Visualisierung von Funktionen einer komplexen Veränderlichen insbesondere im Zusammenhang mit der Riemannschen ζ-Funktion

Tim Paehler, WWU Münster

6. 10. 1999

Inhaltsverzeichnis

'Ohne Sinnlichkeit würde uns kein Gegenstand gegeben, und ohne Verstand keiner gedacht werden. Gedanken ohne Inhalt sind leer, Anschauungen ohne Begriffe sind blind. Daher ist es eben so notwendig, seine Begriffe sinnlich zu machen (d.i. ihnen den Gegendstand in der Anschauung beizufügen), als, seine Anschauungen sich verständlich zu machen (d.i. sie unter Begriffe zu bringen). Beide Vermögen, oder Fähigkeiten, können auch ihre Funktionen nicht vertauschen. Der Verstand vermag nichts anzuschauen, und die Sinne nichts zu denken. Nur daraus, daß sie sich vereinigen, kann Erkenntnis entspringen.'
– Immanuel Kant, Kritik der reinen Vernunft B 76

1 Einleitung in das Thema

In dieser Arbeit soll im Zusammenhang mit der Programmierung eines Funktionenplotters die Problematik und Anwendung der Visualisierung von Funktionen einer komplexen Veränderlichen (im folgenden verkürzt 'komplexe Funktionen' genannt) untersucht werden. Dabei werden - nicht zuletzt mit Blick auf die vielfältigen Aspekte des Mathematiklehrerberufes - verschiedene mathematische und außermathematische Bereiche angeschnitten: Didaktik und Visualistik, Informatik, Reine Mathematik, Numerik sowie Analyse und Anwendung von Computer-Algebra-Systemen (CAS).
Die Unterteilung der Arbeit nach logischen Gesichtspunkten entspricht dabei ihrer chronologischen Gliederung in verschiedene Entwicklungsphasen:

1. **Didaktik und Visualistik: Problematik der Darstellung komplexer Funktionen**
   Nachdem die Frage nach dem wissenschaftlichen Sinn und Nutzen graphischer Darstellungen überhaupt behandelt worden ist, werden die verschiedenen Möglichkeiten der Darstellung von Graphen komplexer Funktionen analysiert und gegeneinander abgewogen.

2. **Informatik: Objektorientierte Programmierung eines Funktionenplotters in Java**
   Durch Entwicklung einer Java-Klassen-Bibliothek, die die Konstruktion, Berechnung und Darstellung von komplexen Funktionen erlaubt, wird anhand eines Baukastensystems ein flexibler und vielseitiger Funktionenplotter programmiert, dessen Spezifikation den in 1. gewonnenen Erkenntnissen Rechnung trägt. Dabei werden u.a. unabhängige Entwurfsmuster einerseits und die technischen Eigenheiten und Idiome der Programmiersprache Java andererseits berücksichtigt.

3. **Reine Mathematik: Berechnung der Riemannschen ζ-Funktion**
   Exemplarisch für eine Erweiterung der Programmbibliothek soll daraufhin die Riemannsche ζ-Funktion berechnet und dargestellt werden. Um dies zu bewerkstelligen, wird zunächst mit Hilfe der Euler-MacLaurin Summenformel eine numerisch hinreichend approximierende Berechnung implementiert.

4. **Numerik: Betrachtung von numerischer Genauigkeit und Zeitverhalten**
   Das numerische Verhalten der Implementation bedarf einer genaueren Untersuchung hinsichtlich Genauigkeit und Laufzeitverhalten. Dabei wird ein erlaubter absoluter Fehler vorgegeben, und anhand von Abschätzungen werden

verschiedene Optimierungsmöglichkeiten bezüglich Konvergenz und Berechnungszeit analysiert.

5. **Analyse und Anwendung von Computer-Algebra-Systemen: Vergleich des Funktionenplotters mit dem CAS MuPAD**
   Es sollen dann der Funktionenplotter mit dem an der GHS Paderborn entwickelten CAS MuPAD verglichen werden und in einem Ausblick Möglichkeiten der Kommunikation zwischen beiden diskutiert werden.

2 Problematik der Darstellung komplexer Funktionen

2.1 Wozu graphische Darstellung komplexer Funktionen?

Der Körper der komplexen Zahlen ist im streng abstrakten Sinne lediglich eine Menge, deren Elemente bezüglich algebraischer Operationen bestimmten Regeln unterworfen sind. Es stellt sich damit in der Tat - verallgemeinert - die Frage nach den Vorteilen einer graphischen Darstellung von algebraischen und analytischen Strukturen. Visualisierung bedeutet ja stets eine Reduktion abstrakter Eigenschaften auf konkrete (nämlich auf die Kantsche Anschauungsraumzeit):[1] Im Hinblick auf die potentielle Vielfalt der Interpretationen eines formalen Zusammenhangs setzt das Bild Parameter ein, nimmt eine Perspektive ein und vollzieht damit Vereinfachungen. Es ist dagegen gerade die Stärke der Mathematik, durch ihre Abstützung auf das Formale einer scheinbar beliebigen Vielfalt an Interpretationen offenzustehen. In rein informationstheoretischer Hinsicht kann also das Bild gegenüber der Formel nur einen begrenzteren Einblick in einen mathematischen Zusammenhang darstellen. Auch historisch-faktisch erscheint die Verlagerung mathematischer Sachverhalte auf die Anschauungsebene ein Rückschritt gegenüber der mit Descartes' analytischer Geometrie begründeten Tradition der Algebraisierung geometrischer Darstellungen.[2]
Die Vorteile des Bildes gegenüber der Formel treten jedoch hervor, wenn man den mathematischen Zusammenhang nicht losgelöst vom Modus des Mathematik-Treibens und diesen wiederum nicht losgelöst vom Mathematik-Treibenden selbst betrachtet, sondern die Notwendigkeit der Einheit aus Mathematik und mathematischem Verstehensprozeß annimmt: Wahr ist in der Mathematik nur, was bewiesen, also auf die Logik reduziert worden ist. Die Stimmigkeit der Logik selbst läßt sich jedoch nicht mehr beweisen, sondern bedarf ihrerseits der unmittelbaren Intuition des Individuums als Rechtfertigung.[3] Das Verständnis von Mathematik als Konstruktionsprozeß begründet insofern auch ihr 'historisches Element', welches seine praktische Anwendung als didaktisches Mittel in den meisten Lehrbüchern und -veranstaltungen findet.

Mit Blick auf die Verbindung von Mensch und Mathematik - sei sie nun philosophisch oder didaktisch motiviert - kann man den Nachteil des Bildes gegenüber dem formalen Zusammenhang - nämlich die Reduktion der Komplexität mathema-

[1] Vgl. [Franke 1999] S. 4-6.
[2] Vgl. [Gericke 1992], [Otte 1994] S. 361-363.
[3] Es ist die grundlegende These der neueren Wissenschafts- und Erkenntnistheorie, daß der für die Geisteswissenschaften konstitutive hermeneutische Zirkel in Minimalform auch von den 'exakten' Wissenschaften berücksichtigt werden muß. Er äußert sich z.B. in der Physik in der Formulierung der Quantentheorie, in der Mathematik zeigt er sich in der Unbeweisbarkeit der Widerspruchsfreiheit hinreichend komplexer axiomatischer Systeme. Vgl. [Otte 1994] v.a. 307-336, [Weizsäcker 1993] S. 113-122, [Hofstadter 1991].

tischer Beziehungen - in einen Vorteil wenden:[4] Von den zahllosen Aspekten, die eine algebraische Formulierung impliziert, werden einige wenige in der Anschauung dargestellt. Der verstehensfördernde Nutzen besteht also darin, durch Reduktion Orientierung zu ermöglichen.[5]

Die Reduktion eines formalen Zusammenhangs auf wenige, innerhalb einer Betrachtungsweise 'wesentliche' Aspekte kann natürlich statt visuell auch auf formale Weise geschehen: etwa durch Betrachtung von Spezialfällen, durch Einsetzen konkreter Zahlen oder durch Nennung von Beispielen.[6] Versteht man aber einerseits mit der Lernpsychologie Lernen als Prozeß, der die gesamten Rezeptionsmöglichkeiten des Menschen in Anspruch nehmen soll[7] und andererseits wissenschaftliche Objektivität als Invarianz unter beliebig großer 'Variation der Perspektive'[8], so wird klar, daß der Prozeß des Erlernens eines wissenschaftlichen Sachverhaltes genau dann optimal verläuft, wenn er durch möglichst verschiedenartige Reize und Aspekte ausgelöst und unterstützt wird. H.W. Franke, dessen Hauptarbeitsgebiet im Bereich der Computergraphik liegt, kommt in seinen Untersuchungen zu vergleichbaren Ergebnissen und stellt dabei die intuitive Überlegenheit der visuellen Erfassung heraus:[9]

'Ein bedenkenswerter Hinweis zur Problematik kommt von den Wahrnehmungspsychologen. Aus ihrer Sicht stellt sich die Aufgabe, bestimmte Zusammenhänge, Aufgaben oder Probleme mathematischer Natur so zu verschlüsseln, daß sie möglichst klar ausgedrückt werden. Wenn man will, kann man den größten Teil der Mathematik sowieso als eine Fülle von Tautologien ansehen, und die Tatsache, daß wir überhaupt Mathematik betreiben, liegt in der Unzulänglichkeit des menschlichen Intellekts, der sich stets nur auf Teilaspekte konzentrieren kann und die großen Zusammenhänge prinzipiell nicht erkennt. Es kommt dann darauf an, welches Kodierungssystem der menschlichen Einsicht am besten angemessen ist. Diese Einsicht beruht zum großen Teil auf der Sinneswahrnehmung und der Datenverarbeitung der einlaufenden Reizmuster im Gehirn. Und es ist eine Tatsache, daß der Mensch ein Augenwesen ist, daß also der größte Teil der Datenanalyse der visuellen Auslese gewidmet ist. Hier liegt der entscheidende Unterschied zwischen der Formel und dem Bild: Sind wir mit einer Formel konfrontiert, dann müssen wir den größten Teil der Denkkapazität ihrer Interpretation widmen. Bei der Konfrontation mit einem Bild dagegen erfolgt die Interpretation in Form von Gestaltbildungsprozessen unterbewußt und wird dem Bewußtsein dadurch unmittelbar zugänglich - und die Denkkapazität kann dann auf weiterführende Aufgaben angewandt werden.'[10]

Wir werden im folgenden Unterkapitel weiter auf die wahrnehmungspsychologischen Aspekte des Erfassungsprozesses eingehen.

Exkurs: Vermutungen

Ich will die Bedeutung von Veranschaulichungen für die mathematische Praxis im folgenden noch an einem kurzen Beispiel beleuchten.

[4] Tatsächlich ist die eine wesentliche Dimension der Didaktik die der 'Reduktion'; vgl. [Meyer, Jank 1991] S. 80-84. Mit der zweiten, der der 'Inszenierung' beschäftigen wir uns im folgenden Unterkapitel.

[5] In der Fachdidaktik wie in der Wissenschaftstheorie wird das Wissen häufig auf die komplementären Begriffe 'Orientierungswissen' und 'Verfügungswissen' projiziert. Vgl. z.B. [Muckenfuß 1995] S. 58-72. Die allgemeine Didaktik verwendet hingegen das geläufigere Begriffspaar 'Bildung' und 'Fachwissen', wobei man mit [Otte 1994] S. 19-46 das Verhältnis der beiden Paare als gleichwertig ansehen kann.

[6] In diesem Zusammenhang fällt mir spontan das Bonmot eines Hochschullehrers ein, demnach es ohne die Funktion $\frac{1}{z}$ 'keine Funktionentheorie gäbe'.

[7] In der Mathematikdidaktik z.B. wird dieser Zusammenhang oft mit den Arbeiten von J. Bruner verbunden. Vgl. [Tietze et al. 1997] S. 55ff, [Führer 1997] S. 248ff.

[8] [Otte 1994] S.28

[9] Vgl. auch [Gibson 1973] S. 80.

[10] [Franke 1999] S. 48

Die wohl fruchtbarsten Prozesse in der Geschichte der mathematischen Forschung sind durch die Formulierung von Vermutungen entstanden, deren Beweis nicht sofort erbracht werden konnte. So hat beispielsweise die Vermutung, Euklids fünftes Axiom, das 'Parallelenaxiom', sei durch die übrigen vier Axiome beweisbar, über 1500 Jahre die Mathematiker beschäftigt und letztlich zur Nichteuklidischen Geometrie geführt und damit - durch die Loslösung der Geometrie vom unmittelbaren Anschauungsraum - den Wandel zur modernen 'Strukturmathematik' eingeleitet.[11] Genauso läßt sich die moderne Algebra vom Ausgangspunkt klassischer geometrischer Probleme (des Würfelverdopplungsproblems, der Konstruktion regelmäßiger N-Ecke, der Quadratur des Kreises) aus motivieren und entwickeln.[12] Die Liste der Befruchtung der Mathematik durch leicht erfaßbare Probleme ließe sich noch weiter fortsetzen (z.B. die Fermatsche Vermutung, das Vierfarben-Problem, etc.);[13] es dürfte aber einleuchten, daß einfach formulierbare - und damit leicht kommunizierbare - Probleme eine große Bündelungskraft mathematischer Bemühungen und damit reichhaltige Resultate zur Folge haben.
Eine der Vermutungen, die zu dieser Liste gehört und die darüber hinaus eine wichtige Beziehung zwischen analytischer und algebraischer Mathematik herstellt, ist die seit über 100 Jahren unbewiesene Riemannsche Vermutung. Sie besagt, daß die 'nichttrivialen' Nullstellen der Riemannschen ζ-Funktion (s. Kapitel 4) allesamt auf der kritischen Geraden $\Re z = \frac{1}{2}$ liegen. Die Riemannsche ζ-Funktion steht in historischem Zusammenhang mit dem Beweis des Primzahlsatzes, wie im folgenden kurz angedeutet werden soll.
Die Riemannsche ζ-Funktion wird im allgemeinen definiert durch

$$\zeta(s) \quad = \quad \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Daraus ergibt sich mit der Eindeutigkeit der Primfaktorzerlegung für jede Zahl n

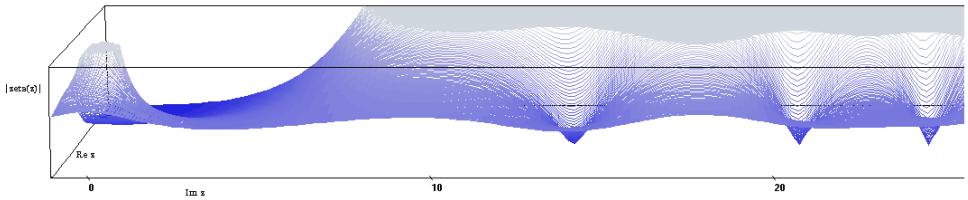
$$\begin{aligned} \zeta(s) \quad &= \quad \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p \in \mathbb{P}} \left(1 + \frac{1}{p^s} + \frac{1}{p^{2s}} + \frac{1}{p^{3s}} + \cdots\right) \\ &= \quad \prod_{p \in \mathbb{P}} \frac{1}{1 - p^{-s}}. \end{aligned}$$

Bildet man von diesem Term nun die logarithmische Ableitung $\frac{\zeta'(s)}{\zeta(s)}$, so erhält man eine Reihe, die eng mit der Primzahlen zählenden Funktion $\pi(x) := \sum_{p \leq x} 1$ ist. Der Primzahlsatz (bewiesen erstmalig 1896 durch de la Vallée de Poussin und Hadamard) besagt

$$\lim_{x \to \infty} \left(\frac{\pi(x)}{\frac{x}{\log x}} \right) \quad = \quad 1. \tag{1}$$

Im Zuge des Beweises dieses Satzes muß nun ein Term, der die logarithmische Ableitung der ζ-Funktion enthält, für eine Abschätzung parallel zur imaginären Achse integriert werden. Man erkennt dabei, daß die Kenntnis der Nullstellen von $\zeta(s)$ (und damit die der Pole von $\frac{\zeta'(s)}{\zeta(s)}$) wesentlich für eine Abschätzung für $\pi(x)$ ist.[14] Genauer gesagt ist - hier allein als Ergebnis angeführt - die Riemannsche Vermutung äquivalent zu der Vermutung, daß für die Differenz $r(x)$ zwischen $\pi(x)$

[11] Vgl. [Koecher, Krieg 1993] S. 4-14, [Hofstadter 1991] S. 96-103, [Otte 1994] S. 361-363.
[12] Dies tut z.B. [Lorenz 1992].
[13] Vgl. die Auflistung unter [MT 1999]
[14] Vgl. [Wolfart 1996] S. 9-11, [Rademacher 1973] S. 88-116, [Freitag, Busam 1995] S. 427-454.



Die ζ-Funktion als analytische Landschaft mit Blick auf die kritische Gerade: Links der (abgeschnittene) Pol bei $z = 1$, daneben drei der nichttrivialen Nullstellen. Die dem Beobachter zugewandte Schnittlinie wird durch $\zeta(3 + ti)$ gebildet.

und dem Integrallogarithmus $\mathrm{Li}(x) = \int_2^x \frac{1}{\log t} dt$ gilt:

$$r(x) := |\mathrm{Li}(x) - \pi(x)| = \mathcal{O}(\sqrt{x} \log x). \tag{2}$$

Diese Abschätzung wäre gegenüber den bislang bewiesenen[15] ein großer Fortschritt in der Berechnung der Auftrittswahrscheinlichkeit von Primzahlen in einem vorgegebenen Intervall.
Es ist nun weitaus zu optimistisch gedacht, daß man durch bloßes Betrachten des Graphen der Riemannschen ζ-Funktion zu einer Beweisidee der Vermutung gelangen könnte; absurd ist hingegen nicht die Vorstellung, anhand des Funktionsgraphen 'kleinere' Vermutungen aufzustellen und zu versuchen, diese entweder zu beweisen oder sie in einen Zusammenhang mit der Riemannschen Vermutung zu bringen. Z.B. könnte man die den nichttrivialen Nullstellen in einer analytischen Landschaft entspringenden 'Rinnen' untersuchen und darüber nachdenken, ob ihnen eine Eigenschaft innewohnt, die sich von den Nullstellen bis in den einfacher zu bearbeitetenden Bereich rechts des kritischen Streifens fortpflanzt. Natürlich sind von dieser Art der Vorgehensweise keineswegs schnellere Ergebnisse im Hinblick auf einen Beweis der Riemannschen Vermutung zu erwarten als von den bislang erprobten Methoden, aber die Orientierung an der graphischen Darstellung würde nicht nur das Problem plastischer (und damit eindringlicher) darstellen, sondern auch seine Komplexität einem weiteren Publikum eröffnen.
Damit nun ein solches Vorhaben Aussicht auf Erfolg hätte, müßte der Betrachter allerdings entweder bereits einen Großteil der mathematischen Kenntnisse in 'graphisch kodiertem' Wissen vorrätig haben, oder aber in der Lage sein, anschauliche und mathematische Eigenschaften einer Funktion ineinander überzuführen. Diese Art der Vorgehensweise würde geleitet aus der Perspektive der Mathematikdidaktik, in der Darstellung und symbolische Form einer mathematischen Struktur als die komplementären Aspekte einer höherliegenden Einheit gesehen werden.[16]

Als Fazit dieses Unterkapitels läßt sich nun also sagen: Eine graphische Darstellung ist ein erweiternder Aspekt der algebraischen Formalstruktur mathematischer Zusammenhänge, der informationstheoretisch nicht notwendig ist (und damit im Rahmen eines praktizierten Informationsminimalismus in der Mathematik häufig als entbehrlich betrachtet wird); in Bezug auf den Mathematik konstituierenden

[15] Vgl. [Weisstein 1998]
[16] Die 'analoge' Denkweise ist demnach der 'digitalen' ebenbürtig, es kann aber keine der beiden die andere ersetzen:
'*Bruner* hebt die Wichtigkeit des intuitiven Denkens für das Problemlösen hervor und meint damit in erster Linie das Arbeiten mit bildhaft analogen Repräsentationen. Während unserer langjährigen Beobachtung der Problemlöseprozesse von Schülern und Studenten hat sich eine zeichnerisch-graphische Problemstellung als besonders effektiv erwiesen. Ikonische Darstellungen können aber auch Fehlvorstellungen nach sich ziehen.' [Tietze et al. 1997] S. 55. Vgl. auch [Otte 1994] S. 370ff.

Erklärungs- und Verstehensprozeß aber sind Abbildungen auf die Anschauungsraumzeit als Komplement abstrakter Formulierungen von großer Wichtigkeit.[17]

2.2 Mögliche Darstellungsformen komplexer Funktionen

Die oben vorgenommene Bindung der graphischen Darstellung an den Erklärungs- und Verstehensprozeß bringt uns zu der allgemeinen und wichtigsten Frage der Visualisierung: *Was* soll *Wie* dargestellt werden, um *Was* zu veranschaulichen? Die erste Variable ist im Thema dieser Arbeit festgelegt. Da mir im Sinne einer Wiederverwendbarkeit von Komponenten an einem möglichst weiten Spektrum von Veranschaulichungsmöglichkeiten gelegen ist, möchte ich die Behandlung der dritten Variable soweit wie möglich hintenanstellen (wir werden gleich sehen, daß dies nur bedingt möglich ist) und betrachte im folgenden zunächst das *Wie*.
Komplexe Funktionen haben als Graphen G eine Teilmenge des $\mathbb{C}^2$, müßten also ohne Informationsverlust vierdimensional dargestellt werden.[18] Obwohl viele Lehrbücher zur Funktionentheorie in unterschiedlichem Maße graphische Darstellungen von Funktionen dem formalen Gedankengang beifügen (es scheint die Faustregel zu gelten: je moderner das Lehrbuch, desto mehr Augenmerk wird den Abbildungen gewidmet), wird von einer *Diskussion* der Darstellungsmöglichkeiten zumeist abgesehen.[19] Ich will hier daher etwas ausführlicher auf diese Thematik eingehen.
Ein möglicher Schritt in der systematischen Analyse der Darstellungsmöglichkeiten besteht zunächst darin, sinnlich (hier: visuell) erfaßbare Bereiche zu suchen, auf deren Variablen die vier Komponenten abgebildet werden. Zu den elementaren Wahrnehmungsgrößen gehören dabei:

- Raumdimensionen (Länge, Breite, Höhe) als Ortsvektor (s.u.)
- Raumdimensionen (Länge, Breite, Höhe) als Richtungsvektor (s.u.)
- Zeit/Bewegung
- Textur/Farbe

Die Liste ließe sich nun noch durch Operationen auf diesen Bereichen erweitern (z.B. Geschwindigkeit als Differential des Raumes nach der Zeit), wodurch allerdings komplexere Wahrnehmungsgrößen[20] entstehen. Ich konzentriere mich daher im folgenden auf die oben genannten Variablen, wobei diese im einzelnen noch einer genaueren Betrachtung bedürfen. Diese Betrachtung soll vor allem unter wahrnehmungspsychologischen und didaktischen Gesichtspunkten durchgeführt werden:

- Die Darstellung soll - im Hinblick auf das Fazit von 2.1 - größtmögliche intuitive (im Gegensatz zu: kognitiver) Erfaßbarkeit garantieren. Dies tut sie nach Darstellung der Wahrnehmungspsychologie dann, wenn sie auf eingeübte Rezeptionssysteme trifft.[21] Im allgemeinen bedeutet dies, eine graphische

[17] Auf die gesellschaftliche Perspektive visueller Darstellung von Spezialistenwissen kann ich hier leider nicht eingehen; die Rechtfertigung von Visualisierung als Faktor zur Integration der nach C.P. Snow dissoziierten 'Zwei Kulturen' ist jedoch ebenfalls ein starkes Argument für eine vermehrte Erforschung und Anwendung graphischer Darstellungsmöglichkeiten. Vgl. [Schirra, Strothotte 1999].

