{with } A = \begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix}.$$

The eigenvalues are

$$\lambda_1 = \frac{a + \sqrt{a^2 + 4b}}{2} \quad \text{and} \quad \lambda_2 = \frac{a - \sqrt{a^2 + 4b}}{2}.$$

The eigenpairs are denoted by $(v_1, \lambda_1), (v_2, \lambda_2)$. We can choose the initial values

$$\begin{pmatrix} x_1 \\ x_0 \end{pmatrix} = \mu_1 v_1 + \mu_2 v_2 \ , \quad \mu_1, \mu_2 \neq 0$$

as a linear combination of eigenvectors, because the assumptions of a and b guarantee $\text{rang}(A) = 2$. Based on Theorem 2.1 and Theorem 2.2 of [5] there holds

$$|\lambda_2| < 1 < |\lambda_1|.$$

For $k = 1$, we get

$$\begin{aligned}\begin{pmatrix} x_2 \\ x_1 \end{pmatrix} &= A \begin{pmatrix} x_1 \\ x_0 \end{pmatrix} \\ &= \mu_1 A v_1 + \mu_2 A v_2 \\ &= \mu_1 \lambda_1 v_1 + \mu_2 \lambda_2 v_2 .\end{aligned}$$

This leads to

$$\begin{pmatrix} x_{k+1} \\ x_k \end{pmatrix} = \mu_1 \lambda_1^k v_1 + \mu_2 \lambda_2^k v_2 .$$

For $k \to \infty$, there holds

$$\lambda_1^k v_1 \to +\infty \quad \text{and} \quad \lambda_2^k v_2 \to 0.$$

Due to this fact, a three-term recurrence relation can be represented as a linear combination of a *dominat* solution, because it converges to infinity, and a *minimal* solution, because it converges to zero, if they exist.

We define the minimal and dominant solution of a three-term recurrence relation, without making constraint for the recurrence matrix.

Definition 3.0.2 (Minimal and dominant solution) *A solution $u_k(z)$ of a three-term recurrence relation is said to be minimal, if there exists a linearly independent solution $v_k(z)$ of the same three-term recurrence relation such that*

$$\lim_{k\to\infty} \frac{u_k(z)}{v_k(z)} = 0, \tag{3.1}$$

and $v_k(z)$ is called dominant solution.

3.1 Forward Evaluation

The forward evaluation is the simpliest and more intuitive way to calculate a solution of a three-term recurrence relation. We consider the three-term recurrence relations as in the first chapter

$$(k-n+1) f_{k+1+s}(z) = (2k+1) z f_{k+s}(z) - (k+n) f_{k-1+s}(z) \ , \quad k \in \mathbb{N}, s, n \in \mathbb{Z}$$

and a vector $f \in \mathbb{C}^j$ with the related initial values.

3.1.1 Implementation in Matlab

Listing 3.1: forward3term.m

```
1 function f = forward3term(f,p,z,n,s)
2
3 j = length(f)-1;
4
5 %Calculate the remaining values
6 for i=j:(p-1)
7   tmp = (2*(i-s)+1)*z*f(i+1) - (i-s+n)*f(i);
8   f(i+2) = tmp/(i-s-n+1);
9 end
```

Line 1: The function requires the parameters f, p, z, n, s. $f \in \mathbb{C}^j$ is a vector with initial values of the three-term recurrence relation. $p \in \mathbb{N}_0$ is the maximal polynomial degree. $z \in \mathbb{C}$ is the point of evaluation. $n \in \mathbb{Z}$ is the constant and $s \in \mathbb{Z}$ is the shift parameter in the three-term recurrence relation.

Line 6-9: We calculate the remaining values of f from $f_j(z)$ up to $f_p(z)$ with formula (2.5).

3.1.2 Analysis

As an example, we calculate the integral $\tilde{Q}^0_{30}(z)$ with the method *forward3term.m*. Figure 3.1 shows the real part of the result with $z \in [-1.1, 1.1] \times [-1.1, 1.1]$, Figure 3.2 shows the real part of the result with $z \in [-1.3, 1.3] \times [-1.3, 1.3]$ and finally Figure 3.3 shows the real part of the result with $z \in [-1.6, 1.6] \times [-1.6, 1.6]$. For example, Listing 3.1.2 describes, how to create Figure 3.1.

```
%Inital values
initialvalues = @(z) [log((z+1)/(z-1));
                      z.*(log((z+1)/(z-1))) - 2];
%Define grid
sx = linspace(-1.1,1.1,400);
sy = linspace(-1.1,1.1,400);
[X,Y] = meshgrid(sx,sy);
Z = X + sqrt(-1) * Y;
%Allocate fw
fw = zeros(length(sx),length(sy),31);

%Calculate via forward3term
for m = 1:length(sy)
    for j = 1:length(sx)
        z = Z(m,j);
```

```
        f = initialvalues(z);
        fw(m,j,1:end) = forward3term(f,30,z,0,0);
    end
end

%Surf result
surf(X,Y,real(fw(:,:,end)))
shading flat
axis tight
```

We recognize, that in general the forward evaluation is numerically unstable computing a minimal solution. In fact, three-term recurrence relations have a bad condition with respect to minimal solutions. There is no problem calculating a dominant solution. That is the reason, why we discuss alternative algorithms to calculate a minimal solution of a three-term recurrence relation. In Figure 3.3, you may detect, that the area, in which the relative error of the forward evaluation is small, is located around an ellipse relative to the real interval $[-1, 1]$. The main object will be the determination of the ellipse parameters to gain a border, which divides the complex plane into two areas. Inside the ellipse, we can calculate the minimal solution via forward evaluation and outside, we use an alternative algorithm.

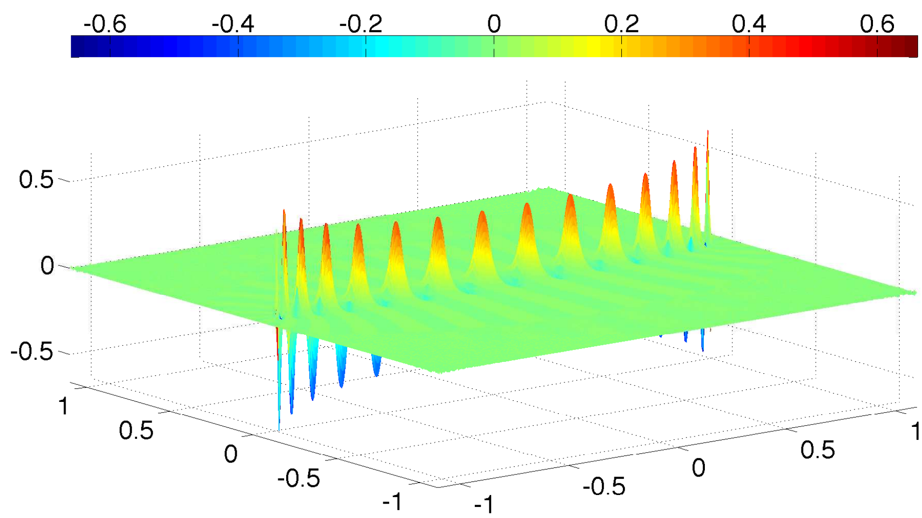

Figure 3.1: Real part of the result of forward evaluation of $Q^0_{30}(z)$ on $[-1.1, 1.1]^2$.

3.2 Miller's Backward Algorithm

To get a better condition for calculating a minimal solution of a three-term recurrence relation, Miller suggests to compute the values backward by using the property

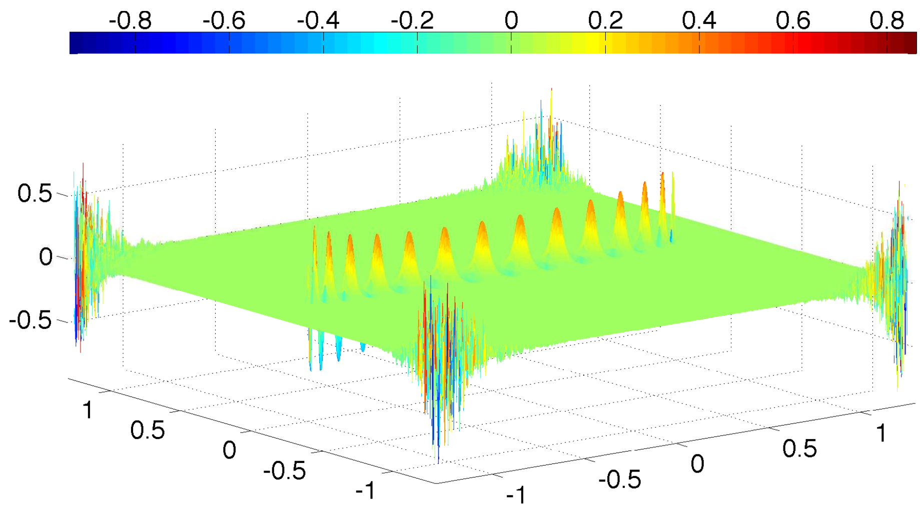

Figure 3.2: Real part of the result of forward evaluation of $Q_{30}^0(z)$ on $[-1.3, 1.3]^2$.

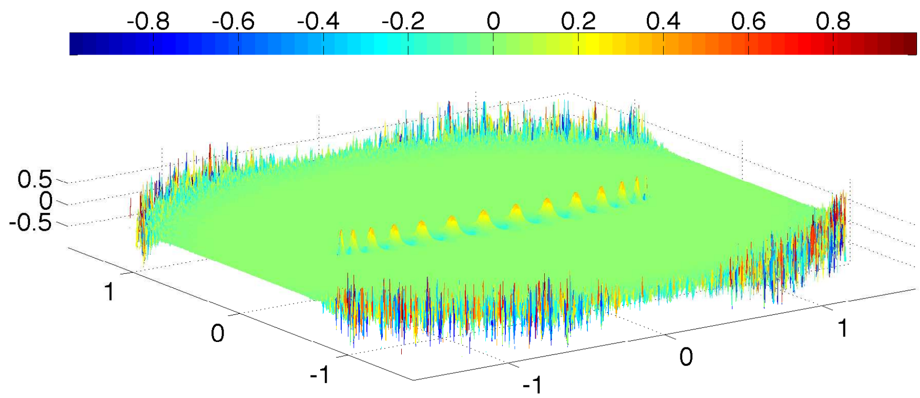

Figure 3.3: Real part of the result of forward evaluation of $Q_{30}^0(z)$ on $[-1.6, 1.6]^2$.

of a minimal solutions, that they converge to zero. We consider the three-term recurrence relations as in the first chapter

$$(k-n+1)f_{k+1+s}(z) = (2k+1)zf_{k+s}(z) - (k+n)f_{k-1+s}(z)\ , \quad k \in \mathbb{N}, s, n \in \mathbb{Z}$$

and a vector $f \in \mathbb{C}^j$ with the related initial values. We rewrite the last equation and obtain the formula

$$f_{k-1+s}(z) = \frac{2k+1}{k+n} z f_{k+s}(z) - \frac{k-n+1}{k+n} f_{k+1+s}(z)\ , \quad k \in \mathbb{N}, s, n \in \mathbb{Z} \tag{3.2}$$

to calculate the values backward. We choose an arbitrary start index ν, set $f_{\nu+1}(z) = 0$ and $f_\nu(z) = 1$ and calculate the remaining values of $f_k(z)$ down to $f_p(z)$. We repeat this procedure until the relative error between the previous and the current values of $f_{p+1}(z)$ and $f_p(z)$ are smaller than a tolerance or ν is larger than an upper bound ν_{max}.

3.2.1 Implementation in Matlab

Listing 3.2: backward3term.m

```
function [f,nu] = backward3term(f,p,z,n,s)

%Preperations
tol = 1e-14;
told = [realmax;1];
%Set nu arbitrary (nu > p )
nu = p+2;
nu_max = 5000;
%Set initial values
t = [1;0];

while nu < nu_max
  %Save only the last two entries
  for i = nu:-1:p
    t= [((2*(i-s)+1)*z*t(1)-((i-s)-n+1)*t(2))/((i-s)+n);
         t(1)];
    t = t./norm(t);
  end
  t = t./t(2);
  %Check, if t is a good approximation
  if sum(abs(told-t)<tol*abs(t)) == 2
      break;
  end
  told = t;
  nu = nu + 1;
end

j = length(f)-1;
tmp = f(end);
f(p:p+1) = t;
%Calculate the remaining values
for i=p-1:-1:j+1
  f(i)=((2*(i-s)+1)*z*f(i+1)-((i-s)-n+1)*f(i+2))/((i-s)+n);
end
%Multiply with tmp/f(j+1), because f(j+1) was overwritten
f(j+1:end) = f(j+1:end) * tmp/f(j+1);
```

Line 1:	The function requires the parameters f, p, z, n, s. $f \in \mathbb{C}^j$ is a vector with initial values of the three-term recurrence relation. $p \in \mathbb{N}_0$ is the maximal polynomial degree. $z \in \mathbb{C}$ is the point of evaluation. $n \in \mathbb{Z}$ is the constant and $s \in \mathbb{Z}$ is the shift parameter in the three-term recurrence relation.
Line 4-10:	We set the tolerance $tol = 1e - 14$ and $\nu = p + 2$ arbitrary. We define an upper border for ν to avoid an endless loop. In Line 10 we set the initial values such that $t(1) = f_\nu(z) = 1$ and $t(2) = f_{\nu+1}(z) = 0$
Line 12, 21-23:	The breaking conditions for the while-loop. If ν become larger than an upper bound or the relative error between the previous and current values of $t(1) = f_p(z)$ and $t(2) = f_{p-1}(z)$ is smaller than a tolerance tol, we don't continue calculating the values for $f_{\nu-1}(z) \dots f_{p-1}(z)$ again.
Line 14-18:	We calculate the values $f_{\nu-1}(z)$ down to $f_p(z)$ with formula (3.2). We have to save only the last two entries, because after the for- loop we only compare t with $told$.
Line 24-25:	If t doesn't satisfy the break condition in Line 21, we set $told = t$, increase $\nu = \nu + 1$ and repeat the procedure from Line 12-26.
Line 29:	We store the last initial value $tmp = f_{j-1}(z)$, because we overwrite it.
Line 32-34:	We calculate the values $f_{p-2}(z)$ down to $f_{j-1}(z)$ with formula (3.2).
Line 36:	We scale the computed values of f with $\frac{tmp}{f_{j-1}(z)}$ to get the correct values.

3.3 Gautschi's Continued Fraction Algorithm

We consider a the three-term recurrence relations as in the first chapter

$$(k - n + 1) f_{k+1+s}(z) = (2k + 1) z f_{k+s}(z) - (k + n) f_{k-1+s}(z) \ , \quad k \in \mathbb{N}, s, n \in \mathbb{Z}$$

and a vector $f \in \mathbb{C}^j$ with the related initial values. Gautschi's continued fraction algorithm is based on the theory of continued fractions and their convergence. For further information see [6]. In Lemma 2.2.2 we proved a transformation formula for $f_k(z)$ in order that the leading coefficients are equal to one, because Gautschi's continued fraction algorithm requires a system of polynomials of this type. While computing $f_k(z)$, instead of $\tilde{f}_k(z)$, we modify Gautschi's continued fraction algorithm.

3.3.1 Implementation in Matlab

Listing 3.3: gautschi.m

```
function [f, nu] = gautschi(f,p,z,n,s)

fold = 1;
fn = 0;
tol = 1e-14;
nu = p+1;
nmax = 1e8;
j = length(f)-1;


while abs(fold-fn) > tol*abs(fold) && nu < nmax

    %Calculate vector r
    r(nu+1) = 1;
    for m = nu:-1:j
        bm = ((m-s)+n)*((m-s)-n)/((2*(m-s)-1)*(2*(m-s)+1));
        r(m) = bm / (z - r(m+1));
    end

    %Store the last initial value
    tmp = f(j+1);
    f(j+1) = 1;
    %First factor is based on transformation formula
    for m = j+1:p
            f(m+1) = (2*(m-s)-1)/((m-s)-n) * r(m)*f(m);
    end
    %Scale the computed value
    f(j+1) = tmp;
    f(j+2:end) = tmp*f(j+2:end);

    fold = fn;
    fn = f(p+1);
    nu = nu + 1;
end
```

Line 1: The function requires the parameters f, p, z, n, s. $f \in \mathbb{C}^j$ is a vector with initial values of the three-term recurrence relation. $p \in \mathbb{N}_0$ is the maximal polynomial degree. $z \in \mathbb{C}$ is the point of evaluation. $n \in \mathbb{Z}$ is the constant and $s \in \mathbb{Z}$ is the shift parameter in the three-term recurrence relation.

