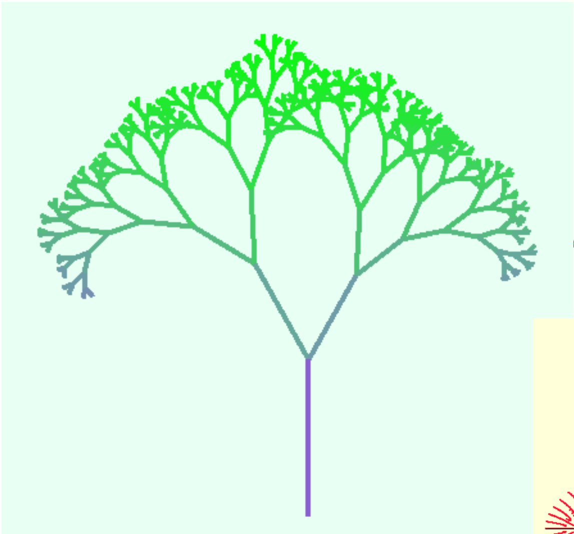
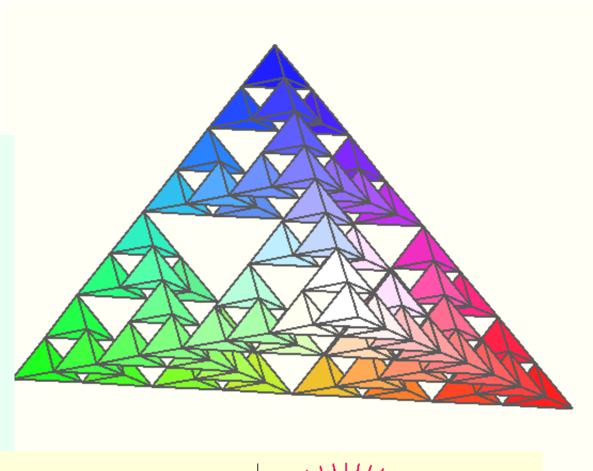
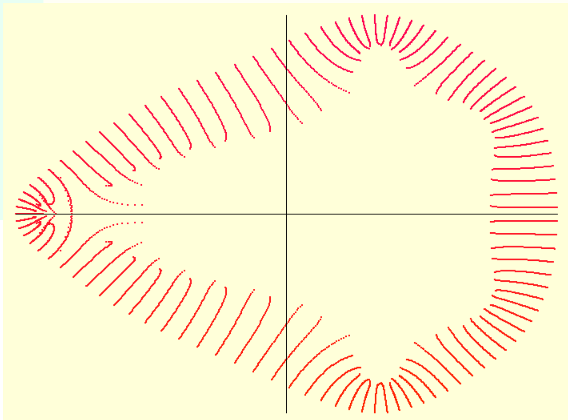
mathPAD

Vol 4 / No2 / Dezember 1994







In dieser Ausgabe:

mathPAD ist eine Publikation der mathPAD-Gruppe der Universität Paderborn

mathPAD

Vol 4 / No2 / Dezember 1994

in dieser Ausgabe:

Liebe Leserin, lieber Leser...

vor Ihnen liegt die neue mathPAD. Diesmal hat es etwas länger gedauert, dafür ist das Heft auch etwas dicker geworden und enthält neben Berichten und Meldungen in eigener Sache, also in MuPAD-Sache, auch Informationen, die auf ein allgemeineres Interesse zählen dürfen. So gibt Karsten Morisse eine Übersicht über parallele Computeralgebra-Systeme, eine Information, die man in dieser Ausführlichkeit wohl selten findet, und Andreas Kemper setzt seine Serie zur Einführung in Neuronale Netze fort.
Besonders herzlich begrüßen wir in diesem Heft Sergei Ivanov aus Donetsk, den jüngsten Autor, den die mathPAD jemals hatte. Bei einem Besuch in Paderborn hat Sergei uns durch seine Auffassungsgabe, seine Kenntnisse in Computeralgebra und seinen Enthusiasmus überrascht. Er sagte uns damals, daß er 13 Jahre alt sei, später stellte sich dann heraus, daß er etwas geflunkert hatte und sich um ein Jahr älter gemacht hatte. Wir sind froh, Sergei zum Kreis treuer MuPAD-Freunde zählen zu können. In diesem Heft berichtet er über die Konstruktion einer konformen Abbildung des Äußeren des Einheitskreises auf das Komplement der Mandelbrotmenge.

B.F. & K.D.

Redaktion:
Benno Fuchssteiner
Klaus Drescher
Universität GH Paderborn
33095 Paderborn, Germany
e-mail:
`benno@uni-paderborn.de`
`kg@uni-paderborn.de`

ISSN 0941 – 9187
V.i.S.d.P.: B. Fuchssteiner
Nachdruck gegen Belegexemplar erlaubt.

Neuronale Netze

Andreas Kemper

Im zweiten Teil dieser kleinen Reihe über Neuronale Netze möchte ich zunächst eine formale Definition Neuronaler Netze geben und dann eine Typisierung durchführen. Anschließend werde ich näher auf das Perzeptron eingehen und lineare sowie nichtlineare Neuronen betrachten.

Formale Definition

In diesem Abschnitt möchte ich zunächst eine allgemeine Definition eines neuronalen Netzes geben.

Definition: Zellulare Struktur

Sei W eine beliebige Menge (Werte der Zellen) und I eine abzählbare Menge (Zellen). Eine Abbildung $c{:}\mathrm{I} \to W$ sei eine Konfiguration von Werten aus W auf den Zellen aus I.
$C = W^I = \{c{:}I \to W\}$ bezeichne die Menge aller Konfigurationen.
Zu jeder Zelle i aus I gebe es eine endliche, geordnete Teilmenge N(i) von I, die die Nachbarn bzw. die Nachbarschaft der Zelle i definiert.
Außerdem existiere zu jedem i aus I eine lokale Funktion $f_i{:}\mathrm{W}^{N(i)} \to W$ die jeder Konfiguration von Werten auf den Zellen der Nachbarschaft von i einen neuen Wert zuweist.
$f = \{f_i | i \in I\}$ sei die Menge der lokalen Funktionen.
Das Tupel $Z = (I, W, N, f)$ wird als *zellulare Struktur* bezeichnet.
Ist I endlich, so heißt Z eine *endliche zellulare Struktur.*
Ist W endlich, so heißt Z *zellularer Automat*.
Durch gleichzeitige Anwendung der lokalen Funktionen in allen Zellen erhält man eine globale Funktion $f{:}\, C \to C$ mit
$F(c)\ (i) = f_i(c(N(i)))$.
Hierbei wird $c(N(i))$ wie folgt gebildet:
Ist $N(i) = (j_1,\ldots,j_n)$ so ist

$c(N(i)) = (c(j_1), \ldots, (j_n))$.

Definition: Zellulares Netz

Eine zellulare Struktur $N = (i, W, N, f)$ mit W Teilmenge von R und

$$f_i(c(N(i))) = \theta_i\Big(\sum_{j \in N(i)} \omega_{i,j} c(j)\Big)$$

mit $\omega_{i,j}$ aus R und monoton wachsenden Funktionen $\theta_i{:}\mathrm{R} \to W$ heißt (deterministisches) *neuronales Netz*. Die Funktion θ_i heißt *Ausgabe–funktion* der Zelle i. $\omega_{i,j}$ wird als *Gewicht* von Zelle i nach Zelle j bezeichnet.

Typisierung formaler Netze

Neuronale Netze können unter verschiedenen Gesichtspunkten typisiert werden. Zum einen kann man sie nach ihrer *Topologie* unterscheiden. So gibt es Netze die vollständig miteinander verbunden sind, ein Beispiel hierfür ist das *Hopfield*-Modell. Andere Netze sind in Schichten unterteilt und geben Informationen immer von einer Schicht an die nächste weiter (*Feed-Forward* - Netze).
Ein weiteres Unterscheidungsmerkmal ist der Aufgabentyp, für den ein Netz eingesetzt wird.

Assoziative Speicher: Hierbei werden im System eine Menge von Mustern gespeichert. Wird nun ein Muster in einer unvollständigen oder teilweise veränderten Form eingegeben, so assoziiert das Netz das vollständige, korrekte Muster.

Inhaltadressierbare Speicher: Das System soll auf ein eingegebenes Muster in einer bestimmten Weise reagieren. Diese Ausgabe kann z.B. in der Erzeugung der Aussprache eines Wortes bestehen.

Kategorisierung: Bei dieser Aufgabe soll das Eingabemuster in eine von mehreren gegebenen Klassen eingeteilt werden.

Ein weiterer wichtiger Aspekt ist die Lernfähigkeit neuronaler Netze. Hierbei unterscheidet man zwischen *Überwachtem Lernen* (Supervised Learning) und *Unüberwachtem Lernen* (Unsupervised Learning).
Im ersten Fall existiert eine spezielle Lernphase, in der dem System Paare aus Eingabe und erwarteter Ausgabe präsentiert werden (*Stimulus-Response*-Paare). Während dieser Phase wird das Netz trainiert und hierbei die Gewichte der Synapsen eingestellt. In der zweiten Phase, der Arbeitsphase, bleiben diese Gewichte dann unverändert. Ziel des Netzes ist es, aus den in der Lernphase vorgegebenen Paaren Verhaltensweisen zu verallgemeinern, so daß das Netz später auch auf Eingaben außerhalb der Trainingsmenge angewendet werden kann.
Beim Unüberwachten Lernen werden dem System nur Eingaben dargeboten. Aus ihrer Häufigkeit, ihrem gemeinsamen Auftreten und ihrer Ähnlichkeit untereinander werden Informationen gewonnen, die dann in den Gewichten gespeichert werden. Diese Art des Lernens findet nicht immer in einer expliziten Lernphase statt, sondern kann auch während der eigentlichen Arbeitsphase passieren. Sie ist damit stärker am biologischen Vorbild orientiert.
Die Eigenschaft, daß beim Erscheinen eines Reizes, der bisher oft in Verbindung mit einem anderen aufgetreten ist, dieser auch assoziiert wird, ist ein grundlegendes Prinzip und findet sich auf der Ebene der Neuronen in der bereits beschriebenen Regel von *Donald Heb*b wieder.

Das Perzeptron

Das *Perzeptron* ist ein Beispiel für ein Netzwerk, das mit der Methode des Überwachten Lernens arbeitet.
Zunächst möchte ich eine speziellere Form eines neuronalen Netzes etwas formaler definieren. Bei diesem Modell wird das Netz in *Schichten* aufgeteilt und die Informationen werden nur von einer Schicht zur nächst folgenden weitergegeben. Die Information fließt also immer in einer Richtung. Verbindungen von Neuronen innerhalb derselben Schicht sind ebenso wenig möglich, wie Verbindungen zu einer vorangegangen Schicht oder einer nicht direkt folgenden Schicht.

Definition: Feed-Forward-Netz

Durch die Nachbarschaftsbeziehungen N einer zellularen Struktur ist ein Graph auf der Knotenmenge der Zellen I gegeben.
Die Menge $E = \{i \in I | N(i) = \emptyset\}$ wird als die Menge der Eingabezellen bezeichnet. Alle Zellen, die durch einen Weg der Länge n mit einer Ausgabezelle e aus E verbunden sind, werden in der Menge S_n zusammengefaßt.
Sind alle S_n disjunkt und nicht leer, so nennen wir sie die Schichten des Netzes und sprechen von einem n+1-*schichtigen Feed-Forward-*Netz. Die Eingabezellen bilden die erste Schicht S_0, die letzte Schicht S_n wird als *Ausgabeschicht* bezeichnet.
Das in diesem Abschnitt behandelte (einfache) Perzeptron ist ein dreischichtiges Feed-Forward-Netz. Es wurde 1962 von Frank Rosenblatt veröffentlicht.
Die Zählweise der einzelnen Schichten ist in der Literatur sehr unterschiedlich. In vielen Fällen wird die Eingabeschicht nicht mitgezählt.