[18] Genauer gesagt ist der Graph Teilmenge eines vierdimensionalen $\mathbb{R}$-Vektorraumes, ein Informationsverlust ist dabei natürlich bereits durch die im Rahmen der Computergraphik notwendige Abbildung von kontinuierlichen Intervallen auf eine diskrete Punktmenge gegeben.

[19] Die einzige von mir entdeckte Ausnahme ist [Freitag, Busam 1995] S. 53-57. Dort findet sich eine Auflistung von drei Möglichkeiten der Visualisierung: transformiertes Koordinatennetz (s. u.), Höhenlinienmodell und analytische Landschaft.

[20] Der Wahrnehmungspsychologe Gibson nennt diese 'Variablen höherer Ordnung' und weist darauf hin, daß ihnen erst mit fortgeschrittenem Lernprozeß Bedeutung zugemessen werden kann. Vgl. [Gibson 1973] S. 301/302.

[21] Vgl. [Rock 1985] S. 82-87, S. 110ff, [Gibson 1973] S. 325ff.

Darstellung zu wählen, die an bereits gebräuchliche Darstellungen anknüpft bzw. eine, die möglichst nahe an der phänomenologischen Lebenswelt des Betrachters liegt.[22]

- Die Anordnungsrelation innerhalb der einzelnen Koordinaten (die in $\mathbb{C} \simeq \mathbb{R}^2$ gegeben ist), soll ebenso möglichst intuitiv dargestellt werden, um Orientierung und Vergleiche innerhalb des Graphen zu ermöglichen.

- Die einzelnen Koordinaten des Definitions- und Wertebereiches sollen untereinander in einem größeren wahrnehmungspsychologischen Zusammenhang stehen als zwischen den Bereichen (Beispiel s.u.).

2.2.1 Raumdarstellung mit Ortsvektoren

Die Darstellung eines Wertes durch einen Ortsvektor im Raum $\vec{x} = (x_1, x_2, x_3)$ ist vor allem aufgrund der stärksten intuitiven Wirkung wohl die verbreiteteste Art der Visualisierung; deshalb soll dieser hier auch die detaillierteste Untersuchung gewidmet werden. Die Darstellungen reichen von eindimensionaler Variation (Balkendiagramme: $A \sim \Delta x_1$, Tortendiagramme: $A \sim \Delta\varphi$ etc.) über zweidimensionale (Graphen reeller Funktionen $x_2 = f(x_1)$) bis zur Ausnutzung aller drei Raumdimensionen, deren einfachster Fall die Abbildung eines Funktionsgraphen auf ein 'Gebirge' im $\mathbb{R}^3$ ist. In Bezug auf komplexe Funktionsgraphen bilden die drei Raumdimensionen zunächst einen unbefriedigenden Darstellungsraum, da die fehlende Dimension durch einen zusätzlichen eindimensionalen Variablenbereich geliefert werden muß. Dieser Bereich (der ebenfalls eine elementare Wahrnehmungsgröße sein sollte) hat jedoch gegenüber den drei Raumdimensionen wahrnehmungspsychologisch einen herausgehobenen Status. Als konkretes Beispiel für diese Problematik kann man sich die Visualisierung im dreidimensionalen Raum mit einer Farbe als zusätzlichem Darstellungsparameter vorstellen, wobei man die (x_1, x_2)-Ebene für die Darstellung des Definitionsbereiches verwendet, den Realteil des Bildes $\Re f(z)$ in der x_3-Achse aufträgt und den Imaginärteil $\Im f(z)$ einem Farbwert an dem jeweiligen Punkt zuordnet. Eine solche Darstellung würde intuitiv betrachtet z und $\Re f(z)$ in einen engeren Zusammenhang rücken als $\Re f(z)$ und $\Im f(z)$, was nicht wünschenswert ist.

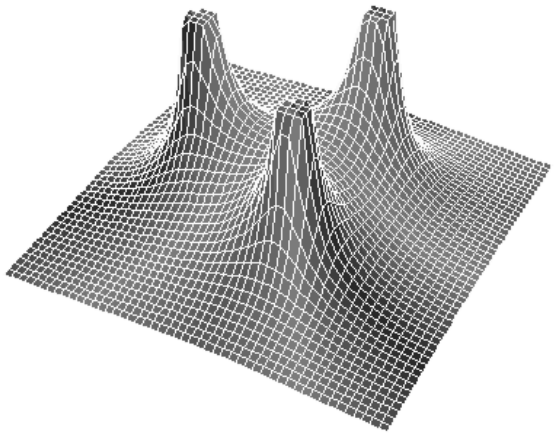
Analytische Landschaft und Koordinatennetz-Darstellung

Das Problem wird zumindest teilweise dadurch gelöst, daß man nicht $f(z)$ abbildet, sondern $|f(z)|$ oder $|f(z)|^2$ auf der dritten Raumdimension über der komplexen Ebene aufträgt, beide Darstellungsarten werden im allgemeinen als 'analytische Landschaft' bezeichnet.[23] Diese Art der Darstellung läßt nun zwar verschiedene Eigenschaften der Funktion unsichtbar werden (so etwa das in der Funktionentheorie fundamentale Kriterium der Holomorphie), es bleiben aber Nullstellen und Pole (bei hinreichend 'gutartigen', z.B. meromorphen, Funktionen) sichtbar.

Die Holomorphie einer Funktion dagegen läßt sich auf eine andere Art und Weise veranschaulichen: Die Abbildung eines transformierten Koordinatennetzes in der

[22] Im besonderen bedeutet dies, Darstellungen in Lehrbüchern zur Funktionentheorie zu betrachten und zu vergleichen bzw. die lebensweltlichen Anschauungsformen, seien sie nun natürlich (z.B. räumliches Denken) oder kulturbedingt (z.B. Informations-Graphiken in Fernsehen und Zeitung, etwa zu Wahl- und Umfrageergebnissen) zu untersuchen. Natürlich kann dies im Rahmen dieser Arbeit nur stichpunktartig geschehen.

[23] Ich verwende diesen Terminus im folgenden für die Darstellung von $|f(z)|^2$. Die Abbildung als analytische Landschaft wird in z.B. in [Fischer/Lieb 1994] gegenüber anderen Darstellungsarten bevorzugt. Auch in [Freitag, Busam 1995] finden sich mehrfach Abbildungen dieser Art.



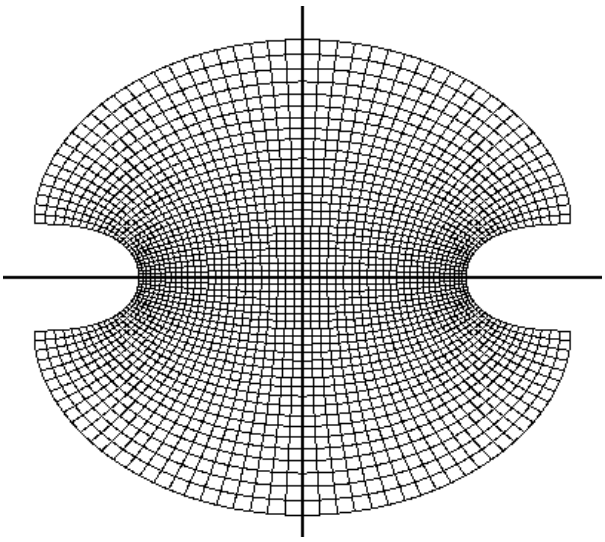
Darstellung einer Funktion als analytische Landschaft (mit abgeschnittenen Polen)

zweidimensionalen Ebene. Dabei wird im Definitionsbereich ein rechteckförmiges, achsenparalleles Netz aus Koordinatenlinien gebildet und das Bild dieses Netzes unter einer Funktion f in der Werte-Ebene betrachtet. Sofern f nun in dem betrachteten Gebiet holomorph ist, sind die Winkel zwischen je zwei Linien im Definitionsbereich und zwischen ihren Bildern identisch. Man spricht dabei auch von (lokaler) 'Konformität'.[24] Die zweidimensionale Darstellung eines transformierten Netzes hat allerdings wiederum den Nachteil, daß nicht injektive (und damit so gut wie alle im Rahmen der Funktionentheorie interessierenden) Funktionen sich nur in stark eingeschränkten Gebieten eindeutig visualisieren lassen. Zudem genügt diese Art der Darstellung nur in geringem Maße der Forderung nach einer intuitiv vermittelten Anordnungsrelation. Da lediglich benachbarte Knoten im Urbild auf benachbarte Knoten im Bild abgebildet werden, ansonsten aber fast beliebige Drehungen und Verzerrungen des 'Netzes' möglich sind, ist es häufig nicht auf den ersten Blick erkennbar, welchem Definitionswert ein Funktionswert zugeordnet ist - man ist, nicht zuletzt aus der reellen Analysis, stets Graphen und Graphiken gewohnt, in denen die Koordinaten auf geraden Achsen aufgetragen werden und wo die intuitiven Metaphern "je weiter rechts, desto mehr x" und "je weiter oben, desto mehr y" gelten.
Wir haben damit zwei verschiedene Visualisierungsstrategien, die verschiedene Aspekte von komplexen Funktionen zu veranschaulichen in der Lage sind. Es gilt also bei der Auswahl der Darstellung die Frage nach dem zu veranschaulichenden *Was* erneut in den Vordergrund zu rücken, da sie vom *Wie* in diesem Fall offenbar nicht zu trennen ist.

2D- vs. 3D-Darstellung

Eine weitere Frage, die durch den Vergleich der oben beschriebenen Abbildungsmöglichkeiten aufgeworfen wird, ist die nach der Anzahl der zu verwendenden räum-

[24] Die Darstellung von Funktionen als transformiertes Koordinatennetz wird in den meisten Lehrbüchern der Funktionentheorie für verschiedene Beispiele verwendet, in [Herz 1996] werden sogar Übungsaufgaben gestellt, in denen das Bild eines Quadranten oder Streifens unter der Abbildung durch eine spezielle Funktion eingezeichnet werden muß. Die dieser Darstellung zugrundeliegende geometrische Lesart der Funktionentheorie ist die von Bernhard Riemann, der holomorphe Funktionen in erster Linie als konforme Abbildungen zwischen Riemannschen Flächen betrachtet. Vgl. [Remmert 1992] S. 3, S. 57-71.



Darstellung einer (holomorphen) Funktion als transformiertes Koordinatennetz

lichen Dimensionen. Zunächst erscheint - angesichts unserer dreidimensionalen Lebenswelt - eine dreidimensionale Abbildung anschaulicher als eine zweidimensionale. Die räumliche Darstellung verliert aber an Attraktivität, wenn man folgendes bedenkt:

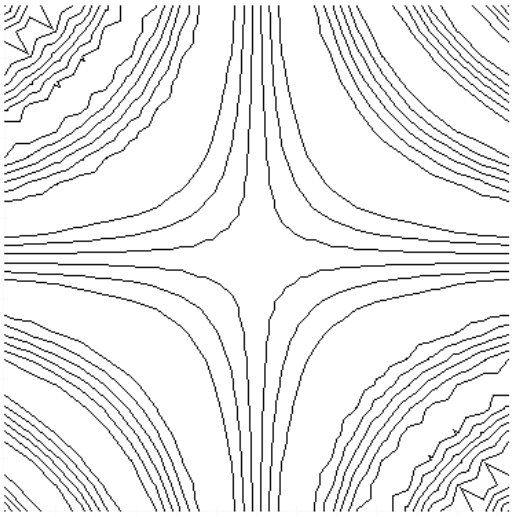
Erstens sind die gebräuchlichsten visuellen Schnittstellen zum Menschen selbst nur zweidimensional (Computermonitor, Fernseher, Papier).[25] Dreidimensionale Erweiterungen wie z.B. Shutter-Brillen sind hingegen teuer und aufwendig im Gebrauch. Die räumlichen Abbildungen würden für gewöhnlich also ihrerseits wieder auf ein zweidimensionales Sichtgerät projiziert, was eine Orientierung erschwert (der in virtuellen 3D-Welten häufig vorkommende Orientierungsverlust ist als *Lost in Space*-Phänomen bekannt[26]). Dem ist allerdings entgegenzuhalten, daß eben aufgrund der häufig anzutreffenden Projektion von räumlichen Darstellungen (diese beginnen bereits mit der Entdeckung der Perspektive in der spätgotischen Malerei und enden vorerst mit dem Fernsehen und der stark zunehmenden Entwicklung und Verbreitung von 3D-Computerspielen), der Mensch durch die Alltagskultur in stets steigendem Maße an diese gewöhnt wird.

Zweitens ist die Berechnung von dreidimensionalen Gebilden ungleich aufwendiger als die von zweidimensionalen. Dies gilt vor allem bei zunehmend fotorealistischer Darstellung (eine Darstellung, die nach unseren Kriterien anzustreben ist).

Es bietet sich nun als ein möglicher Kompromiß an, anstelle einer dreidimensional navigierbaren Repräsentation eine 'Aufsicht' mit Hilfe von Höhenlinien oder Farbübergängen zu wählen. In diesem Fall wäre es auch möglich, Punkte oder Bereiche mit der Maus anzuwählen und weitere Informationen über das Verhalten der

[25] Das gilt beim Computer auch für die Steuerungsmedien. Mit einer Maus kann man lediglich zwei Dimensionen abfahren. Eine räumliche Navigation sowie die interaktive Behandlung von dreidimensionalen Gebilden, wie z.B. das Anwählen von Punkten im Raum ist damit zunächst nicht einfach lösbar.

[26] Vgl. [Schlüter 1998] S. 92ff. Um dem Orientierungsverlust Abhilfe zu schaffen, könnte man gemäß einer 'ökologischen Optik' (Gibson) zusätzlich in der Darstellung einen 'Boden' und einen 'Himmel' einführen, ich halte dies bei rein mathematischen Objekten jedoch für eine etwas irreführende Metaphorik.



Darstellung einer Funktion als Höhenlinienmodell

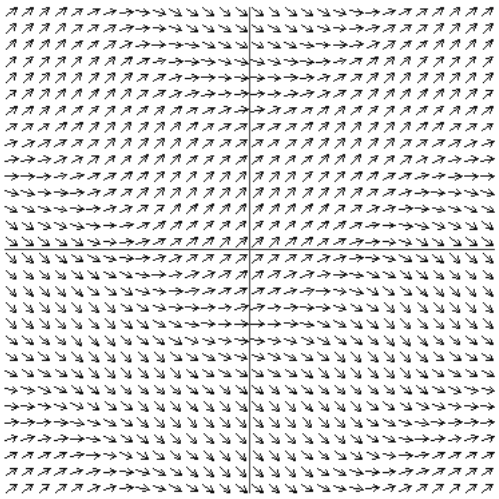
Funktion dort zu erhalten. Dabei wäre ein Farbverlauf als Darstellungsbereich reinen Höhenlinien sicherlich vorzuziehen, allerdings müßte dieser Bereich einer intuitiv erfaßbaren Anordnungsrelation (s.u.) genügen.

2.2.2 Raumdarstellung mit Richtungsvektoren

Vor allem in Lehrbüchern der Physik werden Funktionen der Form $f : \mathbb{R}^n \to \mathbb{R}^n$ als Vektorfelder dargestellt, in vereinfacher Form zumeist in der Ebene ($n = 2$). Dabei wird für ausgewählte Werte an der Stelle (x_1, x_2) in der Ebene ein Pfeil der Länge $r = \sqrt{f(x_1)^2 + f(x_2)^2}$ unter dem Winkel $\varphi = \tan^{-1} \frac{f(x_2)}{f(x_1)}$ zur x_1-Achse eingezeichnet. Diese Darstellung dient vor allem zur Veranschaulichung der Integralsätze für Vektorfelder. Diese Integralsätze tauchen in der Funktionentheorie zwar in ähnlicher Form ebenfalls auf, so daß eine Darstellung im Hinblick auf die Ersichtlichkeit der Holomorphie einer Funktion reizvoll wäre. Aufgrund der Verschiedenheit von komplexer Multiplikation und der Bildung des Skalarprodukts tragen aber in einem durch eine komplexe Funktion gegebenen Vektorfeld so anschauliche Größen wie die Divergenz oder die Rotation keine mathematische Bedeutung, so daß z.B. die Gültigkeit des Cauchyschen Integralsatz - und damit der Holomorphie - nicht vergleichbar intuitiv dargestellt werden kann.[27] Des weiteren ist zur Vektordarstellung zu bemerken, daß die Länge der Pfeile r höchstens in sehr eingeschränkter Weise als Abbildung des Funktionswertbetrages dienen kann, da die in einer Darstellung maximal auftretende Länge - der Übersichtlichkeit halber - die Zellengröße und damit die Anzahl der dargestellten Pfeile bedingen sollte. Bei großer Länge der Pfeile würde dies eine geringe Auflösung zur Folge haben.[28] Aufgrund dieser Problematik wird im allgemeinen die Länge überhaupt nicht dargestellt und allein die Phase abgebildet, wodurch bei komplexen Funktionen nur noch das Verhältnis von Real- und Imaginärteil des jeweiligen Funktionswertes betrachtet werden könnte.

[27] Vgl. aber den Zusammenhang zwischen Harmonischen (reellen) Funktionen und holomorphen Funktionen: [Remmert 1992] S. 43-45.

[28] Wir werden im Unterkapitel 3.4 sehen, daß dieses Problem in abgewandelter Form auch bei den anderen Darstellungsarten auftritt.



Darstellung einer Funktion als Vektorfeld

2.2.3 Dynamische Darstellung durch Einbeziehung der Zeit

Die Zeit ist als Parameter einer Darstellung ebenfalls von großer intuitiver Nähe. Gegenüber dem Raum als Darstellungsbereich ist vor allem ihre strenge natürliche Anordnungsrelation hervorzuheben: Das 'Früher' und das 'Später' werden in größter Weise verschieden wahrgenommen. Die Möglichkeit der dynamischen Darstellung ist das Privileg der elektronischen Medien, und es braucht wohl kaum betont zu werden, daß eine solche im Vergleich zu statischer Darstellung erheblich stärkere Aufmerksamkeit erregen kann. Schon deshalb verdient die Zeit eine Würdigung als Qualität jenseits der des bloßen Parameters. Es unterbindet die Einbeziehung der Zeit als Darstellungsparameter so in der Regel eine Interaktivität zwischen Darstellung und Beobachter.[29] Diese Interaktivität (bezogen auf den Beobachter in der Didaktik 'Selbsttätigkeit' genannt[30]) ist aber gerade eine entscheidende Bedingung für das Erfolgen eines Lernprozesses jenseits bloßer Informationsaufnahme. Insofern sollten (nicht nur) im Rahmen interaktiver Darstellungen Animationen mit Rücksicht auf die funktionale Flexibilität der Darstellung und dem Einbinden der Aktivitäten des Beobachters äußerst vorsichtig eingesetzt werden. Für die Darstellung komplexer Funktionen ergibt sich darüber hinaus erneut das Problem, daß die Abbildung eines Parameters auf die Zeit diesen gegenüber den übrigen stark hervorheben - und damit isolieren - würde.

2.2.4 Farbe als Darstellungsdimension

Farbe ist ebenfalls eine äußerst intuitive visuelle Wahrnehmungsgröße. Die Bindung von Information an Farben wird so auch im Alltag vielfach vorgenommen. In Karten beispielsweise werden verschiedene Bereiche farblich unterschieden, wobei sowohl kontrastierende Farben (thematische Karten) als auch die für unsere Zwecke interessanteren Farbverläufe (physische Landkarten) zur Darstellung eingesetzt werden.

[29] Einen solchen Interaktionsverlust bewegter Bilder stellt man nicht nur - in extremer Weise - vor dem Fernseher fest, wo man nur wenige Wahlmöglichkeiten bei der Informationsbeschaffung hat, sondern auch bei Computerprogrammen, die anstelle von Echtzeitgraphik vorberechnete Animationen verwenden.

[30] Vgl. z.B. [Führer 1997] S. 46-70, [Wagenschein 1999] S.33/34 [Meyer, Jank 1991] S. 298ff.

Vor allem das Vorhandensein der farblichen Darstellungsmöglichkeit in nahezu jeder Computer-Monitor-Einheit macht eine Verwendung von Farbe als Parameter in diesem Rahmen attraktiv. Es ist dabei zu beachten, daß eine Farbe physiologisch gesehen selbst als dreidimensionales Gebilde betrachtet werden muß: Beispielsweise kann in additiver Farbmischung ein beliebiger Farbton erst durch die Mischung der Basisfarben rot, grün und blau - in verschiedenen Intensitäten - erreicht werden (RGB-Modell).[31] Alternativ ist z.B. auch die sogenannte HSB-Farbdarstellung anhand der Parameter Farbwinkel im Farbenkreis (Hue), Farbsättigung (Saturation) und Helligkeit (Brightness) möglich.
Mit dem RGB-Modell ist nun zwar ebenfalls eine Anordnungsrelation innerhalb der einzelnen Basisfarben gegeben, allerdings erkennt der im subtraktiven Farbmodell geschulte Mensch diese in der Regel nur schwer. Greift man allerdings aus der Lebenswelt bekannte eindimensionale Anordnungsmuster wie z.B. Helligkeit, Farbtemperatur (z.B. bei Wetterkarten) oder das Mischungsverhältnis in einem Farbübergang zwischen zwei Farben heraus, so werden diese intuitiv erkannt.

2.2.5 Fazit

Die Vielzahl der Darstellungsmöglichkeiten scheint zunächst keine eindeutige Bevorzugung zu erlauben. Wie wir gesehen haben, läßt sich die Darstellungsart nicht völlig von der Intention der Darstellung abkoppeln. Wenn man nun das Ziel eines möglichst flexiblen Visualisierungsprogrammes weiterhin im Auge behält, erscheint es sinnvoll, in diesem Programm mehrere Darstellungsmöglichkeiten anzubieten. Auch im Hinblick auf einen praktischen Vergleich der Darstellungen, der damit ermöglicht wird, ist dieses Vorgehen ertragreich.