Line 5-6: We set the tolerance $tol = 10^{-14}$ and $\nu = p + 1$ arbitrary.

Line 11: The break condition of the while-loop. If ν become larger than an upper bound or the relative error between the previous and current values of $f_p(z)$ is smaller than a tolerance *tol*, we don't continue calculating the values for $f_j(z) \dots f_p(z)$ again.

Line 14-18: The calculation of vector r as Gautschi suggests in his paper [6] (Eq 5.1).

$$r_{m-1}(z) = \frac{b_m}{z - a_m - r_m}\ , \quad m = \nu - 1, \dots, 1$$

with $r_\nu = 0$. Note, that in our case $a_m = 0$ for all $m \in \mathbb{N}$. Lemma 2.2.2 supplies a formula for

$$b_m = \frac{(m-n)(m+n)}{(2m-1)(2m+1)},$$

without an index shift. In Line 16, we insert the shift.

Line 20-21: We store the last initial value in *tmp*, because we compute the remaining values respectively to $f_{j-1}(z)$, which is set to 1.

Line 24-27: The formula to calculate the minimal solution is given by [6] (Eq 5.1). In fact, we don't expect, that the system of $f_k(z)$ has the leading coefficient equal to one, we modify the formula. Lemma 2.2.2 supplies

$$f_m(z) = \frac{2m-1}{m-n} \cdot r_{m-1} \cdot f_{m-1}(z).$$

In Line 25, we insert the shift again.

Line 28-29: Due to the fact, that we set $f_{j-1}(z) = 1$, we have to scale the computed values with *tmp*.

3.4 Estimate of ν as Against Adaptive Determination

We want to compare Gautschi's continued fraction algorithm and Miller's backward algorithm with respect to to the maximal ν of each point to get a exact solution. In fact, we have two algorithm to calculate a minimal solution, which are numerical stable on the complete complex plane. But close to the real interval $[-1, 1]$, we need many iterations to get an exact solution, in other words ν increases very fast, when $z \in \mathbb{C}$ is close to $[-1, 1]$. Figure 3.4 shows the maximal ν calculating $Q_k^0(z)$ for $k = 0 \dots 100$ (left: via Miller's backward algorithm, right: via Gautschi's continued fraction algorithm) - more precisely, the figure shows the maximal ν for each point, whose the break condition of the while-loop (in *backward3term.m* Line 12 and in *gautschi.m* Line 11) is satisfied. This figure illustrates the biggest disadvantage of both algorithms - the low convergence close to the real interval $[-1, 1]$. It is desirable to have a stable algorithm with a high convergence. We can optimize

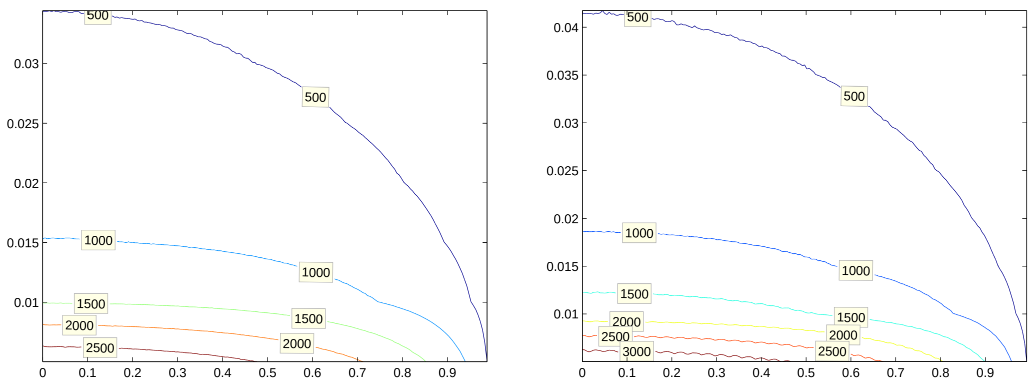

Figure 3.4: Maximal ν for each point. Left: Calculated via backward3term. Right: Calculated via gautschi.m.

Gautschi's continued fraction algorithm estimating a useful initial value of the index ν to prevent the while-loop in Line 11.

Let be $u_k(z)$ the minimal solution and $p_k(z)$ the dominant solution with $p_{-1}(z) = 0$ and $p_0(z) = 1$ of the three-term recurrence relation

$$(k - n + 1) f_{k+1+s}(z) = (2k + 1) z f_{k+s}(z) - (k + n) f_{k-1+s}(z) , \quad k \in \mathbb{N}, s, n \in \mathbb{Z}.$$

It holds $\frac{u_k(z)}{p_k(z)} \to 0$ for $k \to \infty$ and for sufficiently large N, the relative error can be approximated by

$$\max_{1 \le n \le N} |\epsilon_n^{(\nu)}| = |\epsilon_N^{(\nu)}| = \left| \frac{u_\nu(z)}{p_\nu(z)} \cdot \frac{p_N(z)}{u_N(z)} \right| ,$$

see also [5] (3.18). An asymptotic formula for $\frac{u_k(z)}{p_k(z)}$ is given in the appendix of [3] (Eq. A.1). Thus we have

$$\frac{u_\nu(z)}{p_\nu(z)} \cdot \frac{p_N(z)}{u_N(z)} \sim \left(\frac{1}{z + \sqrt{z^2 - 1}} \right)^{2(\nu - N)} .$$

Given a tolerance $\epsilon > 0$, we want to determine ν, such that the relative error is smaller than ϵ, therefore

$$\epsilon \stackrel{!}{=} \left(\frac{1}{z + \sqrt{z^2 - 1}} \right)^{2(\nu - N)}$$

must apply. This is equivalent to

$$\begin{aligned}
\log(\epsilon) &= \log\left(\left(\frac{1}{z+\sqrt{z^2-1}}\right)^{2(\nu-N)}\right)\\
&= 2(\nu-N)\log\left(\frac{1}{z+\sqrt{z^2-1}}\right)\\
&= 2(\nu-N)\left(\log(1)-\log(z+\sqrt{z^2-1})\right)\\
&= -2(\nu-N)\log(z+\sqrt{z^2-1})
\end{aligned}$$

We multiply the last equation with -1, divide by $\log(z+\sqrt{z^2-1})$ and obtain

$$\begin{aligned}
2(\nu-N) &= \frac{\log\left(\frac{1}{\epsilon}\right)}{\log(z+\sqrt{z^2-1})}\\
\nu &= N+\frac{\log\left(\frac{1}{\epsilon}\right)}{2\log(z+\sqrt{z^2-1})}.
\end{aligned}$$

Thus, to have a precision ϵ for $u_n(z)$, we have to choose

$$\nu \geq n+\frac{\log\left(\frac{1}{\epsilon}\right)}{2\log|z+\sqrt{z^2-1}|}. \tag{3.3}$$

3.5 Error Analysis and Ellipse Parameters

In order to choose a favorite algorithm, we make an error analysis with Miller's backward algorithm (MBA) and Gautschi's continued fraction algorithm (GCFA). We fix two points in $\mathbb{C}$ - one close to the singularity and one in the far field - and compare the exact solution, which is calculated by Maple, with the results of both algorithms. Afterwards, we want to assign the ellipse parameter by observing the relative error between the forward evaluation and Gautschi's continued fraction algorithm. We noted previously, that the forward evaluation has a small relative error with respect to the exact solution inside an ellipse. Consequently, we want to determine the ellipse parameter to have a tool, which allows us to decide, when we have to switch between the forward evaluation and an alternative algorithm. Therefore, we avoid the problem of the low convergence close to the real interval $[-1, 1]$, because there, we use the forward evaluation.

Table 3.4 contains the absolute error between both algorithms with respect to the exact solution, which is evaluated close to the singularity, thus we chose $z = 1+0.1i$. Table 3.5 contains the absolute error between both algorithms with respect to to the exact solution, which is evaluated in the far field, therefore we chose $z = 2+3i$. We note, that machine precision is reached. That is the reason, why there are a few irregularities, especially for the initial values.

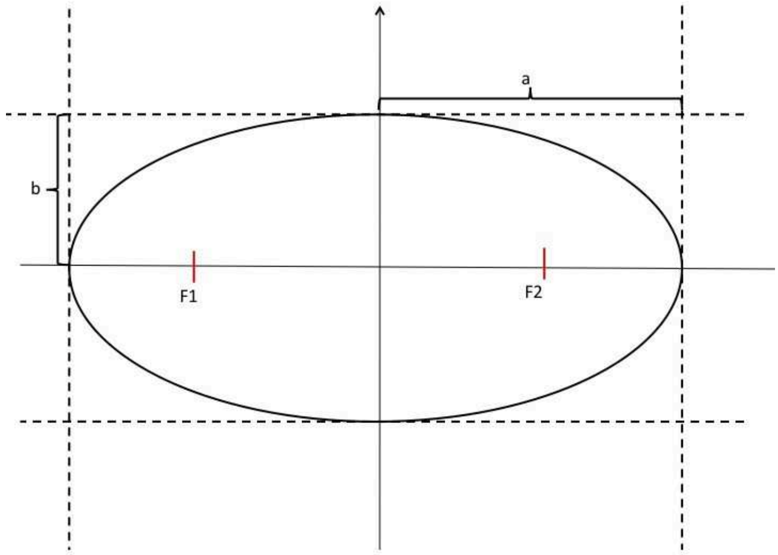

Figure 3.5: An ellipse.

Furthermore we investigate the relative error of $\tilde{Q}_k^0(z)$ calculating via Miller's backward algorithm and Gautschi's continued fraction algorithm respectively to the exact solution at $z = 1 + 0.1i$, which is calculated by Maple, as depicted in Figure 3.6. Next, we want to assign the ellipse parameter looking at the relative error between the forward evaluation and Gautschi's continued fraction algorithm. We give a short summary of relevant ellipse parameter and properties.

Figure 3.5 shows an ellipse with the parameters

a,b	-	length of the half-axis,
F1,F2	-	foci of the ellipse.

In our case, the foci of the ellipse are F1 $= -1$ and F2 $= 1$. Two formulas of the ellipse parameters of a point $z \in \mathbb{C}$ are given by

$$\begin{aligned} a &= \frac{1}{2}(|z+1| + |z-1|), \\ b &= \sqrt{\frac{(|z+1| + |z-1|)^2}{4} - 1}. \end{aligned}$$

Now, we investigate the relative error between the forward evaluation and Gautschi's continued fraction algorithm. We determine for each polynomial degree the minimal b such that the relative error is greater than a given tolerance $tol \in \{10^{-8}, 10^{-9}, 10^{-10}, 10^{-11}, 10^{-12}, 10^{-13}\}$. As an example Figure 3.7 shows the results of calculation theses ellipse parameters for $\tilde{Q}_k^0(z)$ for $k = 0 \ldots 1000$ and Figure 3.7 shows the results of calculation theses ellipse parameters for $R_k^0(z)$ for $k = 1 \ldots 1000$. We find a lower bound with the function

$$b(k) = \min\left(1, \frac{4.5}{(k+1)^{1.17}}\right) \tag{3.4}$$

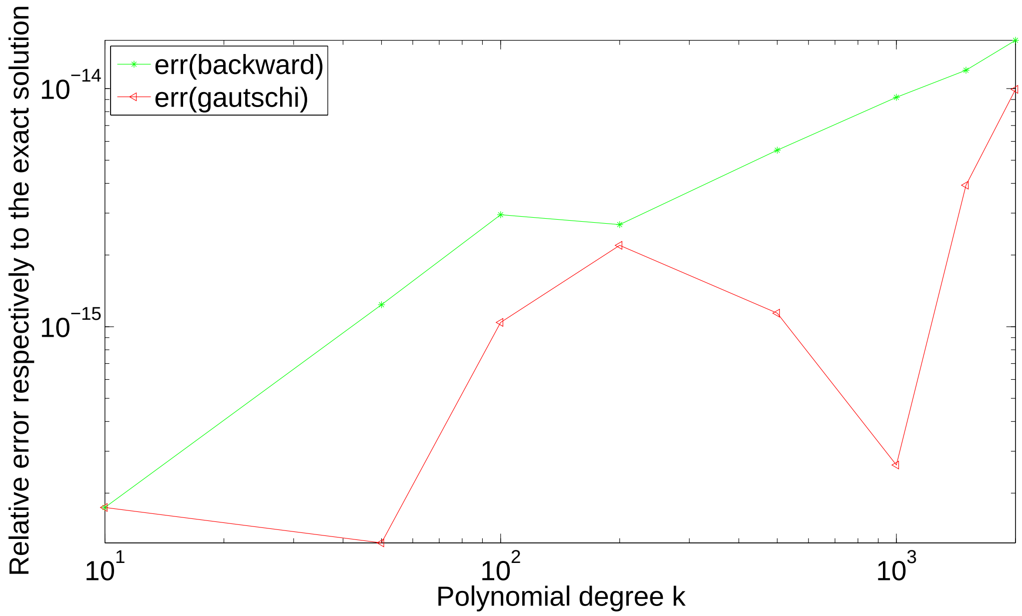

Figure 3.6: Relative error at $z = 1 + 0.1i$.

$z = 1 + 0.1i$	MBA	GCFA
$\tilde{Q}_0^0(z)$	0.0000	0.0000
$\tilde{Q}_1^0(z)$	0.0000	0.0000
$\tilde{Q}_2^0(z)$	0.1110e-15	0.1110e-15
$\tilde{Q}_3^0(z)$	0.1110e-15	0.2220e-15
$\tilde{Q}_4^0(z)$	0.2776e-15	0.1110e-15
$\tilde{Q}_5^0(z)$	0.4996e-15	0.0555e-15
$\tilde{Q}_6^0(z)$	0.4163e-15	0.2776e-15
$\tilde{Q}_7^0(z)$	0.0139e-15	0.7494e-15

$z = 1 + 0.1i$	MBA	GCFA
$\tilde{Q}_0^1(z)$	0.0000	0.0000
$\tilde{Q}_1^1(z)$	0.0000	0.0000
$\tilde{Q}_2^1(z)$	0.0089e-13	0.0001e-13
$\tilde{Q}_3^1(z)$	0.0089e-13	0.0003e-13
$\tilde{Q}_4^1(z)$	0.0266e-13	0.0089e-13
$\tilde{Q}_5^1(z)$	0.0533e-13	0.0222e-13
$\tilde{Q}_6^1(z)$	0.0400e-13	0.0577e-13
$\tilde{Q}_7^1(z)$	0.0266e-13	0.1243e-13

$z = 1 + 0.1i$	MBA	GCFA
$R_1^0(z)$	0.0000	0.0000
$R_2^0(z)$	0.0000	0.0000
$R_3^0(z)$	0.0000	0.0000
$R_4^0(z)$	0.0278e-15	0.0833e-15
$R_5^0(z)$	0.0971e-15	0.0416e-15
$R_6^0(z)$	0.1735e-15	0.1110e-15
$R_7^0(z)$	0.1665e-15	0.1388e-15

$z = 1 + 0.1i$	MBA	GCFA
$R_1^1(z)$	0.0000	0.0000
$R_2^1(z)$	0.0000	0.0000
$R_3^1(z)$	0.0000	0.0000
$R_4^1(z)$	0.0444e-14	0.0333e-14
$R_5^1(z)$	0.1332e-14	0.0222e-14
$R_6^1(z)$	0.1277e-14	0.1110e-14
$R_7^1(z)$	0.0312e-14	0.1887e-14

Tab. 3.4: Absolute error of MBA and GCFA for $z = 1 + 0.1i$.