Definition: Perceptron

Ein endliches dreischichtiges neuronales Feed-Forward-Netz P = (I, W, N, f) mit einer Zerlegung in drei Schichten:

$$R = S_0 \subset I; D = S_1 \subset I; \{a\} = S_2 \subset I$$

und W = {0,1} sowie linearen Schwellwertfunktionen als lokalen Funktionen der Zellen aus $D \cup \{a\}$ heißt (einfaches) *Perzeptron*. Die Schicht R wird als *Retina* bezeichnet, die Zellen in D heißen *Feature Detektoren*. a ist die *Ausgabezelle*. *Eingabemuster* sind diejenigen Konfigurationen, die auf den Zellen aus $D \cup \{a\}$ gleich 0 sind.
In den meisten Beschreibungen des Perzeptrons wird die erste Schicht (Retina) weggelassen, da die Gewichte, die die Zellen der Retina mit den Feature Detektoren verbindet, als konstant

angenommen und während des Lernprozesses nicht verändert werden. Das einfache Perzeptron besteht daher aus zwei Schichten von Neuronen, von denen die erste lediglich als Eingabeschicht fungiert. Die zweite Schicht besteht im einfachsten Fall aus einem einzelnen Neuron. In der folgenden Beschreibung besteht die zweite Schicht ebenfalls aus mehreren Ausgabezellen, also das Modell strenggenommen aus mehreren Perzeptron-Elementen.
Da die Zählweise der Schichten, wie bereits erwähnt, in der Literatur oft sehr unterschiedlich ist, wird im Gegensatz zur obigen Beschreibung dieses einfache Perzeptron auch oft als *einschichtiges* Perzeptron bezeichnet. Werden weitere Schichten eingefügt, so werden diese *Hidden Units* genannt.
Sei eine Schicht mit N Eingabezellen gegeben. An diesen Eingaben liegen die Werte ξ_k an. Die zweite Schicht besteht aus mehreren Ausgabezellen, ihre Werte werden mit O_i bezeichnet. Die Berechnung des Netzes besteht aus:

$$O_i = g\left(\sum_{k=0}^{N} \omega_{ik}\zeta_k\right)$$

Hierbei ist g die Ausgabefunktion. Die Angabe eines Schwellwertes fehlt an dieser Stelle, da er durch eine weitere Eingabezelle mit permanentem Wert -1 und einem Gewicht in der Größe des Schwellwertes θ_i simuliert werden kann. Die Ausgabefunktion ist im folgenden die Vorzeichenfunktion *sign*, so daß die Ausgabezellen lediglich die Werte 1 und -1 annehmen können.

$$O_i = g\left(\sum_{k=0}^{N} \omega_{ik}\zeta_k\right) = g\left(\sum_{k=1}^{N} \omega_{ik}\zeta_k - \theta_i\right)$$

Die allgemeine Aufgabe des Perzeptrons besteht darin, auf eine Anzahl gegebener Eingabemuster ξ_k^μ mit ebenfalls vorgegebenen Ausgabemustern ζ_k^μ zu antworten. Das heißt, es soll gelten

$$O_i = g\left(\sum_{k=0}^{N} \omega_{ik}\xi_k\right) = \zeta_i$$

für alle Paare $\mu = 1, \ldots ,p$ der Trainingsmenge.
Der Fall $\xi_i^\mu = \zeta_i^\iota$ wird als *Auto-Assoziation* bezeichnet. Anderenfalls spricht man von *Hetero-Assoziation*. Diese umfaßt auch das *Klassifikationsproblem*, bei dem die Eingaben in eine Menge von vorgegebenen Klassen einsortiert werden sollen. In diesem Fall wird erwartet, daß für jede Eingabe genau eine Ausgabezelle reagiert.

Lineare Separabilität

Für das Perzeptron existiert eine Lernregel, die sicherstellt, daß wenn es Gewichte gibt, die obige Aufgabe erfüllen können, diese mit Hilfe der Lernregel auch gefunden werden.
Die Bedingung für die Existenz solcher Gewichte ist die *lineare Separabilität* der Klassen von Eingabemustern, auf die eine spezielle Ausgabezelle mit einer 1 bzw. einer -1 reagieren soll. Können diese Mengen (im N-dimensionalen Eingaberaum $(\xi_1, \ldots ,\xi_N)$ durch eine N-1-dimensionale Hyperebene getrennt werden, so spricht man von linearer Separabilität. Ein Beispiel für eine nicht linear separable Funktion ist die Funktion XOR. Trägt man ihre 4 möglichen Eingabewerte auf einer zweidimensionalen Ebene ab, so sieht man sehr leicht, daß die beiden Klassen {(1, -1), (-1, 1)} und {(1, 1), (1, -1)}, für die die Funktion jeweils 1 und -1 liefert, nicht durch eine 1-dimensionale Hyperebene, also in diesem Fall eine Gerade, getrennt werden können. Dies ist der einfachste Fall der *Parity-Rule*, die von Minsky und Papert gründlich untersucht worden ist.
Ist eine solche Separabilität nicht gegeben, so kann durch eine Erweiterung der Merkmalsvektoren die Unterscheidbarkeit der Klassen verbessert werden.

Die Lernregel für das Perzeptron

Im folgenden betrachten wir lediglich linear separable Probleme, bei denen also die gesuchten Gewichte ω_{ij} existieren. Der Lernalgorithmus entnimmt ein Element aus der Menge der Trainigspaare und wendet das Perzeptron, das initial beliebige Gewichte besitzt, auf die Eingabe an. Die gelieferte Ausgabe wird mit der erwarteten Ausgabe verglichen und im Fall einer Abweichung, in Anlehung an die Regel von Hebb, die Gewichte um einen Faktor proportional zu dem Produkt von Eingabe und erwarteter Ausgabe verändert. Wir wählen:

$$\omega_{ik}^{new} = \omega_{ik}^{old} + \Delta\omega_{ik}$$

mit

$$\Delta\omega_{ik} = \eta\,(1 - \zeta_i^\mu O_i^\mu)\,\zeta_i^\mu \xi_k^\mu$$

Nach dem Konvergenzsatz für das Perzeptron konvergiert dieser Algorithmus in endlich vielen Schritten gegen eine Lösung für die Gewichte ω_{ik}. Der Wert η gibt die Lernschrittweite an und sollte $\eta << 1$ gewählt werden.

Grenzen des einfachen Perzeptrons

Wie schon erwähnt, scheitert das einfache Perzeptron bereits an relativ einfachen Aufgaben, wie der Berechnung der XOR Funktion. Ein weiteres wichtiges Ergebnis von Minsky und Papert *Perceptrons: An introduction to computational geometry* (1969) besagt, daß ein im Durchmesser seiner Feature Detektoren auf der Retina beschränktes Perzeptron nicht in der Lage ist, den Zusammenhang eines Objektes zu erkennen. Dies zeigt deutlich, daß das biologische Vorbild nicht dem einfachen Perzeptron-Modell entsprechen kann. Eine andere zentrale Beobachtung von Marvin und Minsky betrifft das Parity-Problem. Hierbei wird die Anzahl der verbundenen Figuren innerhalb eines Bildes betrachtet und daraufhin überprüft, ob diese Anzahl gerade oder ungerade ist. Dieses Problem kann nur mit einem Perzeptron der Ordnung k gelöst werden, wobei k die Anzahl der Punkte des Bildes ist und die Ordnung eines Perzeptrons als die Anzahl der Punkte der Retina definiert ist, mit der jede Zelle der Feature Detektoren verbunden ist.

Lineare Neuronen

Bisher wurden in diesem Kapitel lediglich einfache Schwellwertelemente mit der Ausgabefunktion g(h) = sgn(h) betrachtet. Im folgenden werden Neuronen mit nichtdiskreten, differenzierbaren Ausgabefunktionen g(h) behandelt. Dies hat den Vorteil, daß eine Fehlerfunktion $E(\omega)$ definiert und nach den Gewichten ω_{ik} differenziert werden kann. Diese Funktion kann dann mit verschiedenen Techniken minimiert werden. Zunächst wollen wir den einfachen Fall g(h) = sgn(h) betrachten. In diesem einfachen Fall ergibt sich die Ausgabe eines Neurons zu:

$$O_i^\mu = \sum_{k=1}^{N} \omega_{ik} \xi_k^\mu$$

Für diesen Fall einer linearen Ausgabefunktion können die Gewichte explizit berechnet werden

$$\omega_{ik} = \frac{1}{N} \sum_{k=1}^{N} \zeta_i^\mu \,(Q^{-1})_{\mu\nu} \xi_k^\nu$$

wobei

$$Q_{\mu\nu} = \frac{1}{N} \sum_{k=1}^{N} \xi_k^\mu \xi_k^\nu$$

ist. Es ist zu beachten, daß Q hierzu invertierbar sein muß und daß dies nur der Fall ist, wenn die Eingabevektoren ξ^μ linear unabhängig sind. Jedoch kann eventuell auch im Fall einer linearen Abhängigkeit der Eingabevektoren ξ^μ eine Lösung gefunden werden, wenn die Zielvektoren ζ^μ speziell gewählt sind. Da maximal N Eingabevektoren linear unabhängig gewählt werden können, ist die Anzahl der möglichen Assoziationen p, die gespeichert werden können, auf N begrenzt.

Sind die Eingabevektoren linear abhängig, so kann die sogenannte Moore-Penrose *Pseudoinverse* berechnet werden, die den im nächsten Abschnitt definierten Fehler minimiert.

Die lineare Unabhängigkeit für den linearen Fall impliziert die lineare Separabilität für die zuvor beschriebenen diskreten Schwellwertelemente. Die Umkehrung ist i.a. falsch. Viele Probleme im Zusammenhang mit diskreten Schwellwert-Netzen, wie z.B. die XOR-Funktion, erfüllen die Bedingung der linearen Unabhängigkeit der Eingabevektoren nicht.

Ein weiteres Verfahren zur Bestimmung der Gewichte stellt das *Gradientenabstiegsverfahren* dar. Hier werden die Gewichte nicht explizit vorausbestimmt, sondern wiederum mit Hilfe einer Lernregel Schritt für Schritt modifiziert. Hierzu wird eine Fehlerfunktion definiert, die in Abhängigkeit der Gewichte ω_{ik} den jeweiligen

Fehler berechnet. Die Funktion ist gegeben durch:

$$E(\omega) = \frac{1}{2}\sum_{i=1}^{N}\sum_{\mu=1}^{p}(\zeta_i^\mu - O_i^\mu)^2$$

$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{\mu=1}^{p}\left(\zeta_i^\mu - \sum_{k=1}^{N}\omega_{ik}\xi_k^\mu\right)^2$$

E wird umso kleiner, desto besser die Gewichte ω_{ik} die Bedingung erfüllen. Beim Gradientenabstiegsverfahren werden nun die Gewichte proportional zur Richtung des negativen Gradienten der Fehlerfunktion E verändert. Der Gradient der Funktion E ergibt sich zu

$$\frac{\partial E}{\partial \omega_{ik}} = \sum_{\mu=1}^{p}(\zeta_i^\mu - O_i^\mu)\,\xi_k^\mu$$

und damit ergibt sich die Änderung für ein Gewicht ω_{ik} zu:

$$\Delta\omega_{ik} = -\eta\frac{\partial E}{\partial \omega_{ik}} = -\eta\sum_{\mu=1}^{p}(\zeta_i^\mu - O_i^\mu)\,\xi_k^\mu$$

Damit ist für ein Eingabemuster μ die Änderung eines Gewichtes ω_{ik} für jede Komponente ξ_k^μ durch

$$\Delta\omega_{ik} = \eta(\zeta_i^\mu - O_i^\mu)\,\xi_k^\mu$$

gegeben.
Diese kann auch als

$$\Delta\omega_{ik} = \eta\delta_i^\mu\xi_k^\mu$$

geschrieben werden, wenn δ_i^μ als

$$\delta_i^\mu = \zeta_i^\mu - O_i^\mu$$

definiert wird. Diese Regel wird oft auch als *Delta-Rule*, *Adeline-Rule* oder *Widrow-Hoff-Rule* bezeichnet. Das Gradientenabstiegsverfahren ist äquivalent zur Lernregel im diskreten Fall. Dieser kann als Spezialfall des Gradientenabstiegsverfahren interpretiert werden. Das Verfahren konvergiert gegen ein globales Minimum der Fehlerfunktion E, da E eine quadratische Funktion der Gewichte ω_{ik} ist.

Nichtlineare Neuronen

Das Verfahren für den linearen Fall kann ganz natürlich auf den Fall einer nichtlinearen, differenzierbaren Ausgabefunktion g übertragen werden. In diesem Fall ergibt sich die Kostenfunktion zu

$$E(\omega) = \frac{1}{2}\sum_{i=1}^{N}\sum_{\mu=1}^{p}(\zeta_i^\mu - O_i^\mu)^2$$

$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{\mu=1}^{p}\left[\zeta_i^\mu - g\left(\sum_{k=1}^{N}\omega_{ik}\xi_k^\mu\right)\right]^2$$

und für den Gradienten ergibt sich

$$\frac{\partial E}{\partial \omega_{ik}} = \sum_{\mu=1}^{p}[\zeta_i^\mu - \xi_k^\mu g(h_i^\mu)]\,g'(h_i^\mu)\,\xi_k^\mu$$

wobei h_i^μ als

$$h_i^\mu = \sum_{k=1}^{N}\omega_{ik}\xi_k^\mu$$

definiert ist. Analog zur linearen Ausgabefunktion kann man auch jetzt die Veränderung von ω_{ik} angeben, die bei der Bearbeitung eines Eingabemusters μ durchzuführen ist:

$$\Delta\omega_{ik} = \eta\delta_i^\mu\xi_k^\mu$$

Nun gilt jedoch:

$$\delta_i^\mu = [\zeta_i^\mu - O_i^\mu]\,g'(h_i^\mu)$$

Wie im Fall einer linearen Ausgabefunktion existiert auch hier eine Lösung, wenn die Eingabemuster linear unabhängig sind. Diese Lösung muß durch das Gradientenabstiegsverfahren jedoch nicht in jedem Fall gefunden werden. Da die Kostenfunktion i.a. lokale Minima besitzt, ist es möglich, daß das Verfahren in einem solchen Minimum stecken bleibt. Einige Methoden, dies zu verhindern, werden im folgen Teil dieser kleinen Reihe vorgestellt.