3 Objektorientierte Programmierung eines Funktionenplotters in Java

3.1 Einführung

Mit den im vorangehenden Kapitel beschlossenen Voruntersuchungen sind wir nun in der Lage, eine ausführliche Spezifikation für ein Programm zur Darstellung von komplexen Funktionen zu erstellen. Das Hauptmerkmal des Programmes soll dabei die Flexibilität sowohl in Bezug auf den Einsatz wie auf die Programmierung von Erweiterungen sein. Die entwicklungstechnische Erweiterbarkeit wird im allgemeinen durch das Paradigma der objektorientierten Programmentwicklung gesichert, indem das Programm nicht als monolithischer Block, sondern mehr als Sammlung von interagierenden, weitgehend unabhängigen Objekten implementiert wird. Es wäre also eher angebracht, von der Entwicklung eines Baukastensystems ('Framework') als allein von der Erstellung eines lauffähigen Programmes zu sprechen.
Die Entwicklung der einzelnen Komponenten für den Funktionenplotter soll und kann sich nun allerdings nicht an das Prinzip der beliebigen Erweiterbarkeit halten, sondern es werden - wie im Rahmen fortgeschrittener objektorientierter Programmentwicklung üblich - zu Beginn feste Schnittstellen und Abhängigkeiten definiert, über welche die einzelnen Komponenten miteinander in Verbindung stehen. Auf diese Weise läßt sich die wichtigste Aufgabe der Spezifikation benennen: Die Trennung von konstanter und veränderlicher Struktur.[32] Flexibel soll dabei unsere Anwendung

[31] Diese - physikalisch gesehen - seltsam anmutende Tatsache erklärt sich dadurch, daß das menschliche Auge rot (am unteren Rand des sichtbaren Spektrums), blau (am oberen Rand) und grün (in der Mitte) durch separate Zäpfchenarten in der Retina wahrnimmt und aus diesen Einzelfarbreizen im Gehirn den Gesamtfarbeindruck rekonstruiert. Vgl. [Vogel 1995] S. 576-579, 1228-1230.

[32] [Eckel 1998] nennt dies die Festlegung des 'Gradienten der Veränderung' ('Vector of Change').

auf die Erweiterung um neue Ansichtskomponenten einerseits und auf die Implementation zusätzlicher mathematischer Funktionen andererseits reagieren. Konservativ soll dagegen der Mechanismus der Funktionsberechnung gehalten werden, was zur Folge hat, daß Optimierungen hinsichtlich der Berechnungszeit mehr auf die mathematische Ebene verlagert werden müssen.
Zur Beschreibung der Implementation greife ich auf das sich mittlerweile in der Informatik verbreitende, von der Implementationssprache unabhängige Darstellungsmittel des 'Entwicklungsmusters' zurück.[33] Nach den in dem nächsten Unterkapitel behandelten implementationspraktischen Aspekten möchte ich zunächst die Spezifikation und den dadurch festgelegten Grundaufbau vorstellen. Im darauf folgenden Unterkapitel sollen dann ausgewählte Objekte in ihrer spezifizierten Funktion sowie Aspekte ihrer Implementation besprochen werden (der vollständige Quellcode der Komponenten befindet sich im Anhang) und einige Anwendungsbeispiele illustrativ angeführt werden.

3.2 Praktische Vorüberlegungen

3.2.1 Zu Java als Implementationssprache

Daß für die Implementation des Funktionenplotters die Programmiersprache Java gewählt wird, hat vor allem zwei Gründe:

1. Die Programmiersprache ist plattformunabhängig,[34] damit entfallen Betrachtungen bezüglich der zu wählenden Hardware und des zu wählenden Betriebssystems ebenso wie eine Auswahl einer Graphikbibliothek (s.u.). Alle diese Überlegungen müßten bei jeder anderen Implementationssprache vorgenommen werden. Die Plattformunabhängigkeit hat auch zur Folge, daß das Programm in eine WWW-Seite eingebettet, und ohne Installation direkt aus dem Internet heraus aufgerufen werden kann.

2. Die Programmiersprache verfügt über eine weitgefächerte Standardbibliothek, die vor allem im graphischen Bereich äußerst komfortabel ist. Sie bietet zwar keine mathematische Funktionalität, die die elementaren Berechnungsmöglichkeiten reeller Zahlen überschreitet, so daß die Objekte für komplexe Zahlen, Polynome etc. selbst implementiert werden müssen. Dies ist jedoch bei den meisten 'Allzweck'-Programmiersprachen der Fall.

3.2.2 Benutzte Programme und Hilfsmittel

Für die Entwicklung dieser Arbeit wurden ausschließlich Programme benutzt, die nach einem standardisierten Verfahren arbeiten und/oder frei erhältlich sind. Der Funktionenplotter läuft auf dem JDK 1.2.x von Sun mit der dreidimensionalen Graphik-Erweiterung Java 3D[35]. Als Entwicklungsumgebung wurde GNU Linux unter X11 mit XEmacs[36] als Editor verwendet. Die benutzte Versionierungssoftware ist GNU CVS[37] (Concurrent Versions System). Graphiken wurden entweder mit dem Vektorzeichenprogramm xfig[38] erstellt oder mit Gimp[39] (GNU Image Mani-

[33] Ich halte mich dabei an das in [Buschmann et al.] vorgenommene Schema.
[34] Dies ist sie nicht nur im technischen Sinne, sondern durch das laufende Verfahren zur Anerkennung als 'Publicly Available Specification' durch die ISO bald auch im ökonomischen Sinne. Vgl. `http://java.sun.com/aboutJava/standardization/javapas.html`.
[35] [Sun 1999b]
[36] `http://www.xemacs.org`
[37] `http://www.cyclic.com`
[38] `http://www.xfig.org`
[39] `http://www.gimp.org`

pulation Programm). Die Darstellung der Klassendiagramme erfolgt in UML 1.1[40] (Unified Modeling Language, einer durch die OMG standardisierten Darstellungssprache von objektorientierten Modellen).

3.3 Die Grobstruktur des Funktionenplotters

Da gemäß dem Fazit in Unterkapitel 2.2 mehrere Darstellungsarten für eine spezielle Funktion möglich sein sollen, wird die Gesamtarchitektur des Programmes nach dem Model-View-Controller-Muster (MVC) entworfen.[41] Dieses spaltet eine interaktive Anwendung in drei Komponenten auf: Das Modell enthält die Daten und die Funktionalität zu ihrer Generierung, die Ansichten (Views) implementieren allein die graphische Darstellung dieser Daten, und das Kontrollelement nimmt Benutzereingaben entgegen und vermittelt zwischen Datenmodell und Ansicht. Das Programm soll im groben wie folgt ablaufen:

1. Der Benutzer gibt in einem Textfeld der Kontrollkomponente eine Funktion $f(z)$ als Zeichenkette ein. Anhand dieser Zeichenkette wird durch einen Mini-Compiler ein Operationsbaum erstellt.

2. Der Operationsbaum nimmt daraufhin für einen ebenfalls eingegebenen Wertebereich (ein achsenparalleles Koordinatennetz aus $\mathbb{C}$) die Berechnung vor und legt die Funktionswerte in einem zweidimensionalen Feld ab.

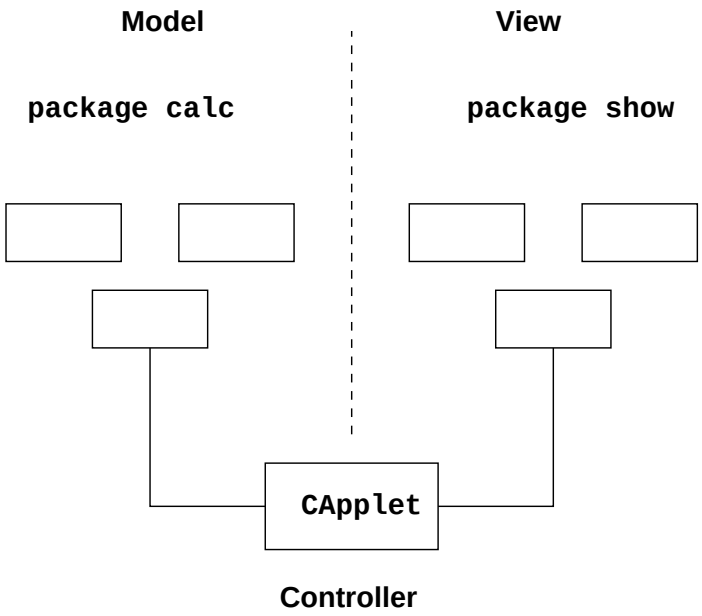
3. Das Feld wird an das Kontrollelement übergeben. Dieses ruft dann die ausgewählte Ansicht auf und übergibt ihr über eine standardisierte Schnittstelle die Daten.

4. In der vom Betriebssystem aufgerufenen `paint()`-Methode (die die Graphikkomponente auffordert, sich auf den Bildschirm zu zeichnen) regelt die Ansicht die spezifische Darstellung der Funktionswerte.

5. Für bereits errechnete Daten können weitere Ansichten ohne Neuberechnung geöffnet werden. Eine einmal generierte Ansicht ist danach im wesentlichen vom Kontrollelement und dem Modell mit seinen Datenbeständen unabhängig.

Will man nun auf die Implementationsebene übergehen, so stellt sich die Frage, ob die drei (logischen) Komponenten dort direkt durch einzelne Objekte repräsentiert werden sollen, oder ob ihre Funktionalität durch ein Java-Paket (package) zu bündeln ist. Ich entscheide mich angesichts der zu erwartenden geringen Funktionalität beim Controller für eine einzelne Klasse (die aufgrund der praktischen Verwendbarkeit die `Applet`-Klasse erweitert, aber gleichzeitig über eine `main()`-Methode verfügt, um als Applikation ausgeführt werden zu können) und nenne diese `CApplet`. Der Berechnungs- und Datenerzeugungsbereich soll der Spezifikation nach beliebige eingegebene Zeichenketten in entsprechende Funktionsberechnungen umwandeln und verfügt so über einen größeren Funktionalitätsumfang, weshalb ihm ein eigenes Paket mit Namen `calc` zugewiesen wird. Die Ansichten sind ohnehin mehrere Objekte, sie werden in dem Paket `show` zusammengefaßt. Die Kontrolleinheit ist in ihrer Spezifikation recht einfach: Sie nimmt die Benutzerkommandos entgegen und übernimmt die Verbindung zwischen der Berechnung und der Anzeige. Wir beschäftigen uns daher in den folgenden Unterkapiten lediglich mit der funktionalen Differenzierung innerhalb der Pakete `calc` und `show`.

[40] `http://www.omg.org/news/pr97/umlprimer.html`. Vgl. [Fowler, Scott 1998]
[41] Vgl. [Buschmann et al.] S. 124 ff.



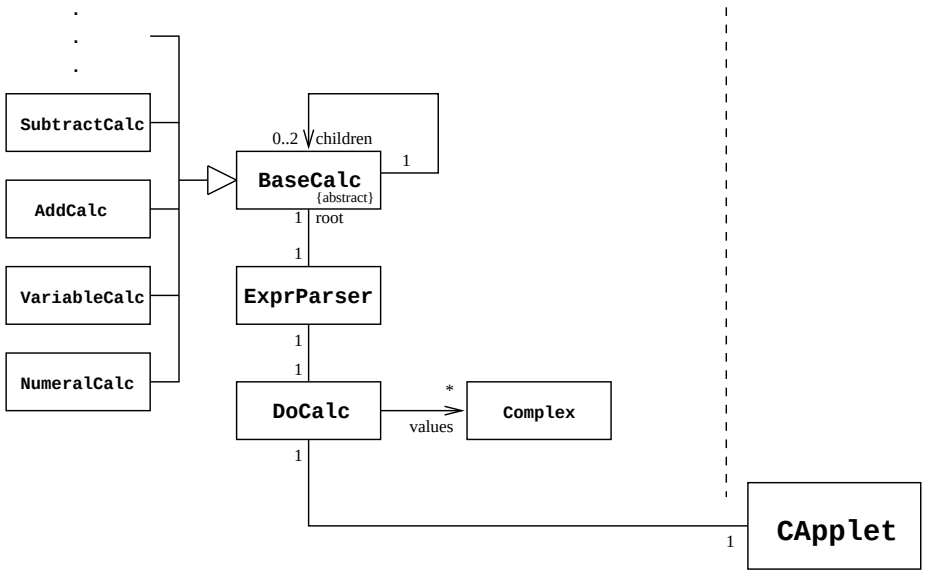
Grobstruktur des Funktionenplotters

3.3.1 Zum Implementationsprozeß

Das Produkt einer Entwicklung kann nur so gut sein wie der Prozeß der Entwicklung es zuläßt. Deshalb wird bei der Erstellung von Software dem Entwicklungsprozeß zunehmend mehr Beachtung zugewandt, was sich z.B. in der stark ansteigenden Anzahl von 'Methoden'-Büchern niederschlägt. Ich möchte hier die Entwicklungsschritte des Funktionenplotters soweit kurz skizzieren, wie dies in den folgenden Unterkapiteln nicht schon durch die Beschreibung der Programmstruktur geschieht. Der objektorientierte Entwicklungsprozeß verläuft *inkrementell* und *iterativ*.[42] Inkrementell, weil er durch stetes Hinzufügen von Objekten bzw. Methoden zu Objekten gekennzeichnet ist; iterativ, weil jedes Objekt während und nach der Fertigstellung einer Anzahl von Tests auf Spezifikationstreue unterworfen werden muß.[43] So beginne ich mit dem 'konservativen' Teil, der Berechnungseinheit, deren Kern die Klasse `calc.Complex` darstellt. Nach der Implementation der elementaren Rechenmethoden für diese Klasse wird der Miniatur-Compiler `ExprParser` (s.u.) mit dem dazugehörigen Operationsbaum erstellt. Nachdem dieser symbolische Ausdrücke in der gewünschten Grammatik (die über die Kommandozeile übergeben werden) verarbeitet und in einen Operationsbaum umsetzt, erstelle ich ein Kontrollelement mit minimaler Funktionalität und beginne mit der graphischen Darstellungseinheit. In dieser wird zunächst die Grundschnittstelle implementiert; dann wird eine möglichst einfache und lauffähige Darstellung programmiert (eine Abbildung als Vektorfeld), wodurch erstmalig Berechnungs- und Darstellungsgeschwindigkeit des Programmes praktisch ausgelotet und verglichen werden können. Es stellt sich dabei heraus, daß die Berechnung gegenüber der Darstellung sehr schnell ist (z.B. dauert die Berechnung von $\exp(1/z)$ für 250000 Werte wenig mehr als 6 Sekunden, was einer durchschnittliche Berechnungsdauer von ca. 0.000024 s entspricht). Dieses Faktum

[42] Vgl. z.B. [Fowler, Scott 1998] S. 27-52

[43] Getreu dem Grundsatz, daß die Hälfte des Programm-Codes Test-Routinen ausmachen sollen ([Fowler, Scott 1998] S. 41), ist nicht nur nahezu jedem größeren Objekt eine Test-Methode beigefügt, die wesentliche Funktionalität überprüft, sondern es werden auch innerhalb einzelner Methoden Zustandsmeldungen ausgegeben. Ich habe die meisten dieser Meldungen jedoch aus Gründen der Geschwindigkeitsoptimierung und der Übersichtlichkeit im Quelltext auskommentiert.



Klassendiagramm des `calc`*-Paketes*

wiederum erlaubt überhaupt erst die Berechnung von aufwendigeren Funktionen (s. Kapitel 4), zunächst aber müssen weitere Graphikdarstellungen implementiert werden. Dabei und in der weiteren Folge wird durch die Implementation teilweise der flexible Teil der Spezifikation verändert, teilweise wird er durch zusätzliche Funktionalität erweitert. Die Beschreibung der einzelnen Klassen im nächsten Unterkapitel ist also lediglich eine sequentielle Auflistung von statischen Endresultaten, die bezüglich ihrer dynamischen, nichtlinearen Entwicklungsgeschichte ein wenig in die Irre führt.

3.4 Das Berechnungspaket `calc`

3.4.1 Einführung

Das Berechnungspaket ist, wie bereits erwähnt, für die Verarbeitung der eingegebenen Funktion und die Berechnung der Funktionswerte zuständig. Dabei besteht seine Hauptaufgabe darin, die vom Benutzer entgegengenommene Zeichenkette nach Art eines Miniatur-Compilers in einen Berechnungsbaum umzuwandeln. Aus dieser Perspektive betrachtet, kann also der Benutzer in einer begrenzten formalen Sprache (nämlich der der Arithmetik inklusive elementarer Funktionen) eine Berechnungsvorschrift programmieren, nach der für einen beliebigen (netzförmigen) Definitionsbereich die Berechnung vorgenommen wird. Da die Arithmetik sich funktional formulieren läßt (z.B. die Addition als Funktion $+ : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$), reicht es aus, zur Synthese der Rechenvorschrift einen Baum aus Funktionen zu konstruieren, deren Knoten derart verbunden sind, daß die Ausgabekanäle der Kinderknoten die Eingabekanäle des zugehörigen Elternknotens bilden. Bei der Berechnung selbst werden die Daten als Strom betrachtet, der an den Blättern beginnt und im Zuge der Berechnung zur Wurzel fließt (die dann das Ergebnis enthält). Das der Berechnungsarchitektur zugrundeliegende Entwurfsmuster ist das vor allem in der UNIX-Welt bekannte 'Pipes-and-Filters'-Muster.[44]

[44] Vgl. [Buschmann et al.] S. 54ff. Das Entwurfsmuster genügt dabei auch der oben geforderten Erweiterbarkeit in Bezug auf Implementation weiterer Operationen.

3.4.2 Der Miniatur-Compiler `ExprParser`

Im Rahmen der Spezifikation soll der Miniatur-Compiler die aus den 'elementaren' Funktionen unter Verwendung der gebräuchlichen arithmetischen Symbole hervorgehende Funktionsmenge verarbeiten können. Er muß damit zunächst die folgende (in EBNF-Form gegebene) kontextfreie Grammatik verarbeiten können:[45]

```
G(T,N,s,P)

T:      NUM 'z', '+', '-', '*', '/', '^', 's', 'c', 'e'
N:      expr, term, fac, pot, prim
s:      expr
P:
(1) <expr>      ->      <term> { '+' <term> } | <term> { '-' <term> } .
(2) <expr>      ->      <fact> { '*' <fact> } | <fact> { '/' <fact> } .
(3) <fact>      ->      's' <pot> | 'c' <pot> | 'e' <pot> | <pot>.
(4) <pot>       ->      <prim> '^' <prim> | <prim> .
(5) <prim>      ->      'z' | NUM | '(' <expr> ')' .
```

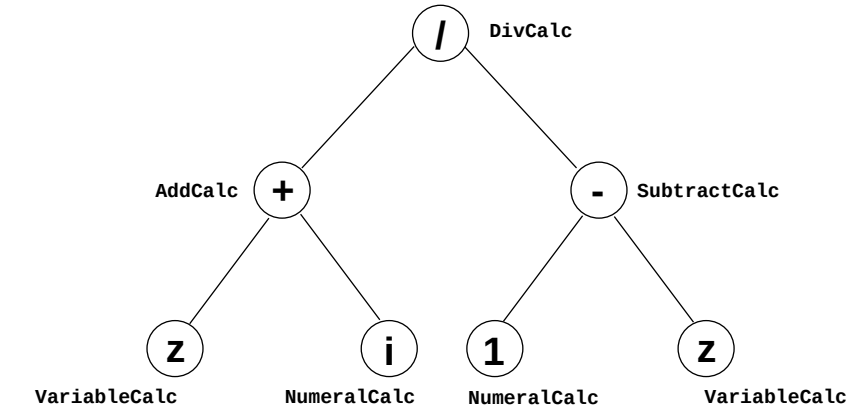
Dabei stehen die Terminale '`s`' für die Sinus-, '`c`' für die Cosinus- und '`e`' für die Exponentialfunktion. `NUM` stellt ein Numeral (eine Fließkommazahl oder '`i`') dar, das ein unäres Minus enthalten kann.
Das Objekt, das für die Generierung des Berechnungsbaumes zuständig ist, ist `ExprParser`. Da die zu verarbeitende Grammatik relativ simpel ist, wird in diesem Objekt die gesamte Analyse des Eingabetextes und die Synthese der zugehörigen Operationen durchgeführt. Die Funktionalität des Scanners bzw. Lexers erledigt die Methode `getToken()`, die das nächst angeforderte Token und seinen Typ in Objektvariablen ablegt. Die Nonterminale `N` haben ihre Entsprechungen in gleichnamigen Methoden (`expr()`, `fact()`, etc.), die den Startausdruck `<expr>` durch rekursiven Abstieg vermittels der Produktionen `P` in Terminale auflösen. Der Rückgabewert der Nonterminal-Methoden ist vom Typ `BaseCalc`, welche die (abstrakte) Basisklasse aller Operationsknoten darstellt (s.u.). Der Parsing- und Kompilationsvorgang wird durch den Aufruf des Konstruktors `ExprParser(String input)` in Gang gebracht, die Methode `getOperationTree()` liefert schließlich die Wurzel des konstruierten Operationsbaumes.

3.4.3 Der Berechnungsbaum: `BaseCalc` und ihre Unterklassen

Der von `ExprParser` konstruierte Berechnungsbaum besteht ausschließlich aus Knoten, die von der Klasse `BaseCalc` abgeleitet worden sind. Ihre Namen reflektieren die ihnen zugehörigen Operationen: `AddCalc`, `SubtractCalc`, `NumeralCalc` etc. Die Klasse `BaseCalc` dient einerseits als Schnittstelle ihrer Unterklassen, andererseits implementiert sie die diesen allen gemeinsame Funktionalität. Sie wird daher als `abstract` deklariert. Da der Quelltext der Klasse exemplarisch die Struktur und (in der `main()`-Methode) das dynamische Verhalten des Berechnungsbaumes klar macht, sei er hier aufgelistet:

```
package calc;

/** This class is the base for all kind of calculations. It's subclasses
  represent the nodes in the Operational Tree constructed by ExprParser */

public abstract class BaseCalc {
```

[45] Zum Compilerbau [Gumm, Sommer 1998] S. 535-559. Zu kontextfreien Grammatiken: [Wirtz 1998] S. 178-194.