$z = 2 + 3i$	MBA	GCFA
$\tilde{Q}_0^0(z)$	0.0000	0.0000
$\tilde{Q}_1^0(z)$	0.8327e-16	0.8327e-16
$\tilde{Q}_2^0(z)$	0.0781e-16	0.0781e-16
$\tilde{Q}_3^0(z)$	0.0108e-16	0.0108e-16
$\tilde{Q}_4^0(z)$	0.0015e-16	0.0014e-16
$\tilde{Q}_5^0(z)$	0.0002e-16	0.0001e-16
$\tilde{Q}_6^0(z)$	0.0000e-16	0.0000e-16
$\tilde{Q}_7^0(z)$	0.0000e-16	0.0000e-16

$z = 2 + 3i$	MBA	GCFA
$\tilde{Q}_0^1(z)$	0.0000	0.0000
$\tilde{Q}_1^1(z)$	0.0000	0.0000
$\tilde{Q}_2^1(z)$	0.0434e-16	0.0607e-16
$\tilde{Q}_3^1(z)$	0.0087e-16	0.0065e-16
$\tilde{Q}_4^1(z)$	0.0012e-16	0.0012e-16
$\tilde{Q}_5^1(z)$	0.0002e-16	0.0002e-16
$\tilde{Q}_6^1(z)$	0.0000e-16	0.0000e-16
$\tilde{Q}_7^1(z)$	0.0000e-16	0.0000e-16

$z = 2 + 3i$	MBA	GCFA
$R_1^0(z)$	0.0000	0.0000
$R_2^0(z)$	0.0000	0.0000
$R_3^0(z)$	0.0000	0.0000
$R_4^0(z)$	0.1285e-14	0.1285e-14
$R_5^0(z)$	0.0101e-14	0.0101e-14
$R_6^0(z)$	0.0009e-14	0.0009e-14
$R_7^0(z)$	0.0001e-14	0.0001e-14

$z = 2 + 3i$	MBA	GCFA
$R_1^1(z)$	0.0000	0.0000
$R_2^1(z)$	0.0000	0.0000
$R_3^1(z)$	0.0083e-14	0.0083e-14
$R_4^1(z)$	0.1343e-14	0.1344e-14
$R_5^1(z)$	0.0158e-14	0.0159e-14
$R_6^1(z)$	0.0019e-14	0.0019e-14
$R_7^1(z)$	0.0002e-14	0.0002e-14

Tab. 3.5: Absolute error of MBA and GCFA for $z = 2 + 3i$.

whereas k is the polynomial degree. In fact, we gain a border, which divides the complex plane into two area. Inside the ellipse with the parameters a and b, we use the forward evaluation and outside, we use Gautschi's continued fraction algorithm. Let be $z = u + iv$, then

$$\text{Calculation at } z \begin{cases} \text{via forward evaluation} & \text{, if } \frac{a^2}{u^2} + \frac{b^2}{v^2} < 1 \\ \text{via Gautschi's continued fraction algorithm} & \text{, else.} \end{cases}$$

The reasons, why we choose Gautschi's continued fraction algorithm, are the precision and the overflow of Miller's backward algorithm. Numerical experiments showed, that for $p \gg 1$, overflow occurs calculating the minimal solution with Miller's backward algorithm.

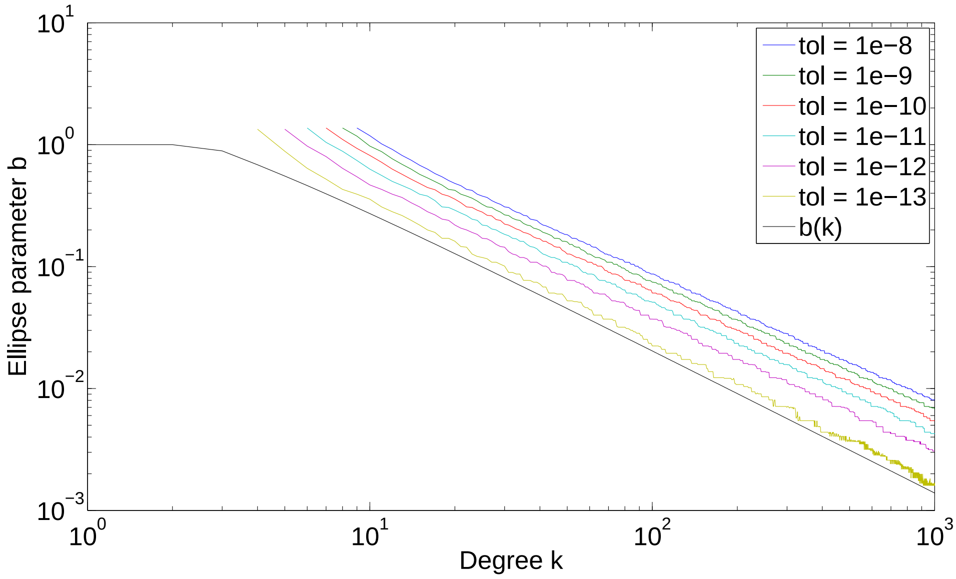

Figure 3.7: Result of calculating the ellipse parameter b for $\tilde{Q}_k^0(z)$.

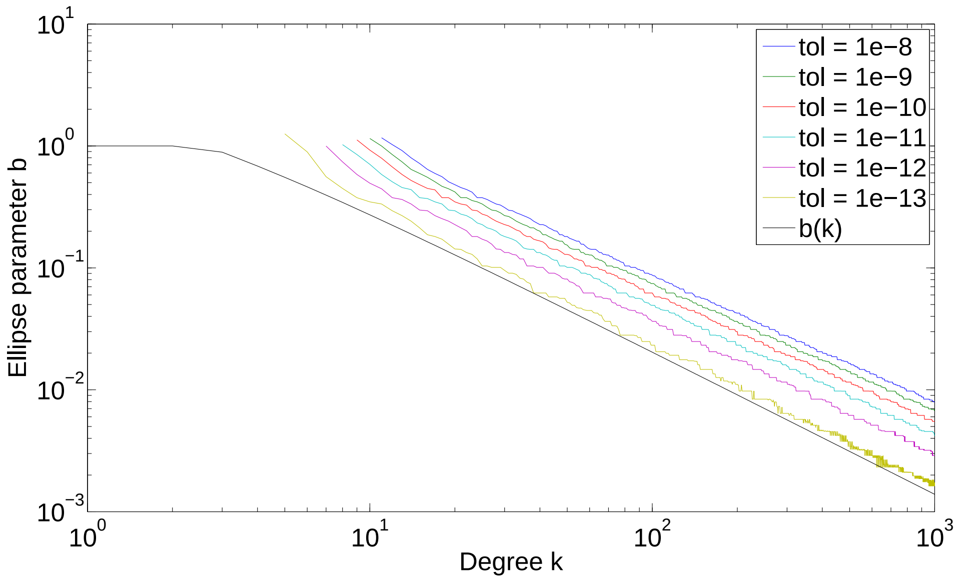

Figure 3.8: Result of calculating the ellipse parameter b for $R_k^0(z)$.

Chapter 4
C-Library 'liblegfct.c'

We want to give an detailed description of the C-Library 'liblegfct.c', which is found in the folder 'libbem_c'. We use the results of the previous chapter to create a library, which contains functions calculating minimal solutions numerically stable and deciding autonomously, which algorithm should be used. The functions exists as MATLAB files as well in the folder 'libbem_mat'. We prefer to calculate the integrals with the C functions, because we reach more performance parallelizing the computation e.g. via OPENMP or via NVIDIA CUDA. Due to this fact, a interface between MATLAB an C is required. In [2] a MEX-interface is introduced, which uses OPENMP. Instead of explaining the MEX interface, we limit the explanation to the C-Library. In the end of this chapter, we describe testing the routines in MATLAB. Recall the definitions

$$
\begin{aligned}
\tilde{Q}_k^{-1}(z) &:= \int_{-1}^{1} P_k(t)\log(t-z)dt \quad , \ k \in \mathbb{N}_0, \\
\tilde{Q}_k^{m}(z) &:= \int_{-1}^{1} \frac{P_k(t)}{(z-t)^{m+1}}dt \quad , \ k, m \in \mathbb{N}_0, \\
R_k^{-1}(z) &:= \int_{-1}^{1} N_k(t)\log(t-z)dt \quad , \ k \in \mathbb{N}, \\
R_k^{m}(z) &:= \int_{-1}^{1} \frac{N_k(t)}{(z-t)^{m+1}}dt \quad , \ k \in \mathbb{N}, m \in \mathbb{N}_0.
\end{aligned}
$$

The C-Library 'liblegfct.c' includes the following functions:

qtm1 Calculates the integrals $\tilde{Q}_k^{-1}(z)$ for one point $z \in \mathbb{C}$ and $k = 0 \ldots p$.

qt0 Calculates the integrals $\tilde{Q}_k^{0}(z)$ for one point $z \in \mathbb{C}$ and $k = 0 \ldots p$.

qt1 Calculates the integrals $\tilde{Q}_k^{1}(z)$ for one point $z \in \mathbb{C}$ and $k = 0 \ldots p$.

qtn Calculates the integrals $\tilde{Q}_k^{m}(z)$ for one point $z \in \mathbb{C}$, $m = 0 \ldots n$ and $k = 0 \ldots p$.

q0 Calculates the integrals $Q_k^{0}(z)$ for one point $z \in \mathbb{C}$ and $k = 0 \ldots p$.

q1 Calculates the integrals $Q_k^{1}(z)$ for one point $z \in \mathbb{C}$ and $k = 0 \ldots p$.

qn Calculates the integrals $Q_k^{m}(z)$ for one point $z \in \mathbb{C}$, $m = 0 \ldots n$ and $k = 0 \ldots p$.

r0 Calculates the integrals $R_k^0(z)$ for one point $z \in \mathbb{C}$ and $k = 1 \ldots p$.

r1 Calculates the integrals $R_k^1(z)$ for one point $z \in \mathbb{C}$ and $k = 1 \ldots p$.

4.1 The Function qtm1

We give a detailed explanation of **qtm1**. The functions **qt0, qt1, r0** and **r1** work similar. We only adjust the three-term recurrence relations and the related initial values. The function

void qtm1(fcomplex const z, int const p, double const fac, double *rretmp, double *rimtmp, double *Vre, double *Vim)

calculates the integrals $fac \cdot \tilde{Q}_k^{-1}(z)$ for $k = 0 \ldots p$. The parameters are

fcomplex const z z is the point of evaluation. *fcomplex* is a struct, which defines complex numbers (See 'complex.h').

int const p p is the maximal polynomial degree.

double const fac *fac* is a factor, which makes the calculation more flexible. Special cases maybe require a multiplicative factor.

double *rretmp *rretmp* is a pointer to an array, which is used by Gautschi's continued fraction algorithm. The array contains the real part of the vector r, see Listing 3.3 Line 14-18.

double *rimtmp *rimtmp* is a pointer to an array, which is used by Gautschi's continued fraction algorithm. The array contains the imaginary part of the vector r, see Listing 3.3 Line 14-18.

double *Vre *Vre* is a pointer to an array, which contains the real part of the computed integrals.

$$Vre = fac \cdot \left(\operatorname{Re} \tilde{Q}_0^{-1}(z), \operatorname{Re} \tilde{Q}_1^{-1}(z), \ldots, \operatorname{Re} \tilde{Q}_p^{-1}(z) \right)$$

double *Vim *Vim* is a pointer to an array, which contains the imaginary part of the computed integrals.

$$Vim = fac \cdot \left(\operatorname{Im} \tilde{Q}_0^{-1}(z), \operatorname{Im} \tilde{Q}_1^{-1}(z), \ldots, \operatorname{Im} \tilde{Q}_p^{-1}(z) \right)$$

The implementation is given in the following listing.

Listing 4.1: Function qtm1 in liblegfct.c

```
void qtm1(fcomplex const z, int const p, double const fac,
          double *rretmp, double *rimtmp,
```

```
        double *Vre, double *Vim){

int j,n,nu;
double tmp00, tmp01;
double a1, a2, a3, u, u2, v, v2, b, b2, b1_2, b2_2;
fcomplex z2, ctmp0, ctmp1, ctmp2, ctmp3;
double *rre = rretmp;
double *rim = rimtmp;

/* Get real(z) and imag(z)*/
u = z.r;
v = z.i;
u2 = u*u;
v2 = v*v;

/* Set V[0] and V[1]*/
/* case: z is complex*/
if ( v2 > eps ){
      a1 = atan((1-u)/v);
      a2 = atan((1+u)/v);
      b1_2 = (1+u)*(1+u)+v2;
      b2_2 = (1-u)*(1-u)+v2;
      a3 = log(b1_2/b2_2)*0.5;

      Vre[0] = 0.5*log(b1_2*b2_2)+u*a3-2+v*(a1+a2);
      Vim[0] = a1-a2-u*(a1+a2)+v*a3-pi*v/fabs(v);
      if(p>0){
          Vre[1] = 0.5*a3*(u2-v2-1) + u*v*(a1+a2)-u;
          Vim[1] = u*v*a3-0.5*(u2-v2-1)*(a1+a2)-v;
      }
/* case: z is real, z != +/- 1*/
} else if(u2!=1){
      a1 = fabs(1+u);
      a2 = fabs(1-u);

      Vre[0] = log(a1*a2)+u*log(a1/a2)-2;
      Vim[0] = 0;
      if(p>0){
          Vre[1] = 0.5*(u2-1)*log(a1/a2)-u;
          Vim[1] = 0;
      }
/* case: z is real, z = +/- 1*/
```

```
} else {
      Vre[0] = 2*log(2)-2;
      Vim[0] = 0;
      if(p>0){
          Vre[1] = -u;
          Vim[1] = 0;
      }
}

/* Multiply with fac*/
Vre[0] *= fac;
Vim[0] *= fac;
if(p>0){
      Vre[1] *= fac;
      Vim[1] *= fac;
}

/*Calculate V[2]..V[p]*/
if(p>1){

  /* Ellipse parameters */
  b = MIN(1,4.5/pow(p+1.0,1.17));
  b2 = b*b;
  a2 = 1.0+b2;

  /* Inside ellipse: calculation via forward eval */
  if (u2/a2+v2/b2 < 1) {
    /* Calculate initial values */
    ctmp0 =  Cmul(z,Complex(Vre[1],Vim[1]));
    Vre[2] = (fac*2.0)/3.0 + ctmp0.r;
    Vim[2] = 0.0 + ctmp0.i;

    /* Forward recurrence */
    for(j=2; j<=p-1; j++){
      tmp00 = (2.0*j+1.0)/(j+2.0);
      tmp01 = (1.0-j)/(j+2.0);
      ctmp0.r = tmp00*z.r;
      ctmp0.i = tmp00*z.i;
      ctmp1 = Cmul(ctmp0, Complex(Vre[j],Vim[j]));
      ctmp2.r = tmp01* Vre[j-1];
      ctmp2.i = tmp01* Vim[j-1];

```

```
        Vre[j+1] = ctmp1.r + ctmp2.r;
        Vim[j+1] = ctmp1.i + ctmp2.i;
      }
    /* Outside of ellipse, calculation via Gautschi */
    } else {
      /* Assign nu */
      z2 = Complex(fabs(u),fabs(v));
      ctmp0 = Csqrt(Cadd(Complex(-1.0,0), Cmul(z2,z2)));
      tmp00 =  2*log(Cabs(Cadd(z2, ctmp0)));
      nu = (int) (p+ceil(log(INVTOL)/tmp00));

      /* Catch case of overflow: Allocate more memory */
      if(nu>MAX_NU){
          FILE* of = fopen("nu_overflow.txt","w+");
          fprintf(of, "nu = %d\n", nu);
          fclose(of);
          rre = (double*) malloc(nu*sizeof(double));
          rim = (double*) malloc(nu*sizeof(double));
      }

      /* Set initial values */
      rre[nu-1] = 1;
      rim[nu-1] = 0;
      /* Calculate vector r */
      for (n=nu; n>=2; n--){
        ctmp3 = Csub(z,Complex(rre[n-1],rim[n-1]));
        tmp00 = (n*n-1.0)/((4*n*n-1.0)*(ctmp3.r*ctmp3.r
                                          + ctmp3.i*ctmp3.i));
        rre[n-2] =  tmp00*ctmp3.r;
        rim[n-2] = -tmp00*ctmp3.i;
      }

      /* Calculate remaining values of V */
      for (n=2; n<p+1; n++){
        ctmp3 = Cmul(Complex(rre[n-2],rim[n-2]),
                     Complex(Vre[n-1],Vim[n-1]));
        tmp00 = (2*n-1.0)/(n+1.0);
        Vre[n] = tmp00*ctmp3.r;
        Vim[n] = tmp00*ctmp3.i;
      }
    }/* end else */
  }/* end if p > 1*/
```

```
129 }/* end qtm1 */
```

Line 18-60: Set initial values and multiply with factor *fac*. The idea is to multiply only the initial values with fac, because based on the recurrence relation, the other values are multiplied with fac, too. We differentiate several cases to avoid complex arithmetic and to treat the singularities $z = 1$ and $z = -1$.