Nichtlineare Wechselwirkungen

Thorsten Schulze

Im Vordergrund dieses Artikels stehen Experimente zu Wechselwirkungsphänomenen bei nichtlinearen Teilchensystemen. Diese Systeme werden durch nichtlineare partielle Differentialgleichungen beschrieben, die aufgrund von Linearisierungseigenschaften vollständig integrabel genannt werden. Beispiele hierfür sind die Korteweg-de Vries Gleichung (KdV), die modifizierte Korteweg-de Vries Gleichung und die Sine-Gordon Gleichung. Die Wechselwirkungen werden anhand der KdV untersucht und mit Hilfe des Computeralgebrasystems MuPAD grafisch dargestellt.

Einleitung

Im allgemeinen betrachten wir Gleichungen der Form

$$u_t = K(u)$$

wobei $u = u(x, t)$ und $K(u)$ ein Vektorfeld seien. Gleichungen dieser Art lassen sich in *geeigneten Koordinaten* linearisieren und somit auch lösen. Dabei verfügen diese Gleichungen über besondere, stabile Lösungen, die sogenannten *Soliton-Lösungen*.

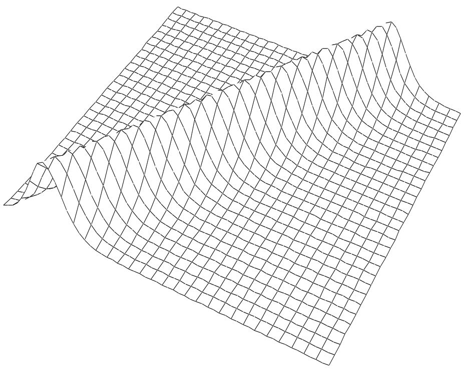


Abbildung 1: Ein-Soliton-Lösung der KdV

Dabei heißt eine Lösung u der Form $u(x, t) = s(x + ct)$ *travelling wave solution* oder *Ein-Soliton-Lösung*. Eine solche haben wir in Abbildung 1 für die KdV

$$u_t = u_{xxx} + 6uu_x$$

dargestellt. Es handelt sich dabei um eine Welle, die sich mit konstanter Geschwindigkeit ausbreitet. Die Geschwindigkeit wird dabei durch den Parameter c beschrieben.
Unter einer *Multi-Soliton-Lösung* verstehen wir eine Lösung, die sich asymptotisch für $t \longrightarrow \pm\infty$ vollständig in Solitärwellen zerlegen läßt. Dieses läßt sich anhand der folgenden Zwei-Soliton-Lösung der KdV gut beobachten:

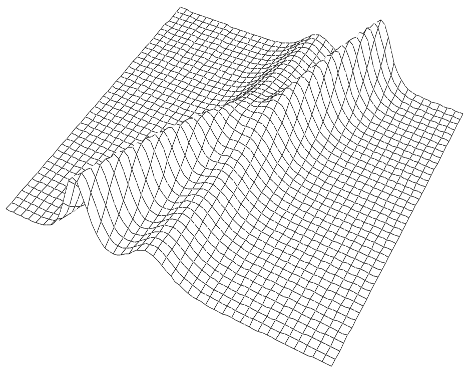


Abbildung 2: Zwei-Soliton-Lösung der KdV

Beide Wellen wandern mit unterschiedlichen Geschwindigkeiten von links nach rechts. Die größere und damit auch schnellere Welle holt die langsame, kleinere Welle ein, es kommt zur Kollision. Aber, und das ist der wichtige Punkt, nach der Kollision tauchen beide Wellen wieder unverändert auf. Es hat scheinbar nur eine Phasenverschiebung stattgefunden. Dieses Phänomen soll im folgenden genauer untersucht werden. Doch zuvor noch eine Eigenschaft der betrachteten Gleichungen: Bei den betrachteten Systemen ist eine informationserhaltende

Beschränkung auf Teilsysteme unter Beibehaltung aller strukturellen Eigenschaften möglich. Im Falle des Zweisolitons führt dieses zu den Begriffen der *interacting* und *virtuellen Solitonen*. Dabei beschreiben erstere eine Zerlegung der ursprünglichen Soliton-Lösung. Für die obige Zwei-Soliton-Lösung ergeben sich die folgenden interacting Solitonen:

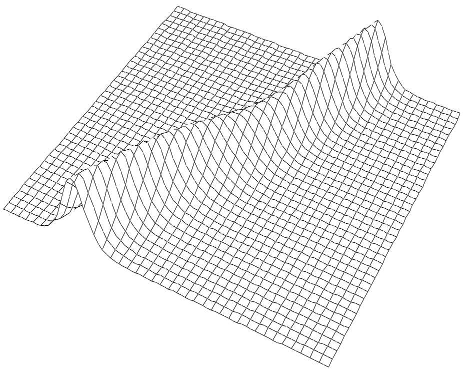


Abbildung 3: Größeres interacting Soliton der Zwei-Soliton-Lösung

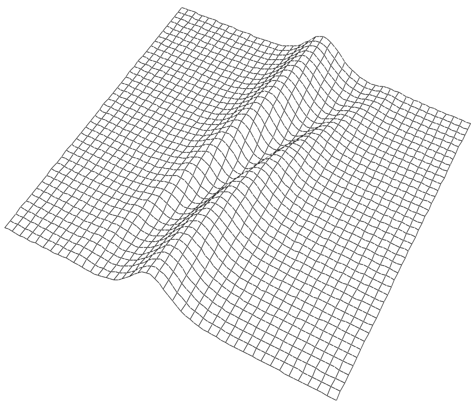


Abbildung 4: Kleineres interacting Soliton der Zwei-Soliton-Lösung

Würde man diese beiden Grafiken übereinanderlegen, so erhielte man wiederum die in Abbildung 2 dargestellte Zwei-Soliton-Lösung. Die virtuellen Solitonen treten dann bei einer weiteren Zerlegung der interacting Solitonen auf. Es handelt sich dabei um Objekte, die nur zeitlich begrenzt, während der Wechselwirkung auftauchen.

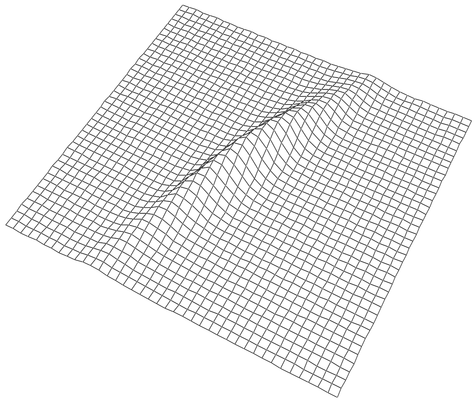


Abbildung 5: Virtuelles Soliton des Zweisolitons

Die Phänomene, die bei der Kollision von Einzelwellen auftreten, werden im folgenden mit der als *analytische Fortsetzung* bekannten Methode untersucht.

Analytische Fortsetzung

In verschiedenen Artikeln stellte man fest, daß sich Soliton-Lösungen in der komplexen Ebene als Summe von Polen darstellen lassen. Wir wollen nun mit Hilfe der analytischen Fortsetzung die Dynamik der einzelnen Singularitäten untersuchen. Betrachten wir dazu die Ein-Soliton-Lösung der KdV, welche sich zu

$$u(x, t) = \frac{\partial^2}{\partial x^2}(ln(f(x, t))) \qquad (1)$$

ergibt mit

$$f(x, t) = 1 + e^{cx - c^3 t} . \qquad (2)$$

Somit erhält man für die Ein-Soliton-Lösung den folgenden Ausdruck:

$$u(x, t) = 2\left(\frac{f_{xx}f - f_x^2}{f^2}\right). \qquad (3)$$

Hierbei steht f_x stellvertretend für die partielle Ableitung von f nach x. Bei der analytischen Fortsetzung fixieren wir eine der unabhängigen Variablen und setzen die andere ins Komplexe fort. In unserem Fall fixieren wir den Zeitparameter t und setzen x ins Komplexe fort. Wir suchen dann nach Singularitäten der Lösung $u(x_{re} + Ix_{im}, t_{fixed})$. Diese ergeben sich aber anhand der obigen Lösungsformel gerade als Nullstellen der Funktion $f(x_{re} + Ix_{im}, t_{fixed})$, welche sich zu

$$x_{re} = c^2 t_{fixed}$$
$$x_{im} = \frac{2k+1}{c}\pi$$

ergeben. Anhand dieser Formeln erkennt man, daß die Imaginärteile der Singularitäten der Ein-Soliton-Lösung äquidistant in der komplexen Ebene liegen. Für die zeitliche Entwicklung berechnet man die Singularitäten für verschiedene Werte des Zeitparameters t_{fixed}. Für die Ein-Soliton-Lösungen mit den asymptotischen Geschwindigkeiten 0.8 und 0.5 erhält man die folgende Grafik:

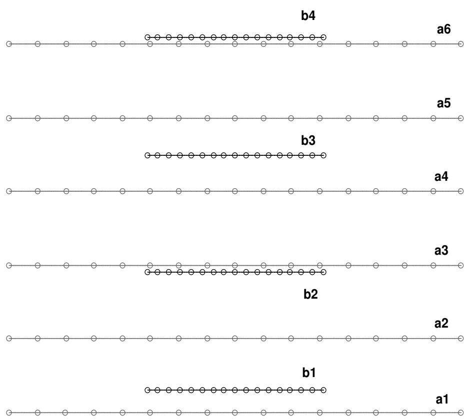


Abbildung 6: Dynamik der Ein-Soliton-Lösungen

Dabei sind die Singularitäten des größeren Einsolitons mit **a1 – a6** und die des kleineren Solitons mit **b1** - **b4** beschriftet. Die Realteile der einzelnen Singularitäten sind dabei an der horizontalen, die Imaginärteile an der vertikalen Achse abgetragen. Die Singularitäten für den minimalen Wert des Zeitparameters sind am rechten Rand der Grafik gezeichnet. Man erkennt zum einen, daß die Imaginärteile der Singularitäten äquidistant verteilt sind, und sich die Abstände nicht verändern. Ferner bewegen sich die Singularitäten mit konstanter Geschwindigkeit, was man anhand der Kreise erkennen kann.

Kommen wir nun zur Zwei-Soliton-Lösung der KdV. Diese ergibt sich, wenn man in den obigen Lösungsformeln (1) und (2) nun folgendes f verwendet:

$$f(x, t) = 1 + e_1 + e_2 + c_{12} e_1 e_2$$
$$e_i = e^{(c_i - c_i^3 t)}$$

Für diese Lösung lassen sich die Nullstellen der Funktion f nicht explizit berechnen, weshalb dazu ein numerisches Verfahren, das Newton-Verfahren, verwendet wird. Dabei gehen wir wie folgt vor: Für den minimalen Wert des Zeitparameters überziehen wir den gewählten Bereich, in dem wir die Nullstellen berechnen wollen, mit einem Gitter und berechnen für jeden Gitterpunkt die Newton-Iteration. Konvergiert diese Iteration, so verwenden wir den Iterationswert als Näherung für eine Nullstelle von f. In jedem weiteren Zeitschritt verwenden wir dann die Näherungen des vorherigen Schrittes als Startwerte für die Newton-Iteration. Mit Hilfe dieses Vorgehens erhalten wir für die Zwei-Soliton-Lösung die folgende Grafik:

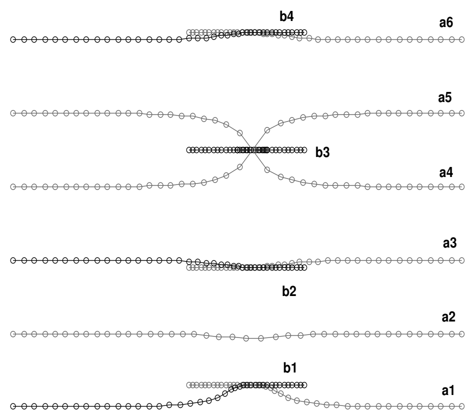


Abbildung 7: Dynamik der Zwei-Soliton-Lösung

Wiederum erkennt man zehn Singularitäten, die mit **a1** - **a6** und **b1** - **b4** beschriftet sind. Dabei gehören die Singularitäten **a1** - **a6** zum größeren Soliton und die Singularitäten **b1** - **b4** zum kleineren Soliton. Dieses gilt jedoch nur für den minimalen Wert des Zeitparameters, denn, wie wir sehen werden, gehören die Singularitäten

nach der Kollision nicht unbedingt zu dem Soliton, zu dem sie vorher gehört haben.
Vergleicht man Abbildung 7 mit der der zugehörigen Ein-Soliton-Lösungen (Abbildung 6), so fällt auf, daß sie in den Randbereichen, also für $t \longrightarrow \pm\infty$ übereinstimmen, was als Charakteristikum für Solitonlösungen schon erwähnt wurde. Interessant jedoch sind die Bereiche für kleine Werte des Zeitparameters, also der Bereich, in dem die Kollision der beiden Wellen stattfindet. An den unteren drei Singularitäten `a1`, `a2` und `b1` kann man gut erkennen, was während der Kollision passiert: Die Singularitäten `a1` und `a2` nähern sich der Singularität `b1` an. Dabei scheint der Abstand der Imaginärteile ein Maß für die Größe dieser Annäherung zu sein, denn die Singularität `a1` wird mehr angezogen als `a2`. Schließlich verringert `a1` ihre Geschwindigkeit und wird zu einer Singularität des kleineren Solitons, wohingegen `b1` an Geschwindigkeit gewinnt und zu einer Singularität des größeren Solitons wird, was in dem folgenden vergrößerten Ausschnitt zu erkennen ist.