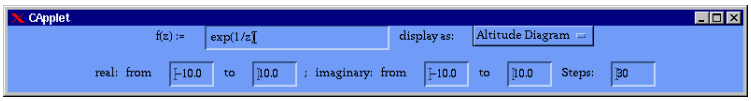
Der Operationsbaum von $\frac{z+i}{1-z}$: Beim Berechnen der Werte wird in den Blättern vom Typ `VariableCalc` *der jeweils zu berechnende Wert gesetzt (funktional gesprochen: ein Wert an die Variable gebunden), das Ergebnis wird in der Wurzel (hier vom Typ* `DivCalc`*) ausgelesen.*

```
protected Complex result;
protected BaseCalc[] children = null;
protected boolean calculated = false;

/** overridden by inheriting class, needed for consistency-checks */
public abstract int getArgsNr();

/** connects the Object to another BaseCalc and uses it's output as input */
public void insertChildren(BaseCalc[] children) throws IllegalArgException{
  if(children.length != getArgsNr())
    throw new IllegalArgException("Illegal Number of Arguments to "
                                  +this.getClass().getName()+"!\nneeds "
                                  +getArgsNr()+" - got "+ children.length);
  this.children = children;
}

/** returns an array with the child-nodes */
public BaseCalc[] getChildren(){
  return children;
}

/** wrapper for doCalculate. Keeps track of the state */
public void calculate(){
  if(calculated)
    return;
  //System.out.println(getClass().getName()+": calculating...");
  doCalculate();
  calculated = true;
}

/** implemented by subclasses, does the real calculation by recursively
    calling it's children */
protected abstract void doCalculate();

/** return the result calculated by calculate() */
public Complex result(){
```

```
    if(!calculated)
      calculate();
    //System.out.println("result is "+result);
    return result;
  }

  /** remove the calculated-flag */
  public void clear(){
    if(children != null)
      for(int i = 0; i < children.length; i++)
        children[i].clear();
    calculated = false;
  }

  /** returns a string representing the arithmetic operation */
  public String toString(){
    String retVal = "";
    if(children != null){
      for(int i=0; i < children.length; i++)
        retVal += " "+children[i].toString();
      return getArithmeticSymbol()+"("+retVal+" )";
    }
    return getArithmeticSymbol();
  }


  /** must be implemented for debugging purposes: returns "+" for AddCalc,
      etc. */
  public abstract String getArithmeticSymbol();

  /** for testing purposes */
  public static void main(String[] args){

    // make some nodes:
    MultiplyCalc m = new MultiplyCalc();
    AddCalc a = new AddCalc();
    NumeralCalc n1 = new NumeralCalc(new Complex(1,2));
    NumeralCalc n2 = new NumeralCalc(new Complex(3,4));
    NumeralCalc n3 = new NumeralCalc(new Complex(1,0));

    // construct the Operational Tree: n1 * n2 + n3
    try{
      BaseCalc[] args1 = {n1,n2};
      m.insertChildren(args1);
      BaseCalc[] args2 = {m,n3};
      a.insertChildren(args2);
    } catch (IllegalArgException e){ System.out.println(""+e); }
    System.out.println("Result is: "+a.result());
  }
}
```

3.4.4 Grundfunktionalität komplexer Zahlen: `Complex`

Da die Java-Bibliothek keine Klasse für komplexe Zahlen beinhaltet und eine Recherche im Internet kein Auffinden eines befriedigenden Drittanbieter-Paketes ergab, mußte die `Complex`-Klasse als Repräsentation komplexer Zahlen und ihrer Rechenoperationen von mir selbst erstellt werden. Sie ist die allen Berechnungen zugrundeliegende Klasse. Da Java (im Gegensatz zu z.B. C++) kein Überladen von Ope-



Die Graphische Benutzerschnittstelle von `CApplet`

ratoren erlaubt, können Objekte dieser Klasse allerdings nicht mit der Einfachheit von primitiven Datentypen (wie etwa `double`) verwendet werden. Arithmetische Operationen beispielsweise müssen entweder als Instanzmethoden oder als statische Methoden implementiert werden. Zudem muß ebenfalls im Unterschied zu den primitiven Datentypen beachtet werden, daß Objekte in Java stets als Referenz übergeben werden, was im weiteren diverse Kopieroperationen notwendig macht. Um den Objektstatus einer komplexen Zahl möglichst transparent zu halten, werden in der weiteren Verwendung von Berechnungen die zunächst statischen Methodenaufrufe gegenüber Instanzmethodenaufrufen bevorzugt, auch wenn das Erzeugen eines neuen Objektes in Java natürlich ein (zeit-)aufwendigerer Akt ist als das Ändern von Instanzvariablen.[46]

3.4.5 Durchführung und Organisation der Berechnung: `DoCalc`

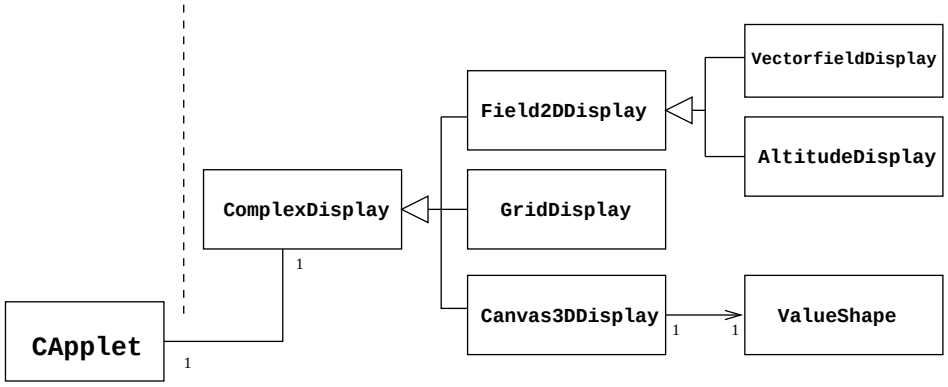
Die Klasse `DoCalc` übernimmt die eigentliche Berechnung der Funktionswerte und stellt gleichzeitig die Schnittstelle nach außen (in unserem Fall zu `CApplet`) dar. Sie nimmt die Eingabe des Benutzers entgegen, führt dabei in `preparse()` einige Vereinfachungen durch (z.B. die Abbildung auf Kleinbuchstaben, Ersetzung von '`exp`' durch '`e`', etc.) und erzeugt ein `ExprParser`-Objekt; diesem wird daraufhin der Berechnungsbaum bzw. eine Referenz auf seine Wurzel entnommmen sowie ein `Vector`, in dem die Objekte vom Typ `VariableCalc`, also die Platzhalter für die mathematische Variable z, enthalten sind. Die Berechnung läuft nun derart ab, daß das durch den Benutzer angegebene $n \times m$-Netz (das `CApplet` über `DoCalc.setRange()` und `DoCalc.setSteps()` propagiert) hindurchiteriert wird und die jeweiligen komplexen Werte in diesen Platzhaltern eingesetzt werden. Die Ergebnisse werden in einem zweidimensionalen `Complex[n][m]`-Array abgelegt und per `getValues()` an `CApplet` abgegeben.

3.5 Das Darstellungspaket `show`

3.5.1 Überblick und Basisklasse: `ComplexDiplay`

Nachdem die vom Benutzer eingegebene Funktion im `calc`-Paket in eine maschinelle Berechnungsvorschrift umgesetzt worden ist und die Werte der Funktion gemäß dieser Vorschrift errechnet worden ist, übergibt der Controller (also das `CApplet`-Objekt) das Werte-Feld an die Darstellungseinheit, die durch das `show`-Paket organisiert wird. Konkret übergibt er den Funktionswertevorrat (nebst zugehörigem Definitionsbereich) über die Schnittstelle der abstrakten Klasse `ComplexDisplay`. Eine mögliche Darstellung muß dann in einer von dieser abgeleiteten Klasse realisiert werden. Um zu demonstrieren, daß mit dieser Basisklasse nur die elementaren Eigenschaften einer Darstellung (im Rahmen der gewählten Grundfunktionalität,

[46] Anders als in C++ können temporäre Objekte in Java nicht auf einem im Zwischenspeicher befindlichen Kellerspeicher (Stack) erzeugt werden, sondern werden direkt auf den 'Object-Heap' im Hauptspeicher geschoben. Dadurch ergibt sich nach [Roulo 1998] ein Geschwindigkeitsunterschied von Faktor 10 gegenüber C++. Methodenaufrufe und Variablenzugriffe sind dagegen in beiden Sprachen fast gleich schnell.



Klassendiagramm des `show`*-Paketes*

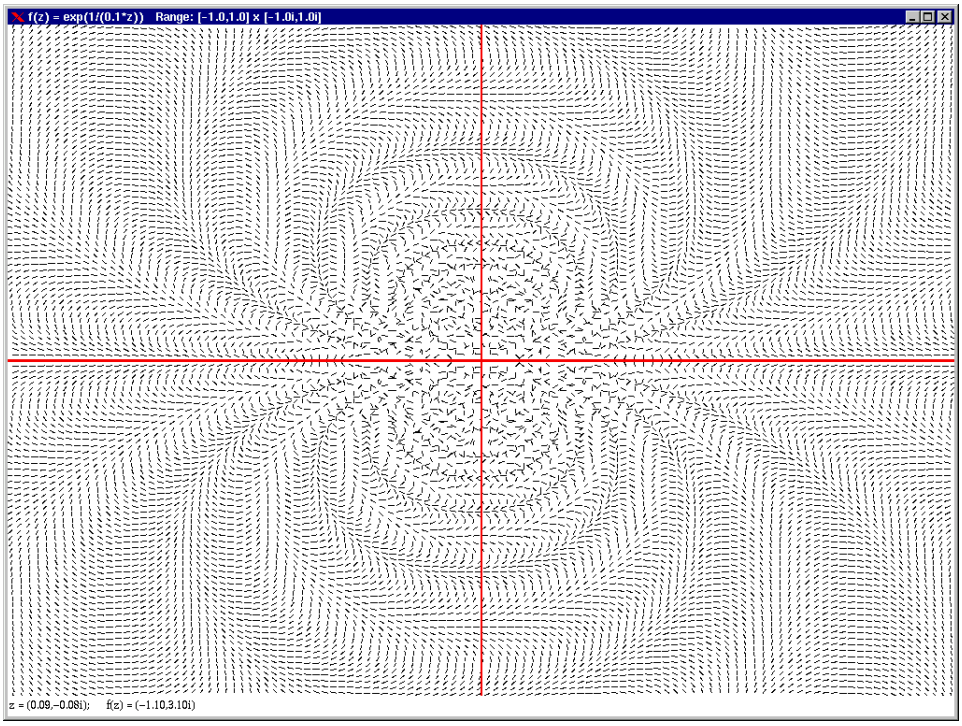
nämlich der graphischen Darstellung der Funktionswerte über einer rechteckförmigen Teilmenge von $\mathbb{C}$) spezifiziert wurden und um die Orientierung im `show`-Paket zu vereinfachen, sei der Quelltext dieser Klasse hier angeführt:

```
package show;

import java.awt.*;
import java.awt.event.*;

/** generalizes functionality for graphically representing complex
    functions */

public abstract class ComplexDisplay extends Container {

  /** sets the calculated values, that need to be displayed */
  public abstract void setValues(calc.Complex[][] inValues);

  /** sets the ranges (each as a two element-array containing min- and
      max-value) for both real and imaginary direction */
  public abstract void setRange(double[] realRange, double[] imagRange);

}
```

Wir betrachten im folgenden die Ableitungen dieser Basisklasse. Die Implementationen sind dabei einerseits als exemplarische Realisierungen der im 2. Kapitel untersuchten Darstellungsmöglichkeiten zu verstehen. Andererseits werden sie ihre rein praktische Daseinsberechtigung - zumindest teilweise - in der im 4. Kapitel ermöglichten visuellen Analyse der Riemannschen ζ-Funktion beweisen müssen.
Im Sinne der oben angestrebten Einbindung der Selbsttätigkeit des Benutzers beim Darstellungsprozeß verfügen die Implementationen allesamt über interaktive Muster zur weiteren Informationsaneignung über die Betrachtungszeit. Diese interaktiven Erweiterungen werden im Anschluß an die Beschreibung der einzelnen Klassen unter den ihnen gemeinsamen Gesichtspunkten besprochen.

3.5.2 Darstellung in einem zweidimensionalen Feld: `Field2DDisplay`

Die abstrakte Klasse `Field2DDisplay` bildet den Grundstock für alle Darstellungen, die den rechteckigen Definitionsbereich in einem (beliebig großen und propor-



Darstellung von $\exp(1/z)$ *mittels* `VectorField`

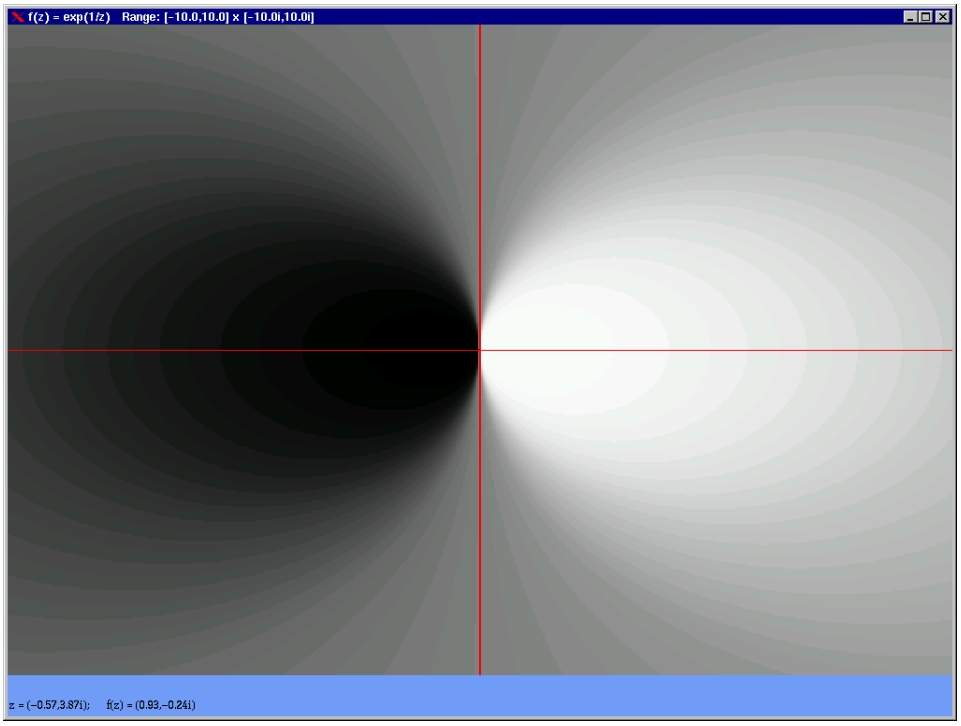
tionierten) Rechteck auf dem Bildschirm darstellen. Auf die von der Graphikverwaltung zugewiesene Bildschirmfläche wird dabei mittels einer affinen Streckung mit unterschiedlichem Skalierungsfaktoren in x- und y-Richtung der ausgewählte Wertebereich paßgenau abgebildet.[47] Ist die physikalische Auflösung größer, als die ausgewählte Wertdichte (was sinnvollerweise der Fall sein sollte), entstehen so auf dem Bildschirm mehrere Pixel große Zellen, die einem bestimmten Wert aus dem Definitionsbereich zugeordnet sind. Diese Zelle wird dann anhand des Funktionswertes zu diesem Definitionswert[48] gekennzeichnet. Wie diese Kennzeichnung innerhalb der Zelle geschieht, unterliegt der abgeleiteten Klasse: In `VectorFieldDisplay` wird (wie in 2.2.2 dargelegt) die 'Richtung' durch einen Strich angezeigt, in `AltitudeDisplay` wird dagegen die Zelle gemäß einem Algorithmus eingefärbt (s.u.).

3.5.3 Darstellung als Vektorfeld: `VectorFieldDisplay`

Diese Klasse implementiert gegenüber ihrer Oberklasse lediglich das Zeichnen eines Striches in 'Richtung' des zu der Zelle gehörigen Funktionswertes. Sie verdankt ihre Existenz im wesentlichen entstehungshistorischen Gründen und der experimentellen Überprüfbarkeit der in 2.2.2 aufgestellten These, daß die Darstellung einer komplexen Funktion als Vektorfeld einen eher geringen anschaulichen Fortschritt bewirkt.

[47] Das Java 2D API erlaubt die interne affine Transformation einer Graphik, wodurch eine Trennung von Koordinatenberechnung und Zeichenvorgang begünstigt wird. Vgl. [Sun 1999].

[48] Genauer genommen handelt es sich bei diesem 'Wert' natürlich erneut um das Abbild eines rechteckigen Bereiches der komplexen Zahlenebene, ich approximiere aber die Funktionswerte des Bereichs durch den Funktionswert des Rechteckschwerpunktes.



Darstellung von $\exp(1/z)$ *mittels* `AltitudeDisplay`

3.5.4 Darstellung als Höhendiagramm: `AltitudeDisplay`

Diese Klasse ist für die Darstellung (gemäß den Betrachtungen in 2.2.4) der 'analytischen Landschaft' als Höhendiagramm zuständig. Als Farbübergang soll zunächst der zwischen Schwarz und Weiß in 255 Stufen gewählt werden. Die Aufteilung der Helligkeitsstufen erweisen sich dabei zunächst als Problem: Verwendet man - wie bei Höhenkarten oft üblich - eine äquidistante Progression, so kann es passieren, daß die Graphik im wesentlichen aus einer einzigen Farbe besteht. Dies ist vor allem bei im untersuchten Bereich betragsmäßig stark anwachsenden Funktionen (wie etwa der Exponentialfunktion) der Fall. Universelle Abhilfe leistet allein eine dynamische, werteabhängige Farbbalancierung. Eine sinnvolle Spezifikation wäre, die Unterteilung des aktuellen Wertevorrats[49] in gleichgroße Partitionen und die Zuweisung einer gemeinsamen Farbe für alle Elemente einer Partition zu fordern. Die Implementation realisiert dies durch einen Algorithmus, der zunächst die Funktionswerte sortiert und diese dann partitioniert, indem er dem sortierten Feld `sortedValues[]` in festen Abständen Werte entnimmt, die er als Intervallgrenzen der Partitionen definiert und in einem neuen Feld, `valuesForColor[]` ablegt. Beim Zeichenprozeß wird dann der einzelne Funktionswert mit den Elementen dieses Feldes in aufsteigender Reihenfolge verglichen, bis er kleiner als das ausgewählte Element ist. Der Index des Elements innerhalb des Feldes `valuesForColor[]` wird dann als Graustufe beim Ausfüllen der Zelle verwendet.[50]

[49] Hier und in der Folge wird der sprachlichen Einfachheit halber von 'Funktionswerten' anstelle von 'Funktionswertbeträgen bzw. 'Funktionswertbetragsquadraten' gesprochen.

[50] Ist der Wert aufgrund einer Bereichsüberschreitung oder der Division durch Null vom Wert `NaN` (not a number), so erhält die Zelle die Farbe grün.

3.5.5 Darstellung im dreidimensionalen Raum: `Canvas3DDisplay`

Es soll trotz der in 2.2.1 geäußerten Nachteile auch eine echt dreidimensionale Darstellung im Raum implementiert werden. Die Gründe dafür liegen vor allem in der Zukunftsfähigkeit dieser Art der Darstellung und in dem gegenwärtigen Vorhandensein einer leistungsstarken 3D-Visualisierungsbibliothek für Java: das Java 3D API.[51] Diese in vielerlei Hinsicht an das Konzept von VRML[52] angelehnte Bibliothek arbeitet allein mit logischen Koordinaten und übernimmt die Berechnung der Perspektive sowie das Rendering der Objekte. Die einzelnen Objekte (zu denen auch der Beobachter zählt) werden als Blätter in einem Baum eingehängt. Den darüberliegenden Knoten werden affine Transformationen zugeordnet, wodurch sich die Anordnung der Objekte im Raum ergibt. In diesem Fall bedeutet das: Die errechnete Funktion wird zunächst als Drahtgittermodell durch die Klasse `ValueShape` konstruiert. Dabei wird der Funktionswert zu einem Definitionswert mit den Funktionswerten von dessen vier Nachbarn im Definitionsbereichsnetz durch eine Linie verbunden. Das fertige Objekt wird daraufhin in `Canvas3DDisplay` in den inhaltlichen Ast des Szenegraphen eingebaut. Der Beobachtungsast (die `ViewBranchGroup`) wird demgegenüber (durch eine Transformation von der Wurzel aus) räumlich nach hinten verschoben und leicht (um $\frac{\pi}{15}$) gedreht, so daß der Beobachter aus geringer Entfernung auf die Funktionsdarstellung blickt. Ein Problem stellt nun die 'Höhe' der Landschaft dar: Will man den höchsten Punkt der Funktion mit in die Darstellung einbeziehen, so muß man eine hinreichend große Entfernung vorwählen. Bei dieser Entfernung können aber bereits wieder die Details in einem anderen Bereich aus der Sicht verschwinden. Eine Möglichkeit der Lösung bestünde nun - wie in Abbildung 1 gezeigt - den Wertebereich oberhalb einer bestimmten Grenze einfach 'abzuschneiden', ich will jedoch in 3.5.7 einer flexibleren Lösung nachgehen.

3.5.6 Darstellung als transformiertes Netz: `GridDisplay`

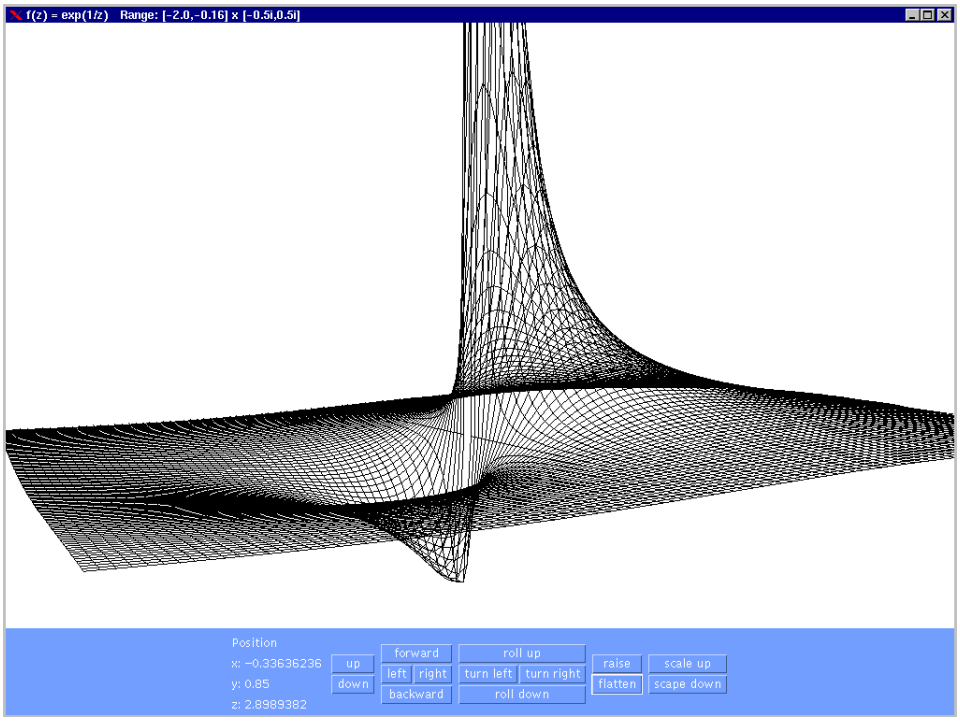
Die Darstellung einer Funktion durch ein transformiertes Koordinatennetz wird auf eine ähnliche Weise durchgeführt wie die des zweidimensionalen Feldes in `Field2D-Display`. Vom Gesichtspunkt der Implementation gibt es also wenig neues hinzuzufügen. Geometrisch unterscheidet sich die Darstellung des transformierten Netzes aber: Die Abmessungen des Graphen sind nicht wie in den anderen zweidimensionalen Darstellungen durch den Definitionsbereich vorgegeben, sondern durch den errechneten Wertebereich der Funktion. Es stellt sich also ähnlich wie in 3.5.3 und 3.5.5 das Problem, daß von einem Graphen häufig entweder ein Teil außerhalb der betrachteten Fläche liegt oder aber ein Großteil des Netzes in einen jenseits der Auflösung befindlich kleinen Bereich abgebildet wird.