Line 65-68: We define the ellipse parameter. See Chapter 3.5 and Equation (3.4). We choose

$$b = \min\left(1, \frac{4.5}{(p+1)^{1.17}}\right).$$

Line 71-91: Check, if z is inside the ellipse with parameter a and b. Inside, we use the forward evaluation and compute the remaining values with Lemma (2.3.2) with $m = -1$ and $s = 0$. Note, that the three-term recurrence relation for $\tilde{Q}_k^{-1}(z)$ only holds for $k > 1$. Thus we set another initial value in Line 74-75.

Line 92-96: The else-case. We are outside the ellipse and use Gautschi's continued fraction algorithm. Thus we assign ν with Equation (3.3) as we suggest in Chapter 3.4,

$$\nu = p + \frac{\log\left(\frac{1}{\text{INVTOL}}\right)}{2\log|z + \sqrt{z^2-1}|}.$$

Line 98-105: We catch the case of overflow respecting to r. We allocate the storage outside of the functions, because it isn't necessary to allocate the memory for every point. We only have to allocate the storage one time and overwriting the data.

Line 107-117: The computation of the vector r analogically to Listing 3.3 Line 14-18.

Line 119-126: We calculate the remaining values of V analogically to Listing 3.3 Line 24-26.

As we said before, the functions **qt0, qt1, r0** and **r1** work similar, but note, that **r0** and **r1** calculate the integrals $R_k^0(z)$, $R_k^1(z)$ for $k = 1 \dots p$ as against the other functions, which calculates the integrals for $k = 0 \dots p$. For example in MATLAB, the function be called by

$$\textbf{num\_value} = \textbf{qtm1}(1+3i, 100)$$

and returns a vector **num_value** of length 101 containing the result of **qtm1** evaluated on $z = 1 + 3i$ up to degree 100.

4.2 The Function q0

We give a detailed explanation of **q0**. The function **q1** works similar. We use the relation *(i)* of Lemma 2.3.1 and the routines **qt0** and **qt1** to get the values for **q0** and **q1**. The function

> **void q0(** fcomplex const z, int const p, double const fac, double *rretmp, double *rimtmp, double *Vre, double *Vim)

calculates the integrals $fac \cdot Q_k^0(z)$ for $k = 0 \ldots p$. The parameters **fcomplex const z**, **int const p** and **double const fac** are similar to **qtm1**. The other parameters are

double *rretmp *rretmp* is a pointer to an array, which is used by **qt0**.

double *rimtmp *rimtmp* is a pointer to an array, which is used by **qt0**.

double *Vre *Vre* is a pointer to an array, which contains the real part of the computed integrals.

$$Vre = fac \cdot \left(\operatorname{Re} Q_0^0(z), \operatorname{Re} Q_1^0(z), \ldots, \operatorname{Re} Q_p^0(z)\right)$$

double *Vim *Vim* is a pointer to an array, which contains the imaginary part of the computed integrals.

$$Vim = fac \cdot \left(\operatorname{Im} Q_0^0(z), \operatorname{Im} Q_1^0(z), \ldots, \operatorname{Im} Q_p^0(z)\right)$$

Listing 4.2: Function q0 in liblegfct.c

```
void q0(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim){

    int k;
    /* Get qt0 */
    qt0(z,p,fac,rretmp,rimtmp,Vre,Vim);

    /* Multiply V with 0.5 and fac */
    for (k=0;k<p+1;k++) {
            Vre[k] = fac*0.5*Vre[k];
            Vim[k] = fac*0.5*Vim[k];
    }
}
```

Line 7: Calculate the values with **qt0** and store them in *Vre* and *Vim*.

Line 10-13: We use the relation

$$Q_k^0(z) = \frac{0!}{2}\tilde{Q}_k^0$$

and override the values in *Vre* and *Vim*.

4.3 The Function qtn

We give a detailed explanation of **qtn**. The function **qn** works similar. In Chapter 2.3.1 we proved a three-term recurrence relation for $\tilde{Q}_k^m(z)$ over m. As against the functions, we treated before, the method

void qtn(fcomplex const z, int const p, int const n, double const fac,
double *rretmp, double *rimtmp, double *Vtmpre, double *Vtmpim,
double *Vre, double *Vim)

calculates all integrals from $fac \cdot \tilde{Q}_k^0(z)$ up to $fac \cdot \tilde{Q}_k^m(z)$ for $k = 0 \ldots p$. The parameters **fcomplex const z**, **int const p** and **double const fac** are similar to **qtm1**. We take a look at the other parameters.

double *rretmp *rretmp* is a pointer to an array, which is used by **qt0** and **qt1**

double *rimtmp *rimtmp* is a pointer to an array, which is used by **qt0** and **qt1**.

double *Vretmp *Vretmp* is a pointer to an array, which is used by **qt0** and **qt1**

double *Vimtmp *Vimtmp* is a pointer to an array, which is used by **qt0** and **qt1**.

double *Vre *Vre* is a pointer to an array, which contains the real part of the computed integrals.

$$\begin{aligned} Vre = fac\cdot \Big(& \operatorname{Re}\tilde{Q}_0^0(z), \ldots, \operatorname{Re}\tilde{Q}_p^0(z), \\ & \operatorname{Re}\tilde{Q}_0^1(z), \ldots, \operatorname{Re}\tilde{Q}_p^1(z), \\ & \qquad \vdots \\ & \operatorname{Re}\tilde{Q}_0^m(z), \ldots, \operatorname{Re}\tilde{Q}_p^m(z)\Big) \end{aligned}$$

double *Vim *Vim* is a pointer to an array, which contains the imaginary part of the computed integrals.

$$\begin{aligned} Vim = fac\cdot \Big(&\operatorname{Im}\tilde{Q}_0^0(z),\ldots,\operatorname{Im}\tilde{Q}_p^0(z),\\ &\operatorname{Im}\tilde{Q}_0^1(z),\ldots,\operatorname{Im}\tilde{Q}_p^1(z),\\ &\vdots\\ &\operatorname{Im}\tilde{Q}_0^m(z),\ldots,\operatorname{Im}\tilde{Q}_p^m(z)\Big) \end{aligned}$$

The implementation is given in the following listing.

Listing 4.3: Function qtn in liblegfct.c

```c
void qtn(fcomplex const z, int const p, int const n,
    double const fac, double *rretmp, double *rimtmp,
    double *Vtmpre, double *Vtmpim,
    double *Vre, double *Vim){

    int k,i;
    double u,u2,v2,ar,br;
    fcomplex z2, a,b, ctmp, ctmp1, ctmp2, ctmp3;

    /* Get real(z) and imag(z) */
    z2 = Cmul(z,z);
    u  = z.r;
    u2 = u*u;
    v2 = z.i * z.i;
    ctmp = Complex(z2.r-1.0,z2.i);

    /* Set qt0  V */
    qt0(z,p,fac,rretmp,rimtmp,Vtmpre,Vtmpim);
    for (k=0;k<p+1;k++) {
        Vre[k] = Vtmpre[k];
        Vim[k] = Vtmpim[k];
    }

    /* Set qt1  V */
    if (n>0) {
        qt1(z,p,fac,rretmp,rimtmp,Vtmpre,Vtmpim);
        for (k=0;k<p+1;k++) {
            Vre[k+p+1] = Vtmpre[k];
            Vim[k+p+1] = Vtmpim[k];
```

```
30         }
31     }
32 
33     if (n>1) {
34         /* Loop for calculation of qti upto degree p*/
35         for (k=0;k<p+1;k++) {
36             /* Loop for caluclation upto qtn */
37             for (i=1;i<n;i++) {
38                 /* case: z is complex */
39                 if (v2 > eps) {
40                     a = Cdiv(RCmul(2*i,z),RCmul(i+1,ctmp));
41                     b = Cdiv(  RCmul((k-i+1.0)*(k+i),
42                                Complex(1.0,0)),
43                              RCmul(i*(i+1.0),ctmp));
44                     ctmp1 = Cmul(a,Complex(Vre[k+i*(p+1)],
45                                            Vim[k+i*(p+1)]));
46                     ctmp2=Cmul(b,Complex(Vre[k+(i-1)*(p+1)],
47                                        Vim[k+(i-1)*(p+1)]));
48                     ctmp3 = Cadd(ctmp1,ctmp2);
49                     Vre[k+(i+1)*(p+1)] = ctmp3.r;
50                     Vim[k+(i+1)*(p+1)] = ctmp3.i;
51                 /* case: z is real and not +1 or -1*/
52                 } else if (u2 != 1) {
53                     ar = 2.0*i*u / ((i+1)*(u2-1.0));
54                     br = (k-i+1.0)*(k+i) /
55                             (i*(i+1.0)*(u2-1.0));
56                     Vre[k+(i+1)*(p+1)] = ar*Vre[k+i*(p+1)]
57                                     + br*Vre[k+(i-1)*(p+1)];
58                     Vim[k+(i+1)*(p+1)] = 0.0;
59 
60                 /* case: z = +1 or z = -1 */
61                 } else {
62                     Vre[k+(i+1)*(p+1)] = 0.0;
63                     Vim[k+(i+1)*(p+1)] = 0.0;
64                 }
65             }
66         }
67     }
68 }/* end qtn */
```

Line 16-30: Calculation of the initial values with **qt0** and **qt1**.

Line 33-65: We compute the remaining values with

$$\tilde{Q}_k^{m+1}(z) = a_m \tilde{Q}_k^m(z) + b_m \tilde{Q}_k^{m-1}(z)$$

and

$$a_m = \frac{2mz}{(z^2-1)(m+1)} \; , \; b_m = \frac{(k+m)(k-m+1)}{(z^2-1)(m+1)m}.$$

We differentiate several cases to avoid complex arithmetic and to treat the singularities $z = 1$ and $z = -1$.

The function **qn** work similar, because in Lemma 2.1.1 *(vi)* there is given a three-term recurrence relation for $Q_k^m(z)$ over m. Instead of calculating the initial values with **qt0** and **qt1**, we have to use the functions **q0** and **q1**.

4.4 Test Files

All tests are computed on an Apple Macbook Pro with the following technical lineup:

Kernel:	Intel Core 2 Duo, 2,26 GHz
RAM:	4 GB DDR3 RAM, 1067 MHz.

We describe the way testing the C-Library. We compare the numerical results against the exact values which are computed with Maple. The test files are implemented in MATLAB. Thus, we need a MEX-interface, see [2]. The choice of the testing points should be cover the far field, the near field respecting to the real interval $[-1, 1]$ and this interval, too. We choose the following points

$$z \in \{0 \; , \; i \; , \; -i \; , \; 2+3i \; , \; \frac{101}{100} \; , \; \frac{1}{2} + \frac{101}{100}i \; , \; \frac{1}{2} \; , \; -\frac{1}{7}\}.$$

The exact solution of $\tilde{Q}_k^0(z)$ for $z = 2+3i$ and $k = 0 \ldots 4$ can be calculated with the following Maple script, where the *orthopoly*-package is included to get the Legendre polynomials.

Listing 4.4: exactqt0.mw

```
restart;
with(orthopoly);
Digits := 100;
z := 2+3*I;
Qt0 := proc (z, k) options operator, arrow;
        int(P(k, t)/(z-t), t = -1 .. 1) end proc;
zm := z-1/100000000000000000*I;
zp := z+1/100000000000000000*I;
```

```
#if z in [-1,1]
#seq(evalf((Qt0(zm, k)+Qt0(zp, k))*(1/2)), k = 0 .. 4);
#else
seq(evalf(Qt0(z, k)), k = 0 .. 4)
```

The missing exact values can be determined similarly. Listing 4.5 gives an example for a MATLAB script, which tests the routine **qt0**.

Listing 4.5: test_qt0.m

```
% Routine to test output of qt0
function test_qt0
flag_err = 0;
fprintf('Test function QT0 at various points.\n');
try
    qt0(0,1);
catch
    fprintf('---------- Can''t find QT0!  --------- \n');
    return
end
% Test for small p
z = 0;
ex_value = [0,-2,0,4/3,0];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = 1i;
ex_value=[-pi/2*1i, pi/2-2, (pi-3)*1i,19/3-2*pi,...
            (40/3-17/4*pi)*1i];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = -1i;
ex_value=[pi/2*1i, pi/2-2, (3-pi)*1i,19/3-2*pi,...
            (17/4*pi-40/3)*1i];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = 2+3i;
ex_value=[0.29389333245105950409-0.46364760900080611621i,...
        -0.02127050809546264316-0.04561522064843372014i, ...
        -0.00548969759396594090-0.00073914387447999657i, ...
        -0.00042293387717805800+0.00049784621419278698i, ...
        0.00002331200083912096+0.00007641680034994737i];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = 1.01;
```

```
ex_value = [ 5.3033049080590757, ...
             3.3563379571396665, ...
             2.4331995510370568, ...
             1.8583272728192680, ...
             1.4596937914302636];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = 1/2+0.01i;
ex_value = [ 1.0984345503858559-3.1149287517127744i, ...
            -1.4196334372899442-1.5464800303525286i, ...
            -1.5907451527050982+0.3763098515326415i, ...
            -0.3854704999198297+1.3180658106338021i, ...
             0.8327060254128811+0.8643294619064986i];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = 1/2;
ex_value = [1.0986122886681096, ...
           -1.4506938556659448, ...
           -1.6373265360835132, ...
           -0.3973095429589645, ...
            0.8803490519735407];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%
z = -1/7;
ex_value = [-0.2876820724517809, ...
            -1.9589025610783166, ...
             0.5636058707426725, ...
             1.1717431667325269, ...
            -0.7156401947401360];
flag_err = flag_err | check_and_text(z,ex_value,qt0(z,4));
%

if flag_err
  fprintf('---------  ERRORS in QT0.\n');
else
  fprintf('-------------------------------------\n');
  fprintf('---------  NO ERRORS in QT0  ---------\n');
  fprintf('-------------------------------------\n');
end

function flag_err = check_and_text(z,ex_value,num_value)
abserr = 1e-14;
```

```
flag_err = 1;
if (size(ex_value,1)~=size(num_value,1) ...
        | size(ex_value,2)~=size(num_value,2))
  fprintf('\n*** Error / dimensions do not match!\n');
  fprintf('\t z = (% 5.12f,% 5.12fi)\n',real(z),imag(z));
  fprintf('\t p = % 5d\n',size(ex_value,2));
  fprintf('\n');
  return
end
if sum(abs(num_value-ex_value) > abserr)
  fprintf('\n*** Error at  z = (% 5.12f,% 5.12fi)\n',...
            real(z),imag(z));
  for j=1:size(ex_value,2)
    fprintf('\t exact value = (% 5.12f,% 5.12fi)\n', ...
            real(ex_value(j)),imag(ex_value(j)));
    fprintf('\t num value   = (% 5.12f,% 5.12fi)\n', ...
            real(num_value(j)),imag(num_value(j)));
  end
  fprintf('\n');
else
  flag_err = 0;
end
```

There exists the following test files and they work similar to *test_qt0.m*.

test_qtm1.m	Tests the function **qtm1**.
test_qt0.m	Tests the function **qt0**.
test_qt1.m	Tests the function **qt1**.
test_qt2.m	Tests the function **qtn** with $n = 2$.
test_qt3.m	Tests the function **qtn** with $n = 3$.
test_q0.m	Tests the function **q0**.
test_q1.m	Tests the function **q1**.
test_q2.m	Tests the function **qn** with $n = 2$.
test_q3.m	Tests the function **qn** with $n = 3$.
test_r0.m	Tests the function **r0**.
test_r1.m	Tests the function **r1**.