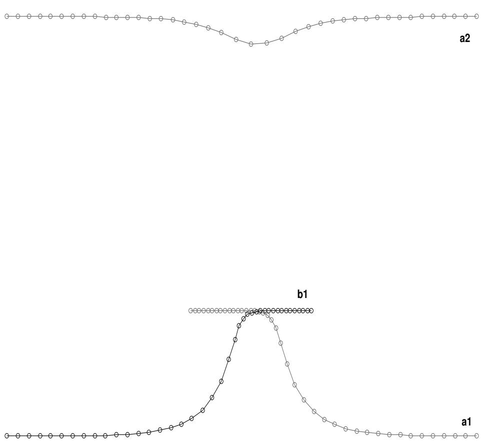


Abbildung 8: Singularitäten `a1`, `a2` und `b1`

Ähnliches gilt auch für die Singularitäten `a3` und `b2` sowie `a6` und `b4`. Eine Ausnahme scheinen hingegen die Singularitäten `a4`, `a5` und `b3` zu bilden, denn hier scheinen sich die Singularitäten `a4` und `a5` auszutauschen. Dieses hat seinen Grund darin, daß der Imaginärteil der kleineren Singularität `b3` genau zwischen den Imaginärteilen von `a4` und `a5` liegt. In diesem Fall scheinen sich nicht Singularitäten unterschiedlicher Zugehörigkeit auszutauschen, sondern die beiden zum größeren Soliton gehörenden Pole tauschen ihre Imaginärteile aus, was anhand folgender Vergrößerung zu sehen ist.

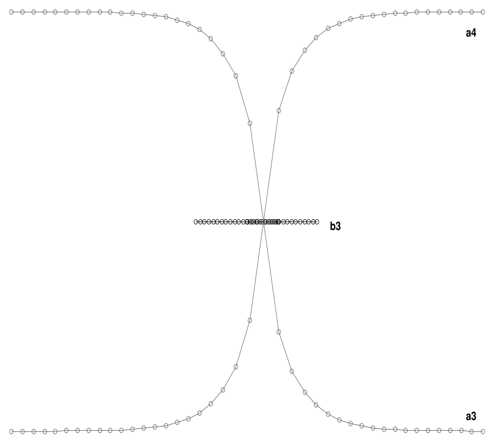


Abbildung 9: Singularitäten `a4`, `a5` und `b3`

Betrachten wir nun die Gleichung, welche die interacting Solitonen der Zwei-Soliton-Lösung beschreibt. Diese ergibt sich aus der Ausgangslösung

$$u(x,t) = \frac{\partial^2}{\partial x^2}(ln(f(x,t,c_1,c_2))) ,$$

indem man eine x-Ableitung durch eine Ableitung nach einer der Phasenverschiebungen p_i ersetzt. Somit lautet die Lösung der interacting Solitonen:

$$s_i(x,t) = \frac{\partial}{\partial p_i}\frac{\partial}{\partial x} ln(f(x,t,c_1,c_2)) .$$

Somit sind bei beiden Gleichungen dieselben Funktionen im Nenner der Lösung zu finden. Da dieser Nenner aber für die Singularitäten verantwortlich ist, ergeben sich dieselben Singularitäten für das Zweisoliton und die zugehörigen interacting Solitonen. Gleiches gilt dann auch für die Dynamik der Singularitäten. Somit können wir also mit Hilfe der Dynamik der Singularitäten keine weiteren Aussagen über die Wechselwirkung machen. Deshalb untersuchen wir diese Phänomene, indem wir die analytische Fortsetzung dreidimensional darstellen. Dieses ist möglich, indem wir den Realteil als x-, den Imaginärteil als y- und den Betrag der analytischen Fortsetzung als z-Koordinate darstellen. Dieses gibt uns dann ein dreidimensionales Bild für den aktuellen Wert des Zeitparameters t_{fixed}. Der Vorteil liegt darin, daß wir nun auch Aussagen über die Massen der einzelnen Singularitäten machen können. Berechnet man dann diese Oberflächen für verschiedene Werte des Zeitparameters, so kann man feststellen, wie sich die Polmassen verändern. Für die Ein-Soliton-Lösung der KdV ergibt sich dabei die in Abbildung 10 gezeigte Grafik. Man erkennt hier drei Singularitäten mit gleichen Polmassen. Diese Massen ändern sich während der zeitlichen Entwicklung nicht.

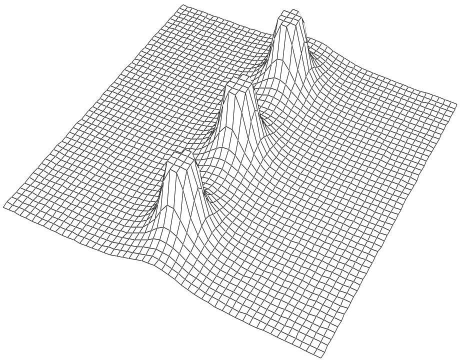


Abbildung 10: Betrag des Einsolitons

Bei der analytischen Fortsetzung der Zwei-Soliton-Lösung hingegen ergibt sich folgendes Verhalten:

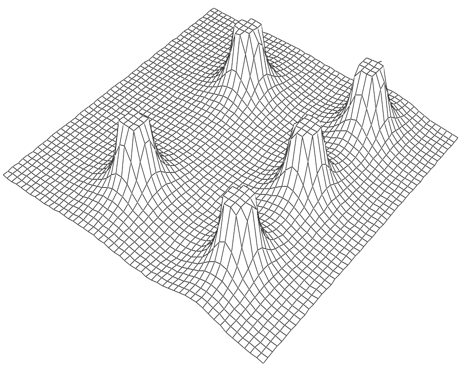


Abbildung 11: Betrag des Zweisolitons für $t_{fixed} = -10$

Betrachtet man diese Grafik und vergleicht sie mit der Abbildung 7, so erkennt man hier die Singularitäten `a1`, `a2` und `a3` am rechten Rand und die beiden Singularitäten `b1` und `b2` im mittleren Bereich der Grafik. Der zeitliche Verlauf wird auf den folgenden Abbildungen deutlich.

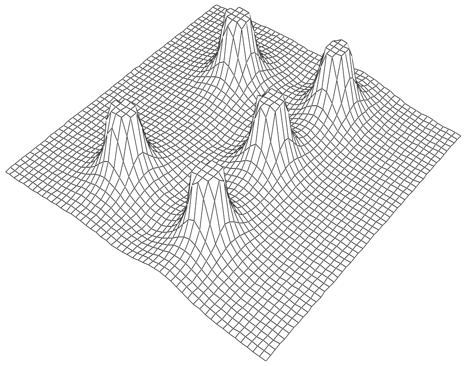


Abbildung 12: Betrag des Zweisolitons für $t_{fixed} = -5$

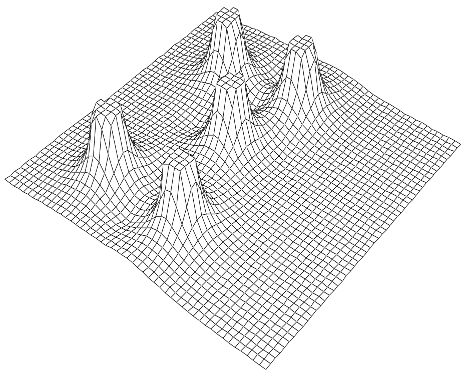


Abbildung 13: Betrag des Zweisolitons für $t_{fixed} = 0$

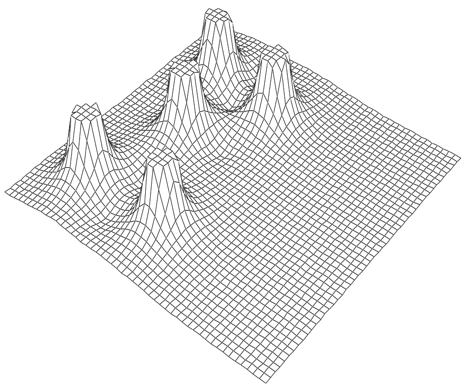


Abbildung 14: Betrag des Zwei-Solitons für $t_{fixed} = 5$

Die Singularitäten **a1** und **a3** verlieren im Vergleich mit der Singularität **a2** an Geschwindigkeit. Gleichzeitig gewinnen die Singularitäten **b1** und **b2** an Geschwindigkeit. Die Polmassen der einzelnen Singularitäten bleiben gleich. Für $t_{fixed} = 10$ ergibt sich dann die Grafik:

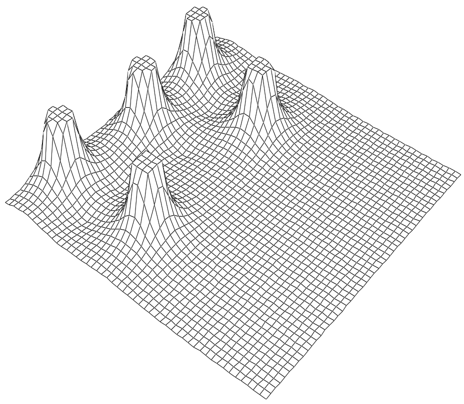


Abbildung 15: Betrag des Zwei-Solitons für $t_{fixed} = 10$

Was sagt dieses Verhalten jedoch über die physikalische Interpretation aus?
Bisherige Arbeiten zu Wechselwirkungsphänomenen bei Solitonen der Korteweg-de Vries Gleichung bestätigten einerseits die klassische physikalische Intuition, daß es sich bei der Wechselwirkung um eine Art *elastischer Streuung* handele, legten aber andererseits auch nahe, daß sich die einzelnen Teilchen nicht nur streuen, sondern daß sie auch einen temporären Identitätsaustausch vornehmen. Betrachtet man die vorliegenden Grafiken unter diesem Gesichtspunkt, so mag man auf den ersten Blick ein wenig verwirrt sein, denn sie zeigen keine elastische Streuung, sondern vielmehr einen Identitätsaustausch. Die Lösung dieses Problems liegt bei den interacting Solitonen. Im vorherigen Abschnitt stellten wir fest, daß die Dynamik der Singularitäten der interacting Solitonen mit denen des ursprünglichen Zweisolitons übereinstimmen. Betrachten wir nun den Betrag der analytischen Fortsetzung des größeren interacting Solitons zum Zeitpunkt $t_{fixed} = -10$, so erkennen wir drei Singularitäten mit großen Polmassen und zwei Singularitäten mit kleinen Polmassen. Die Polmassen dieser Singularitäten verschwinden asymptotisch für $t_{fixed} \to -\infty$.

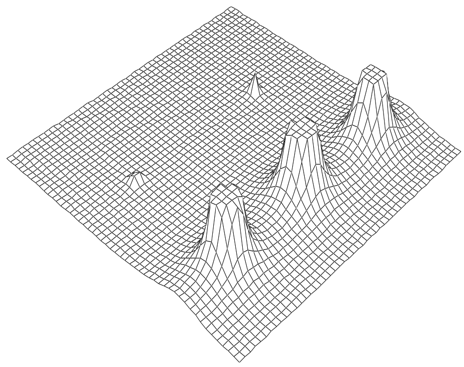


Abbildung 16: Betrag des größeren interacting Solitons für $t_{fixed} = 10$

Dabei sind die Pole mit den großen Polmassen dort zu sehen, wo beim Zweisoliton (vgl. Abbildung 2) die größere Welle zu sehen ist. Die Pole mit den kleinen Polmassen hingegen befinden sich am Ort des kleineren Solitons. Betrachtet man nun weitere Grafiken, die für größere Werte des Zeitparameters erzeugt wurden, so fällt auf, daß die Singularitäten ihre Massen verändern.

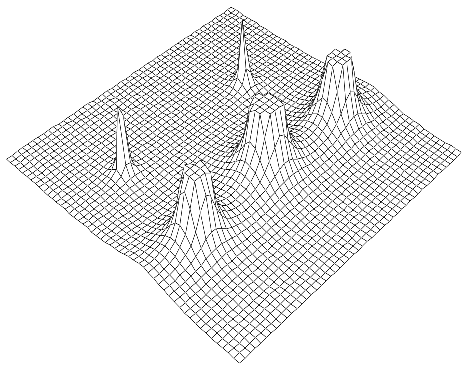


Abbildung 17: Betrag des größeren interacting Solitons für $t_{fixed} = -5$

Dabei werden die zu den Singularitäten **a1** und **a3** gehörenden Polmassen immer kleiner, und die Polmassen der Singularitäten **b1** und **b2** immer größer, wie man auf den folgenden Grafiken erkennen kann.