3.5.7 Verbesserung der Darstellungen durch Interaktivität

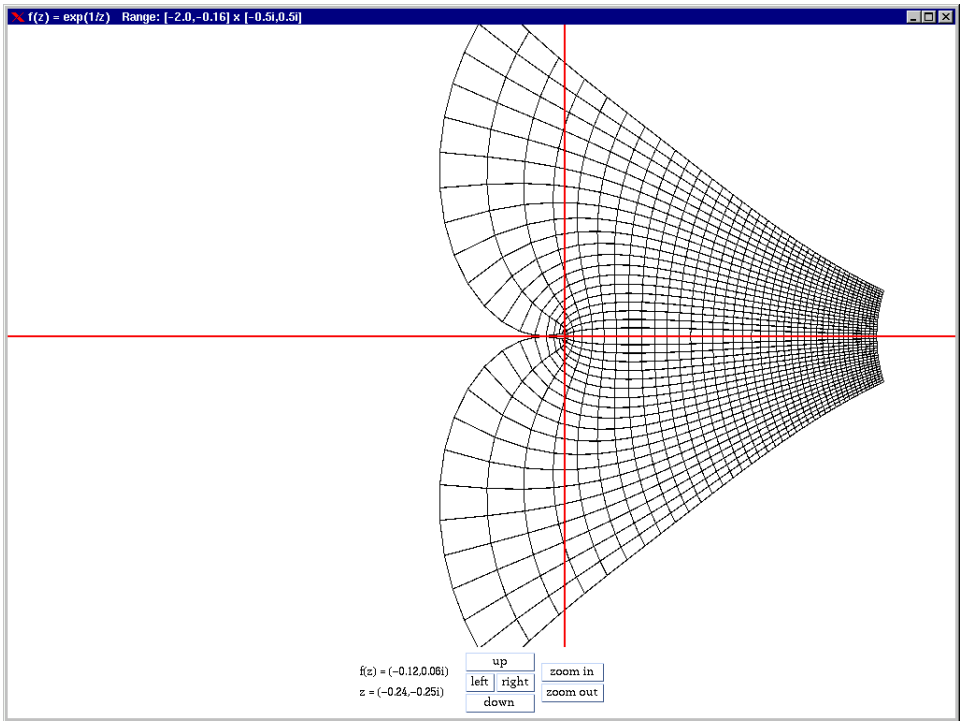
Will man die Probleme in den einzelnen Unterklassen von `ComplexDisplay` zusammenfassen, so ergibt sich folgender Mißstand: Die unvorhersehbare Streuung der Funktionswerte erlaubt keine konstante, einheitliche Betrachtungsweise. Das Problem läßt sich natürlich durch eine monotone nichtlineare Transformation der Funktionswerte beheben, wie dies in 3.5.4 geschieht. Allerdings wird dadurch die Information der absoluten Werte zugunsten der reinen Anordnungsrelation zwischen

[51] Die Funktionalität dieses Pakets kann hier ebenso nur marginal erklärt werden wie seine Verwendung. Eine genaue Spezifikation mit Beispielen findet sich in [Sun 1999b].

[52] Vgl. z.B. [Schlüter 1998].



Darstellung von $\exp(1/z)$ *mittels* `Canvas3DDisplay`



Darstellung von $\exp(1/z)$ *mittels* `GridDisplay`

ihnen eliminiert.[53] Allgemein betrachtet, stehen die Darstellung des Ganzen und die Darstellung des Details in einem komplementären Verhältnis. Ihre scheinbar widersprüchlichen Anforderungen lassen sich damit allein durch die Abbildung des Darstellungsprozesses auf die Zeit aufheben. Dies soll allerdings nach den Überlegungen in 2.2.3 nicht durch Animation, sondern durch Interaktion mit dem Benutzer geschehen. Erlaubt man nämlich dem Benutzer, den Abbildungsmaßstab und seine Perspektive zu wandeln, so wird das Problem auf elegante Weise gelöst. Darüber hinaus ist es in den zweidimensionalen Darstellung möglich, den Benutzer über die Maus einzelne Koordinaten abfragen zu lassen. Dazu müssen die physikalischen Mauskoordinaten durch das Inverse der Abbildungstransformation (des Definitionsbereichs auf den Bildschirmbereich) in Werte aus dem Definitionsbereich zurückgerechnet und der zugehörige Funktionswert ausgerechnet werden.
Es werden somit an den einzelnen Komponenten die folgenden konkreten Erweiterungen vorgenommen:

Field2DDisplay

Die oben erwähnte Abfrage der Funktionswerte mit der Maus wird implementiert. Zunächst geschieht die Ausgabe über die Ermittlung des zum angewählten Definitionsbereichswert gehörigen im Wertefeld gespeicherten Funktionswert. Da diese Angabe aber durch die zum Zeitpunkt der Berechnung angegebene Schrittweite begrenzt und damit unnötig ungenau ist, wird in der Folge ein generelleres Verfahren gewählt: Es wird eine Schnittstelle für ein 'Proxy-Objekt'[54] definiert (`calc.Calculator`), über welche die Ansichtskomponente Zugriff auf das der Funktionsberechnung zugehörige `DoCalc`-Objekt erhält. Dies erlaubt damit die genaue Zuordnung des Funktionswertes mit einer der physikalischen Auflösung entsprechenden Genauigkeit.
Da zusätzlich zum Funktionswert auch die Richtung des Gradienten in der Aufsicht der analytischen Landschaft von Interesse sein dürfte (über ihn lassen sich z.B. Sattelpunkte, 'Rinnen' und 'Rücken' aufspüren), wird auch eine Möglichkeit implementiert, die Richtung des Gradienten experimentell zu bestimmen: 'zieht' man mit der Maus von einem bestimmten Punkt z_0 aus einen Kreis K (mit z_0 als Mittelpunkt) 'auf', so wird auf ∂K die Funktion ausgewertet und die Verbindungslinie $\overline{z_0 z_1}$ eingezeichnet, wobei z_1 in Richtung des maximalen Abstieges liegt: $|f(z_0)| - |f(z_1)| = \max_{z \in \partial K}(|f(z_0)| - |f(z)|)$. Wendet man dieses Verfahren iterativ an, indem man stets am Endpunkt einer Linie einen neuen Kreis aufzieht, so approximiert man auf graphische Weise eine Gradientenlinie durch Sekanten und erhält bei hinreichend kleiner Schrittweite die Spur, die z.B. eine (trägheitslose) Kugel beim Herunterrollen im 'Funktionsgebirge' hinterlassen würde.

Canvas3DDisplay

Im Falle einer dreidimensionalen Landschaft läßt sich das Problem der Skalierung und Perspektivewahl über die Ermöglichung von Navigation durch den Benutzer beheben.
Da das Java 3D API gerade für die Navigation in virtuellen dreidimensionalen Räumen vorgesehen ist, fällt die Implementation dieses Vorhabens nicht schwer: Es müssen lediglich Steuerungsknöpfe in der Benutzerschnittstelle integriert werden, deren Betätigung die Anwendung entsprechender affiner Transformationen auf die Beobachterposition zur Folge hat. So wird beispielsweise beim Drücken des 'right'-Knopfes eine Translation um 0.05 Einheiten auf den Beobachter angewandt, das

[53] Die Farben im Höhendiagramm lassen sich natürlich zusätzlich über eine Legende bestimmten Wertebereichen zuordnen, so daß zumindest auf kognitiver Ebene die Information über die Absolutwerte zugänglich bleibt.
[54] Zum Proxy-Muster: [Buschmann et al.] S. 263ff.

Drücken des 'roll down'-Knopfes hat eine Rotation um die x-Achse um den Winkel $\frac{\pi}{120}$ zur Folge. Darüber hinaus muß die räumliche Repräsentation des Funktionsgraphen in der Größe skalierbar sein, da Definitionsbereich wie Wertebereich von beliebiger Größe sein können. Ich entscheide mich für die separate Aufnahme von Steuerungselementen, deren Betätigung eine Streckung, bei der alle Achsen um denselben Faktor gedehnt werden, bzw. eine Streckung nur in y-Richtung (in der der Funktionswert aufgetragen ist) bewirkt.

GridDisplay

Im Falle der Darstellung einer Funktion durch ein transformiertes Koordinatennetz ist ebenfalls die Navigation durch den Graphen ein wichtiges Hilfsmittel. Auch hier nehme ich - analog zur dreidimensionalen Darstellung - Steuerungskomponenten in die Ansicht mit auf, die eine Translation ('Panning') oder Skalierung ('Zooming') des Graphen zur Folge haben.

Ausblick

Die Liste der interaktiven Erweiterungen könnte noch um einiges fortgesetzt werden (z.B. 'Zooming & Panning' in den 2D-Feld-Darstellungen), ich will mich aber exemplarisch auf die vorgenommenen Implementationen beschränken.
Was zeigt dieser Abschnitt? Erst durch das Einbauen interaktiver Elemente werden die Darstellungen den Möglichkeiten elektronischer Medien gerecht. In der Fähigkeit, mit dem Benutzer in einen tieferen Dialog zu treten, emanzipiert sich die Rolle der elektronischen Medien von der der konventionellen. Statische Informationen werden dagegen im allgemeinen durch gedruckte Medien flexibler und (aufgrund der höheren Auflösung von Druckern gegenüber Monitoren) exakter dargestellt, was z.B. durch das häufig praktizierte Ausdrucken von elektronischen Dokumenten belegt wird. Mit der Propagation elektronischer Darstellungen soll natürlich nicht die bislang notwendige Fähigkeit des menschlichen Geistes, die Statik in der Phantasie zu dynamisieren, ausgebremst werden. Vielmehr entsteht m. E. durch die Ausnutzung der interaktiven Möglichkeiten elektronischer Medien ein Angebot, intuitive Vorstellungen intersubjektiv sichtbar zu machen und sie damit zu kommunizieren.

4 Berechnung der Riemannschen ζ-Funktion

In diesem Kapitel kommen wir auf das in Kapitel 1.1 angeführte Beispiel zurück. Es wird nun - zum einen, um die Erweiterbarkeit der Programmbibliotheksarchitektur zu demonstrieren, zum anderen, um die konkreten Mittel für den in 1.1 beschriebenen Analyseprozeß bereitzustellen - die Darstellung der Riemannschen ζ-Funktion durch die Euler-MacLaurin-Summenformel hergeleitet und diese programmiertechnisch umgesetzt.

4.1 Mathematische Herleitung

Die folgende Herleitung entspricht dem Vorgehen von [Rademacher 1973]; eine Einführung zu der Euler-MacLaurin Summenformel findet sich aber in einer Vielzahl von Lehrbüchern.[55]. Es wird zunächst ausgehend von der Definition der Bernoulli-Polynome ihre fundamentale Rekursionsformel hergeleitet und anhand dieser die Euler-MacLaurin Summenformel durch iterierte partielle Integration konstruiert. Genauere Abschätzungen zu den Bernoulli-Polynomen und -zahlen werden auf das folgende Kapitel verschoben, dessen Thema die numerische Betrachtung ist.

[55] So z.B. kürzere in [Knopp 1996] S. 537-554, [Stoer 1993] S.122-125.

4.1.1 Bernoullizahlen und -polynome

Die Bernoulli-Polynome wurden durch Jacob Bernoulli (1654-1705) zur einheitlichen Beschreibung von Summen des Typs $\sum_{n=1}^{N} n^q$ eingeführt. Dabei machte er sich die summatorische Eigenschaft der Binomialkoeffizienten zunutze:

Definition 1 *Der Binomialkoeffizient* $\binom{x}{j}$ *wird für den Körper* $\mathbb{Q}$ *definiert als*

$$\binom{x}{j} := \frac{x(x-1)\cdots(x-j+1)}{1\cdot 2\cdots j} \text{ für } x \in \mathbb{Q},\ j \in \mathbb{N}. \tag{3}$$

Damit läßt sich $\binom{X}{j}$ als Polynom $P \in \mathbb{Q}[X]$ verstehen mit grad $P = j$.
Man überprüft direkt durch Ausrechnen:

$$\binom{x}{j} = \binom{x+1}{j+1} - \binom{x}{j+1}. \tag{4}$$

Betrachtet man nun den Polynomring $\mathbb{Q}[X]$ als einen $\mathbb{Q}$-Vektorraum, dann ist $\left\{\binom{X}{j} | 0 \le j \le q\right\}$ eine Basis von $\{P \in \mathbb{Q}[X] | 0 \le \text{grad} P \le q\}$. Damit existieren zu gegebenen $n, q \in \mathbb{N}$ eindeutig bestimmte Koeffizienten $A_{qj} \in \mathbb{Q}$, mit

$$n^q = \sum_{j=0}^{q} A_{qj} \binom{n}{j}. \tag{5}$$

Aus (4) ergibt sich damit:

$$n^q = \sum_{j=0}^{q} A_{qj} \left\{\binom{n+1}{j+1} - \binom{n}{j+1}\right\}. \tag{6}$$

bzw. bei mehrfacher Anwendung

$$\sum_{n=M}^{N-1} n^q = \sum_{j=0}^{q} A_{qj} \left\{\binom{N}{j+1} - \binom{M}{j+1}\right\}. \tag{7}$$

Wir führen nun die Bernoulli-Polynome (zunächst als Funktionen $B_q : \mathbb{N}_0 \to \mathbb{Q}$) ein:

Definition 2

$$B_q(n) := q \sum_{j=0}^{q-1} A_{q-1,j} \binom{n}{j+1} + C_q \ ,\ q \in \mathbb{N}, \tag{8}$$

mit einer noch zu bestimmenden Konstante C_q. Aus der Definition folgt für $q > 1$ sofort:

$$B_q(0) = C_q, \tag{9}$$
$$B_q(1) = C_q. \tag{10}$$

Es gilt nun wegen (7):

$$\sum_{n=M}^{N-1} n^q = \frac{1}{q+1}\{B_{q+1}(N) - B_{q+1}(M)\}. \tag{11}$$

Die Summierung von beliebigen Potenzen ist also einfach durch das Auswerten des zugehörigen Bernoulli-Polynoms zu errechnen. Um nun eine einfachere Konstruktionsvorschrift für $B_q(x)$ zu finden, erweitern wir den Definitionsbereich auf $\mathbb{R}$ und betrachten im folgenden wieder den Spezialfall für ein einzelnes Monom n^q, also $N-1=M=n$. Differenziert man diese Gleichung auf beiden Seiten so erhält man:

$$qn^{q-1} = \frac{1}{q+1}\left\{B'_{q+1}(n+1) - B'_{q+1}(n)\right\}, \tag{12}$$

der Vergleich mit der ursprünglichen Gleichung liefert:

$$B_q(n+1) - B_q(n) = \frac{1}{q+1}\left\{B'_{q+1}(n+1) - B'_{q+1}(n)\right\} \tag{13}$$

bzw.

$$\frac{1}{q+1}B'_{q+1}(n+1) - B_q(n+1) = \frac{1}{q+1}B'_{q+1}(n) - B_q(n). \tag{14}$$

Dies zeigt, daß $\frac{1}{q+1}B'_{q+1}(n) - B_q(n)$ die Periode 1 hat, was bei einem Polynom allerdings nur für eine konstante Funktion gelten kann. Es folgt also:

$$\frac{1}{q+1}B'_{q+1}(n) - B_q(n) = K_q. \tag{15}$$

Wir können nun die Konstante C_q so wählen, daß K_q verschwindet. Es gilt dann

$$\frac{1}{q+1}B'_{q+1}(n) = B_q(n). \tag{16}$$

Nun lassen sich die Bernoulli-Polynome rekursiv berechnen. Definition 2 ergibt für $q=1$:

$$B_1(n) = n + C_1. \tag{17}$$

Integriert man diese Gleichung, so erhält man mit (9) und (16):

$$\frac{1}{2!}B_2(n) = \frac{n^2}{2!} + \frac{C_1}{1!}n + \frac{C_2}{2!}. \tag{18}$$

und durch $(q-1)$-fache Integration ergibt sich:

$$\frac{1}{q!}B_q(n) = \frac{n^q}{q!} + \frac{C_1 \cdot n^{q-1}}{1!(q-1)!} + \cdots + \frac{C_{q-1}\cdot n}{(q-1)!1!} + \frac{C_q}{q!}. \tag{19}$$

Durch Einsetzen von $B_q(1) = C_q$ in diese Gleichung ergibt sich damit die Ermittlung der Integrationskonstante $\frac{C_q}{q!}$ aus den vorangehenden, bereits berechneten $C_r, r<q$. Die Zahlen C_q nennen wir von nun an B_q, die *Bernoullizahlen.*
Multipliziert man die obige Gleichung mit $q!$, so erhält man für sie den folgenden Zusammenhang:

$$0 = 1 + \binom{q}{1}B_1 + \binom{q}{2}B_2 + \cdots + \binom{q}{q-1}B_{q-1}, q \geq 2. \tag{20}$$

Wir haben damit zunächst die Bernoull-Polynome aufgrund ihrer Eigenschaft, Summen von Potenzen darzustellen, betrachtet. Die einfache Rekursionsbeziehung ergibt aber im Zusammenhang mit dem Prozeß der partiellen Integration eine weitere interessante Anwendung bei der Darstellung von analytischen Funktionen:

4.1.2 Die Euler-MacLaurin-Summenformel

Sei $f : \mathbb{R}_+ \to \mathbb{R}; x \mapsto f(x)$ eine hinreichend oft stetig differenzierbare Funktion. Dann kann man durch wiederholte partielle Integration mit (16) (und unter Berücksichtigung, daß $B_1'(x) \equiv 1$) folgende Identität herleiten:

$$\begin{aligned}\int_0^1 f(x)dx &= [B_1(x)f(x)]_0^1 - \int_0^1 B_1(x)f'(x)dx \\ &= \sum_{r=1}^{q}(-1)^{r-1}\left[\frac{B_r(x)}{r!}f^{(r-1)}(x)\right]_0^1 \\ &\quad +(-1)^q \int_0^1 \frac{B_q(x)}{q!}f^{(q)}(x)dx. \end{aligned} \tag{21}$$

Dies läßt sich (erneut unter spezieller Berücksichtigung von $B_1(x) = x - 1/2$) umformen zu:

$$\begin{aligned}f(1) &= \int_0^1 f(x)dx + \sum_{r=1}^{q}(-1)^r \frac{B_r}{r!}\left\{f^{(r-1)}(1) - f^{(r-1)}(0)\right\} \\ &\quad +(-1)^{q-1}\int_0^1 \frac{B_q(x)}{q!}f^{(q)}(x)dx. \end{aligned} \tag{22}$$

Wenden wir nun auf dieses Ergebnis auf die Funktion $x \mapsto f(n-1+x)$ an, so erhalten wir

$$\begin{aligned}f(n) &= \int_0^1 f(n-1+x)dx + \sum_{r=1}^{q}(-1)^r \frac{B_r}{r!}\left\{f^{(r-1)}(n) - f^{(r-1)}(n-1)\right\} \\ &\quad +(-1)^{q-1}\int_0^1 \frac{B_q(x)}{q!}f^{(q)}(n-1+x)dx. \end{aligned} \tag{23}$$

Durch Summieren von $a+1$ bis b, $a, b \in \mathbb{N}$, erhält man

$$\begin{aligned}\sum_{n=a+1}^{b} f(n) &= \int_a^b f(x)dx + \sum_{r=1}^{q}(-1)^r \frac{B_r}{r!}\left\{f^{(r-1)}(b) - f^{(r-1)}(a)\right\} \\ &\quad +\frac{(-1)^{q-1}}{q!}\int_a^b B_q(x - [x])f^{(q)}(x)dx. \end{aligned} \tag{24}$$

4.1.3 Die Riemannsche ζ-Funktion

Die Riemannsche ζ-Funktion wurde durch Riemann definiert als:

Definition 3

$$\zeta(s) := \sum_{n=1}^{\infty}\frac{1}{n^s}\ ,\ s = \sigma + it \in \mathbb{C}; \sigma > 1. \tag{25}$$

Betrachtet man die Funktion als Grenzwert der Folge der Partialsummen $\sum_{n=1}^{m} n^{-s}$, so folgt aus deren Holomorphie und lokal gleichmäßiger Konvergenz in der Halbebene $\sigma > 1$ nach dem Weierstraßschen Konvergenzsatz die Holomorphie von ζ in diesem Gebiet.
Wir wollen nun über die Euler-MacLaurin-Summenformel eine analytische Fortsetzung der Riemannschen ζ-Funktion konstruieren. Dazu schreibt man sie zunächst in der Form

$$\zeta(s) = 1 + \lim_{N\to\infty} \sum_{n=2}^{N} n^{-s} \tag{26}$$

und stellt sodann fest, daß mit

$$f(x) \quad = \quad x^{-s}, \qquad f^{(k)}(x) = (-1)^k s(s+1)\cdots(s+k-1)x^{-s-k} \tag{27}$$

und (24) für $\sigma > 1$ folgende Gleichung gilt

$$\begin{aligned} \zeta(s) - 1 \quad = \quad & \lim_{N\to\infty} \left\{ \int_1^N x^{-s} dx + \frac{1}{2}(N^{-s} - 1) \right. \\ & - \sum_{r=2}^{q} \frac{B_r}{r!} s(s+1)\cdots(s+r-2)(N^{-s-r+1} - 1) \\ & \left. - \frac{1}{q!} s(s+1)\cdots(s+q-1) \int_1^N B_q(x - [x]) x^{-s-q} dx \right\}. \end{aligned} \tag{28}$$

Wir berechnen nun den Limes:

$$\begin{aligned} \zeta(s) \quad = \quad & \frac{1}{s-1} + \frac{1}{2} + \sum_{r=2}^{q} \frac{B_r}{r!} s(s+1)\cdots(s+r-2) \\ & - \frac{1}{q!} s(s+1)\cdots(s+q-1) \int_1^\infty B_q(x - [x]) x^{-s-q} dx. \end{aligned} \tag{29}$$

Das Integral konvergiert nun für $\Re s > 1 - q$ lokal gleichmäßig in s und stellt somit dort eine holomorphe Funktion dar. Da $q \in \mathbb{N}$ beliebig ist, ist somit die holomorphe Fortsetzbarkeit von ζ auf $\mathbb{C} \setminus \{1\}$ (in $s = 1$ liegt ein vom Term $\frac{1}{s-1}$ herührender Pol 1. Ordnung vor) bewiesen und gleichzeitig eine Formel gefunden, die wir zur numerischen Berechnung von $\zeta(s)$ verwenden wollen.

4.2 Implementation des Berechnungsmechanismus

4.2.1 Einführung

Anhand der oben hergeleiteten Beziehung läßt sich nun die Implementation der ζ-Funktion vornehmen. Um die Umsetzung der mathematischen in programmiertechnische Beziehungen weiterhin im objektorientierten Rahmen zu vollziehen (und damit flexibel gegenüber folgenden Erweiterungen zu halten), müssen auf der Ebene der letzteren Klassen eingeführt werden, die den mathematischen Begriffen entsprechen. So sollte z.B. der Begriff des Polynoms abgebildet werden auf eine Klasse gleichen Namens, welche die gängigen Operationen mit Polynomen (Multiplikation, Auswertung für spezielle Zahlenwerte, Integration, etc.) implementiert. Im folgenden möchte ich - ausgehend von der Anforderung durch den Benutzer der neuen Funktion - einen kurzen Blick auf die zusätzlich zu implementierenden Klassen (die natürlich allesamt im `calc`-Paket liegen) werfen.