Finally we can test all routines and the result is shown in Listing C.1.
The MATLAB script *show_complex.m*, which is described in Listing 4.6, is a tool to visualize on the left side the real part, and on the right side the imaginary part of the result of the functions in 'liblegfct.c'. The parameters are

- **p** The maximal degree of the integrand, which we want to plot.
- **fct** A string with the function name, we want to use.
- **sx** Optional: The scale of the x axis.
- **sy** Optional: The scale of the y axis.

Listing 4.6: show_complex.m

```
function show_complex(p,fct,sx,sy)
% Example:
% show_complex(1,'r1')
% show_complex(1,'qtm1')
% show_complex(1,'qt0')
% show_complex(1,[],linspace(0.5,1.5,30),linspace
   (-0.5,0.5,30))

if nargin<2 || isempty(fct)
  fct = 'r0'
end

if nargin<=2
  sx = linspace(-1.5,1.5,61);
  sy = linspace(-1.5,1.5,61);
end
[X,Y] = meshgrid(sx,sy);

Z = ones(size(X));
for j=1:size(X(:))
  tmp = feval(fct,complex(X(j),Y(j)),p);
  Z(j) = tmp(p+1);
end
Z(find(abs(Z)>1e10))=NaN;

subplot(1,2,1);
surf(X,Y,real(Z));
subplot(1,2,2);
surf(X,Y,imag(Z));
```

Figure 4.1 shows the result of the function call

show_complex(3,'qt0').

As an example we take a look at the computation time of the method *qt0.c*. Figure 4.2 shows the computation time of *qt0.c* for $p \in \{10, 100, 500, 1000, 5000\}$. Thereby, we fix a polynomial degree p and investigate the time, which is required to calculate

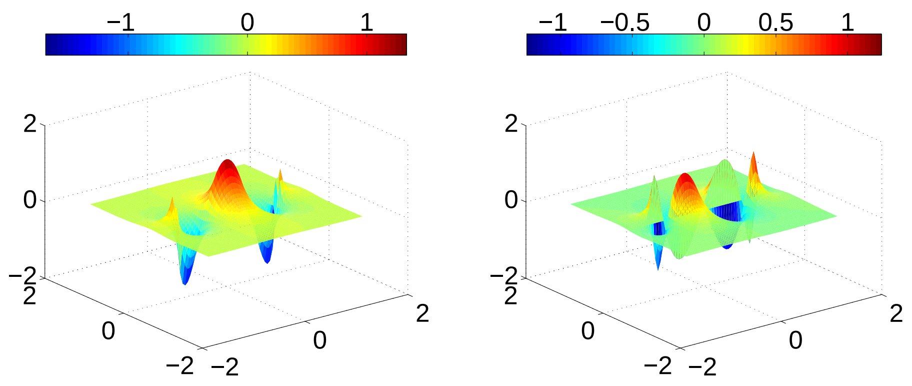

Figure 4.1: Result of function call: show_complex(3,'qt0').

the integrals for $k = 0 \ldots p$ for all points inside the grid, we defined before. The x axis describe the number of points of the grid, and the y axis the required time. Moreover, we fixed a number of points and investigate the computation time as depicted in Figure 4.3.

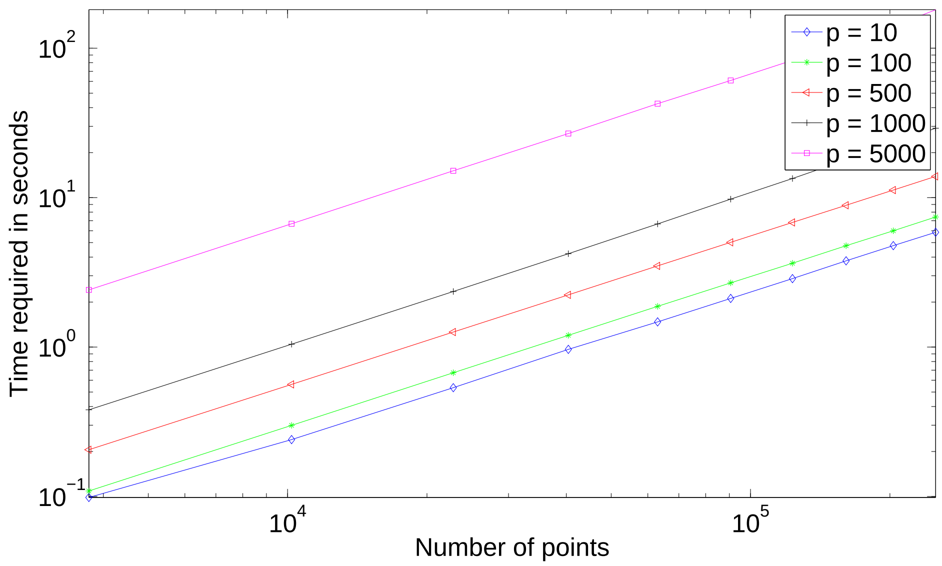

Figure 4.2: Computation time of *qt0.c* with a fixed p.

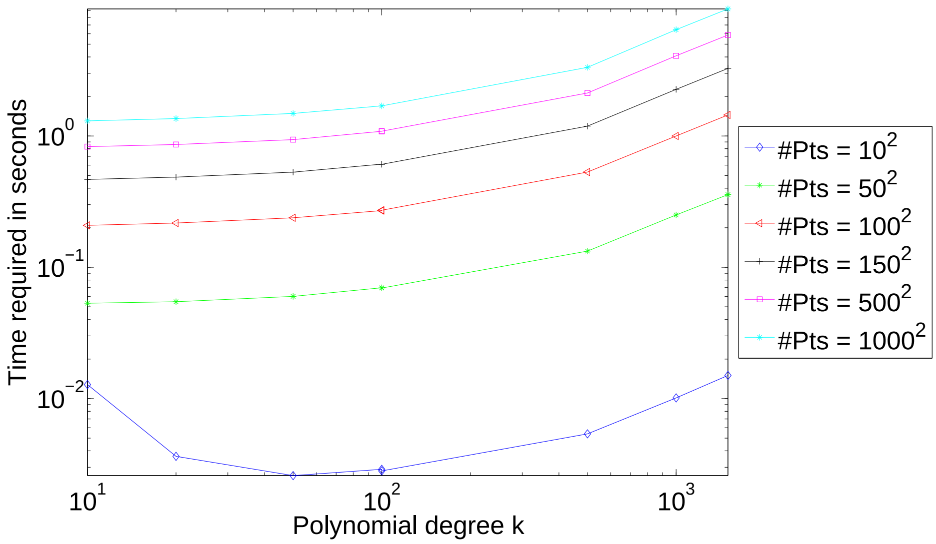

Figure 4.3: Computation time of *qt0.c* with a fixed number of points.

--

Chapter 5
Conclusion

We gave an short overview on the Legendre polynomials and functions as an example for a three-term recurrence relation in the second chapter. Moreover, we treated the theory of three-term recurrence relations and proved, that the integrals $\tilde{Q}_k^m(z)$, $R_k^m(z)$ satisfy a three-term recurrence relation. Therefore, we computed the initial values. Hence, we gained a method, to compute integrals in a fast and efficient way. Chapter 3 showed us, that in general this computation isn't stable, but we demonstrated, how to solve the problem of instability. Furthermore, we noticed, that the algorithms, we introduced, had a high effort close to the real interval $[-1, 1]$. We analyzed, that the area, in which the relative error is smaller than a tolerance with respect to exact solution, is located around an ellipse respecting to the real interval $[-1, 1]$ and took a look on the relative error between the forward evaluation and Gautschi's continued fraction algorithm. This analysis led to a division of the complex plane and we were able to decide, which algorithm is better to use. Thus, we could create a method, which decides autonomously, in which case it is better to use the forward evaluation or Gautschi's continued fraction algorithm.
Chapter 4 gave a detailed description of the C-Library 'liblegfct.c' containing the stable implementation of $Q_k^m(z)$, $\tilde{Q}_k^m(z)$ and $R_k^m(z)$. We compared the numerical results with the exact solution, which was calculated by Maple, and noticed, that we gained a high precision. Thus, the aim of this thesis is reached.
For the future, it will be conceivable to implement the methods for $\tilde{Q}_k^m(z)$ with $m \in \mathbb{Z}$, $m < -1$. Moreover, $R_k^m(z)$ have to be implemented with $m \in \mathbb{N}$, $m > 1$. As there only exist procedures to calculate $R_k^0(z)$ and $R_k^1(z)$, it is desirable to find a three-term recurrence relation for $R_k^m(z)$ over m and implement it.

Appendix A
The file 'liblegfct.h'

Listing A.1: Function qtm1 in liblegfct.c

```
#define MAX_NU  100000
#define MAX_P   30000
#define MAX_NP   90000

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "complex.h"

/* qtm1
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre        (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *          / 1
 *          |
 *    fac * |   P_k(t) * log(t-z) dt; z = u+iv, k = 0,...,p
 *          |
 *          / -1
 *
 *    where P_k(t) are the Legendre polynomials
 *
 */

void qtm1(fcomplex const z, int const p, double const fac,
          double *rretmp, double *rimtmp,
          double *Vre, double *Vim);
```

```c


/* qt0
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *         / 1
 *         |
 *   fac * |  P_k(t) / (z-t) dt; z = u+iv, k = 0,...,p
 *         |
 *         / -1
 *
 *    where P_k(t) are the Legendre polynomials.
 *
 */

void qt0(fcomplex const z, int const p, double const fac,
         double *rretmp, double *rimtmp,
         double *Vre, double *Vim);

/* qt1
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *         / 1
 *         |
 *   fac * |  P_k(t) / (z-t)^2 dt; z = u+iv, k = 0,...,p
 *         |
 *         / -1
 *
```

```
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void qt1(fcomplex const z, int const p, double const fac,
         double *rretmp, double *rimtmp,
         double *Vre, double *Vim);


/* qtn
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           n        (max. exponent of (z-t)^n)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 *           Vtmpre   (temporary variables)
 *           Vtmpim   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)

 *         / 1
 *         |
 *  fac *  |  P_k(t) * log(t-z) dt; z = u+iv, k = 0,...,p
 *         |
 *         / -1
 * and for j = 0..n
 *         / 1
 *         |
 *  fac *  |  P_k(t) / (z-t)^j dt; z = u+iv, k = 0,...,p
 *         |
 *         / -1
 *
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void qtn(fcomplex const z, int const p, int const n,
         double const fac, double *rretmp, double *rimtmp,
         double *Vtmpre, double *Vtmpim,
         double *Vre, double *Vim);
```

```

/* q0
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *             / 1
 *             |
 *   fac*0.5 * |  P_k(t)/(z-t) dt; z = u+iv, k = 0,...,p
 *             |
 *             / -1
 *
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void q0(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim);

/* q1
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *                              / 1
 *                              |
 *   fac * -0.5*(z^2-1)^(1/2) * | P_k(t)/(z-t)^2 dt;
 *                              |
 *                              / -1
 *   for z = u+iv, k = 0, ..., p, or
 *                             / 1
```

```
 *                            |
 *  fac * 0.5*(1-x^2)^(1/2) * | P_k(t)/(x-t)^2 dt;
 *                            |
 *                            / -1
 *  for |x| < 1 real,k = 0,...,p and
 *  where P_k(t) are the Legendre polynomials.
 *
 */

void q1(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim);

/* qn
 *
 * INPUT:   z       (point of evaluation)
 *          p       (max. polynomial degree)
 *          n       (upper limit of calculation of qn)
 *          fac     (factor to multiply)
 *          rretmp  (temporary variables)
 *          rimtmp  (temporary variables)
 *          Vtmore  (temporary variables)
 *          Vtmpim  (temporary variables)
 * OUTPUT:  Vre     (real part of the integral)
 *          Vim     (imaginary part of the integral)
 *
 *                             / 1
 *                             |
 *  fac * -0.5*(z^2-1)^(1/2) * | P_k(t)/(z-t)^m dt;
 *                             |
 *                             / -1
 *  for z = u+iv, k = 0, ..., p, or
 *                            / 1
 *                            |
 *  fac * 0.5*(1-x^2)^(1/2) * | P_k(t)/(x-t)^m dt;
 *                            |
 *                            / -1
 *  for |x| < 1 real,k = 0,...,p, m = 0..n and
 *  where P_k(t) are the Legendre polynomials.
 *
 */
```

```
void qn(fcomplex const z, int const p, int const n,
        double const fac, double *rretmp, double *rimtmp,
        double *Vtmpre, double *Vtmpim,
        double *Vre, double *Vim);


/* r0
 *
 * INPUT:   z       (point of evaluation)
 *          fac     (factor to multiply)
 *          p       (max. polynomial degree)
 *          rretmp  (temporary variables)
 *          rimtmp  (temporary variables)
 * OUTPUT:  Vre     (real part of the integral)
 *          Vim     (imaginary part of the integral)
 *
 *         / 1
 *         |   N_k(t)
 *   fac * |  -------- dt;   z = u+iv, k = 1, ..., p
 *         |    z-t
 *         / -1
 *
 *                  / t
 *   where N_k(t) = |    P_(k-2)(xsi) dxsi
 *               -1 /
 *
 *
 */

void r0(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim);


/* r1
 *
 * INPUT:   z       (point of evaluation)
 *          fac     (factor to multiply)
 *          p       (max. polynomial degree)
 *          rretmp  (temporary variables)
 *          rimtmp  (temporary variables)
 * OUTPUT:  Vre     (real part of the integral)
```

```
 *            Vim      (imaginary part of the integral)
 *
 *           / 1
 *           |      N_k(t)
 *    fac *  |   ----------- dt;    z = u+iv, k = 1, ..., p
 *           |     (z-t)^2
 *           / -1
 *
 *                    / t
 *    where N_j(t) =  |     P_(k-2)(xsi) dxsi
 *                 -1 /
 *
 *
 */

void r1(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim);
```

Appendix B

The file 'liblegfct.c'

Listing B.1: Function qtm1 in liblegfct.c

```
#include "liblegfct.h"
#include "mex.h"

#define eps 1e-15
#define BIG 1e+99
#define pi 3.14159265358979323846264338
#define MIN(a,b) ((a)>(b)?(b):(a))
#define INVTOL 1e14

/* qtm1
 *
 * INPUT:   z        (point of evaluation)
 *          p        (max. polynomial degree)
 *          fac      (factor to multiply)
 *          rretmp   (temporary variables)
 *          rimtmp   (temporary variables)
 * OUTPUT:  Vre        (real part of the integral)
 *          Vim      (imaginary part of the integral)
 *
 *         / 1
 *         |
 *    fac * |  P_k(t) * log(t-z) dt; z = u+iv, k = 0,...,p
 *         |
 *         / -1
 *
 *    where P_k(t) are the Legendre polynomials
 *
 */

void qtm1(fcomplex const z, int const p, double const fac,
          double *rretmp, double *rimtmp,
          double *Vre, double *Vim){