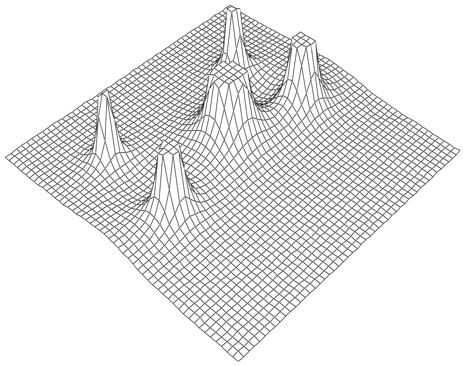


Abbildung 18: Betrag des größeren interacting Solitons für $t_{fixed} = 0$

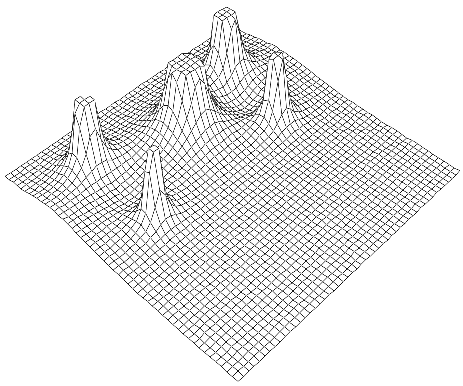


Abbildung 19: Betrag des größeren interacting Solitons für $t_{fixed} = 5$

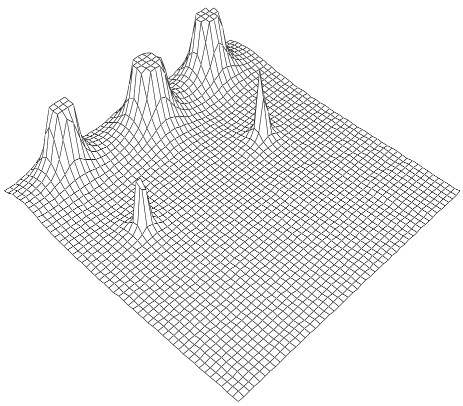


Abbildung 20: Betrag des größeren interacting Solitons für $t_{fixed} = 10$

Ähnliche Grafiken ergeben sich für das kleinere interacting Soliton.
Aufgrund dieses Verhaltens der interacting Solitonen kann man die Wechselwirkung bei Soliton-Kollisionen als Selbstwechselwirkung der interacting Solitonen beschreiben, also als Wechselwirkung zwischen den Singularitäten und den Polmassen. Es scheint dabei folgendes zu gelten: Es findet ein ständiger Massenaustausch statt. Jeder Pol sendet ständig Masse an alle anderen Singularitäten aus. Dabei scheint dieser Austausch jeweils vom Abstand der einzelnen Singularitäten abzuhängen. Das größere interacting Soliton hat dabei Singularitäten mit großen Polmassen, wo es im Zwei-Soliton-Bild auch zu sehen ist, und Singularitäten mit kleinen Polmassen, wo das kleinere Soliton zu sehen ist. Während das große Soliton das kleinere einholt gibt es ständig Masse an dieses ab. Nach dem Überholvorgang nimmt es ständig Masse von diesem auf. Da aber während des Überholvorganges aus dem kleinen Pol mitunter ein großer Pol wird, sieht es dann so aus, als ob die Teilchen ihre Identitäten austauschen.

Parallele Computeralgebra-Systeme

Karsten Morisse

Die verfügbaren sequentiellen Computeralgebra-Systeme sind bei der Bearbeitung *schwieriger* Probleme unter den Gesichtspunkten Laufzeit und/oder Speicherbedarf oftmals überfordert. Abhilfe können hier parallel arbeitende Systeme schaffen. In diesem Beitrag möchte ich einen kurzen Überblick über parallele Computeralgebra-Systeme geben. Alle Systeme befinden sich noch in einem experimentellen Stadium. Das Hauptaugenmerk bei dieser Vorstellung liegt auf dem Systementwurf. Parallele Lisp-Umgebungen, wie z.B. Multilisp oder Tlisp, werden in diesem Artikel nicht beschrieben.

ALTS

ALTS [15] ist ein System für den Einsatz auf Netzwerken von Workstations. Es basiert auf der SACLIB Library, eine C-Version des SAC-II Systems in Library-Form. Die Kommunikation basiert auf den Standards RPC (Remote Procedure Call) und XDR (External Data Representation). Die Verteilung von Tasks ist nicht auf lokale, homogene Netzwerke beschränkt, es können auch heterogene Netzwerke und räumlich verteilte Netzwerke (WAN) eingesetzt werden. Das parallele Programmierparadigma ist wie folgt. Die Versorgung der beteiligten Prozessoren mit Arbeit geschieht über einen *Shared Data Space*, der von einem Prozessor verwaltet wird. Der Zugriff der übrigen Prozessoren geschieht über diesen Server (der bei häufigem Zugriff dann natürlich zum Flaschenhals wird).
In [15] sind als Beispiele die Multiplikation von Matrizen über multivariaten Polynomen und die Berechnung von Determinanten angeben. Im ersten Fall wird mit 10 Prozessoren ein Speed-Up von nahezu 9 erreicht. Im zweiten Beispiel ist das beste Ergebnis mit 10 Prozessoren ein Speed-Up von 8.4.

DSC - Distributed Symbolic Computation

DSC [4] ist ein Software-System für die verteilte Bearbeitung von großen Computeralgebra-Problemstellungen. DSC verwendet die Internet Standard-Protokolle UDP (User Datagram Protocol) und TCP (Transmission Control Protocol) für die Kommunikation und ist somit auf heterogenen Unix-Netzwerken einsetzbar. Das System erlaubt die Verteilung von C- und Lisp-Code sowie der notwendigen Datenfiles. Der Anwender braucht sich um die Verteilung parallel ausführbarer Subtasks nicht zu kümmern, dies wird vom System durchgeführt.
Es gibt eine DSC-Schnittstellen in C und Common Lisp. Die C User Library umfaßt 5 Funktionen zur Generierung, Beenden, Warten auf und Debugging von parallelen Subtasks. Das Common Lisp Interface umfaßt daneben zwei weitere zur Suspendierung und zur Zeitermittlung.
Das System selbst ist aus mehreren Schichten aufgebaut. Die Basisschicht realisiert die Interprozeß-Kommunikation auf Basis der DARPA Internet Standard Protokolle. Die erste Schicht enthält die internen DSC Routinen, Dämon-Prozesse und die C-Library Funktionen. Weitere drei Schichten bestehen aus: Controller Programm, Lisp/C Interface, Lisp Library, high-level Lisp Funktionen.
Mit Hilfe des DSC Controllers können interaktiv neue Applikationen gestartet werden. Weiterhin ist das Überwachen und die Kontrolle von Applikationsprogrammen und paralleler Subtasks möglich.
Auf jedem beteiligten Knoten (Workstation) wird ein *Server-Prozeß* gestartet. Dieser Server überwacht eine UDP Datagram Adresse auf die Ankunft von externen Anfragen (von anderen Servern). Kommt dieser Request vom Controller, so startet der Server einen neuen Subprozeß

zur Ausführung einer Task. Nach der Generierung einer Task sendet der Server an einen geeigneten Server eine Nachricht, um die Ausführung der Subtask zu initiieren. Weiterhin wird ein TCP Stream Socket zur Übermittlung der entsprechenden Daten eingerichtet. Die Auswahl des Remote-Servers bzw. des Prozessors geschieht nach einem Round Robin Verfahren unter verschiedenen Gesichtspunkten: Wo ist bereits ein Server gestartet, welche Sprache (C, Lisp) wird unterstützt und wieviele parallele Subtasks werden bereits bearbeitet. Ist die Ausführung einer parallelen Subtask beendet, so sendet der ausführende Server das Ergebnis an den aufrufenden Server.
In [5] ist ein Version von DSC mit einem verbesserten Scheduling-Algorithmus vorgestellt. Als Anwendung sind der Goldwasser-Kilian/ Atkin Primzahl-Test und die Lösung dünner linearer Systeme behandelt.
DSC ist kein Computeralgebra-System im eigentlichen Sinne. Es stellt eine Basis für verteilte Applikationen in C und Lisp dar. Das besondere ist, daß auch Source-Code verschickt werden kann, der dann von einem Remote-Server kompiliert wird.

||MAPLE||

||Maple|| [14] ist dem Wunsch entsprungen, bereits vorhandene Libraries von Computeralgebra-Systemen mit der Rechenleistung heutiger Parallelrechner zu verbinden. Das System ist im wesentlichen eine Schnittstelle zwischen Maple und der parallelen Programmiersprache Strand. Die Kommunikation erfolgt über Strings. Entworfen ist das System für verteilte Systeme, und Prototypen sind implementiert auf Sequent (20 Prozessoren), Sun Workstation Netzwerk und einem Transputer-System (16 Prozessoren).
Das System ist zweischichtig aufgebaut. Die obere Schicht wird durch Strand gebildet. Hier wird die parallele Ausführung der Algorithmen gesteuert. Auf den beteiligten Prozessoren laufen sequentielle Maple-Prozesse. Sämtliche Parallelverarbeitung geschieht über die deklarative Sprache Strand. Die parallele Programmierung erfolgt über, im Vergleich zu anderen imperativen Sprachen recht gewöhnungsbedürftige, *Guarded Horn Klauseln*.
Verschiedene Programmierparadigmen (paralleles Divide&Conquer, Pipelining, Master/Worker) lassen sich über diesen Ansatz realisieren. Als Anwendungsbeispiele sind in [14] die Isolation reeller Nullstellen von Polynomen und die Berechnung von Gröbner Basen vorgestellt.

MuPAD

Das Paderborner System MuPAD [6] ist das einzige, welches fest in der Programmiersprache verankerte Programmkonstrukte besitzt. Um den in der Praxis auftretenden verschiedenartigen Problemen gerecht zu werden, bietet MuPAD zwei Formen der Parallelverarbeitung an. Basierend auf dem Architekturmodell einer Shared-Memory Maschine stellt die *Mikroparallelität* mehrere Sprachkonstrukte zur Verfügung:

- `parbegin...end_par`: Die Anweisungen des Rumpfes werden gleichzeitig ausgeführt.
- `seqbegin...end_seq`: Sequentielle Ausführung einer Anweisungssequenz innerhalb einer parallelen Umgebung.
- `for...parallel`: Parallele Ausführung der Schleifenrümpfe (für jede Iteration).

Außerdem können *private Variablen* deklariert werden. Das Ende eines parallelen Konstruktes bildet einen Synchronisationspunkt.
Die Mikroparallelität ist ein einfach zu bedienendes, paralleles Programmierparadigma. Der Benutzer hat hierbei nur die parallel ausführbaren Anweisungen zu spezifizieren. Die Erzeugung der korrespondierenden Tasks und deren Verteilung an Prozessoren übernimmt das System. Ein einfaches Beispiel ist die folgende Modifikation des bekannten Fibonacci-Zahlen Algorithmus:

```
fib := proc(n)
begin
if n < 2 then 1
elif n < 10 then
   fib(n-1) + fib(n-2)
else
   _plus((parbegin
         fib(n-1); fib(n-2)
      end_par))
end_if
end_proc:
```

Hier werden die rekursiven Aufrufe von `fib` gleichzeitig ausgeführt. Um eine zu feine Granularität zu vermeiden, wurde die zweite if-Abfrage eingefügt. Andernfalls wäre der Auf-

wand für die Task-Erzeugung und Verteilung zu groß.
Die Mikroparallelität arbeitet nur dann zufriedenstellend, wenn die Kommunikation zwischen den Prozessoren gering bzw. schnell ist. Dies ist jedoch bei verteilten Systemen[1] häufig nicht der Fall. Zudem wird der mögliche Durchsatz durch eine Reihe verschiedener Randbedingungen beeinflußt: Geschwindigkeit, Latenzzeit, Datengröße, Task-Granularität, Datenabhängigkeiten etc. Ein Automatismus, der derartige Größen bei der Aufgabenverteilung berücksichtigt, ist insbesondere im Bereich der Computeralgebra schwierig, wenn nicht gar unmöglich, zu entwickeln. Daher bietet MuPAD mit der *Makroparallelität* eine zweite Form der Parallelverarbeitung an. Diese erlaubt eine optimale Anpassung des Problems an die zugrundeliegende Hardware. Der Benutzer besitzt die vollständige Kontrolle über parallele Programmierung und Lastverteilung/Scheduling. Das Architekturmodell ist eine beliebige Anzahl von *Clustern*, die mittels einiger weniger Konstrukte kommunizieren können. Jeder Cluster besteht wiederum aus einer beliebigen Zahl von Prozessoren, die durch einen gemeinsamen Speicher verbunden sind[2]. Die verfügbaren Konstrukte sind:

- *Queues*: Kommunikationskanal zu einem Cluster. Jeder andere Cluster darf in diesen Kanal schreiben.
- *Pipes*: Unidirektionaler Kommunikationskanal zwischen genau zwei Clustern.
- *Netzvariablen*: Globaler Speicher.

Andere parallele Programmierparadigmen können mittels dieser Konstrukte einfach nachgebildet werden. Als Besonderheit stehen die parallelen Konstrukte auch in jeder sequentiellen MuPAD-Version zur Verfügung. Die Ausführungsreihenfolge der erzeugten Tasks erfolgt dabei nach einem Zufallsprinzip.

PAC

PAC [10] ist ein Computeralgebra-System für den Einsatz auf verteilten Systemen. Der Schwerpunkt der parallelen Algorithmen liegt auf numerischen Computeralgebra-Algorithmen. Der symbolische Teil wird von einem üblichen sequentiellen System auf einem Host ausgeführt, an den die MIMD-Maschine über ein spezielles Interface angeschlossen ist. Dieses System kann parallele Algorithmen auf der MIMD-Maschine aufrufen. Ein Prototyp läuft auf einem FPS Hypercube Tesseract 20 mit 16 Prozessoren (Transputer). Beispiele aus den Bereichen Multiplikation univariater Polynome, lineare Diophantische Gleichungen, Gauß-Elimination, Gröbner-Basen belegen den effektiven Einsatz des Systems.
Eine nachfolgende Implementation wurden auf größeren Netzwerken (Telmat Meganode mit 128 Transputern) durchgeführt [13].