4.2.2 Erweiterungen in `ExprParser`

Wie wir bereits in 3.3.2 festgestellt hatten, sind für die Einführung zusätzlicher Funktionen nur wenige Änderungen an `ExprParser` notwendig: Gibt ein Nutzer z.B. 'zeta(z)' im Funktionsfeld ein, so behandelt der Compiler die Funktion genauso wie etwa 'exp(z)', er muß lediglich anstelle eines `ExpCalc`-Objektes ein (neu zu konstruierendes) Objekt vom Typ `ZetaCalc` in den Berechnungsbaum einhängen. In diesem Objekt wird dann die elementare Berechnung der Funktion für einen speziellen Wert an eine statische Methode in der `Complex`-Klasse weiterdelegiert.

4.2.3 Erweiterungen in `Complex`

Die Implementation in `Complex` muß nun einerseits die Rechenoperationen aus (29) in die funktionale Sprache der Methodenaufrufe übersetzen, andererseits muß dort auch die Erzeugung der notwendigen Funktionsobjekte (die im folgenden besprochen werden) vorgenommen werden. So wird z.B. der mathematische Ausdruck $\frac{1}{s-1} + \frac{1}{2}$ im Code von `Complex.zeta()` in das Programmfragment `add(div(new Complex(1,0), add(s, new Complex(-1,0))), new Complex(0.5d,0))` umgewandelt. Dabei müssen Operationen auf Funktionen (z.B. Integration) im Implementationsbereich auf Methoden der folgenden Funktionsobjekte abgebildet werden.

4.2.4 Die Klasse `Polynom`

Ein (reelles) Polynom $p \in \mathbb{R}[X]$ wird auf Implementationsebene verstanden als ein `double`-Feld von Koeffizienten, wobei die Position im Feld zugleich den zugeordneten Exponenten beschreibt: Z. B. wird $\frac{X^4}{5} - 3X^2 + 1$ abgebildet auf {`1, 0, -3, 0, 0.2`}. Die Operationen verändern nun (gemäß der mathematischen Semantik) lediglich die Einträge dieses Feldes, so erzeugt etwa die Translation eines Polynoms $p(X) \mapsto p(X + k)$ in `Polynom.translate(double k)` das Feld neu und bestimmt deren Koeffizienten gemäß der Ausmultiplikation von $p(X + k)$.

4.2.5 Die Klasse `zeta.BernoulliPolynomBuilder`

In unserem Fall müssen nun nicht nur allgemein Polynome, sondern im besonderen Bernoulli-Polynome erzeugt werden. Die Erzeugung, die nach der Berechnungsvorschrift (16) vorgeht (die Bernoullizahlen werden dabei ebenfalls von dieser Klasse berechnet), wird in eine gesonderte Klasse verlegt, die aufgrund ihrer Spezialisierung gegenüber den anderen Berechnungsklassen in einem Unterpaket namens `calc.zeta` angelegt wird.
`BernoulliPolynomBuilder.buildPolynom(int q)` erzeugt ein `Polynom`, das die Koeffizienten von $B_q(x)$ enthält. Diese (statische) Methode wird von `Complex.zeta()` für ein spezielles Argument aufgerufen. Da diese Berechnung im weiteren Verlauf aber nicht nur für reelle, sondern auch für komplexe Argumente vollzogen werden soll, ist eine Klasse notwendig, die eine ähnliche Funktionalität wie `Polynom` aufweist, darüber hinaus aber nicht nur komplexe Koeffizienten, sondern auch komplexe Exponenten erlaubt; denn das erzeugte Bernoulli-Polynom soll ja vor der Integration in (29) mit x^{-s} multipliziert werden. Das Objekt, das diese Funktionalität bewerkstelligt, ist von der Klasse `ComplexPowerSum`.

4.2.6 Die Klasse `ComplexPowerSum`

Anders als `Polynom`, wo ein Eintrag im `double`-Feld zugleich Koeffizienten wie Exponenten determinierte, sind in `ComplexPowerSum` zwei Felder (gleicher Länge) notwendig: ein Feld für die Koeffizienten und ein Feld für die zugeordneten Exponenten.

Dadurch wird der Verwaltungsaufwand bei der implementationstechnischen Abbildung der mathematischen Operationen komplizierter. Beispielsweise die Addition zweier dieser Objekte macht zunächst einen Vergleich auf gemeinsame Potenzen hin notwendig, eine Translation ist nicht allgemein implementierbar und bei der Integration muß der Fall $f(z) = \frac{1}{z}$ gesondert behandelt werden.[56]

Mit der Implementation dieser Klasse ist die praktische Umsetzung der Formel (29) abgeschlossen. Wir wollen nun im folgenden Kapitel auf die im Zuge dieser Ausführungen sich entwickelnde Frage nach dem zeitlichen Berechnungsaufwand eingehen und aus den sich ergebenden Antworten heraus verschiedene Optimierungsansätze verfolgen.

5 Betrachtung von numerischer Genauigkeit und Zeitverhalten

5.1 Praktische Aspekte der Berechnung

5.1.1 Wahl von q und N

Um überhaupt eine Berechnung mittels der oben vorgenommenen Implementierung durchzuführen, müssen zunächst in (29) die Anzahl der partiellen Integrationen q und die obere Integrationsgrenze N (anstelle von ∞) gesetzt werden. Um diese Parameter vernünftig zu besetzen, müssen zunächst die Rahmenbedingungen betrachtet werden:

- Die numerische Genauigkeit innerhalb der Berechnung wird durch den (IEEE 754-konformen) `double`-Datentyp mit 52 Bits bzw. ca. 16-stelliger Genauigkeit vorgegeben.[57]

- Die ζ-Funktion wird aufgrund der Überlegungen in 2.1 nur im Bereich des kritischen Streifens und geringfügig rechts davon betrachtet werden. Wir gehen daher in der Folge bei Abschätzungen von $0 \leq \sigma \leq 3$ aus. Die Länge des im kritischen Streifen betrachteten Intervalls soll fürs erste durch $|t|_{\max} \leq 100$ beschränkt werden, später (Unterkapitel 5.2) wird eine genaue Abschätzung in Abhängigkeit von t vorgenommen.

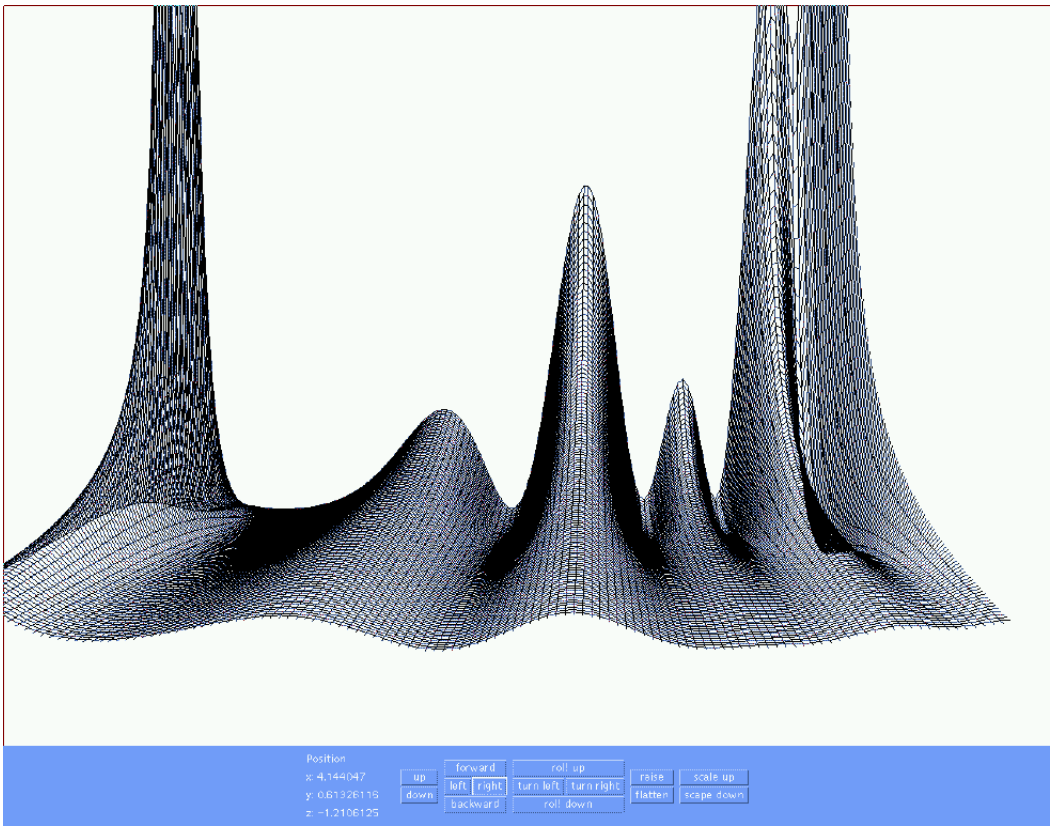
- Für Zwecke der graphischen Darstellung sind wir mit einer Genauigkeit, die Werte bis auf den absoluten Fehler $\Delta\zeta(s) < 10^{-4}$ exakt ausrechnet, zufrieden.

Anhand von praktischen Berechnungen ergeben sich bezüglich der Genauigkeit folgende beobachtbare Effekte abhängig von der Wahl von q und N:

- Wird q zu gering angesetzt, so fällt der Integrand in (29) für $x \to \infty$ nicht stark genug ab, so daß ein großes N verwendet werden muß. Dies hat, da die Integration den aufwendigsten Teil der Berechnung darstellt, zur Folge, daß die Berechnungszeit beim Einhalten der Genauigkeit stark ansteigt.

[56] Dabei wird der logarithmische Anteil in einem dafür vorgesehenen Variablenbereich festgehalten und das Objekt für weitere Operationen als 'ungültig' erklärt, da ansonsten die weitere Aufnahme von Sonderfällen das Objektkonzept aufweichen würde. Die stark zunehmende Komplexität ist im übrigen auch der Grund, dem die Klasse `Polynom` - sowohl unter strukturellen wie berechnungszeitkritischen Gesichtspunkten - ihre Existenzberechtigung verdankt.

[57] [Flanagan 1998] S. 26. Eine Implementation mit Datentypen höherer Genauigkeit (z.B. `java.math.BigDecimal`) ist bei der bestehenden Architektur grundsätzlich möglich, erscheint aber im Rahmen der Spezifikation, die Geschwindigkeit gegenüber hoher Genauigkeit als wichtiger einstuft, wenig sinnvoll.



Darstellung von $\zeta(z)$ mittels `Canvas3DDisplay`

- Wird q zu groß angesetzt, so wirken sich die in mindestens kubischer Ordnung ansteigenden Berechnungen in der Summe in (29) ebenfalls auf die Berechnungszeit aus. Zusätzlich ergibt sich ein hoher numerischer Fehler für betragsmäßig kleine Werte von $\zeta(s)$ (die im uns interessierenden Bereich gegeben sind), da sowohl die Bernoullizahlen als auch die Bernoulli-Polynome stark anwachsen.[58]

Es muß also zwischen dem systematischen Fehler, der durch das 'Abkappen' des Integrals an der Stelle N ensteht, und dem implementationstechnischen Fehler, der sich durch die `double`-Genauigkeit ergibt, unterschieden werden. Während letzterer aufgrund der zahlreichen (und verschachtelten) Operationen sehr schwer vorherzusagen ist,[59] kann ersterer ziemlich genau abgeschätzt werden. Dies soll nun getan werden.

Ich wähle zunächst willkürlich (bzw. anhand der obigen Überlegungen) $q = 6$. Es genügt dann $N = 37$ den obigen Anforderungen.

[58] Als sinnvolle Obergrenze kann dabei $q \leq 18$ angegeben werden, da bereits für diesen Wert $q!$ nicht mehr ohne Verlust als `double`-Wert dargestellt werden. Für große t muß q aufgrund des ansteigenden Vorfaktors (der separat von der Fakultät berechnet wird) jedoch schon entsprechend kleiner gewählt werden.

[59] Seine Vernachlässigbarkeit ergibt sich im Nachhinein aus dem 'Zutreffen' der Abschätzungen bzgl. des systematischen Fehlers in der Praxis (s. z.B. die Berechnung von $\zeta(\frac{1}{2} + 1000000i)$ weiter unten).

5.1.2 Optimierung auf Implementationsebene

Bei der Implementation wurde gemäß dem Grundsatz 'First make it work, then make it fast...'[60] zunächst keinerlei Optimierung vorgenommen. Der erste Testlauf mit den oben eingesetzten Werten ergab eine Berechnungszeit von 0.05 Sekunden[61] für einen Wert von $\zeta(s)$. Damit ergäbe sich z.B. bei einem Gitter von 200×200 Werten eine Berechnungsdauer von 2000 Sekunden; dies ist dem Benutzer nicht zuzumuten. Es stellt sich also die Frage nach Optimierungsmöglichkeiten. In unserer konkreten Situation ergeben sich zwei mögliche Wege:

1. Die Optimierungen finden auf Implementationsebene statt.
2. Die Optimierungen finden auf der konzeptionellen, also mathematischen Ebene statt, indem die Formel (29) durch eine für numerische Belange verbesserte ersetzt wird.

Ich beschränke mich in diesem Unterkapitel auf Optimierungen auf Implementationsebene, da der hierfür notwendige Änderungsaufwand vergleichsweise gering ist. Der 'Flaschenhals' in Java wird, wie bereits erwähnt, durch die Erzeugung von Objekten gebildet; daher gilt es, in häufig durchlaufenen Programmteilen Objekterzeugungen durch Vorhalten ('Caching') und Wiederverwendung ('Recycling') von Objekten zu minimieren. So kann man u.a. in `calc.zeta.BernoulliPolynomBuilder` die bereits generierten Bernoulli-Polynome und -Zahlen speichern und sie bei erneuter Anfrage aus dem Speicher auslesen, anstatt sie neu zu berechnen.[62] Ebenso werden Konstanten für bestimmte, häufig verwendete komplexe Zahlen definiert (`Complex.ONE`, `Complex.ZERO`). Ein erneuter Meßvorgang ergibt eine durchschnittliche Berechnungsdauer von 0.019 Sekunden, also eine Beschleunigung um mehr als das doppelte.

5.2 Berechnung ohne Integral

Der Berechnungsprozeß ließe sich ohne Frage bedeutend mehr beschleunigen, wenn man den aufwendigen Integrationsprozeß komplett vernachlässigen könnte. Dies wäre unter den in 5.1.1 beschriebenen Rahmenbedingungen möglich, wenn man $\zeta(s)$ aufspaltet in eine unendliche und eine endliche Summe bestimmter Länge m:

$$\zeta(s) = \sum_{n=1}^{m} n^{-s} + \sum_{n=m+1}^{\infty} n^{-s}. \tag{30}$$

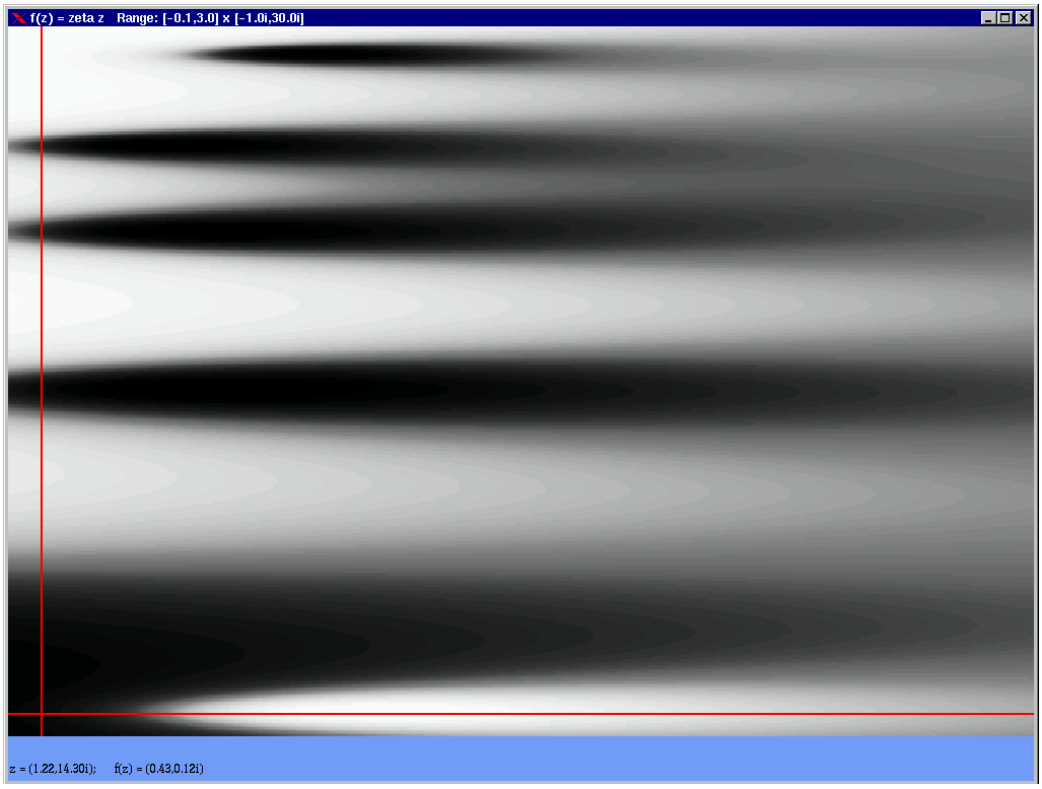
und auf letztere die Euler-MacLaurin Summenformel (24) mit (27) anwendet:

$$\begin{aligned}\sum_{n=m+1}^{\infty} n^{-s} \quad = \quad & \lim_{N\to\infty} \left\{ \int_m^N x^{-s}dx + \frac{1}{2}(N^{-s} - m^{-s}) \right. \\ & - \sum_{r=2}^{q} \frac{B_r}{r!} s(s+1)\cdots(s+r-2)(N^{-s-r+1} - m^{-s-r+1}) \\ & \left. - \frac{1}{q!} s(s+1)\cdots(s+q-1) \int_m^N B_q(x-[x])x^{-s-q}dx \right\}. \end{aligned} \tag{31}$$

[60] '... - if you must.'[Eckel 1998] S. 317

[61] Gemessen auf einer SPARC Sun Ultra-2, 300 MHz, 512 MB, Solaris 2.6

[62] Dies ist ebenso für die Berechnung von $n!$ möglich, allerdings stellt sich dort bei kleinem n - wie es hier benötigt wird - die Frage, ob die Speicherung eines Wertes gegenüber seiner Berechnung einen Vorteil bringt. In der Praxis ($n \leq 17$, 10000 Aufrufe) erwiesen sich beide Verfahren als annähernd gleich schnell. Der Vergleich macht das praktische Verhältnis von Speicherzugriffs- und Prozessorgeschwindigkeit heutiger Rechner deutlich.



Darstellung von $\zeta(z)$ mittels `AltitudeDisplay`

Damit erhält man:

$$\begin{aligned}\zeta(s) &= \sum_{n=1}^{m} n^{-s} + \frac{m^{-s+1}}{s-1} - \frac{m^{-s}}{2} + \sum_{r=2}^{q} \frac{B_r}{r!} s \cdots (s+r-2) m^{-s-r+1} \\ &\quad -\frac{1}{q!} s(s+1)\cdots(s+q-1) \int_m^\infty B_q(x-[x]) x^{-s-q} dx, \end{aligned} \tag{32}$$

wobei das Integral auch hier wie in 4.1.3 für $\Re s > 1-q$ lokal gleichmäßig konvergiert.

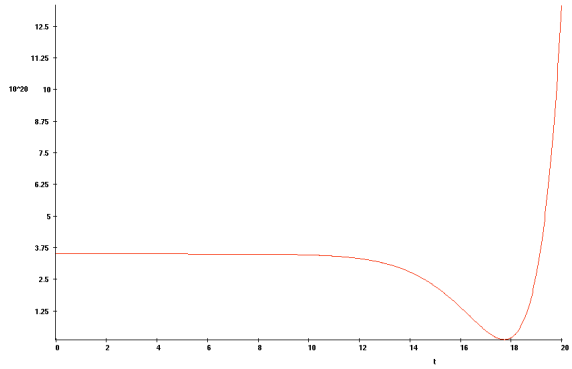
5.2.1 Abschätzung des Integrals

Man kann nun das Integral bei gegebenem q und $\sigma_{\max}$ gemäß der Abschätzung

$$\begin{aligned}&\left| \frac{1}{q!} s(s+1)\cdots(s+q-1) \int_m^\infty B_q(x-[x]) x^{-s-q} dx \right| \\ &\leq \left| \frac{1}{q!} (\sigma_{\max}+it)\cdots(\sigma_{\max}+it+q-1) \right| \max_{x\in[0,1]} |B_q(x)| \, \frac{m^{-\sigma_{\min}-q+1}}{q-1+\sigma_{min}} \end{aligned}$$

für $\Re s \in [\sigma_{\min}, \sigma_{\max}]$ und bei einem hinreichend groß gewähltem $m(q,t)$ vernachlässigen (da $\zeta(\sigma+it) = \overline{\zeta(\sigma-it)}$ gilt, betrachte ich hier und im folgenden o. B. d. A. den Fall $t \geq 0$). Es erscheint nun günstig, ein höheres q als in 5.1.1 zu wählen, ich verwende aus numerischen Gründen $q = 17$.[63] Der erste Schritt besteht in der

[63] 17! hat 15 Stellen, läßt sich also gegenüber größeren Fakultäten ohne Abbruchfehler als `double`-Zahl darstellen. Zudem ist das 17. Bernoulli-Polynom besonders gut abschätzbar.



Graph von $10 \cdot t^{16} + 3.5 \cdot 10^{20} - \left|\Pi_{\sigma=3}^{18}(\sigma + it)\right|$

Abschätzung des Vorfaktors:

$$\begin{aligned} \left|\frac{(3+it)\cdot(4+it)\cdots(18+it)}{17!}\right| &\leq \frac{1}{17!}\prod_{\sigma=3}^{18}(\sigma+t) && (33)\\ &= \frac{1}{17!}(t^{16}+168t^{15}+13060t^{14}+\ldots) \\ &< \begin{cases} \frac{t^{16}}{17!}+48t^{15}+9 & \text{für } t \geq 1 \\ 58 & \text{für } 0 \leq t < 1. \end{cases} && (34) \end{aligned}$$

Der Vorfaktor läßt sich also insgesamt durch ein Polynom 16-ten Grades abschätzen. Da $\prod_{\sigma=3}^{18}(\sigma + it)$ aber ein Polynom mit abwechselnd reellen und imaginären Koeffizienten mit alternierenden Vorzeichen liefert, ist in praktischer Hinsicht bereits (33) eine sehr schlechte Abschätzung, weshalb ich im folgenden eine experimentell ermittelte Abschätzung des Vorfaktors verwenden möchte:

$$\left|\frac{(3+it)\cdot(4+it)\cdots(18+it)}{17!}\right| < \frac{10}{17!}t^{16}+10^6. \qquad (35)$$

Man sieht dabei: Für $t \to 0$ ist die rechte Seite größer. Für große t ist die rechte Seite größer als die rechte Seite in (33) (für $t \geq 1000$ z.B. ergibt sich dies bereits aus Betrachtungen zur Anzahl der Ziffern des Ergebnisses). Der Bereich dazwischen bedarf einer genauen numerischen Analyse, es zeigt sich dabei, daß die minimale Differenz zwischen den beiden Ausdrücke in (35) bei $17 \leq t \leq 18$ liegt (s. Abb. 17).