```

```
int j,n,nu;
double tmp00, tmp01;
double a1, a2, a3, u, u2, v, v2, b, b2, b1_2, b2_2;
fcomplex z2, ctmp0, ctmp1, ctmp2, ctmp3;
double *rre = rretmp;
double *rim = rimtmp;

/* Get real(z) and imag(z)*/
u = z.r;
v = z.i;
u2 = u*u;
v2 = v*v;

/* Set V[0] and V[1]*/
/* case: z is complex*/
if ( v2 > eps ){
      a1 = atan((1-u)/v);
      a2 = atan((1+u)/v);
      b1_2 = (1+u)*(1+u)+v2;
      b2_2 = (1-u)*(1-u)+v2;
      a3 = log(b1_2/b2_2)*0.5;

      Vre[0] = 0.5*log(b1_2*b2_2)+u*a3-2+v*(a1+a2);
      Vim[0] = a1-a2-u*(a1+a2)+v*a3-pi*v/fabs(v);
      if(p>0){
           Vre[1] = 0.5*a3*(u2-v2-1) + u*v*(a1+a2)-u;
           Vim[1] = u*v*a3-0.5*(u2-v2-1)*(a1+a2)-v;
      }
/* case: z is real, z != +/- 1*/
} else if(u2!=1){
      a1 = fabs(1+u);
      a2 = fabs(1-u);

      Vre[0] = log(a1*a2)+u*log(a1/a2)-2;
      Vim[0] = 0;
      if(p>0){
           Vre[1] = 0.5*(u2-1)*log(a1/a2)-u;
           Vim[1] = 0;
      }
/* case: z is real, z = +/- 1*/
} else {
      Vre[0] = 2*log(2)-2;
```

```
        Vim[0] = 0;
        if(p>0){
            Vre[1] = -u;
            Vim[1] = 0;
        }
  }

  /* Multiply with fac*/
  Vre[0] *= fac;
  Vim[0] *= fac;
  if(p>0){
        Vre[1] *= fac;
        Vim[1] *= fac;
  }

  /*Calculate V[2]..V[p]*/
  if(p>1){

    /* Ellipse parameters */
    b = MIN(1,4.5/pow(p+1.0,1.17));
    b2 = b*b;
    a2 = 1.0+b2;

    /* Inside ellipse: calculation via forward eval */
    if (u2/a2+v2/b2 < 1) {
      /* Calculate initial values */
      ctmp0 =  Cmul(z,Complex(Vre[1],Vim[1]));
      Vre[2] = (fac*2.0)/3.0 + ctmp0.r;
      Vim[2] = 0.0 + ctmp0.i;

      /* Forward recurrence */
      for(j=2; j<=p-1; j++){
        tmp00 = (2.0*j+1.0)/(j+2.0);
        tmp01 = (1.0-j)/(j+2.0);
        ctmp0.r = tmp00*z.r;
        ctmp0.i = tmp00*z.i;
        ctmp1 = Cmul(ctmp0, Complex(Vre[j],Vim[j]));
        ctmp2.r = tmp01* Vre[j-1];
        ctmp2.i = tmp01* Vim[j-1];

        Vre[j+1] = ctmp1.r + ctmp2.r;
        Vim[j+1] = ctmp1.i + ctmp2.i;
```

```
      }
    /* Outside of ellipse, calculation via Gautschi */
    } else {
      /* Assign nu */
      z2 = Complex(fabs(u),fabs(v));
      ctmp0 = Csqrt(Cadd(Complex(-1.0,0), Cmul(z2,z2)));
      tmp00 =  2*log(Cabs(Cadd(z2, ctmp0)));
      nu = (int) (p+ceil(log(INVTOL)/tmp00));

      /* Catch case of overflow: Allocate more memory */
      if(nu>MAX_NU){
          FILE* of = fopen("nu_overflow.txt","w+");
          fprintf(of, "nu = %d\n", nu);
          fclose(of);
          rre = (double*) malloc(nu*sizeof(double));
          rim = (double*) malloc(nu*sizeof(double));
      }

      /* Set initial values */
      rre[nu-1] = 1;
      rim[nu-1] = 0;
      /* Calculate vector r */
      for (n=nu; n>=2; n--){
        ctmp3 = Csub(z,Complex(rre[n-1],rim[n-1]));
        tmp00 = (n*n-1.0)/((4*n*n-1.0)*(ctmp3.r*ctmp3.r
                                         + ctmp3.i*ctmp3.i));
        rre[n-2] =  tmp00*ctmp3.r;
        rim[n-2] = -tmp00*ctmp3.i;
      }

      /* Calculate remaining values of V */
      for (n=2; n<p+1; n++){
        ctmp3 = Cmul(Complex(rre[n-2],rim[n-2]),
                     Complex(Vre[n-1],Vim[n-1]));
        tmp00 = (2*n-1.0)/(n+1.0);
        Vre[n] = tmp00*ctmp3.r;
        Vim[n] = tmp00*ctmp3.i;
      }
    }/* end else */
  }/* end if p > 1*/
}/* end qtm1 */
```

```
/* qt0
 *
 * INPUT:   z       (point of evaluation)
 *          p       (max. polynomial degree)
 *          fac     (factor to multiply)
 *          rretmp  (temporary variables)
 *          rimtmp  (temporary variables)
 * OUTPUT:  Vre     (real part of the integral)
 *          Vim     (imaginary part of the integral)
 *
 *         / 1
 *         |
 *  fac * |  P_k(t) / (z-t) dt;   z = u+iv, k = 0,...,p
 *         |
 *         / -1
 *
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void qt0(fcomplex const z, int const p, double const fac,
         double *rretmp, double *rimtmp,
         double *Vre, double *Vim){

    int j,n,nu;
    double tmp00, tmp01;
    double a2, u, u2, v, v2, b, b2;
    fcomplex c1,c2,z2, ctmp0, ctmp1, ctmp2, ctmp3;
    double *rre = rretmp;
    double *rim = rimtmp;

    /* Get real(z) and imag(z) */
    u = z.r;
    v = z.i;
    u2 = u*u;
    v2 = v*v;

    /* Set V[0] and V[1] */
    /* case: z is complex */
    if ( v2 > eps ){
        c1 = Complex(z.r+1.0,z.i);
        c2 = Complex(z.r-1.0,z.i);
```

```
        ctmp0 = Clog(Cdiv(c1,c2));
        Vre[0] = ctmp0.r;
        Vim[0] = ctmp0.i;
        if(p>0){
            ctmp0 = Cmul(z,Complex(Vre[0],Vim[0]));
            Vre[1] = ctmp0.r - 2.0;
            Vim[1] = ctmp0.i;
        }
    /* case: z is real and not +-1*/
    } else if(fabs(fabs(u)-1)>eps){
        Vre[0] = log(fabs((u+1.0)/(u-1.0)));
        Vim[0] = 0.0;
        if(p>0){
            Vre[1] = u*Vre[0] - 2.0;
            Vim[1] = ctmp0.i;
        }
    /* case: z is real, z = +/- 1
     * Integral has singularity set V[j] = 0 for all j.
     */
    } else {
        for(j=0; j<=p; j++){
            Vre[j] = 0.0;
            Vim[j] = 0.0;
        }
        return;
    }

    /* Multiply with fac */
    Vre[0] *= fac;
    Vim[0] *= fac;
    if(p>0){
        Vre[1] *= fac;
        Vim[1] *= fac;
    }

    /* Calculate V[2]..V[p] */
    if(p>1){
        /* Ellipse parameters */
        b = MIN(1,4.5/pow(p+1.0,1.17));
        b2 = b*b;
        a2 = 1.0+b2;

```

```
        /* Inside ellipse, calculation via forward */
        if (u2/a2+v2/b2 < 1) {

            /* Forward recurrence */
            for(j=1; j<=p-1; j++){
                tmp00 = (2.0*j+1.0)/(j+1.0);
                tmp01 = j/(j+1.0);
                ctmp0.r = tmp00*z.r;
                ctmp0.i = tmp00*z.i;

                ctmp1 = Cmul(ctmp0, Complex(Vre[j],Vim[j]));
                ctmp2.r = tmp01* Vre[j-1];
                ctmp2.i = tmp01* Vim[j-1];

                /* Vd[j+1] = Csub(ctmp1, ctmp2); */
                Vre[j+1] = ctmp1.r - ctmp2.r;
                Vim[j+1] = ctmp1.i - ctmp2.i;
            }
        /* Outside ellipse, calculation via Gautschi */
        } else {
            /* Assign nu */
            z2 = Complex(fabs(u),fabs(v));
            ctmp0 =Csqrt(Cadd(Complex(-1.0,0),Cmul(z2,z2)));
            tmp00 =2*log(Cabs(Cadd(z2, ctmp0)));
            nu = (int) (p+ceil(log(INVTOL)/tmp00));

            /* Catch case of overflow:Allocate more memory*/
            if(nu>MAX_NU){
                FILE* of = fopen("nu_overflow.txt","w+");
                fprintf(of, "nu = %d\n", nu);
                fclose(of);
                rre = (double*) malloc(nu*sizeof(double));
                rim = (double*) malloc(nu*sizeof(double));
            }

            /* Set initial value of r */
            rre[nu-1] = 1.0;
            rim[nu-1] = 0.0;

            /* Calculate vector r */
            for (n=nu; n>=2; n--){
                ctmp3 = Csub(z,Complex(rre[n-1],rim[n-1]));
```

```
                    tmp00 = (n*n)/((4*n*n-1.0)*(ctmp3.r*ctmp3.r
                                                +ctmp3.i*ctmp3.i));
                    rre[n-2] =  tmp00*ctmp3.r;
                    rim[n-2] = -tmp00*ctmp3.i;
               }

                /* Calculate the remaining values of V */
                for (n=2; n<p+1; n++){
                    ctmp3 = Cmul(Complex(rre[n-2],rim[n-2]),
                                 Complex(Vre[n-1],Vim[n-1]));
                    tmp00 = (2*n-1.0)/(n);
                    Vre[n] = tmp00*ctmp3.r;
                    Vim[n] = tmp00*ctmp3.i;
                }
            }/* end else */
        }/* end if p > 1 */
}/* end qt0 */

/* qt1
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *         / 1
 *         |
 *   fac * |  P_k(t) / (z-t)^2 dt;   z = u+iv, k = 0,..., p
 *         |
 *         / -1
 *
 *    where P_k(t) are the Legendre polynomials.
 *
 */

void qt1(fcomplex const z, int const p, double const fac,
         double *rretmp, double *rimtmp,
         double *Vre, double *Vim){

```

```
    int j,n,nu;
    double tmp00, tmp01;
    double a2, u, u2, v, v2, b, b2;
    fcomplex c1,c2,c3,z2, ctmp0, ctmp1, ctmp2, ctmp3;
    double *rre = rretmp;
    double *rim = rimtmp;

    /* Get real(z) and imag(z) */
    u = z.r;
    v = z.i;
    u2 = u*u;
    v2 = v*v;

    /* Set V[0] and V[1] */
    /* case: z is complex */
    if ( v2 > eps ){
        c1 = Csub(Cmul(z,z),Complex(1.0,0.0));
        c2 = Cdiv(Complex(2.0,0),c1);
        Vre[0] = c2.r;
        Vim[0] = c2.i;
        if(p>0){
           c3=Clog(Cdiv(Complex(u+1.0,v),Complex(u-1.0,v)));
           ctmp0 = Csub(Cmul(z,c2),c3);
           Vre[1] = ctmp0.r;
           Vim[1] = ctmp0.i;
        }
    /* case: z is real and in (-1,1) */
    } else if(u2!=1){
        Vre[0] = 2.0/(u2-1);
        Vim[0] = 0.0;
        if(p>0){
            Vre[1] = 2.0*u/(u2-1) - log(fabs((u+1)/(u-1)));
            Vim[1] = 0.0;
        }
    /* case: z is real, z = +/- 1
     * Integral has singularity  set V[j] = 0 for all j.
     */
    } else {
        for(j=0; j<=p; j++){
            Vre[j] = 0.0;
            Vim[j] = 0.0;
        }
```

```
        return;
    }

    /* Multiply with fac */
    Vre[0] *= fac;
    Vim[0] *= fac;
    if(p>0){
        Vre[1] *= fac;
        Vim[1] *= fac;
    }

    /* Calculate V[2]..V[p] */
    if(p>1){
        /* Ellipse parameters */
        b = MIN(1,4.5/pow(p+1.0,1.17));
        b2 = b*b;
        a2 = 1.0+b2;

        /* Inside ellipse, calculation via forward */
        if (u2/a2+v2/b2 < 1) {

            /* Forward recurrence */
            for(j=1; j<=p-1; j++){
                tmp00 = (2.0*j+1.0)/j;
                tmp01 = (j+1.0)/j;
                ctmp0.r = tmp00*z.r;
                ctmp0.i = tmp00*z.i;

                ctmp1 = Cmul(ctmp0, Complex(Vre[j],Vim[j]));
                ctmp2.r = tmp01* Vre[j-1];
                ctmp2.i = tmp01* Vim[j-1];

                Vre[j+1] = ctmp1.r - ctmp2.r;
                Vim[j+1] = ctmp1.i - ctmp2.i;
            }
        /* Outside ellipse, calculation via Gautschi */
        } else {
            /* Assign nu */
            z2 = Complex(fabs(u),fabs(v));
            ctmp0 = Csqrt(Cadd(Complex(-1.0,0), Cmul(z2,z2))
                );
            tmp00 =  2*log(Cabs(Cadd(z2, ctmp0)));
```

```
            nu = (int) (p+ceil(log(INVTOL)/tmp00));

            /* Catch case of overflow:Allocate more memory*/
            if(nu>MAX_NU){
                FILE* of = fopen("nu_overflow.txt","w+");
                fprintf(of, "nu = %d\n", nu);
                fclose(of);
                rre = (double*) malloc(nu*sizeof(double));
                rim = (double*) malloc(nu*sizeof(double));
            }


            /* Set initial value of r */
            rre[nu-1] = 1.0;
            rim[nu-1] = 0.0;

            /* Calculate vector r */
            for (n=nu; n>=2; n--){
                ctmp3 = Csub(z,Complex(rre[n-1],rim[n-1]));
                tmp00 = (n*n-1.0)/((4*n*n-1.0)*
                        (ctmp3.r*ctmp3.r+ctmp3.i*ctmp3.i));
                rre[n-2] =  tmp00*ctmp3.r;
                rim[n-2] = -tmp00*ctmp3.i;
           }

            /* Calculate the remaining values of V */
            for (n=2; n<p+1; n++){
                ctmp3 = Cmul(Complex(rre[n-2],rim[n-2]),
                             Complex(Vre[n-1],Vim[n-1]));
                tmp00 = (2*n-1.0)/(n-1.0);
                Vre[n] = tmp00*ctmp3.r;
                Vim[n] = tmp00*ctmp3.i;
            }

        }/* end else */

    }/* end if p > 1 */
}/* end qt1 */

/* qtn
 *
 * INPUT:   z       (point of evaluation)
```

```
 *           p        (max. polynomial degree)
 *           n        (max. exponent of (z-t)^n)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 *           Vtmpre   (temporary variables)
 *           Vtmpim   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)

 *         / 1
 *         |
 *  fac * |  P_k(t) * log(t-z) dt;   z = u+iv, k = 0,...,p
 *         |
 *         / -1
 * and for j = 0..n
 *         / 1
 *         |
 *  fac * |  P_k(t) / (z-t)^j dt;   z = u+iv, k = 0,...,p
 *         |
 *         / -1
 *
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void qtn(fcomplex const z, int const p, int const n,
    double const fac, double *rretmp, double *rimtmp,
    double *Vtmpre, double *Vtmpim,
    double *Vre, double *Vim){

    int k,i;
    double u,u2,v2,ar,br;
    fcomplex z2, a,b, ctmp, ctmp1, ctmp2, ctmp3;

    /* Get real(z) and imag(z) */
    z2 = Cmul(z,z);
    u  = z.r;
    u2 = u*u;
    v2 = z.i * z.i;
    ctmp = Complex(z2.r-1.0,z2.i);

