# Inhaltsverzeichnis

# 1. Einleitung

Um den Begriff Computeralgebra zu erklären, möchten wir die Berechnungen in der Computeralgebra mit numerischen Rechnungen vergleichen. Beide werden durch einen Computer unterstützt, doch gibt es grundlegende Unterschiede, die wir im folgenden erläutern wollen.

## 1.1 Numerische Berechnungen

In numerischen Rechnungen wird ein mathematisches Problem näherungsweise gelöst, die Rechenschritte finden mit *Zahlen* statt. Diese Zahlen sind intern in *Gleitpunktdarstellung* gespeichert, wodurch arithmetische Operationen schnell ausgeführt werden können. Diese Darstellung hat allerdings den Nachteil, dass sowohl die Berechnungen als auch die Lösungen nicht exakt sind, da es u. A. durch Rundungen zu Fehlern kommt. Numerische Algorithmen sind in der Regel so konstruiert, dass sie möglichst schnell eine Näherungslösung liefern. Näherungen sind oftmals die einzige Möglichkeit, eine mathematische Aufgabenstellung zu bearbeiten, wenn nämlich eine exakte Lösung in geschlossener Form nicht existiert. Außerdem sind Näherungslösungen dort nützlich, wo exakte Resultate gar nicht benötigt werden (z. B. bei der Visualisierung).

## 1.2 Computeralgebra

Im Gegensatz zu numerischen Berechnungen mit Zahlen werden in der Computeralgebra *symbolische* Berechnungen durchgeführt, es handelt sich gemäß [Hec 93] um *„Berechnungen mit mathematischen Objekten“*. Ein *Objekt* kann z. B. eine Zahl, aber auch ein Polynom, eine Gleichung, ein Ausdruck oder eine Formel, eine Funktion, eine Gruppe, ein Ring oder ein beliebiges anderes mathematisches Objekt sein. Symbolische Berechnungen mit Zahlen werden

im Gegensatz zu numerischen Berechnungen immer *exakt* durchgeführt, da intern eine genaue Darstellung von beliebig langen ganzen und rationalen Zahlen verwendet wird. Man nennt solche exakten Berechnungen in der Computeralgebra *symbolische* und *algebraische* Berechnungen. In [Hec 93] wird dafür die folgende Definition gegeben:

1. „Symbolisch“ bedeutet, dass es das Ziel ist, eine möglichst geschlossene Form einer Lösung in einer guten (d. h. einfachen) symbolischen Darstellung zu finden.
2. „Algebraisch“ steht für *exakte* Berechnungen im Gegensatz zu den Näherungslösungen, die auf Gleitpunktarithmetik beruhen.

Manchmal wird Computeralgebra auch mit „symbolischer Manipulation“ oder „Formelmanipulation“ erklärt, da mit Formeln und Symbolen gerechnet wird. Beispiele dafür sind die symbolische Integration oder die Differentiation wie

$$\int x\,\mathrm{d}x = \frac{x^2}{2}, \quad \int_1^4 x\,\mathrm{d}x = \frac{15}{2}, \quad \frac{\mathrm{d}}{\mathrm{d}x}\ln\ln x = \frac{1}{x\ln x}$$

oder die Berechnung symbolischer Lösungen von Gleichungen. Als Beispiel sei hier die Gleichung $x^4+p\,x^2+1=0$ in $x$ mit einem Parameter $p$ betrachtet, die die Lösungsmenge

$$\left\{\pm\frac{\sqrt{2}\cdot\sqrt{-p-\sqrt{p^2-4}}}{2},\ \pm\frac{\sqrt{2}\cdot\sqrt{-p+\sqrt{p^2-4}}}{2}\right\}$$

besitzt. Für die symbolische Berechnung einer exakten Lösung wird fast immer mehr Rechenzeit und mehr Hauptspeicher benötigt als für die Berechnung einer numerischen Lösung. Aber eine symbolische Lösung ist exakt, allgemeiner und liefert meist weitere Informationen zu dem Problem und seiner Lösung. Betrachten wir z. B. die obige Lösungsformel, die eine Lösung der Gleichung für beliebige Werte des Parameters $p$ liefert: Sie zeigt die funktionale Abhängigkeit von $p$. Damit kann beispielsweise ermittelt werden, wie empfindlich die Lösungen gegen Änderungen des Parameters sind.

Für spezielle Anwendungen sind Kombinationen von symbolischen und numerischen Methoden sinnvoll. Es gibt z. B. Algorithmen in der Computeralgebra, die von der effizienten Gleitpunktarithmetik der Hardware profitieren. Auf der anderen Seite kann es sinnvoll sein, ein Problem aus der Numerik zunächst symbolisch zu vereinfachen, bevor der eigentliche approximative Algorithmus angewendet wird.

## 1.3 Eigenschaften von Computeralgebra-Systemen

Die meisten der bekannten Computeralgebra-Systeme sind interaktiv zu benutzende Programmpakete: Der Benutzer gibt dem System eine Reihe von Formeln und Befehlen, die dann vom System bearbeitet (man sagt auch, *ausgewertet*) werden. Das System gibt anschließend eine Antwort zurück, die weiter manipuliert werden kann.

Zusätzlich zu exakten symbolischen Berechnungen können die meisten Computeralgebra-Systeme Lösungen auch numerisch approximieren. Dabei kann die Genauigkeit vom Benutzer auf eine beliebige Anzahl von Stellen vorgegeben werden. In MuPAD geschieht dies durch die globale Variable `DIGITS`. Beispielsweise wird mit dem einfachen Befehl `DIGITS:=100` erreicht, dass MuPAD Gleitpunktberechnungen mit einer Genauigkeit von 100 Dezimalstellen ausführt. Natürlich benötigen solche Berechnungen mehr Rechenzeit und mehr Hauptspeicher als das Benutzen der Gleitpunktarithmetik der Hardware.