Als nächstes betrachten wir das 17. Bernoulli-Polynom:

$$\begin{aligned} B_{17}(x) &= x^{17} - \frac{17}{2}x^{16} + \frac{68}{3}x^{15} - \frac{238}{3}x^{13} + \frac{884}{3}x^{11} - \frac{2431}{3}x^9 \\ &\quad + \frac{4420}{3}x^7 - \frac{23494}{15}x^5 + \frac{2380}{3}x^3 - \frac{3617}{30}x. \end{aligned}$$

Da der Betrag eines Polynoms zwischen 0 und 1 kleiner ist als die Summe seiner Koeffizientenbeträge, gilt damit $|B_{17}(x - [x])| \leq 5170$. Für die Gesamtabschätzung ergibt sich somit:

$$\begin{aligned} &\left|\frac{(s+it)\cdot(s+1+it)\cdots(s+15+it)}{17!}\int_m^\infty B_{17}(x-[x])x^{-17}dx\right| \\ < &\left(\frac{10}{17!}t^{16}+10^6\right)\left|\int_m^\infty \frac{5170}{x^{17}}dx\right| \end{aligned}$$

$$< \quad (10^{-11} \cdot t^{16} + 3.3 \cdot 10^8) \cdot m^{-16}.$$

Damit ergibt sich bei einem angestrebten *absoluten* Fehler von $\Delta\zeta(s) < 10^{-4}$:

$$m \quad > \quad 0.37 \cdot |t| + 6.05 > \sqrt[16]{10^{-7} \cdot t^{16} + 3.3 \cdot 10^{12}}.$$

5.3 Verbesserung der Abschätzung

Die oben vorgenommene Abschätzung des Integrals ist sicherlich viel zu hoch angesetzt. Dies erkennt man bereits daran, daß $B_{17}(1) = B_{17}(0) = 0$ gilt und die höchsten Koeffizienten unterschiedliches Vorzeichen haben. Zudem wäre es günstig, für ein beliebiges $B_q(x)$ eine Abschätzung angeben zu können. Eine solche Abschätzung erhält man, wenn man $B_q(x - [x])$ als periodische Funktion betrachtet und einer Fourier-Analyse unterzieht.

5.3.1 Fourier-Analyse der Bernoulli-Polynome

Betrachtet man $f_q(x) := B_q(x - [x])$ als Funktion mit Periode 1, so stellt man fest, daß diese beschränkt und stetig differenzierbar auf $\mathbb{R} \setminus \mathbb{Z}$ ist. Für $q \geq 2$ ist f_q sogar wegen $B_q(0) = B_q(1)$ überall stetig. Damit ist f_q in eine Fourier-Reihe entwickelbar, wobei diese für $q \geq 2$ gleichmäßig gegen f_q konvergiert:[64]

$$B_q(x - [x]) \quad = \quad \frac{a_0^{(q)}}{2} + \sum_{n=1}^{\infty} (a_n^{(q)} \cos(2\pi n x) + b_n^{(q)} \sin(2\pi n x)), \tag{36}$$

mit den Projektionen auf die Basisfunktionen

$$a_n^{(q)} \quad = \quad 2 \int_0^1 B_q(x) \cos(2\pi n x) dx \tag{37}$$

$$b_n^{(q)} \quad = \quad 2 \int_0^1 B_q(x) \sin(2\pi n x) dx. \tag{38}$$

Für $a_0^{(q)}$ ergibt sich mit (16):

$$\begin{aligned} a_0^{(q)} \quad &= \quad 2 \int_0^1 B_q(x) dx \\ &= \quad \frac{2}{q+1} \int_0^1 B'_{q+1}(x) dx \\ &= \quad \frac{2}{q+1} (B_{q+1}(1) - B_{q+1}(0)) = 0. \end{aligned} \tag{39}$$

für alle $q \in \mathbb{N}$. Wir betrachten nun den Fall $n > 0$. Durch partielle Integration erhält man

$$a_n^{(q)} \quad = \quad 2 \left[B_q(x) \frac{\sin(2\pi n x)}{2\pi n} \right]_0^1 - \frac{2}{2\pi n} \int_0^1 B'_q(x) \sin(2\pi n x) dx \tag{40}$$

[64] Vgl. z.B. [Barner, Flohr 1983] S. 450ff.

$$= -\frac{2q}{2\pi n}\int_0^1 B_{q-1}(x)\sin(2\pi nx)dx$$
$$= -\frac{q}{2\pi n}b_n^{(q-1)}. \tag{41}$$

für $q \geq 2$. Für $q = 1$ ergibt sich aus (40) wegen $B_1(x) = x - \frac{1}{2}$

$$a_n^{(1)} = 0. \tag{42}$$

für alle $n \in \mathbb{N}$. Ebenso ergibt sich

$$b_n^{(q)} = -2\left[B_q(x)\frac{\cos(2\pi nx)}{2\pi n}\right]_0^1 - \frac{2}{2\pi n}\int_0^1 B_q'(x)\cos(2\pi nx)dx$$
$$= \frac{2q}{2\pi n}\int_0^1 B_{q-1}(x)\cos(2\pi nx)dx$$
$$= \frac{q}{2\pi n}a_n^{(q-1)} \tag{43}$$

für $q \geq 2$. Für $q = 1$ dagegen

$$b_n^{(1)} = -2\left[\left(x - \frac{1}{2}\right)\frac{\cos(2\pi nx)}{2\pi n}\right]_0^1$$
$$= -\frac{2}{2\pi n}. \tag{44}$$

Dies läßt sich zusammenfassen zu

$$b_n^{(q)} = \frac{q}{2\pi n}a_n^{(q-1)} = -\frac{q(q-1)}{(2\pi n)^2}b_n^{(q-2)} \quad \text{und} \tag{45}$$
$$a_n^{(q)} = -\frac{q}{2\pi n}b_n^{(q-1)} = -\frac{q(q-1)}{(2\pi n)^2}a_n^{(q-2)}. \tag{46}$$

Substituiert man nun in diesen Formeln je einmal $q = 2k$ bzw. $q = 2k-1$, $k \in \mathbb{N}$, so ergibt sich induktiv mit dem Induktionsanfang (42) und (44):

$$a_n^{2k-1} = 0, \qquad a_n^{2k} = (-1)^k\frac{2(2k)!}{(2\pi n)^{2k}} \tag{47}$$
$$b_n^{2k-1} = (-1)^k\frac{2(2k-1)!}{(2\pi n)^{2k-1}} \qquad b_n^{2k} = 0 \tag{48}$$

für $n \in \mathbb{N}$. Damit ergibt sich als Fourier-Reihendarstellung:

$$B_{2k-1}(x - [x]) = 2(-1)^k(2k-1)!\sum_{n=1}^{\infty}\frac{\sin(2\pi nx)}{(2\pi n)^{2k-1}} \tag{49}$$
$$B_{2k}(x - [x]) = 2(-1)^{k-1}(2k)!\sum_{n=1}^{\infty}\frac{\cos(2\pi nx)}{(2\pi n)^{2k}} \tag{50}$$

Wir sehen dabei jetzt und in Folge von dem für unsere Aufgabenstellung uninteressanten Spezialfall $B_1(x)$ ab und nehmen für (49) $k \geq 2$ an.

5.3.2 Folgerungen und Abschätzungen aus der Fourier-Analyse

Es ist zunächst zu bemerken, daß sich aus (49) direkt $B_{2k-1} = B_{2k-1}(0) = 0$ ergibt: Ein Ergebnis, das sich in `BernoulliPolynomBuilder.getBernoulliNumber()` verwerten läßt. Darüber hinaus erkennt man, daß $|B_{2k}(x)|$ bei $x = 0$ ein Maximum einnimmt (also $|B_{2k}(x)| \leq |B_{2k}|$ gilt). Es folgt aus (49) ebenfalls

$$\frac{B_{2k}}{(2k)!} = \frac{2(-1)^{k-1}}{(2\pi)^{2k}} \sum_{n=1}^{\infty} \frac{1}{n^{2k}}. \tag{51}$$

Dadurch erhält man nicht nur einen algebraischen Ausdruck für $\zeta(2k)$, sondern auch die Abschätzung

$$2\frac{(2k)!}{(2\pi)^{2k}} < |B_{2k}| = 2\frac{(2k)!}{(2\pi)^{2k}} \sum_{n=1}^{\infty} \frac{1}{n^{2k}} \tag{52}$$

und mit $\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2} \geq \sum_{n=1}^{\infty} \frac{1}{n^{2k}}$ ergibt sich daraus

$$\frac{2(2k)!}{(2\pi)^{2k}} < |B_{2k}| \leq 2\frac{(2k)!}{(2\pi)^{2k}} \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{(2k)!}{12(2\pi)^{2k-2}} < \frac{4(2k)!}{(2\pi)^{2k}}. \tag{53}$$

Welche Abschätzungen ergeben sich nun für $q = 2k - 1$? Formuliert man die Fourierdarstellungen (49) und (50) für das Funktionsmaximum, so gilt für $q > 1$:

$$\max|B_q(x - [x])| \leq 2q! \sum_{n=1}^{\infty} \frac{1}{(2\pi n)^q}. \tag{54}$$

Entsprechend gilt analog den Schritten (52-53) somit:

$$\frac{2q!}{(2\pi)^q} < \max|B_q(x - [x])| < \frac{4q!}{(2\pi)^q} \quad , q > 1 \tag{55}$$

5.3.3 Verbesserte Abschätzung des Integrals

Mit den so gewonnenen Kenntnissen können wir jetzt das Integral bedeutend besser abschätzen als in Unterkapitel 5.1:

$$\begin{aligned} & \left| \frac{(3+it)\cdot(4+it)\cdots(18+it)}{17!} \int_m^{\infty} B_{17}(x - [x]) x^{-17} dx \right| \\ < & \left(\frac{10}{17!} t^{16} + 10^6 \right) \int_m^{\infty} \frac{\frac{4(17!)}{(2\pi)^{17}}}{x^{17}} dx \\ < & (6.75 \cdot 10^{-14} \cdot t^{16} + 5.91 \cdot 10^5) \cdot m^{-16}. \end{aligned} \tag{56}$$

Für einen absoluten Fehler von $\Delta\zeta(s) < 10^{-4}$ ergibt sich

$$m > 0.27 \cdot |t| + 4.08 > \sqrt[16]{6.75 \cdot 10^{-10} \cdot t^{16} + 5.91 \cdot 10^9}. \tag{57}$$

Diese Abschätzung gilt es nun zusammen mit dem neuen Berechnungsmechanismus zu implementieren.

5.3.4 Implementation

Die Implementation der neuen Berechnung gestaltet sich relativ einfach, da vollständig auf bereits vorhandene Objekte zurückgegriffen werden kann.
Dem `Complex`-Objekt wird eine Methode `zeta_fast(Complex s)` hinzugefügt, die die neuen Berechnungsschritte in das funktional-objektorientierte Schema übersetzt.
Für die durchschnittliche Berechnungszeit eines Wertes nach der neuen Methode ergibt sich eine Dauer von 0.0015 Sekunden,[65] also gegenüber der vorherigen Implementation eine Beschleunigung um mehr als den Faktor 12.[66]
Die oben errechnete Abschätzung wird durch die Praxis sehr gut verifiziert. Es zeigt sich zudem, daß sie für große t erwartungsgemäß[67] noch ein wenig zu pessimistisch ist. So ist z.B. selbst für $\zeta(\frac{1}{2} + 1000000i)$ der absolute Fehler $\Delta\zeta < 10^{-7}$.

Als Fazit läßt sich sagen, daß durch die Optimierung auf mathematischer Ebene - zumindest bei einer komplexen Darstellung wie der der ζ-Funktion - ein höheres Maß an Berechnungszeit eingespart werden kann als durch die 'konservative' Umsetzung mathematischer Formeln auf der Implementationsebene, auch wenn diese durch Wiederverwendung einen flexibleren und weitreichenderen Umgang mit mathematischen Objekten erlaubt. Computerprogramme, in denen dieser Flexibilität gegenüber der Berechnungszeit ein - notwendig - höheres Gewicht eingeräumt wird (die also unsere erste Implementationsmethode prinzipiell der zweiten vorziehen würden), sind Computer-Algebra-Systeme (CAS). Wir werden uns im folgenden Kapitel mit einem solchen genauer auseinandersetzen.

6 Vergleich des Funktionenplotters mit dem CAS MuPAD

6.1 Einführung

Computer-Algebra-Systeme sind Programme, die mathematische Ausdrücke nicht nur numerisch, sondern auch symbolisch behandeln können und damit in der Lage sind, mathematische Operationen auch abstrakt durchzuführen, so z.B. die analytische Lösung einer Differentialgleichung oder die Transformation von Matrizen und Gleichungssystemen. Um sich einem breitest möglichen Anwenderbereich offenzuhalten, müssen diese Programme über eine flexible Handhabung von mathematischen Objekten verfügen. Wir wollen - in Kenntnis der Architektur und Funktionsweise unseres Funktionenplotters - uns in diesem Kapitel exemplarisch mit dem CAS MuPAD auseinandersetzen, um dieses in Aufbau und Anwendung mit unserem Programm vergleichen zu können.

6.2 Aufbau und Funktionsweise von MuPAD

Ähnlich wie die meisten Interpreter-Systeme ist MuPAD eingeteilt in einen maschinennah programmierten Kern, der die Basisfunktionalität bereitstellt, und in eine Bibliothek, die die in der Interpretersprache geschriebenen Erweiterungen umfaßt,

[65] Um eine praktisch relevante Vergleichbarkeit zu ermöglichen, wird ein (äquidistantes) 200×200-Gitter mit Imaginärteil $98 \leq t \leq 100$ berechnet, für größere t ergibt sich natürlich eine etwas längere durchschnittliche Berechnungszeit.

[66] Weitere Optimierungen auf Implementationsebene ergeben nochmals eine Verbesserung auf eine durchschnittliche Berechnungsdauer von 0.0008 Sekunden, was z.B. bei der Berechnung eines 200×200-Gitters eine Wartezeit von 32 Sekunden zur Folge hat.

[67] Ein Blick auf Abb. 17 zeigt, daß die vorgenommene Abschätzung für sehr große t nicht optimal ist: $\left|\Pi_{\sigma=3}^{18}(s+it)\right|$ ist asymptotisch gleich t^{16}, der Tern wird aber durch $10 \cdot t^{16}$ abgeschätzt.

deren Funktionalität also bereits auf dem Kern aufbaut und von diesem umgesetzt wird.

6.2.1 Der Kern

Da MuPAD trotz seiner freien Verfügbarkeit für Lehrzwecke keine direkten Einblicke in den Quell-Code seines Kerns zuläßt, sind wir auf die technischen Beschreibungen des MuPAD-Teams angewiesen.[68] Von diesen gibt es jedoch glücklicherweise genügend, so daß im folgenden ein kurzer Überblick über die Kern-Architektur gegeben werden soll.
Der MuPAD-Kern ist in C/C++ geschrieben, wodurch ihm neben der Organisation der Basisobjekte auch deren Speicherverwaltung obliegt.[69] Für das System wurde so eine 'universell einsetzbare Speicherverwaltung' geschrieben, 'die jedoch als Grundlage zur Implementation eines Computer-Algebra-Systems nahezu optimal geeignet ist.'[70] Sieht man nun von den speichermodellabhängigen Implementationsdetails ab, so erkennt man einen ähnlichen Aufbau wie in unserem Funktionenplotter: Algebraische Ausdrücke werden in MuPAD durch n-äre Bäume ausgedrückt, wobei bei der Auswertung Werte oder Funktionen an Identifier gebunden werden. Da im Unterschied zu unserem Programm aber auch Ausdrücke durch einen Evaluierer algebraisch umgeformt bzw. vereinfacht werden sollen, sind die Berechnungsbäume wesentlich dynamischer angelegt. Bei der Evaluierung wird der Baum rekursiv durchlaufen und Unterbäume, die entweder keine Identifier enthalten oder solche, an die Werte gebunden sind, auswertet. So wird z.B. der Ausdruck `f(4+3)` $\simeq$ `(f (+ 3 4))` durch den Evaluierer auf `f(7)` $\simeq$ `(f 7)` reduziert, wenn an `f` kein Wert (also keine Funktion/Prozedur) gebunden ist. Des weiteren haben die Objekte für algebraische Objekte (z.B. Polynome) eine Normalform, in die sie der Evaluierer gegebenenfalls umformen muß. Wollte man also die Funktionalität unseres Funktionenplotters weiter in Richtung der eines echten CAS ausdehnen, so müßte man das `Calc`-Schema um weitere Elemente (z.B. Zuweisungsoperatoren, komplexere Funktionsobjekte, etwa Polynome, etc.) erweitern, und dem System einen Zustandskellerspeicher (Stack) mit den vorgenommenen Bindungen hinzufügen. Zudem müßte eine Evaluierungseinheit mit bestimmten (objektabhänigen) Regeln implementiert werden. Prinzipiell ist alles dies ohne größere Veränderung der bestehenden Objektarchitektur machbar (die Einführung der in Unterkapitel 4.2 beschriebenen Klassen ist bereits ein Schritt in diese Richtung, wenngleich auch auf einer untergeordneten, weniger abstrakten Ebene), es käme allerdings eine große Anzahl an komplexen Objektklassen hinzu.

Die Langzahlarithmetik

Der im Hinblick auf die Geschwindigkeit wichtigste Teil, die Arithmetik, verdient eine gesonderte Betrachtung. MuPAD nutzt für numerische Berechnungen das ursprünglich für Zahlentheorie-Anwendungen geschriebene Langzahlarithmetik-Paket PARI.[71] Dieses stellt Funktionalität zur Verfügung, die mit beliebig vorgegebener Genauigkeit elementare numerische Berechnungen vornimmt und dabei im Vergleich zu komplexeren Systemen durch maschinennahe Programmierung vor allem auf Geschwindigkeit optimiert ist. Die Optimierung des Paketes leitet sich dabei aus einem dreistufigen Kernkonzept ab, auf dessen unterster Stufe dem 'basic kernel', lediglich die vier arithmetischen Grundoperationen für unterschiedliche Zahlendarstellungen

[68] [Kemper 1993], [Gottheil 1993], [Morisse 1993].
[69] Letztere wird in unserem Funktionenplotter durch die Virtuelle Maschine des Java-Interpreters organisiert.
[70] [Morisse 1993] S. 11.
[71] Vgl. [Batut et al. 1995], [Batut et al. 1991], [Fuchssteiner et al. 1993] S. 523, `ftp://ftp.math.ucla.edu/pub/pari/`

sowie die Umformungen zwischen den Darstellungen implementiert sind. Auf der nächsthöheren Stufe, dem 'generic kernel', befindet sich die darauf aufbauende komplexere Funktionalität sowie weitere algebraische Typen wie Restklassen, Polynome, etc. Auf der höchsten Stufe finden sich wiederum komplexere Operationen wie z.B. Prim-Überprüfung, Faktorisierung, etc. Da die Operationen der letzten Schicht bereits in MuPAD selbst ausgeführt werden können, ist es klar, daß es von PARI lediglich den 'basic kernel' (komplett) und den 'generic kernel' (teilweise) übernimmt.[72] Im Gegensatz zu MuPAD ist der Quellcode von PARI frei verfügbar, so daß sich in diesem Paket detaillierte Untersuchungen zur numerischen Optimierung vornehmen lassen.

6.2.2 Die Bibliothek

Sieht man mit den Betrachtungen im vorangehenden Kapitel den Unterschied zwischen dem Funktionenplotter und einem CAS vor allem in der von ihnen verarbeiteten unterschiedlich mächtigen Sprache (und damit in der Komplexität ihrer Grammatik und Semantik), so zeigt sich dies auch darin, daß ein Großteil der Funktionalität von MuPAD bereits in der Interpretersprache implementiert ist. Dies hat den Vorteil, daß die mathematischen Funktionen - sofern sie komplett in der Bibliothek definiert sind - unabhängig von der verwendeten Hardware- und Betriebssystemplattform sind. Der damit auf der Hand liegende Nachteil ist die niedrigere Verabeitungsgeschwindigkeit.

6.2.3 Funktionsweise von MuPAD

Nach diesem kurzen Überblick über den Aufbau von Kern und Bibliothek, wollen wir nun ihr Zusammenspiel betrachten. Exemplarisch soll dies erneut für die für unsere Belange interessante Berechnung der ζ-Funktion geschehen.