```

```
    /* Set qt0  V */
    qt0(z,p,fac,rretmp,rimtmp,Vtmpre,Vtmpim);
    for (k=0;k<p+1;k++) {
        Vre[k] = Vtmpre[k];
        Vim[k] = Vtmpim[k];
    }

    /* Set qt1  V */
    if (n>0) {
        qt1(z,p,fac,rretmp,rimtmp,Vtmpre,Vtmpim);
        for (k=0;k<p+1;k++) {
            Vre[k+p+1] = Vtmpre[k];
            Vim[k+p+1] = Vtmpim[k];
        }
    }

    if (n>1) {
        /* Loop for calculation of qti upto degree p*/
        for (k=0;k<p+1;k++) {
            /* Loop for caluclation upto qtn */
            for (i=1;i<n;i++) {
                /* case: z is complex */
                if (v2 > eps) {
                    a = Cdiv(RCmul(2*i,z),RCmul(i+1,ctmp));
                    b = Cdiv(  RCmul((k-i+1.0)*(k+i),
                                Complex(1.0,0)),
                               RCmul(i*(i+1.0),ctmp));
                    ctmp1 = Cmul(a,Complex(Vre[k+i*(p+1)],
                                           Vim[k+i*(p+1)]));
                    ctmp2=Cmul(b,Complex(Vre[k+(i-1)*(p+1)],
                                         Vim[k+(i-1)*(p+1)]));
                    ctmp3 = Cadd(ctmp1,ctmp2);
                    Vre[k+(i+1)*(p+1)] = ctmp3.r;
                    Vim[k+(i+1)*(p+1)] = ctmp3.i;
                /* case: z is real and not +1 or -1*/
                } else if (u2 != 1) {
                    ar = 2.0*i*u / ((i+1)*(u2-1.0));
                    br = (k-i+1.0)*(k+i) /
                            (i*(i+1.0)*(u2-1.0));
                    Vre[k+(i+1)*(p+1)] = ar*Vre[k+i*(p+1)]
                                   + br*Vre[k+(i-1)*(p+1)];
                    Vim[k+(i+1)*(p+1)] = 0.0;
```

```

                    /* case: z = +1 or z = -1 */
                    } else {
                        Vre[k+(i+1)*(p+1)] = 0.0;
                        Vim[k+(i+1)*(p+1)] = 0.0;
                    }
                }
            }
        }
}/* end qtn */

/* q0
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *               / 1
 *               |
 *  fac * 0.5 *  |  P_k(t) / (z-t) dt; z = u+iv, k = 0,...,p
 *               |
 *               / -1
 *
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void q0(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim){

    int k;
    /* Get qt0 */
    qt0(z,p,fac,rretmp,rimtmp,Vre,Vim);

    /* Multiply V with 0.5 and fac */
    for (k=0;k<p+1;k++) {
            Vre[k] = fac*0.5*Vre[k];
```

```
            Vim[k] = fac*0.5*Vim[k];
    }
}

/* q1
 *
 * INPUT:    z        (point of evaluation)
 *           p        (max. polynomial degree)
 *           fac      (factor to multiply)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *                               / 1
 *                               |
 *  fac * -0.5*(z^2-1)^(1/2) *  | P_k(t)/(z-t)^2 dt;
 *                               |
 *                               / -1
 *  for z = u+iv, k = 0, ..., p, or
 *                              / 1
 *                              |
 *  fac * 0.5*(1-x^2)^(1/2) *  | P_k(t)/(x-t)^2 dt;
 *                              |
 *                              / -1
 *  for |x| < 1 real,k = 0,...,p and
 *  where P_k(t) are the Legendre polynomials.
 *
 */

void q1(fcomplex const z, int const p, double const fac,
     double *rretmp, double *rimtmp,
     double *Vre, double *Vim){

    int k;
    double u,v,v2,u2,tmp;
    fcomplex z2, z_tmp, ctmp, ctmp2;

    /* Get real(z) and imag(z) */
    u = z.r;
    v = z.i;
    u2 = u*u;
```

```
    v2 = v*v;
    z2 = Cmul(z,z);

    /* Get qt1 */
    qt1(z,p,fac,rretmp,rimtmp,Vre,Vim);

    /* Multiply V with factor*/
    for (k=0;k<p+1;k++) {
        /* z is complex and not in [-1,1] */
        if (v2 > eps || u2 > 1) {
            z_tmp = Complex(Vre[k],Vim[k]);
            ctmp = RCmul(-0.5*fac,
                         Csqrt(Complex(z2.r-1.0,z2.i)));
            ctmp2 = Cmul(z_tmp,ctmp);
            Vre[k] = ctmp2.r;
            Vim[k] = ctmp2.i;
        }
        /* z is real and |z| < 1*/
        else  {
            tmp = sqrt(1-u2)/2;
            Vre[k] = fac*tmp*Vre[k];
            Vim[k] = fac*tmp*Vim[k];
        }
    }
}

/* qn
 *
 * INPUT:   z        (point of evaluation)
 *          p        (max. polynomial degree)
 *          n        (upper limit of calculation of qn)
 *          fac      (factor to multiply)
 *          rretmp   (temporary variables)
 *          rimtmp   (temporary variables)
 *          Vtmore   (temporary variables)
 *          Vtmpim   (temporary variables)
 * OUTPUT:  Vre      (real part of the integral)
 *          Vim      (imaginary part of the integral)
 *
 *                            / 1
 *                            |
 *  fac * -0.5*(z^2-1)^(1/2) * | P_k(t)/(z-t)^m dt;
```

```
 *                             |
 *                            /  -1
 *   for z = u+iv, k = 0, ..., p, or
 *                           /  1
 *                           |
 *   fac * 0.5*(1-x^2)^(1/2) * | P_k(t)/(x-t)^m dt;
 *                           |
 *                           /  -1
 *   for |x| < 1 real,k = 0,...,p, m = 0..n and
 *   where P_k(t) are the Legendre polynomials.
 *
 */

void qn(fcomplex const z, int const p, int const n,
        double const fac, double *rretmp, double *rimtmp,
        double *Vtmpre, double *Vtmpim,
        double *Vre, double *Vim){

    int k,i;
    double u,u2,v2,ar,br;
    fcomplex z2, a,b, ctmp, ctmp1, ctmp2, ctmp3;

    /* Get real(z) and imag(z) */
    z2 = Cmul(z,z);
    u  = z.r;
    u2 = u*u;
    v2 = z.i * z.i;
    ctmp = Csqrt(Complex(z2.r-1.0,z2.i));

    /* Set q0 -> V */
    q0(z,p,fac,rretmp,rimtmp,Vtmpre,Vtmpim);
    for (k=0;k<p+1;k++) {
        Vre[k] = Vtmpre[k];
        Vim[k] = Vtmpim[k];
    }

    /* Set q1 -> V */
    if (n>0) {
        q1(z,p,fac,rretmp,rimtmp,Vtmpre,Vtmpim);
        for (k=0;k<p+1;k++) {
            Vre[k+p+1] = Vtmpre[k];
            Vim[k+p+1] = Vtmpim[k];
```

```
            }
        }

        if (n>1) {
            /* Loop for calculation of qi upto degree p*/
            for (k=0;k<p+1;k++) {
                /* Loop for caluclation upto qn */
                for (i=0;i<n-1;i++) {
                    /* case: z is complex */
                    if (v2 > eps) {
                        a = Cdiv(RCmul(-2.0*(i+1),z),ctmp);
                        b = Complex((k-i)*(k+i+1.0),0.0);
                        ctmp1=Cmul(a,Complex(Vre[k+(i+1)*(p+1)],
                                     Vim[k+(i+1)*(p+1)]));
                        ctmp2 = Cmul(b,Complex(Vre[k+i*(p+1)],
                                     Vim[k+i*(p+1)]));
                        ctmp3 = Cadd(ctmp1,ctmp2);
                        Vre[k+(i+2)*(p+1)] = ctmp3.r;
                        Vim[k+(i+2)*(p+1)] = ctmp3.i;
                    /* case: z is real and not in [-1,1] */
                    } else if (u2 > 1) {
                        ar = -2.0*(i+1)*u / sqrt(u2-1.0);
                        br = (k+i+1.0)*(k-i);
                        Vre[k+(i+2)*(p+1)]=ar*Vre[k+(i+1)*(p+1)]
                                            + br*Vre[k+i*(p+1)];
                        Vim[k+(i+2)*(p+1)] = 0.0;
                    /* case: z is real and in (-1,1) */
                    } else if (u2 < 1) {
                        ar = -2.0*(i+1)*u / sqrt(1.0-u2);
                        br = (k+i+1.0)*(k-i);
                        Vre[k+(i+2)*(p+1)]=ar*Vre[k+(i+1)*(p+1)]
                                            - br*Vre[k+i*(p+1)];
                        Vim[k+(i+2)*(p+1)] = 0.0;
                    /* case: z = +1 or z = -1 */
                    } else {
                        Vre[k+(i+2)*(p+1)] = 0.0;
                        Vim[k+(i+2)*(p+1)] = 0.0;
                    }
                }
            }
        }
}/* end qn */
```

```
/* r0
 *
 * INPUT:   z        (point of evaluation)
 *          fac      (factor to multiply)
 *          p        (max. polynomial degree)
 *          rretmp   (temporary variables)
 *          rimtmp   (temporary variables)
 * OUTPUT:  Vre      (real part of the integral)
 *          Vim      (imaginary part of the integral)
 *
 *         / 1
 *         |    N_k(t)
 *    fac * |   -------- dt;    z = u+iv, k = 1,..., p
 *         |     z-t
 *         / -1
 *
 *                   / t
 *    where N_k(t) = |     P_(k-2)(xsi) dxsi
 *                -1 /
 *
 *
 */

void r0(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim){

    int j,n,nu;
    double tmp00, tmp01;
    double a1, a2, u, u2, v, v2, b, b2;
    fcomplex c1,c2,c3,z2, ctmp0, ctmp1, ctmp2, ctmp3;
    double *rre = rretmp;
    double *rim = rimtmp;

    /* Get real(z) and imag(z) */
    u = z.r;
    v = z.i;
    u2 = u*u;
    v2 = v*v;

```

```
    /* Set initial values V[0], V[1] and V[2] */
    /* case: z is complex */
    if ( v2 > eps ){
        c1 = Clog(Complex( z.r+1.0, z.i));
        c2 = Clog(Complex( z.r-1.0, z.i));
        c3 = RCmul(0.5,Csub(c1,c2));

        ctmp0 = Cmul(c3,Complex(-z.r+1.0,-z.i));
        ctmp1 = Cmul(c3,Complex(z.r+1.0,z.i));
        Vre[0] = ctmp0.r + 1.0;
        Vim[0] = ctmp0.i;
        Vre[1] = ctmp1.r - 1.0;
        Vim[1] = ctmp1.i;
        if(p>1){
            ctmp0 = Csub(Cmul(c3,Csub(Cmul(z,z),
                        Complex(1.0,0.0))),z);
            Vre[2] = ctmp0.r;
            Vim[2] = ctmp0.i;
        }
    /* case: z is real and z != +/- 1 */
    } else if(u2!=1){
        a1= 0.5*log(fabs((1.0+u)/(u-1.0)));

        Vre[0] =  a1*(1.0-u)+1.0;
        Vim[0] =  0.0;
        Vre[1] =  a1*(1.0+u)-1.0;
        Vim[1] =  0.0;
        if(p>1){
            Vre[2] =  (u2-1.0)*a1-u;
            Vim[2] =  0.0;
        }
    /* case: z = 1*/
    } else if(u == 1.0) {
        Vre[0] =  1.0;
        Vim[0] =  0.0;

        Vre[1] =  BIG;
        Vim[1] =  0.0;

        if(p>1){
            Vre[2] = -u;
            Vim[2] = 1.0;
```

```
        }
    /* case: z = -1*/
    } else if(u == -1.0) {
        Vre[0] =  -BIG;
        Vim[0] =  0.0;

        Vre[1] =  -1.0;
        Vim[1] =  0.0;

        if(p>1){
            Vre[2] = -u;
            Vim[2] = 0.0;
        }
    }


    /* Multiply with fac */
    Vre[0] *= fac;
    Vim[0] *= fac;
    Vre[1] *= fac;
    Vim[1] *= fac;
    if(p>1){
        Vre[2] *= fac;
        Vim[2] *= fac;
    }

    /* Calculate the remaining values of V */
    if(p>2){
        /* Set V[3] */
        ctmp0 = Cmul(z,Complex(Vre[2],Vim[2]));
        Vre[3] = ctmp0.r + 2.0/3.0*fac;
        Vim[3] = ctmp0.i;

        if(p>3){
            /* Ellipse parameter */
            b = MIN(1,4.5/pow(p+1.0,1.17));
            b2 = b*b;
            a2 = 1.0+b2;

            /* Inside ellipse calculation via forward */
            if (u2/a2+v2/b2 < 1) {
                for(j=3; j<=p-1; j++){
```

```
                    tmp00 = (2.0*j-1.0)/(j+1.0);
                    tmp01 = (j-2.0)/(j+1.0);
                    ctmp0.r = tmp00*z.r;
                    ctmp0.i = tmp00*z.i;
                    ctmp1 = Cmul(ctmp0,
                                 Complex(Vre[j],Vim[j]));
                    ctmp2.r = tmp01* Vre[j-1];
                    ctmp2.i = tmp01* Vim[j-1];

                    Vre[j+1] = ctmp1.r - ctmp2.r;
                    Vim[j+1] = ctmp1.i - ctmp2.i;
                }
            /* Outside ellipse, calculation via Gautschi */
            } else {

                /* Assign nu */
                z2 = Complex(fabs(u),fabs(v));
                ctmp0 = Csqrt(Cadd(Complex(-1.0,0),
                                   Cmul(z2,z2)));
                tmp00 =  2*log(Cabs(Cadd(z2, ctmp0)));
                nu = (int) (p+ceil(log(INVTOL)/tmp00));

                /* Catch overflow: Allocate more memory*/
                if(nu>MAX_NU){
                    FILE* of=fopen("nu_overflow.txt","w+");
                    fprintf(of, "nu = %d\n", nu);
                    fclose(of);
                    rre =(double*)malloc(nu*sizeof(double));
                    rim =(double*)malloc(nu*sizeof(double));
                }

                /* Calculate vector r */
                rre[nu-1] = 1.0;
                rim[nu-1] = 0.0;
                for (n=nu; n>=3; n--){
                    ctmp3 = Csub(z,Complex(rre[n-1],
                                           rim[n-1]));
                    tmp00 = (n*n-1.0)/((4.0*n*n-1.0)*
                         (ctmp3.r*ctmp3.r+ctmp3.i*ctmp3.i));
                    rre[n-2] =  tmp00*ctmp3.r;
                    rim[n-2] = -tmp00*ctmp3.i;
                }
```

```
                /* Calcualte vector V */
                for (n=3; n<p; n++){
                    ctmp3 = Cmul(Complex(rre[n-2],rim[n-2]),
                                 Complex(Vre[n],Vim[n]));
                    tmp00 = (2.0*n-1.0)/(n+1.0);
                    Vre[n+1] = tmp00*ctmp3.r;
                    Vim[n+1] = tmp00*ctmp3.i;
                }
            }
        }/*endif p > 3 */
    }/*endif p > 2 */
}/*end r0 */

/* r1
 *
 * INPUT:    z        (point of evaluation)
 *           fac      (factor to multiply)
 *           p        (max. polynomial degree)
 *           rretmp   (temporary variables)
 *           rimtmp   (temporary variables)
 * OUTPUT:   Vre      (real part of the integral)
 *           Vim      (imaginary part of the integral)
 *
 *          / 1
 *          |      N_k(t)
 *    fac * |   ----------- dt;    z = u+iv, k = 1, ..., p
 *          |     (z-t)^2
 *          / -1
 *
 *                   / t
 *    where N_j(t) = |     P_(k-2)(xsi) dxsi
 *                 -1 /
 *
 *
 */

void r1(fcomplex const z, int const p, double const fac,
        double *rretmp, double *rimtmp,
        double *Vre, double *Vim){

    int j,n,nu;
```

```
    double tmp00, tmp01;
    double a1, a2, a3, u, u2, v, v2, b, b2;
    fcomplex c1,c2,c3,c4,c5,z2, ctmp0, ctmp1, ctmp2, ctmp3;
    double *rre = rretmp;
    double *rim = rimtmp;

    /* Get real(z) and imag(z) */
    u = z.r;
    v = z.i;
    u2 = u*u;
    v2 = v*v;

    /* Set initial values V[0], V[1], V[2] and V[3]*/
    /* case: z is complex */
    if ( v2 > eps ){
        c1=Cdiv(Complex(1.0,0.0),Complex(u+1.0,v));
        c2=Cdiv(Complex(1.0,0.0),Complex(u-1.0,v));
        c3=Csub(c2,c1);
        c4=Clog(Complex(-u+1,-v));
        c5=Clog(Complex(-u-1,-v));
        ctmp0 = Csub(RCmul(0.5,Csub(c5,c4)),c1);
        ctmp1 = Csub(c2,RCmul(0.5,Csub(c5,c4)));

        Vre[0] = ctmp0.r;
        Vim[0] = ctmp0.i;
        Vre[1] = ctmp1.r;
        Vim[1] = ctmp1.i;
        if(p>1){
            ctmp0 = RCmul(-1.0,Cmul(z,Csub(c5,c4)));
            Vre[2] = ctmp0.r + 2.0;
            Vim[2] = ctmp0.i;
        }
        if(p>2){
            ctmp0 = Cadd(RCmul(3.0,z),
                    Cmul(Csub(Complex(0.5,0.0),
                        RCmul(1.5,Cmul(z,z))),Csub(c5,c4)));
            Vre[3] = ctmp0.r;
            Vim[3] = ctmp0.i;
        }
    /* case: z is real, z != +/- 1 */
    } else if(u2!=1){
        a1 = u+1;
```

```
        a2 = u-1;
        a3 = log(fabs(a1/a2));

        Vre[0] = -1/a1+0.5*a3;
        Vim[0] = 0.0;
        Vre[1] = 1/a2 - 0.5*a3;
        Vim[1] = 0.0;
        if(p>1){
            Vre[2] = -u*a3+2.0;
            Vim[2] = 0.0;
        }
        if(p>2){
            Vre[3] = (0.5-1.5*u2)*a3+3*u;
            Vim[3] = 0.0;
        }
    /* case: z = 1 */
    } else if(u==1){
        Vre[0] = -0.5;
        Vim[0] = 0.0;
        Vre[1] = 0.0;
        Vim[1] = 0.0;
        if(p>1){
            Vre[2] = 2.0;
            Vim[2] = 0.0;
        }
        if(p>2){
            Vre[3] = 3.0;
            Vim[3] = 0.0;
        }
    /* otherwise */
    } else {
        Vre[0] = 0.0;
        Vim[0] = 0.0;
        Vre[1] = -0.5;
        Vim[1] = 0.0;
        if(p>1){
            Vre[2] = 2.0;
            Vim[2] = 0.0;
        }
        if(p>2){
            Vre[3] = -3.0;
            Vim[3] = 0.0;
```

```
        }
    }

    /* Multiply with factor */
    Vre[0] *= fac;
    Vim[0] *= fac;
        if(p>0){
            Vre[1] *= fac;
            Vim[1] *= fac;
        }
    if(p>1){
            Vre[2] *= fac;
            Vim[2] *= fac;
        }
    if(p>2){
            Vre[3] *= fac;
            Vim[3] *= fac;
        }

    /* Calculate the remaining values of V */
    if(p>3){

        /* Ellipse parameter */
        b = MIN(1,4.5/pow(p+1.0,1.17));
        b2 = b*b;
        a2 = 1.0+b2;

        /* Inside ellipse calculation via forward */
        if (u2/a2+v2/b2 < 1) {
            /* Forward recurrence */
            for(j=3; j<=p-1; j++){
                tmp00 = (2.0*j-1.0)/j;
                tmp01 = (j-1.0)/j;
                ctmp0.r = tmp00*z.r;
                ctmp0.i = tmp00*z.i;

                ctmp1 = Cmul(ctmp0, Complex(Vre[j],Vim[j]));
                ctmp2.r = tmp01* Vre[j-1];
                ctmp2.i = tmp01* Vim[j-1];

                Vre[j+1] = ctmp1.r - ctmp2.r;
                Vim[j+1] = ctmp1.i - ctmp2.i;
```

```
            }
        } else {

            /* Assign nu */
            z2 = Complex(fabs(u),fabs(v));
            ctmp0= Csqrt(Cadd(Complex(-1.0,0),Cmul(z2,z2)));
            tmp00 =  2*log(Cabs(Cadd(z2, ctmp0)));
            nu = (int) (p+ceil(log(INVTOL)/tmp00));

            /* Catch case of overflow:Allocate more memory*/
            if(nu>MAX_NU){
                FILE* of = fopen("nu_overflow.txt","w+");
                fprintf(of, "nu = %d\n", nu);
                fclose(of);
                rre = (double*) malloc(nu*sizeof(double));
                rim = (double*) malloc(nu*sizeof(double));
            }

            /* Calculate vector r */
            rre[nu-1] = 1.0;
            rim[nu-1] = 0.0;
            for (n=nu; n>=3; n--){
                ctmp3 = Csub(z,Complex(rre[n-1],rim[n-1]));
                tmp00 = (n*n)/((4.0*n*n-1.0)*
                       (ctmp3.r*ctmp3.r + ctmp3.i*ctmp3.i));
                rre[n-2] =  tmp00*ctmp3.r;
                rim[n-2] = -tmp00*ctmp3.i;

            }

            /* Calculate vector V */
            for (n=3; n<p; n++){
                ctmp3 = Cmul(Complex(rre[n-2],rim[n-2]),
                             Complex(Vre[n],Vim[n]));
                tmp00 = (2*n-1.0)/(n);
                Vre[n+1] = tmp00*ctmp3.r;
                Vim[n+1] = tmp00*ctmp3.i;
            }
        }/* end else */
    }/* end if p > 3 */
}/*end r1*/
```