Moderne Computeralgebra-Systeme stellen zusätzlich noch eine mächtige Programmiersprache[1] zur Verfügung und bieten Werkzeuge zur Visualisierung und Animation mathematischer Daten. Auch bieten viele Systeme die Möglichkeit zur Vorbereitung druckfertiger Dokumente (so genannte *Notebooks* oder *Worksheets*). Auch in MuPAD existiert ein Notebook-Konzept, welches in dieser Einführung allerdings nicht behandelt werden soll. Das Ziel dieses Buches ist es, eine Einführung in die Benutzung der *mathematischen* Fähigkeiten MuPADs zu geben.

## 1.4 Existierende Systeme

Es gibt viele verschiedene Computeralgebra-Systeme, von denen einige kommerziell vertrieben werden, während andere frei erhältlich sind.

So genannte *special purpose* Systeme dienen zur Behandlung von speziellen mathematischen Problemen. So gibt es das System *Schoonship* für Probleme in der Hochenergiephysik, *DELiA* zur Behandlung von Differentialgleichungen, *PARI* für Anwendungen in der Zahlentheorie[2] und *GAP* für Probleme aus der Gruppentheorie.

---

[1] Die Programmiersprache von MuPAD besitzt eine ähnliche Syntax wie Pascal, wobei Konstrukte für objektorientierte Programmierung zur Verfügung gestellt werden.

[2] Teile dieses Systems werden intern von MuPAD verwendet.

Daneben gibt es so genannte *general purpose* Computeralgebra-Systeme. Dazu gehören das seit 1980 entwickelte und speziell für Kleincomputer ausgelegte *Derive* sowie *MathView* (ehemals *Theorist*), das seit 1990 entwickelt wird und eine ausgefeilte Benutzungsoberfläche, aber nur eingeschränkte mathematische Fähigkeiten besitzt. Außerdem gibt es die Systeme *Macsyma* und *Reduce*, beide seit 1965 entwickelt und in LISP programmiert. *Maxima* ist eine Abspaltung aus dem originalen *Macsyma*, seit 1998 unter eigenem Namen vertrieben. Modernere Systeme wie *Mathematica* und *Maple* befinden sich seit etwa 1980 in Entwicklung und sind in C programmiert. *Mathematica* war das erste System mit einer benutzerfreundlichen Oberfläche. Weiterhin ist *Axiom* zu erwähnen, das ebenfalls seit etwa 1980 entwickelt wird. Im Gegensatz zu den bereits genannten Systemen verfügt *Axiom* über eine komplett typisierte Sprache und lässt Berechnungen nur in speziellen mathematischen Kontexten zu. Unter allen diesen Systemen ist MuPAD das jüngste: Es wird seit 1990 an der Universität Paderborn entwickelt und versucht, die Stärken verschiedener Vorläufer mit modernen, eigenen Konzepten zu verbinden.

## 1.5 MuPAD

Zusätzlich zu den bereits genannten Eigenschaften von Computeralgebra-Systemen hat MuPAD die folgenden Fähigkeiten (die in diesem Buch allerdings nicht näher behandelt werden):

- MuPAD bietet Sprachkonstrukte zum objektorientierten Programmieren. Man kann eigene Datentypen definieren und fast alle Operatoren und Funktionen zu deren Behandlung überladen.
- MuPAD stellt einen interaktiven Quellcode-Debugger zur Verfügung.
- Mit dem Modulkonzept kann man in C oder C++ geschriebene Programme zum MuPAD-Kern hinzufügen.

Das Herzstück von MuPAD ist der so genannte *Kern*, der aus Effizienzgründen im Wesentlichen in C und teilweise in C++ implementiert ist. Dieser Kern wiederum besteht aus den folgenden grundlegenden Teilen:

– Der so genannte *Parser* liest die Eingaben an das System und überprüft sie auf richtige Syntax. Eine fehlerfreie Eingabe wird vom Parser in einen MuPAD-Datentyp umgewandelt.

– Der so genannte *Auswerter* (*Evaluierer*, englisch: *evaluator*) wertet die Eingaben aus und vereinfacht die Ergebnisse. Dieser Vorgang ist in MuPAD genau definiert und wird später näher erläutert.

– Die *Speicherverwaltung* ist für die interne Verwaltung der MuPAD-Objekte zuständig.

– Einige oft benötigte Algorithmen wie z. B. die arithmetischen Funktionen sind aus Effizienzgründen als *Kernfunktionen* auf C-Ebene implementiert.

Parser und Evaluierer definieren die MuPAD-Programmiersprache. Mit Hilfe dieser Sprache sind die zu MuPAD gehörenden Programmbibliotheken implementiert, die das mathematische Wissen des Systems enthalten.

Daneben besitzt MuPAD komfortable Benutzungsoberflächen zur Erzeugung so genannter *Notebooks* oder von Graphiken oder zum Debuggen in der MuPAD-Sprache geschriebener Programme. Das MuPAD-Hilfesystem hat Hypertextfunktionalität. Man kann in Dokumenten navigieren, aber auch Beispiele per Mausklick vom System berechnen lassen. Abbildung 1.1 zeigt die Hauptkomponenten des MuPAD-Systems.