PACLIB

PACLIB [7], [11] ist die parallele Version von SACLIB, einer auf SAC-2 basierenden C-Library, die am RISC-Linz entwickelt worden ist. SAC-2 ist eine Sammlung von Algorithmen für die verschiedensten Operationen auf Integern (Langzahlarithmetik), rationalen Zahlen, endlichen Körpern und Polynomringen mit ebendiesen Koeffizienten. Implementiert sind die Algorithmen in ALDES, welches nach Assembler oder Fortran übersetzt wird. Das ganze System ist listenorientiert. SACLIB ist eine C-Version von SAC-2 (inzwischen gibt es auch noch andere Versionen von SAC-2: LISP, Modula-2). SACLIB kann wie jede andere C-Library zu dem eigenen Anwendungsprogramm hinzugebunden und verwendet werden. PACLIB ist eine parallele Version von SACLIB. Implementiert wurde das Paket auf einer Sequent Symmetry mit 20 Prozessoren. Der Systemaufbau ist dabei wie folgt:

- In einer *Ready Queue* werden alle ausführbaren Tasks abgelegt.
- Der *Shared Heap* ist ein gemeinsamer Speicher und dient als Kommunikationsmedium.
- Es gibt eine Anzahl *virtueller Prozesse*, die die Tasks ausführen. Jeder Prozeß ist dabei ein Unix-Prozeß, der eine lokale Liste freier Speicherzellen des Heaps besitzt. Bei der Ausführung einer Task wird der Speicher aus dieser Liste verwendet.

1. Hierunter fallen als mögliche Architekturen auch Netzwerke (LAN, WAN) von Workstations.
2. Auf jedem Cluster kann somit die Mikroparallelität angewendet werden.

- Die *Available List* besteht aus einer Liste freier Heap-Speicherzellen, die bei Bedarf an die virtuellen Prozesse vergeben werden. Ist diese Liste leer, so wird eine Garbage Collection ausgeführt

Die Verwaltung der Tasks basiert auf dem µ-System, einer C-Library für *light-weight concurrency* auf Shared-Memory Maschinen und Unix-Workstations. Die Effizienz der Task-Verwaltung erlaubt die Anwendung von feinkörniger Parallelität. Es gibt Funktionen für die Erzeugung neuer Tasks (Funktionsaufrufe werden als parallel ausführbar gekennzeichnet), zur Synchronisation und zum Abbruch von Tasks. Nicht-Determinismus und spekulative Parallelität stehen ebenfalls zur Verfügung. Für die Eingabe an und die Ausgabe von Tasks dienen SACLIB-Objekte. Beispielsweise wird mit

```
result = pacStart(func, args)
```

eine Task erzeugt, die den Aufruf `func(args)` gleichzeitig zum Vater ausführt und das Ergebnis in `result` ablegt. Auf Tasks und ihre Ergebnisse kann über Handles (Referenz in den Shared Heap) beliebig oft zugegriffen werden. Zu diesem Zweck wird eine Task durch einen Task-Deskriptor und ein Ergebnis-Deskriptor beschrieben. Ist eine Task beendet, so kann über den Ergebnis-Deskriptor, der eine beliebige SACLIB-Liste sein darf, beliebig oft auf das entsprechende Resultat zugegriffen werden.

Neben der Kommunikation über den Shared Heap wird auch eine Kommunikation über Streams angeboten. Ein Stream ist ein Kommunikationskanal über den eine Task Daten an andere Tasks versenden kann.

Wie auch zum SACLIB-Paket, gehört auch zu PACLIB eine Speicherverwaltung mit einem effizienten Garbage Collection Algorithmus. Dieser wird parallel von allen Prozessoren ausgeführt. Dabei durchsuchen die Prozessoren den Heap nach nicht mehr referenzierten Speicherzellen.

In [11] ist die Anwendung auf das Lösen linearer Gleichungssysteme beschrieben. Als bestes Ergebnis wird mit 18 Prozessoren ein Speed-Up von 15 erzielt.

PARSAC-2

PARSAC-2 [3],[8] ist eine in C realisierte, auf SAC-2 aufbauende, parallele Computeralgebra-Bibliothek. Zielarchitektur sind Shared-Memory Maschinen und Netzwerke von Workstations. Testimplementationen laufen auf einer Encore Multimax (eng-gekoppelter Multiprozessor mit Bus Verbindung) unter dem Betriebssystem Mach.

Das parallele Programmierparadigma bilden *S-Threads*. Ein *Thread* ist ein *light-weight process,* der einen Funktionsaufruf ausführen kann. Ein S-Thread ist ein Thread mit Zugriff auf einen lokalen Speicherbereich und der Möglichkeit der parallelen Listenverarbeitung.

Das System besteht aus 3 Komponenten:

- Das *S-Threads System* bildet eine auf Threads basierende Umgebung für die parallele Listenverarbeitung. Es ist aufgesetzt auf C-Threads, die in der Testimplementation durch das Betriebssystem Mach bereitgestellt werden. Jeder Thread ist ein eigener Prozeß, der beispielsweise einen Funktionsaufruf ausführen kann. Jedem S-Thread steht dabei ein eigener, lokaler Bereich von Speicherzellen zur Verfügung. Dieser Bereich kann nach Beendigung des Threads an einen anderen Thread oder an den Heap übergeben werden. Als Basisoperationen dienen `sthread_fork` und `sthread_join`. Mittels `sthread_fork` kann ein neuer S-Thread generiert werden, der parallel zu dem aufrufenden Thread abläuft. Als Parameter dient dabei ein Funktionsaufruf, der dann von dem Thread ausgeführt wird.
- Als algebraisches Kernstück dient SAC-2. Jede Funktion kann mittels S-Threads parallel zu anderen Funktionen ausgeführt werden.
- Die oberste Ebene bilden in C geschriebene parallele Algorithmen. Die parallelen Elemente werden dabei durch das Thread-System realisiert, sequentielle algebraische Manipulationen werden von SAC-2 übernommen.

Der zugrundeliegende gemeinsame Speicher ist in *Seiten* unterteilt. Er dient als Speicherreservoir für die Threads. Benötigt ein Thread Speicherplatz, so allokiert er dynamisch Seiten aus dem gemeinsamen Speicher. Nachfolgende Speicheranforderungen werden dann lokal befriedigt[3]. Durch dieses Konzept, jede Seite ist

3. Sofern lokal noch genügend Platz vorhanden ist. Andernfalls werden Seiten aus dem gemeinsamen Speicher angefordert.

Teil des gemeinsamen Speichers, kann ein Seitentransfer einfach durch Übergabe einer Referenz realisiert werden. In PARSAC ist kein üblicher Garbage-Collection Algorithmus implementiert, sondern ein *präventiver Garbage-Collection Mechanismus*. Wenn bei der Beendigung einer Task Zwischenresultate und Endergebnis auf einer (oder mehrerer) Seiten stehen, so wird das Ergebnis auf eine neue Seite (des Vaters) kopiert und die alten Seiten können wieder freigegeben werden. Eine echte Garbage-Collection findet nur sehr selten statt. Bei dem in der Computeralgebra häufig auftretenden Effekt *Intermediate Expression Swell* scheint das eine vernünftige Strategie zu sein.
Das parallele Programmiermodell in PARSAC ist das parallele Absetzen von Funktionsaufrufen innerhalb eines gemeinsamen Speichers. Auf diese Weise können recht einfach neue parallele Algorithmen auf Basis von SAC-2 implementiert werden. Es können jedoch auch bereits bestehende SAC-2 Algorithmen parallelisiert werden. Zahlreiche Anwendungen sind vorgestellt worden: Langzahlmultiplikation (Karatsuba- und 3-primes FFT-Methode), GCD multivariater Polynome, Isolation reeller Nullstellen von Polynomen, Bestimmung von Gröbner Basen.
PACLIB und PARSAC-2 ähneln sich in mancherlei Hinsicht (Light-weight Concurrency, virtuelle Prozessoren). Die wesentlichen Unterschiede sind in der Verwaltung des Speichers zu finden. In PARSAC-2 besitzt jeder Prozeß einen eigenen lokalen Speicherbereich des Heaps. Speicheranforderungen werden aus diesem Bereich befriedigt. In PARSAC-2 wird nach Beendigung eines Prozesses das Ergebnis an den Vater zurückgegeben (und kopiert). In PACLIB wird hingegen nur die Referenz auf eine Speicherzelle innerhalb des gemeinsamen Heaps übergeben. Auf diese Referenz kann beliebig oft und von jedem Prozessor zugegriffen werden. In PARSAC-2 kann nur der Vater auf das Ergebnis zugreifen. Weiterhin besteht in PARSAC-2 nicht die Möglichkeit Tasks abzubrechen.

Netzwerk-Version von SAC-2

In [12] ist eine weitere Netzwerk-Version von SAC-2 beschreiben. Das System ist implementiert auf einem lokalen Netzwerk von Sun-Workstations (Ethernet), die via Sockets Daten austauschen (Message-Passing Kommunikation). Auf jedem Knoten des Netzes läuft ein eigener SAC-2 Prozeß (Algorithmen-Server), der von einem Scheduler Tasks zur Bearbeitung erhält. Aufgrund der hohen Kommunikationskomplexität ist nur die grobkörnige Parallelität effektiv einsetzbar. Die SAC-2 Systemsprache ALDES wurde um Konstrukte zur Spezifikation paralleler Prozeduraufrufe erweitert. Die Maschinen des Netzwerkes werden wie übliche andere Peripherie-Geräte behandelt. Die Kommunikation via Message-Passing geschieht damit wie die übliche Ein- und Ausgabe durch Angabe einer entsprechenden Device-Nummer. Parallel auszuführende Funktionsaufrufe werden mit einem Präfix **!** versehen. Die Funktionsaufrufe werden intern umgewandelt in eine *Task Spezifikation* und die Resultate werden im Programm durch *Futures* ersetzt. Das aufrufende Programm fährt in seiner Programmausführung fort. Daten, die Futures enthalten, sind unvollständig. Werden vollständige Daten benötigt, so müssen die Futures durch ihre Werte ersetzt werden. Dies kann explizit durch den Programmierer erfolgen oder aber auch durch den Compiler. Das Ziel mit der Entwicklung der Netzwerk-Version war nicht eine intensive Entwicklung neuer, verteilter Algorithmen, sondern eine einfache Erweiterung eines bestehenden Systems. Alle SAC-2 Programme sind auch in der Netzwerk-Version einsetzbar.

RR

Eine parallele Variante des Computeralgebra-Systems Reduce ist in [9] beschrieben. Es handelt sich hierbei um eine Kombination aus Reduce, PSL (Portable Standard Lisp) und dem PVM-System[4].
Die weite Verbreitung von PVM (es steht auf Workstation-Netzwerken und zahlreichen Multiprozessor-Systemen zur Verfügung) ermöglicht eine einfache Portierung von RR auf verschiedene Architekturen, sofern Reduce und

4. PVM (Parallel Virtual Machine) bietet auf zahlreichen Architekturen eine einheitliche, komfortable Möglichkeit des Message-Passings, unabhängig von der zugrundeliegenden Netzwerktopologie.

PSL dort verfügbar sind.
Das parallele Programmiermodell ist ein Master/Worker-Modell. Es gibt einen Master-Prozeß, der eine beliebige Anzahl von asynchron arbeitenden Worker-Prozessen starten kann. Diese warten auf die Zuteilung von Aufgaben zur Bearbeitung. Jeder solche Prozeß ist eine vollkommen eigenständige Reduce/PSL Sitzung. Darin stehen sämtliche symbolischen und algebraischen Fähigkeiten zur Verfügung. Er verfügt über einen lokalen Speicher zur Speicherung von Programmen und Daten. Der Zugriff auf Prozesse geschieht über *Process Handles*.
Arbeit wird in Form von *Tasks* an die Prozesse zugewiesen. Dies sind typischerweise Funktionsaufrufe in Reduce oder direkt in Lisp. Der Zugriff auf eine Task, z.B. Statusabfrage, geschieht über *Task Handles*. Die Vergabe von Tasks an Worker-Prozesse darf nicht nur durch den Master erfolgen, sondern auch durch die Worker untereinander. Zu diesem Zweck müssen natürlich Process und Task Handle bekannt sein. Bei der Erzeugung einer Task wird spezifiziert, ob ein Ergebnis zurückgegeben werden soll. Falls ja, so wird auf dieses über den Task Handle zugegriffen. Neben den Funktionen zur Erzeugung von Tasks, gibt es Funktionen zur Statusabfrage, Ergebnisabfrage und zum Abbruch. Als Besonderheit kann die Ausführung einer Tasks auf einem Prozessor suspendiert werden, um zunächst in der Warteschlange dieses Prozesses wartende Tasks zu bearbeiten.
Ein globaler Speicher steht den Prozessen nicht zur Verfügung. Jeder kann nur auf einen lokalen Speicher zugreifen. Der Austausch von Daten der beteiligten Prozeß untereinander muß explizit durch das Absetzen von Tasks erfolgen.
Für die Implementation von feinkörniger Parallelität ist dieses Modell weniger geeignet. Typische Anwendungen sind eher die Ausführung großer datenunabhängiger Aufgaben.