Beispiel: Die Berechnung der Riemannschen ζ-Funktion

Ruft man vom MuPAD-Kommandozeilenprompt die ζ-Funktion auf - beispielsweise durch das Kommando

■ `zeta(0.5 + 20*I);`

- so sucht der Interpreter zunächst in der Umgebung nach dem Symbol '`zeta`', um dieses zu evaluieren. Er wird dabei in dem durch die Standardbibliothek erzeugten Modul `stdlib` fündig und ruft daraufhin `stdlib::zeta(0.5 + 20*I)` auf. Der Interpretersprachen-Quelltext zu diesem Modul ist im wesentlichen der Folgende (deutsche Kommentare von mir eingefügt):

```
zeta := proc(x)

begin
    if x::zeta <> FAIL then return(x::zeta(args())) end_if;

    if args(0) <> 1 then
       error("1 argument expected.");
    end_if;

    # Überprüfung des Argumenttyps #
    case domtype(x)

      ...
```

[72] Dies gilt insbesondere für die (in unserem Fall interessanten) zahlentheoretischen Funktionen.

```
      # Test auf Ganzzahligkeit des Arguments, Behandlung von
        Spezialfällen #
      of DOM_INT do
        if x < 0 then
           if x mod 2 = 0 then
             return ( 0 )
           elif -x < 100 then # See apendix #
             return( -bernoulli(1-x) / (1-x) )
           end_if
        elif x > 0 then
           if x mod 2 = 0 and x <= 100 then # See apendix #
             return( 1/2*(2*PI)^x*abs(bernoulli(x))/fact(x) )
           elif x = 1 then
             error("singularity encountered.")
           end_if
        else
           return( -1/2 )
        end_if;

        break;


      # Aufruf der Kernelfunktion für komplexe Argumente #
      of DOM_COMPLEX do
        if domtype(op(x, 1)) = DOM_FLOAT or
          domtype(op(x, 2)) = DOM_FLOAT then
           return( funcattr(zeta, "float")(x) );
        end_if;
        break;

     end_case;

     procname(x)
end_proc:

...
```

Man sieht, daß das Argument s der Funktion auf verschiedene Spezialfälle hin überprüft wird: Für $s = 2n + 1$, $n \in \mathbb{N}$, würde z.B. Beziehung (51) ausgenutzt und der entsprechende analytische Ausdruck in ausgewerteter Form zurückgegeben. Erst wenn festgestellt ist, daß s (wie in unserem Beispiel) nicht durch einen Spezialfall abgedeckt wird, wird das Argument an die darunterliegende Kern-Funktion weitergereicht. Was dort passiert, läßt sich zwar zunächst nur vermuten, allerdings ist es höchst wahrscheinlich, daß der Kern im wesentlichen nur die Funktion `gzeta()` im PARI-Paket aufruft, die ihrerseits nach Überprüfung des Arguments die Funktion `czeta()` aufruft. Dort wird die eigentliche Berechnung vorgenommen. Eine Betrachtung des Quellcodes ergibt, daß PARI die Zeta-Funktion ebenfalls mittels der Euler-MacLaurin-Summenformel berechnet, allerdings wird die Abschätzung für die obere Summationsgröße m - im folgenden Quellcode in der Variable `n` abgebildet - bedeutend aufwendiger (und für mich nicht in verständlicher Weise) vorgenommen. Die wesentlichen Bestandteile des Berechnungsabschnittes sind jedoch besser

nachvollziehbar:[73]

```
0 GEN czeta(GEN s, long prec){
...
1   y=gun;ms=gneg(s);
2   for(i=2;i<=n;i++)
3     y=gadd(y,p2=gexp(gmul(ms,glog(stoi(i),prec+1)),prec+1));
4   flag3=((2*p)<46340);
5   mpbern(p,prec+1);p31=cgetr(prec+1);
6   z=gzero;
7   for(i=p;i>=1;i--)
8   {
9     i2=i<<1;
10    p1=gmul(gaddsg(i2-1,s),gaddsg(i2,s));
11    p1=flag3 ? gdivgs(p1,i2*(i2+1)) : gdivgs(gdivgs(p1,i2),i2+1);
12    p1=flag2 ? gdivgs(p1,n1) : gdivgs(gdivgs(p1,n),n);
13    if(bernzone[2]>prec+1) {affrr(bern(i),p31);p3=p31;}
14    else p3=(GEN)bern(i);
15    z=gadd(divrs(p3,i),gmul(p1,z));
16  }
17  z1=gsub(gdivsg(n,gsubgs(s,1)),ghalf);
18  z=gmul(gadd(z1,gmul(s,gdivgs(z,n<<1))),p2);
19  if(!flag1) {gaffect(gadd(y,z),res);avma=av;}
...
20  return res;
```

Sieht man von den Flag-Variablen ab, die lediglich numerischen Optimierungen dienen, so läßt sich der Code folgendermaßen beschreiben: Zuerst (1-3) nimmt das Programm die Berechnung der Summe $y = \sum_{n=1}^{m} n^{-s}$ vor, wobei m^{-s} für spätere Zwecke festgehalten wird. Daraufhin (5) werden die benötigten Bernoullizahlen mit der geforderten Genauigkeit (die durch den Wert von `prec` gegeben ist) einmalig berechnet und in der Folge lediglich neu ausgelesen. In der sich anschließenden Schleife (7-16) wird der Wert z von `z` iterativ gebildet:

$$\begin{aligned} z &= \frac{(s+1)(s+2)}{2\cdot 3}m^{-2}\left(\frac{(s+3)(s+4)}{4\cdot 5}m^{-2}\cdots\left(0+\frac{2B_q}{q}\right)\cdots+\frac{B_4}{2}\right)+\frac{B_2}{1} \\ &= \sum_{r=2}^{q}\frac{2B_r}{r!}(s+1)\cdots(s+r-2)m^{-r+2}. \end{aligned}$$

Mit $q := 2p$, wobei der zuvor (von m unabhängig) abgeschätzte Wert von p in der Variablen `p` festgehalten wird. In den folgenden Zeilen (17-18) wird `z` der Wert

$$\begin{aligned} z' &= m^{-s}\left(\frac{m}{s-1}-\frac{1}{2}+\frac{sz}{2m}\right) \\ &= \frac{m^{-s+1}}{s-1}-\frac{m^{-s}}{2}+\sum_{r=2}^{q}\frac{B_r}{r!}s(s+1)\cdots(s+r-2)m^{-s-r+1} \end{aligned}$$

zugewiesen und in der Ergebnisvariable `res` der Wert auf die Summe aus z' und y gesetzt (19). Man sieht, daß das Resultat mit dem der Berechnung in Kapitel 5.2

[73] Zu dem verwendeten Datentyp `GEN` und den auf ihm operierenden Funktionen s. [Batut et al. 1995] S. 89-108.

übereinstimmt, wenngleich hier ein Algorithmus für eine andere Umsetzung als in unserem Funktionenplotter sorgt.[74]

6.2.4 Das Graphikpaket von MuPAD

Das MuPAD-Paket verfügt wie die meisten Computer-Algebra-Systeme über ein flexibles und leistungsfähiges Paket zur Darstellung von verschiedenartigen Graphiken. Dieses Paket besteht zum einen aus der Bibliothek, `plotlib`[75], die mittels der MuPAD-Berechnungsfunktionen die zu zeichnenden Knotenpunkte berechnet, und zum anderen aus einer Darstellungseinheit VCam, die die flexible Betrachtung der generierten Objekte erlaubt. Dabei wird ebenso wie in unserem Funktionenplotter auf eine strikte Trennung von Berechnung und Darstellung geachtet: Die Knoten werden erst komplett berechnet und in einem Binärformat (das auch in einer Datei gespeichert werden kann) abgelegt, dann erst wird das Betrachtungsprogramm mit diesen Daten als Parameter aufgerufen.

6.3 Geschwindigkeitsvergleich zwischen MuPAD und dem Funktionenplotter

Selbstverständlich ist ein Geschwindigkeitsvergleich zwischen einem spezialisiertem Programm, wie unserem Funktionenplotter und einem 'general-purpose'-Programm, welches ein CAS darstellt, ein Vergleich zwischen Äpfeln und Birnen. Im Hinblick auf die dieser Arbeit zugrundeliegende Thematik der Visualisierung komplexer Funktionen erscheint eine Gegenüberstellung der Berechnungs- und Darstellungszeiten dennoch sinnvoll, da die für unseren Funktionenplotter spezifizierte Funktionalität bislang von einem CAS übernommen werden mußte. Der Vorteil der Spezialisierung liegt dabei in unserem Fall - wie die folgenden Daten belegen werden - vor allem in einer Geschwindigkeitsoptimierung zugunsten von Flexibilität und hoher numerischer Genauigkeit. Um in Hinblick auf die im Exkurs von 2.1 beschriebenen möglichen Untersuchungen einen einigermaßen repräsentativen Anwendungsquerschnitt zu bemessen, wurde zunächst der Wert von $\zeta(0.5 + 1000000i)$ jeweils auf dem Referenzrechner (s.o.) zuerst mittels `calc.Complex`, dann durch MuPAD und, exemplarisch für ein weiteres CAS, durch Maple berechnet. Als vorgegebene Genauigkeit wurden 5 Stellen angegeben. Es ergaben sich dabei folgende Werte:[76]

Programm	t
MuPAD 1.4.1	37 min 17 s
Maple VR5	33 min 12 s
Funktionenplotter	9 s

Eine probehalber zusätzlich durchgeführte Messung auf der von PARI mitgelieferten Benutzerschnittstelle GP ergab eine Berechnungsdauer von ca. 30 Minuten, allerdings mußte dabei mehrfach von Hand die Größe des Objekt-Kellers (Stack) verdoppelt werden und die Berechnung erneut von vorne durchgeführt werden. Es ist also zu vermuten, daß MuPAD die Stack-Vergrößerung selbst vornimmt, und durch die Durchführung dieser Aktionen die Mehrzeit gegenüber dem PARI-Paket

[74] Eine Implementation nach diesem Algorithmus im Funktionenplotter ergab eine erneute Geschwindigkeitsverbesserung auf eine durchschnittliche Berechnungsdauer von ca. 0.0005 Sekunden, also 20 Sekunden für ein 200×200-Wertegitter.

[75] Vgl. [Schulze 95].

[76] Die Berechnung mittels des Funktionenplotters wurde mit einer initialen Stack-Größe von 64 MB ('`java -Xms64m calc.Complex 0.5 1000000 0 0`') durchgeführt. In MuPAD und Maple sind mir keine vergleichbaren Optimierungsmöglichkeiten bekannt.

entsteht. Über den in Maple verwendeten Berechnungsmechanismus ist mir zwar nichts bekannt, die von PARI benötigte Zeit dürfte aber bei der Verwendung des flexiblen Langzahlarithmetikkonzeptes eine Art untere Grenze darstellen. Da der Funktionenplotter hingegen nur mit fester `double`-Präzision rechnet, ensteht ein sehr viel geringerer Speicherverwaltungsaufwand, der sich in einer über 200 mal schnelleren Berechnung niederschlägt.[77]

Als nächstes wurde, um die für einen typischen Zeichenvorgang dauernde Berechnungszeit zu evaluieren, ein Gitter aus 200×200 Werten im Bereich $[0,3] \times [0,100]i$ berechnet. Dabei wurde für den Funktionenplotter eine Dauer von 20 Sekunden ermittelt, MuPAD benötigte 3 Stunden und 47 Minuten. Da letzteres für einen Benutzer, der graphisch orientiert arbeiten will, nicht akzeptabel ist (auch z.B. die in Abb. 1 gezeigte Graphik wurde in über einer Stunde berechnet), stellt die Verwendung des Funktionenplotters für die in 2.1 skizzierte Vorgehensweise einen echten Fortschritt dar.

6.4 Ausblick: Interoperabilität

Um eine Integration des Funktionenplotters in bereits bestehende Systeme zu erleichtern,[78] soll nun zum Schluß noch ein kurzer Überblick über die Möglichkeiten zur Interoperabilität geliefert werden, dies soll wiederum exemplarisch im Blick auf die Zusammenarbeit mit MuPAD geschehen.
MuPAD verfügt zusätzlich zu der in der UNIX-Welt 'normalen' Kommunikation zwischen Programmen über Ein- und Ausgabedatenströme[79] auch über eine differenziertere Möglichkeit der Interoperabilität: Die Rede ist von dynamischen Modulen, die über den Mechanismus des Bindens zur Laufzeit ('dynamic binding') es einem Programmierer erlaubt, vollen Zugriff auf die MuPAD-Objekte und Methoden zu erlangen.[80] Der Vorteil dieses Systems liegt vor allem darin, daß Daten nicht aufwendig kopiert werden müssen, sondern eine Übergabe per Referenz möglich ist. Da unser Funktionenplotter in reinem Java geschrieben ist, stellt sich eine Verwendung dieser Technologie zunächst als schwierig dar. Denkbar wäre jedoch die Benutzung des Java Native Interface (JNI), einer Kommunikationsschnittstelle zwischen Java und C/C++,[81] um über eine mit MuPAD gemeinsame C++-Schnittstelle Daten auszutauschen, bzw. Methodenaufrufe vorzunehmen. Damit würde aber dem plattformübergreifenden Ansatz von Java eine maschinenabhängige Komponente beigefügt, ein Nachteil allerdings, der aufgrund der Plattformgebundenheit des MuPAD-Kernes (im Gegensatz zur Bibliothek) für Kommunikation auf der MuPAD-Objekt-Ebene nicht zu umgehen ist. Die plattformunabhängige Alternative müßte sich hingegen auf Kommunikation ohne einen gemeinsam genutzten Hauptspeicher (z.B. über Dateien, Sockets, etc.) beschränken.
Für die plattformunabhängige Variante hingegen wäre zwar ein Kopieren von Daten notwendig, allerdings wäre dies unter Verwendung einer skalierbaren Architektur, die die beliebige Verteilung von Diensten auf verschiedene Rechner erlaubt, ohnehin notwendig. Bemerkenswert scheinen in diesem Zusammenhang die auf verschiedenen

[77] Die numerische Ungenauigkeit aufgrund von sich fortpflanzenden Abbruchfehlern wurde dabei in einem Bereich jenseits der 12. Nachkommastelle beobachtet, sie dürfte für unsere Zwecke vernachlässigenswert sein, wenngleich auch keine beweisbare numerische Abschätzung angegeben werden kann.

[78] Es sind hierfür viele Szenarien möglich, denkbar wäre z.B. eines, in dem die Berechnung eines größeren Wertebereichs von unserem Funktionenplotter, die genauere, numerische Untersuchung eines kleineren Bereichs darauf basierend von MuPAD vorgenommen wird.

[79] Eine Verallgemeinerung dieses Konzeptes faßt man im allgemeinen unter dem Stichwort Interprozeß-Kommunikation (IPC) zusammen. Vgl. z.B. [Sorgatz 1997].

[80] Vgl. [Sorgatz 1998], praktisches Beispiel in [Sorgatz 1997].

[81] Vgl. [Sun 1997].

OSI-Schichten[82] arbeitenden Austauschprotokolle. Neben den üblichen standardisierten Kommunikationsprotokollen TCP/IP, HTTP und IIOP[83] gibt es hier bereits auf mathematische Anwendungen spezialisierte Protokolle (deren Standardisierung zur Zeit jedoch noch nicht abgeschlossen ist), z.B. MP[84] und OpenMath[85]. Im Sinne einer Interoperabilisierung unseres Funktionenplotters wäre es nun denkbar, eine Protokollschnittstelle zu erstellen, also ein Objekt, dessen Aufgabe in der Konvertierung von Java-Konstrukten in die jeweiligen Protokolldaten besteht. Die in unserem Programm verwendeten Datenstrukturen bauen im wesentlichen auf Syntaxbäumen, mathematischen Funktionen und dem IEEE-`double`-Datentyp auf. Diese sind aber bereits hinreichend plattformunabhängige Konzepte (und z.B. in OpenMath und MP direkt implementiert), so daß eine solche Schnittstelle mit geringem Aufwand zu programmieren wäre.

[82] Zum OSI/ISO-Schichtenmodell vgl. [San93] S. 20ff.

[83] Vgl. [Paehler 1998], [Paehler 1998b]

[84] `http://symbolicnet.mcs.kent.edu/areas/protocols/mp.html`
In [Bachmann 1998] wird die Implementation einer Verbindung von MuPAD und dem auf Polynomberechnungen spezialisierten CAS Singular durch MP beschrieben.

[85] `http://www.openmath.org`

Literatur

[Bachmann 1998] Olaf Bachmann, Hans Schönemann, Andreas Sorgatz. Connecting MuPAD and Singular with MP. mathPAD Band 8 Ausgabe 1. GHS Paderborn 1998.
`ftp://ftp.mupad.de/MuPAD/mathpad/`

[Barner, Flohr 1983] Martin Barner, Friedrich Flohr. Analysis I. Walter de Gruyter ^{2}1983.

[Batut et al. 1991] C. Batut, D. Bernardi, H. Cohen, M. Olivier, N.-P. Skoruppa. The Pari-GP package. mathPAD Band 1 Ausgabe 3. GHS Paderborn 1991.
`ftp://ftp.mupad.de/MuPAD/mathpad/`

[Batut et al. 1995] C. Batut, D. Bernardi, H. Cohen, M. Olivier. User's Guide to PARI-GP. Elektronische Version enthalten in der Distribution erhältlich unter
`ftp://ftp.math.ucla.edu/pub/pari/unix/pari-1.39.03.tar.gz`.

[Buschmann et al.] Frank Buschmann et al. Pattern-orientierte Software-Architektur - Ein Pattern-System. Addison-Wesley 1998.

[Dress, Jäger 1999] Andreas Dress, Gottfried Jäger (Hrsg.). Visualisierung in Mathematik, Technik und Kunst - Grundlagen und Anwendungen. Vieweg 1999.

[Eckel 1998] Bruce Eckel. Thinking in Java. Prentice Hall 1998.
`http://www.bruceeckel.com/javabook.html`

[Fischer/Lieb 1994] Wolfgang Fischer, Ingo Lieb Einführung in die Funktionentheorie. Vieweg 1994.

[Flanagan 1998] David Flanagan. Java in a Nutshell. Deutsche Ausgabe der 2. Auflage. O'Reilly^{2}1998.

[Fowler, Scott 1998] Martin Fowler, Kendall Scott. UML konzentriert - Die neue Standard-Objektmodellierungssprache anwenden. Addison Wesley 1998.

[Franke 1999] Herbert W. Franke. Schnittstelle Mathematik/Kunst.
In: [Dress, Jäger 1999] S. 3-21.

[Freitag, Busam 1995] Eberhard Freitag, Rolf Busam. Funktionentheorie. Springer 1995.

[Fuchssteiner et al. 1993] B. Fuchssteiner, K. Gottheil, A. Kemper, O. Kluge, K. Morisse, H. Naundorf, G. Oevel, T. Schulze, W. Wiwianka. Mupad Benutzerhandbuch. Birkhäuser Verlag 1993. (HyTeX-Version als Online-Hilfe in jeder MuPAD-Distribution enthalten.)

[Führer 1997] Lutz Führer. Pädagogik des Mathematikunterrichts - Eine Einführung in die Fachdidaktik für Sekundarstufen. Vieweg 1997.

[Gericke 1992] Helmuth Gericke. Mathematik in Antike und Orient. Mathematik im Abendland - von den römischen Feldmessern bis zu Descartes. Fourier 1992.

[Gibson 1973] James J. Gibson. Die Sinne und der Prozeß der Wahrnehmung. Verlag Hans Huber 1973.

[Gottheil 1993] Klaus Gottheil. Evaluierung in MuPAD. mathPAD Band 3 Ausgabe 1. GHS Paderborn 1993.
`ftp://ftp.mupad.de/MuPAD/mathpad/`

[Gumm, Sommer 1998] Heinz-Peter Gumm, Manfred Sommer. Einführung in die Informatik. Oldenbourg 1998.

[Herz 1996] Andreas Herz. Repetitorium Funktionentheorie. Vieweg 1996.

[Hofstadter 1991] Douglas R. Hofstadter. Gödel Escher Bach - Ein endlos geflochtenes Band. Deutscher Taschenbuch Verlag 1991.

[Knopp 1996] Konrad Knopp. Theorie und Anwendung der unendlichen Reihen. Springer 1996.

[Meyer, Jank 1991] Hilbert Meyer, Werner Jank. Didaktische Modelle. Cornelsen Scriptor 1991.

[Kebeck 1997] Günther Kebeck. Wahrnehmung - Theorien, Methoden und Forschungsergebnisse der Wahrnehmungspsychologie. Juventa ^{2}1997.

[Kemper 1993] Andreas Kemper. Entwurf und Implementation des MuPAD Kernes. mathPAD Band 3 Ausgabe 1. GHS Paderborn 1993.
`ftp://ftp.mupad.de/MuPAD/mathpad/`

[Koecher, Krieg 1993] Max Kocher, A. Krieg. Ebene Geometrie. Springer 1993.

[Morisse 1993] Karsten Morisse. Datenstrukturen und Speicherverwaltung in MuPAD. mathPAD Band 3 Ausgabe 2. GHS Paderborn 1993.
`ftp://ftp.mupad.de/MuPAD/mathpad/`

[MT 1999] The MacTutor History of Mathematics archive.
`http://www-groups.dcs.st-and.ac.uk:80/~history/` 1999.

[Muckenfuß 1995] Heinz Muckenfuß. Lernen im sinnstiftenden Kontext. Entwurf einer zeitgemäßen Didaktik des Physikunterrichts. Cornelsen 1995.

[Lorenz 1992] Falko Lorenz. Algebra I. B.I. Wissenschaftsverlag ^{2}1992.

[Otte 1994] Michael Otte. Das Formale, das Soziale und das Subjektive - Eine Einführung in die Philosophie und Didaktik der Mathematik. Suhrkamp 1994.

[Paehler 1998] Tim Paehler. Arbeiten mit CORBA.
`http://wwwmath.uni-muenster.de/~paehler/projekte/corba` 1998.

[Paehler 1998b] Tim Paehler. Sicherheitsmechanismen und Proxifizierung in Verteilten Systemen. Vortrag zum Seminar 'Verteilte Programmierung' WS 1997/1998.
`http://wwwmath.uni-muenster.de/~paehler/projekte/insects/security/`

[Rademacher 1973] Hans Rademacher. Topics in Analytic Number Theory. Springer 1973.

[Remmert 1992] Reinhold Remmert. Funktionentheorie 1. Springer ^{3}1992.

[Rock 1985] Irvin Rock. Wahrnehmung - Vom visuellen Reiz zum Sehen und Erkennen. Spektrum Verlag 1985.

[Roulo 1998] Marc Roulo. Accelerate your Java apps! In: JavaWorld 9/98.
`http://www.javaworld.com/jw-09-1998/jw-09-speed.html`

[San93] Michael Santifaller. TCP/IP und ONC/NFS in Theorie und Praxis. Addison-Wesley 1993.

[Schirra, Strothotte 1999] Jörg R. J. Schirra, Thomas Strothotte. Von Bildern und neuen Ingenieuren. Aspekte eines Studiengangs Computervisualistik. In: [Dress, Jäger 1999] S. 189-205.

[Schlüter 1998] Oliver Schlüter: VRML - Sprachmerkmale, Anwendungen und Perspektiven. O'Reilly Essentials 1998.

[Schulze 95] Thorsten Schulze. Plot-Library. Autmath Technical Report No 6. 1995.
`http://www.sciface.com/support/papers/PS/plotlib.ps.gz`

[Sorgatz 1997] Andreas Sorgatz. Dynamic Modules - A Flexible and Very Efficient Concept to Integrate Software Packages into Computer Algebra Systems. GHS Paderborn 1997.
`http://www.sciface.com/support/papers/PS/modules.ps.gz`

[Sorgatz 1998] Andreas Sorgatz. Dynamische Module - Ein Konzept zur Effizienten Software Integration im General Purpose Computer Algebra System MuPAD.GHS Paderborn 1997.
`http://www.mupad.de/BIB/ONLINE/sorsfb98.ps.gz`

[Stoer 1993] Josef Stoer. Numerische Mathematik 1. Springer ^{6}1993.

[Sun 1997] Sun Microsystems. Java Native Interface Specification.
`ftp://ftp.javasoft.com/docs/jdk1.1/jni.pdf` 1997.

[Sun 1999] Sun Microsystems. Programmers Guide to the Java^{TM} 2D API - Enhanced Graphics und Imaging for Java.
`http://java.sun.com/products/jdk/1.2/docs/guide/2d/index.html` 1999.

[Sun 1999b] Sun Microsystems. Java 3D^{TM} API Specification.
`http://java.sun.com/products/java-media/3D/index.html` 1999.

[Tietze et al. 1997] Uwe-Peter Tietze, Manfred Klika, Hans Wolpers. Mathematikunterricht in der Sekundarstufe II. Band 1: Fachdidaktische Grundfragen - Didaktik der Analysis. Vieweg 1997.

[Vogel 1995] Helmut Vogel. Gerthsen Physik. Springer ^{18}1995.

[Wagenschein 1999] Martin Wagenschein. Verstehen lehren - Genetisch - Sokratisch - Exemplarisch. Beltz 1999.

[Weisstein 1998] Eric Weisstein. CRC Concise Encyclopedia of Mathematics. CRC Press 1998. `http://www.treasure-troves.com/math/pad`

[Weizsäcker 1993] Carl Friedrich v. Weizsäcker. Der Mensch in seiner Geschichte. DTV 1993.

[Wirtz 1998] Guido Wirtz. Skript zur Vorlesung Praktische Informatik WS 1997/1998.
`http://wwwmath.uni-muenster.de/cs/u/versys/courses/WiSe9798/InfoI/index.html`

[Wolfart 1996] Jürgen Wolfahrt. Einführung in die Zahlentheorie und Algebra. Vieweg 1996.

Anhang

A Erklärung

Ich versichere, daß ich die schriftliche Hausarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen wurden, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Das gleiche gilt auch für die beigegebenen Zeichnungen, Kartenskizzen und Darstellungen.