Appendix C
Result of all test files

Listing C.1: Result of all test files.

```
Test function QTM1 at various points.
---------------------------------------
---------  NO ERRORS in QTM1  --------
---------------------------------------
Test function QT0 at various points.
---------------------------------------
---------  NO ERRORS in QT0  ---------
---------------------------------------
Test function QT1 at various points.

*** Error at  z = ( 1.01000000000000, 0.00000000000000i)
    exact value = ( 99.50248756218906, 0.00000000000000i)
    num value   = ( 99.50248756218902, 0.00000000000000i)
    exact value = ( 95.19420752975186, 0.00000000000000i)
    num value   = ( 95.19420752975184, 0.00000000000000i)
    exact value = ( 89.43347369077006, 0.00000000000000i)
    num value   = ( 89.43347369077006, 0.00000000000000i)
    exact value = ( 83.02820977456659, 0.00000000000000i)
    num value   = ( 83.02820977456665, 0.00000000000000i)
    exact value = ( 76.42518278103518, 0.00000000000000i)
    num value   = ( 76.42518278103536, 0.00000000000000i)

*** Error at  z = ( 0.50000000000000, 0.00000000000000i)
    exact value = (-2.66666666666667, 0.00000000000000i)
    num value   = (-2.66666666666667, 0.00000000000000i)
    exact value = (-2.43194562200144, 0.00000000000000i)
    num value   = (-2.43194562200144, 0.00000000000000i)
    exact value = ( 1.68541490033107, 0.00000000000000i)
    num value   = ( 1.68541490033117, 0.00000000000000i)
    exact value = ( 5.75468705841589, 0.00000000000000i)
    num value   = ( 5.75468705841612, 0.00000000000000i)
    exact value = ( 4.46658170104374, 0.00000000000000i)
    num value   = ( 4.46658170104392, 0.00000000000000i)
```

```

*** Error at  z = (-0.14285714285714, 0.00000000000000i)
    exact value = (-2.04166666666667, 0.00000000000000i)
    num value   = (-2.04166666666667, 0.00000000000000i)
    exact value = ( 0.57934873911845, 0.00000000000000i)
    num value   = ( 0.57934873911845, 0.00000000000000i)
    exact value = ( 3.83504101656819, 0.00000000000000i)
    num value   = ( 3.83504101656828, 0.00000000000000i)
    exact value = (-2.23868061459485, 0.00000000000000i)
    num value   = (-2.23868061459491, 0.00000000000000i)
    exact value = (-4.36716115055921, 0.00000000000000i)
    num value   = (-4.36716115055941, 0.00000000000000i)

---------  ERRORS in QT1.
Test function QTN(2) at various points.

*** Error at  z = ( 1.01000000000000, 0.00000000000000i)
    exact value = ( 4999.87624068711193, 0.00000000000000i)
    num value   = ( 4999.87624068710829, 0.00000000000000i)
    exact value = ( 4950.37251553179340, 0.00000000000000i)
    num value   = ( 4950.37251553179067, 0.00000000000000i)
    exact value = ( 4857.08492939248390, 0.00000000000000i)
    num value   = ( 4857.08492939248208, 0.00000000000000i)
    exact value = ( 4726.78883130486884, 0.00000000000000i)
    num value   = ( 4726.78883130486975, 0.00000000000000i)
    exact value = ( 4566.48619518150099, 0.00000000000000i)
    num value   = ( 4566.48619518150826, 0.00000000000000i)

*** Error at  z = ( 0.50000000000000, 0.00000000000000i)
    exact value = ( 1.77777777777778, 0.00000000000000i)
    num value   = ( 1.77777777777778, 0.00000000000000i)
    exact value = ( 3.55555555555556, 0.00000000000000i)
    num value   = ( 3.55555555555556, 0.00000000000000i)
    exact value = ( 5.42569621077994, 0.00000000000000i)
    num value   = ( 5.42569621077994, 0.00000000000000i)
    exact value = (-0.65798169527213, 0.00000000000000i)
    num value   = (-0.65798169527236, 0.00000000000000i)
    exact value = (-14.71570849367567, 0.00000000000000i)
    num value   = (-14.71570849367650, 0.00000000000000i)

*** Error at  z = (-0.14285714285714, 0.00000000000000i)
    exact value = (-0.29774305555556, 0.00000000000000i)
```

```
    num value   = (-0.29774305555556, 0.00000000000000i)
    exact value = ( 2.08420138888889, 0.00000000000000i)
    num value   = ( 2.08420138888889, 0.00000000000000i)
    exact value = (-1.16676616423323, 0.00000000000000i)
    num value   = (-1.16676616423323, 0.00000000000000i)
    exact value = (-7.50340115253159, 0.00000000000000i)
    num value   = (-7.50340115253182, 0.00000000000000i)
    exact value = ( 6.66861598684874, 0.00000000000000i)
    num value   = ( 6.66861598684898, 0.00000000000000i)

---------  ERRORS in QTN(2).
Test function QTN(3) at various points.

*** Error at  z = ( 1.10000000000000, 0.00000000000000i)
    exact value = ( 333.29734010006121, 0.00000000000000i)
    num value   = ( 333.29734010006030, 0.00000000000000i)
    exact value = ( 316.74045279487456, 0.00000000000000i)
    num value   = ( 316.74045279487370, 0.00000000000000i)
    exact value = ( 287.94586617715868, 0.00000000000000i)
    num value   = ( 287.94586617715788, 0.00000000000000i)
    exact value = ( 252.17525403238761, 0.00000000000000i)
    num value   = ( 252.17525403238685, 0.00000000000000i)
    exact value = ( 214.07425898643254, 0.00000000000000i)
    num value   = ( 214.07425898643174, 0.00000000000000i)

*** Error at  z = ( 0.50000000000000, 0.00000000000000i)
    exact value = (-2.76543209876543, 0.00000000000000i)
    num value   = (-2.76543209876543, 0.00000000000000i)
    exact value = (-3.16049382716049, 0.00000000000000i)
    num value   = (-3.16049382716049, 0.00000000000000i)
    exact value = (-6.32098765432099, 0.00000000000000i)
    num value   = (-6.32098765432099, 0.00000000000000i)
    exact value = (-12.20332084512706, 0.00000000000000i)
    num value   = (-12.20332084512707, 0.00000000000000i)
    exact value = (-4.78569703201935, 0.00000000000000i)
    num value   = (-4.78569703201880, 0.00000000000000i)

*** Error at  z = (-0.14285714285714, 0.00000000000000i)
    exact value = (-0.75262827932099, 0.00000000000000i)
    num value   = (-0.75262827932099, 0.00000000000000i)
    exact value = ( 0.40526138117284, 0.00000000000000i)
    num value   = ( 0.40526138117284, 0.00000000000000i)
```

```
    exact value = (-2.83682966820988, 0.00000000000000i)
    num value   = (-2.83682966820988, 0.00000000000000i)
    exact value = ( 2.34987165489488, 0.00000000000000i)
    num value   = ( 2.34987165489488, 0.00000000000000i)
    exact value = ( 14.67110635436383, 0.00000000000000i)
    num value   = ( 14.67110635436438, 0.00000000000000i)

---------  ERRORS in QTN(3).
Test function Q0 at various points.
----------------------------------------
---------  NO ERRORS in Q0  ---------
----------------------------------------
Test function Q1 at various points.
----------------------------------------
---------  NO ERRORS in Q1  ---------
----------------------------------------
Test function Q2 at various points.
----------------------------------------
---------  NO ERRORS in Q2  ---------
----------------------------------------
Test function Q3 at various points.

*** Error at  z = ( 1.01000000000000, 0.00000000000000i)
    exact value = (-2849.66647171210616, 0.00000000000000i)
    num value   = (-2849.66647171210479, 0.00000000000000i)
    exact value = (-2835.41919210819606, 0.00000000000000i)
    num value   = (-2835.41919210819515, 0.00000000000000i)
    exact value = (-2807.34573476058995, 0.00000000000000i)
    num value   = (-2807.34573476058995, 0.00000000000000i)
    exact value = (-2766.21382321825831, 0.00000000000000i)
    num value   = (-2766.21382321826059, 0.00000000000000i)
    exact value = (-2713.05732158954697, 0.00000000000000i)
    num value   = (-2713.05732158955198, 0.00000000000000i)

---------  ERRORS in Q3.
Test function R0 at various points.
----------------------------------------
---------  NO ERRORS in R0  ----------
----------------------------------------
Test function R1 at various points.
----------------------------------------
---------  NO ERRORS in R1  ----------
```

```
--------------------------------------------
```

Bibliography

[1] A. Bantle, M. Bantle, C. Erath, P. Heiter, and S. Funken. epsBEM - Efficient and p-Stable Matlab Implementation of 2d BEM for Laplace and Lamé Problems. Technical report, Univeristy of Ulm, 2010 (in progress).

[2] Markus Bantle. Efficient Implementation of Collocation and Boundary Element Methods for Integral Equations. Diplomarbeit, University of Ulm, May 2010.

[3] J.D. Donaldson and David Elliot. A Unified Approach to Quadrature Rules with Asymptotic Estimates of their Remainders. *SIAM Review*, 9:24–82, 1972.

[4] David Elliot. Uniform Asymptotic Expansions of the Jacobi Polynomials and an Associated Function. *Mathematics of Computation*, 25(114):309–315, April 1971.

[5] Walter Gautschi. Computational Aspects of Three-Term Recurrence Relations. *SIAM Review*, 9(1):24–80, January 1967.

[6] Walter Gautschi. Minimal Solutions of Three-Term Recurrence Relation and Orthogonal Polynomials. *Mathematics of Computation*, 36(154):547–554, April 1981.

[7] I.S. Gradstein and I.M. Ryshik. *Summen-, Produkt- und Integraltafeln.* Verlag Harri Deutsch, 1981.

[8] M. Hermann. *Numerische Mathematik.* Oldenbourg Verlag München Wien, 2006.

[9] Peter Deuflhard und Andreas Hohmann. *Numerische Mathematik 1 - Eine algorithmisch orientierte Einführung.* Walter de Gruyter, 2008.

[10] Robert Schaback und Holger Wendland. *Numerische Mathematik.* Springer-Verlag Berlin-Heidelberg, 2005.

[11] Shanjie Zhang and Jianming Jin. *Computation of Special Functions.* Wiley-Interscience publication, 1996.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ulm, den 14. Juni 2010

(Unterschrift)