Sugarbush

Die in [1] vorgestellte Version basiert auf C/Linda und ist auf einer Sequent Balance (Shared Memory Maschine mit 12 Prozessoren und 16 MByte Hauptspeicher) implementiert. Das Ziel war es, ein bestehendes Computeralgebra-System mit geringem Aufwand zu einem parallelen Computeralgebra-System zu erweitern. Die bereits bestehenden Libraries sollten auch weiterhin nutzbar sein. Das System ist folgendermaßen aufgebaut: Es werden mehrere Maple-Prozesse gestartet. Zu jedem Prozeß gehört ein *parallel communication transceiver (PCT)*. Die Kommunikation zwischen Maple und PCT ist bidirektional (PCT -> Maple: via Pipe, Maple -> PCT: via Files). Jeder Kern wird mit einem speziellen Library-Paket initialisiert, in dem parallele Operationen bereitgestellt werden. Die Ausführung der Operationen erfolgt über den PCT, der die Operationen auf entsprechende Linda-Konstrukte abbildet. Jede Kopie des Maple-Kernes kann jede Prozedur der Maple-Library ausführen und sorgt für eine selbständige Verwaltung des Speichers. An Operationen werden Nachbildungen der Linda Operationen auf dem Tuple Space bereitgestellt. Es gibt keine Konstrukte wie beispielsweise **for parallel do**. Diese können jedoch durch die Operationen auf dem Tuple Space nachgebildet werden (dies kann jedoch sehr teuer sein). Das vorgestellte System eignet sich für grobkörnige Parallelität, da der Kommunikationsaufwand sehr hoch ist. Insgesamt hat sich das System jedoch als sehr langsam erwiesen.
Eine neuere Version von Sugarbush [2] basiert auf einem speziellen Maple Kern (OEM Kernel) und ist auf einer Sequent Symmetry und auf einem Netzwerk von Sun Workstations implementiert. Es handelt sich ebenfalls um ein interaktives System, in dem Zugriff auf jede Funktion der Maple Library besteht. Als parallele Basisprimitive gibt es wiederum den Linda-Konstrukten entsprechende Operationen auf einem Tuple Space. Zusätzlich gibt es noch einige weitere Befehle. Zum Starten neuer Maple Prozesse gibt es verschiedene Versionen eines `eval` Kommandos. Zur Gewährleistung eines exklusiven Zugriffs oder zur Synchronisation können `lock`-Variablen verwendet werden. Es wird ein *Worker/Task* Konzept unterstützt, bei dem eine Anzahl von Worker-Prozessen gestartet werden. Diese Prozesse können dann über den Tuple Space mit Arbeit versorgt werden. Sind die Daten sehr groß, so erfolgt der Datenaustausch über das Filesystem anstelle über den Tuple Space, da dieser hinsichtlich der Objektgröße sehr beschränkt ist. Dieses Scheduling kann manuell durch das Eintragen spezi-

eller Tupel in den Tuple Space erfolgen oder aber durch Aufruf spezieller Routinen (**parmap**, **partasks**, **pardivide**), die mehrere Aufgaben erzeugen, welche dann von den Worker-Prozessen bearbeitet werden. Das Ergebnis der Worker wird als Ergebnis zurückgegeben. Dabei wird der aufrufende Prozeß selbst auch als Worker-Prozeß eingesetzt. Mit Hilfe der Funktion **Start_pipeline** besteht die Möglichkeit, mehrere Worker in einem Pipelining arbeiten zu lassen.
Von dieser Sugarbush-Version sind keine Laufzeiten bekannt. Es soll jedoch weitaus schneller als die erste Version sein.

Schlußbetrachtung

Nahezu alle Systeme bestehen aus der Einbettung eines bestehenden, sequentiellen Systems in eine parallele Programmiersprache bzw. eine parallele Laufzeitumgebung. Dies hat den Vorteil, daß alle sequentiellen Algorithmen weiterhin verfügbar sind. Parallele Algorithmen für spezielle Probleme sind damit natürlich noch nicht gewonnen. Durch das gleichzeitige Aufrufen vorhandener sequentieller Algorithmen können Probleme, die für die Anwendung dieser grobkörnigen Parallelität geeignet sind, relativ einfach parallel gelöst werden.
MuPAD ist das einzige System, welches von Beginn an als parallelverarbeitendes System entwickelt wurde. Dies spiegelt sich in der vergleichsweise einfachen Anwendbarkeit paralleler Konstrukte wider.

Literatur

[1] B.W. Char: *Progress report on a system for general-purpose parallel symbolic algebraic computation*. In T.T. Sasaki (ed), Proceedings of the ISSAC 1990, ACM Press, 1990.

[2] B.W. Char: *A users's guide to Sugarbush - parallel Maple through Linda*. Preprint, September 1993.

[3] G.E. Collins, J.R. Johnson, W. Küchlin: *PARSAC-2: A Multi-Threaded System for Symbolic and Algebraic Computation*. Technical Research Report OSU-CISRC-12/90-TR38, Ohio State University, 1990.

[4] A. Diaz, E. Kaltofen, K. Schmitz, T. Valente: *DSC A System for Distributed Symbolic Computation*. In S.M. Watt (ed), Proceedings of the ISSAC 1991, ACM Press, 1991.

[5] A. Diaz, M. Hitz, E. Kaltofen, A. Lobo, T. Valente: *Process Scheduling in DSC and the Large Sparse Linear Systems Challenge*. In A. Miola (ed), DISCO 93, vol. 722 of LNCS, Springer-Verlag, 1993.

[6] B. Fuchssteiner et al.: *MuPAD Benutzerhandbuch*, Birkhäuser Verlag, Basel-Boston-Berlin, 1993

[7] H. Hong, A. Neubacher, W. Schreiner: *The Design of the SACLIB/PACLIB Kernels*. In A. Miola (ed), DISCO 93, vol. 722 of LNCS, Springer-Verlag, 1993.

[8] W. Küchlin: *The S-Threads Environment for Parallel Symbolic Computation*. In R. Zippel (ed), Computer Algebra and Parallelism, vol. 584 of LNCS, Springer-Verlag, 1990.

[9] H. Melenk, W. Neun: *RR: Parallel Symbolic Algorithm Support for PSL Based Reduce*. Preprint, 1994.

[10] J.L. Roch, P. Senechaud, F. Siebert-Roch, G. Villard: *Computer Algebra on MIMD Machine*. SIGSAM Bulletin, 23(1), January 1989.

[11] W. Schreiner, H. Hong: *The Design of the PACLIB Kernel for Parallel Algebraic Computation*. In J. Volkert (ed), Parallel Computation, vol. 734 of LNCS, Springer-Verlag, 1993.

[12] S. Seitz: Algebraic Computing on a Local Net. In R.E. Zippel (ed), Computer Algebra and Parallelism, vol. 584 of LNCS, Springer-Verlag, 1990.

[13] F. Siebert-Roch, G. Villard: *PAC: First experiments on a 128 Transputer Meganode*. In S.M. Watt (ed), Proceedings of the ISSAC 1991, ACM Press, 1991.

[14] K. Siegl: *Parallelizing Algorithms for Symbolic Computation using* ||MAPLE||. SIGPLAN Notices, 28(7), 1993.

[15] T.C. Torgersen: *Distributing Symbolic Computations on a Network of Workstations*. In K.G. Fischer et al. (eds), Computational Algebra, vol. 151 of Lecture Notes in Pure and Applied Mathematics, Marcel Dekker New York-Basel-Hongkong, 1993.

Threads unter Solaris

Holger Naundorf

Von der Industrie wird die Wichtigkeit von Parallelrechnern immer stärker erkannt. Deshalb werden auch immer mehr parallele Computer mit entsprechender Software angeboten. SUN z.B. bietet die Familie der SparcServer und SparcCenter mit dem Betriebssystem SOLARIS und einer entsprechenden Library an. In diesem Artikel wird ein kurze Einführung in das Konzept dieser Library gegeben. Außerdem wird die Performance dieser Hardware und Software mit einer älteren Sequent Symmetry verglichen.

Threads

In traditionellen Unix-Systemen entspricht ein Prozeß der Ausführung eines sequentiellen Programms oder hat nur einen einzigen 'Thread of Control' oder einfach 'Thread'.

In einem 'multi-threaded' Unix-Prozeß dagegen können mehrere Ausführungen (Threads) gleichzeitig aktiv sein und quasi-parallel oder auch echt parallel ausgeführt werden. Die Anzahl der in einem Prozeß existierenden Threads kann von außen nicht festgestellt werden und kann sich während der Laufzeit ändern.

Die Threads eines Prozesses teilen sich einen Programm-Code — verschiedene Threads können sich bei der Ausführung allerdings an verschiedenen Stellen im Programm befinden — und einen gemeinsamen Speicher. Im gemeinsamen Speicher befinden sich die globalen Variablen des Programms und dynamisch angeforderte Speicherblöcke.

Dieser gemeinsame Speicher ist einer der großen Vorteile bei der Benutzung von mehreren Threads gegenüber der Benutzung von mehreren Prozessen. Der andere große Vorteil ist, daß Threads wesentlich schneller zu erzeugen sind als Prozesse.

Natürlich hat jeder Thread seinen eigenen Programmzähler (PC) und Stack, auf dem die lokalen Variablen eines Threads gespeichert werden. Ein Problem ist, daß die Stacks der einzelnen Threads nicht voreinander geschützt sein müssen — wohl aber können —, so daß durch 'verirrte' Zeiger eines Threads der Stack eines anderen Threads verändert werden kann — auf diese Weise können natürlich schwierig zu findende Fehler erzeugt werden.

Jeder Thread hat alle Möglichkeiten eines konventionellen Programms. So kann ein Thread beliebige System-Aufrufe durchführen — obwohl etwas Vorsicht geboten ist, da manche System-Aufrufe alle Threads beeinflussen können (z.B. Operationen auf Files). Ein Thread hat auch die Möglichkeit, neue Threads zu erzeugen oder selbst zu terminieren. Dazu sollte er allerdings nicht `exit()` verwenden, da dadurch der gesamte Prozeß beendet wird.

Die Thread-Library stellt neben Funktionen zum Erzeugen und terminieren von Threads auch verschiedene Möglichkeiten der Synchronisation wie mutex locks, condition variables und semaphores zur Verfügung. Eine Synchronisation von verschiedenen Prozessen ist mit dieser Library auch möglich.

Lightweight-Prozesse

Mit Hilfe von Threads hat der Programmierer die Möglichkeit, die Parallelität eines Programms auszudrücken. Nun wäre es natürlich möglich, diesen Threads direkt vom Scheduler des Betriebssystems Rechenzeit zur Verfügung stellen zu lassen. Dies hat allerdings den Nachteil, daß ein sehr paralleles Programm beliebig viele Rechen-Resourcen absorbieren kann. Um weniger Resourcen zu benutzen, müßte das Programm verändert werden.

Aus diesem Grund bieten Solaris und andere Betriebssysteme die 'lightweight processes' (LWP) an.

Lightweight-Prozesse sind logische Prozessoren, die ein Prozeß während eines Programmlaufes anfordern und wieder freigeben kann und die die Threads dieses Prozesses ausführen. Das Scheduling der vorhandenen Threads auf den vorhandenen LWPs ist sehr fein steuerbar. Diesen LWPs wird vom Scheduler des Betriebssystems Rechenzeit zugeteilt — sie bestimmen also, wieviel Rechen-Resourcen ein Prozeß verbrauchen kann. Da durch das Binden jedes Threads an einen eigenen LWP logisch jeder Thread vom Scheduler des Betriebssystems ‘behandelt’ werden kann, ist das Modell mit LWPs eine echte Erweiterung des Modells ohne LWPs.

Thread-Operationen

Zustände

Jeder Thread besitzt:

- eine Thread ID, die diesen Thread unter den Threads dieses Prozesses eindeutig kennzeichnet, also etwa der Prozeßnummer in Unix entspricht
- einen Registerzustand, der nötig für die Rechnung eines Threads ist
- einen Stack, auf dem die lokalen Variablen und die Return-Adressen für Funktionsaufrufe gespeichert sind
- einer Signalmaske, die festlegt, welche Signale dieser Thread akzeptiert. Dies entspricht der Signalmaske eines Prozesses. Auf Signal-Verarbeitung wird in diesem Artikel nicht weiter eingegangen.
- eine Priorität, die beim Schedulen der Threads auf den LWPs berücksichtigt wird

In den ursprünglichen Beschreibungen der Thread-Library (und wohl auch in den ersten Versionen) gab es einen ‘Thread-local storage’, in dem Variablen gespeichert werden konnten, die für jeden Thread einen eigenen Wert haben. Dieses Konzept wurde von SUN — ohne die Benutzer auf diese Inkonsistenz aufmerksam zu machen — aufgegeben. Auch bei der Benutzung dieser ‘privaten’ Variablen — sie wurden durch **`#pragma unshared variable`** erzeugt — wurde der Benutzer nicht auf diese Änderung aufmerksam gemacht; ein Musterbeispiel an schlechtem Support. Leider ist der Ersatz, der für dieses Konzept geschaffen wurde — die Funktionen **`thr_keycreate()`**, **`thr_setspecific()`** und **`thr_getspecific()`** — eine wesentlich weniger elegante Lösung, die in bestehende Programme wesentlich schwieriger einzuarbeiten ist und bei der sich die Frage stellt, ob sie genauso effizient ist.

Interface

Auch das Interface zur Thread-Library wurde von SUN ohne Informationen umgestellt. Im Vergleich zu den ersten White-Papern änderten sich dabei nicht nur die Namen — nun beginnen alle Thread-Funktionen mit **`thr_`** statt mit **`thread_`**, sondern auch teilweise die Argumente. Im folgenden wird nur ein kurzer Überblick über einige der verfügbaren Funktionen gegeben.

Mit Hilfe der Funktion **`thr_create()`** kann ein neuer Thread erzeugt werden. Dies entspricht dem Aufruf einer Funktion mit Argumenten, der parallel zum ausrufenden Thread ausgeführt wird. Auf das Ergebnis der Funktion wird also nicht gewartet. Über die Parameter des Aufrufes kann z.B. die Stack-Größe des neuen Threads bestimmt werden. Es kann aber auch festgelegt werden, ob ein neuer LWP erzeugt und ob dieser fest an diesen Thread gebunden werden soll.

Funktionen, mit denen Threads (und LWPs) noch beeinflußt werden können sind z.B.: **`thr_exit()`**, **`thr_continue()`**, **`thr_join()`** und **`thr_setconcurrency()`**.

Mit Hilfe der Funktionen **`mutex_init()`**, **`mutex_destroy()`**, **`mutex_lock()`**, **`mutex_trylock()`** und **`mutex_unlock()`** können mutex Locks erzeugt und manipuliert werden. Ein Thread, der durch ein solches Lock blokkiert wird, führt kein buzy waiting durch, sondern schläft ein und wird später automatisch geweckt. Wie später noch gezeigt werden wird, ist dies auch mit einigen Laufzeitkosten verbunden. Diese Implementation hat den Vorteil, daß die wartenden Prozesse keine Rechenzeit verbrauchen — ein Vorteil der gerade ins Gewicht fällt, wenn der durch das Lock geschützte kritische Abschnitt relativ lang ist. Diese Funktionen können auch benutzt werden, um Prozesse miteinander zu synchronisieren.

Außerdem gibt es Funktionen **`cv_*`** für ‘conditional variables’, **`sema_*`** für Semaphoren und **`rw_*`** für multiple-readers, single-writer locks.

Performance

Nach dieser sehr kurzen Vorstellung des (großenteils gelungenen) Design von Threads, soll ein Test zur Performance der Thread-Library und eines SparcServers 1000 (mit 8 Prozessoren des Typs Supersparc mit 50MHz getaktet) durchgeführt werden. Um Vergleichmöglichkeiten zu haben, wird ein entsprechender Test auf einer Sequent Symmetry (mit 12 Prozessoren des Typs 80386 mit 16MHz getaktet) mit der dort zur Verfügung gestellten Library durchgeführt. Es gibt heute natürlich bereits wesentlich leistungsfähigere Sequents, solche standen dem Autor aber nicht zur Verfügung.

Bei dem Test müssen verschiedene Anzahlen von Threads eine globale Variable herunterzählen, wobei das Herunterzählen durch ein Lock geschützt ist. Mit der Erhöhung der Anzahl der Threads wird dieses Programm natürlich langsamer, da das Programm keine echte Parallelität besitzt und die Threads sich gegenseitig behindern. Um falschen Schlüssen vorzubeugen, sei darauf hingewiesen, daß die SUN einen 10mal größeren Wert als die Sequent herunterzählen muß!

In dem Programm für die SUN wird zum einen das von der Thread-Library zur Verfügung gestellte Lock benutzt (SUN1), zum anderen eine Implementation eines Locks in Assembler-Sprache mit buzy waiting, das in einer ‘leaf procedure’ realisiert ist (SUN2).

In dem Programm für die Sequent wird zum einen ein mittels einer Funktion (Sequent1), zum anderen ein mittels eines Makros (Sequent2) von der Library realisiertes Lock benutzt. Da das Erhöhen und Erniedrigen eines Zählers von der Sequent durch einen unteilbaren Befehl durchgeführt wird, braucht man diese Aktion nicht mit einem Lock zu schützen. In diesem Fall muß aber der Vergleich und das Herunterzählen eine unteilbare Aktion sein, so daß doch ein Lock benutzt werden muß. In vielen anderen Fällen ist dies allerdings nicht nötig.

Um die Kosten des Setzens und Löschens eines Locks abschätzen zu können, wurde die Laufzeit des Programms mit einem Thread auch ohne Lock gemessen.

Die für die Messungen benutzten Programme und die erhaltenen Zeiten sind auf der nächsten Seite aufgeführt.

Es ergibt sich, daß ein Thread auf der SUN, der das Lock der Library benutzt, 6,8mal langsamer ist als ein Thread ohne Lock — bei dem buzy waiting Lock ist der Faktor nur 2,9.

Ein Thread auf der Sequent, der Funktionen für das Lock benutzt, ist 4,6mal langsamer als ein Thread ohne Lock, bei der Implementation mittels eines Makro ist der Faktor nur 2.

Arbeiten mehrere Threads auf der SUN an der Aufgabe, so zeigt sich, daß die Performance im Vergleich zur Sequent drastisch absinkt. Mit 3 Threads ist die SUN 16,7mal langsamer als mit einem Thread ohne Lock. Bei der Sequent beträgt dieser Faktor nur 5,2.

Wieso die SUN mit 2 Threads und dem Lock der Library überhaupt keine Performance-Einbußen hat, ist dem Autor nicht klar.

Zusammenfassend läßt sich sagen, daß die von der Library der SUN zur Verfügung gestellten Locks relativ hohe Grundkosten haben. Da anders implementierte Locks allerdings ein sehr schlechtes Verhalten bei mehreren Threads haben und natürlich Rechenzeit beim Warten verbrauchen, ist es fraglich, ob es sinnvoll wäre, noch anders implementierte Locks in der Library anzubieten. Nach den Ergebnissen läßt sich auch vermuten, daß die Hardware der SUN sich nicht besonders gut verhält, wenn Speicherzellen von mehreren Prozessoren gleichzeitig benötigt werden. Ob dies so sein muß, da die Geschwindigkeiten der Prozessoren stärker gestiegen sind als die Geschwindigkeiten der Übertragungssysteme kann an dieser Stelle nicht beantwortet werden.

Programme, die häufig kurze unteilbare Aktionen durchführen, können durch diese Eigenschaft auf jeden Fall ein sehr schlechtes Speedup verhalten aufweisen bzw. wegen der hohen Kosten eines Locks immer sehr langsam sein.

Diese Erfahrung hat den Autor zum Durchführen dieses Benchmarks veranlaßt.

Programm für die SUN

```
#include <synch.h>
#include <thread.h>

mutex_t lock ;
long global_counter = 10000000 ;

void Count(void)
{
  for ( ; global_counter > 0 ; )
    {
      mutex_lock( &lock ) ;
      if ( global_counter>0 )
        global_counter-- ;
      mutex_unlock( &lock ) ;
    }
}

main()
{
  long i = 2 ; /*Anzahl der Threads*/
  thread_t id ;

  mutex_init( &lock,
               USYNC_THREAD, NULL ) ;
  for ( ; i>1 ; i-- )
    {
      thr_create(NULL, 0,
             (void *(*)(void *))Count,
             NULL, THR_BOUND, &id) ;
    }
  Count() ;
  return 0 ;
}
```

Programm für die Sequent

```
#include <parallel/microtask.h>
#include <parallel/parallel.h>

#pragma sequent_shared
slock_t lock ;
long global_counter = 1000000 ;
#pragma sequent_shared_end

void Count(void)
{
  for ( ; global_counter > 0 ; )
    {
      S_LOCK ( &lock ) ;
      if ( global_counter>0 )
        global_counter-- ;
      S_UNLOCK ( &lock ) ;
    }
}

main()
{
  long i = 2 ;/*Anzahl der Threads*/

  s_init_lock ( &lock ) ;
  m_set_procs ( i ) ;
  m_fork( Count ) ;
  return 0 ;
}
```

Laufzeiten

	SUN1	SUN2	Seqeunt1	Seqeunt2
ohne Lock	2.8s	2.8s	1.7s	1.7s
1 Thread.	19.3s	8.0s	7.9s	3.4s
2 Threads	19.5s	43s	12s	8s
3 Threads	47s	56s	13s	9
4 Threads	61s	62s	--	10s
5 Threads	65s	66s	--	12s
6 Threads	72s	71s	--	--
9 Threads	--	--	21s	17s

Indefinite summation in MuPAD

Paul Zimmermann

This short article describes the implementation of indefinite summation in MuPAD. Indefinite summation is useful in many areas of computer science (limit computations and complexity analysis for example).

Description of the problem

Given an expression *f* and a variable *x*, the problem of indefinite summation of *f* with respect to *x* is to find an expression *g* such that the following identity holds:

$$f(x) = g(x+1) - g(x)$$

For example if $f(x) = x^2 + 1$, then a possible solution is $g(x) = x^3/3 - x^2/2 + 7x/6$:

```
>> g:=x^3/3-x^2/2+7*x/6:
>> normal(subs(g,x=x+1)-g);
```

$$x^2 + 1$$

Like the companion problem of indefinite integration, the solution *g* is defined up to a constant factor, and enables one to compute some definite sums of *f*:

$$\sum_{k=m}^{n} f(k) = g(n+1) - g(m)$$

$$\sum_{k=m}^{n} f(k) = g(n+1) - g(m)$$

The class of rational functions

Several algorithms were developed for the indefinite summation of rational functions, especially by Paule and Abramow. To summarize, a closed-form expression for *g* does not always exist, unless the denominator of *f* splits into linear factors in the variable *x*. When a solution exists, then it is of the form *g=h+s*, where *h* is rational and *s* is a sum of derivatives of the ψ function (logarithmic derivative of the Gamma function). The part *s* may be zero. For example, the expression $1/(x^2-1)$ admits a rational sum:

```
>> sum(1/(x^2-1),x);
```

$$-\frac{1}{2x} - \frac{1}{2(x-1)}$$

but $(4x^2+3x+1)/x/(x+1)^2$ does not:

```
>> sum((4*x^2+3*x+1)/x/(x+1)^2,x);
```

$$-\frac{1}{x} + 4\mathrm{psi}(x+1, 0) + 2\mathrm{psi}(x+1, 1)$$

In this MuPAD output, the expression psi(x+1,0) represents the 0-th derivative of the ψ function taken at the point *x+1*, that is simply ψ(x+1), whereas psi(x+1,1) represents $\psi'(x+1)$.
When no closed-form sum over rational expressions and derivatives of the ψ function exists, the program simply returns the given input:

```
>> sum(1/(x^3+x+1),x);
```

$$\mathrm{sum}\left(\frac{1}{x^3+x+1}, x\right)$$

Limit computations

The limit of a rational sum can be easily computed once we know how to perform indefinite summation. For example, suppose we want to compute the limit of:

$$\sum_{k=1}^{n} \frac{1}{k(k+2)}$$

when n goes to infinity. We first compute an indefinite sum for *1/k/(k+2)*:

```
>> g := sum(1/k/(k+2),k):
```

from which we deduce the value of the sum from 1 to *n:*

```
>> subs(g,k=n+1)-subs(g,k=1);
```

$$-\frac{1}{2(n+1)}+\frac{1}{-2n-4}+3/4$$

and we easily deduce that the limit is 3/4.
Another nice example is the following sum:

$$\sum_{k=1}^{n}\frac{1}{k(k+2)^2}$$

The indefinite summation gives:

```
>> g:=sum(1/k/(k+2)^2,k);
```

$$-\frac{1}{4k}+\frac{\mathrm{psi}\ (k+2,1)}{2}+\frac{1}{-4k-4}$$

and therefore the sum equals:

```
>> subs(g,k=n+1)-subs(g,k=1);
```

$$\frac{\mathrm{psi}\ (n+3,1)}{2}+\frac{1}{-4n-8}+3/8$$

$$-\frac{1}{4(n+1)}-\frac{\mathrm{psi}\ (3,1)}{2}$$

Knowing that $\psi'(n)$ behaves like *1/n*, we deduce that the given sum admits as limit $3/8-\psi'(3)/2$, that is $1-\pi^2/12$, when n goes to infinity.

Conclusion and extensions

We have presented here a first stage of the indefinite summation in MuPAD. This implementation is currently limited to rational functions, but complete in this restricted domain. As shown above, it can help to deduce non trivial results.
Further developments include the extension to factors of degree two, for example

$$\sum\frac{1}{2-x^2}$$

should give as result

$$\frac{\sqrt{2}}{4}\psi(x+\sqrt{2})-\frac{\sqrt{2}}{4}\psi(x-\sqrt{2})$$

For this we will need a full implementation of ψ and its derivatives to all of the complex plane, because complex numbers can appear, for instance with:

$$\sum\frac{1}{1+x^2}$$

Another possibility of extension is towards the class of exponential expressions, for example to compute sums of geometric series:

$$\sum_k ka^k$$

Finally the most wanted extension is perhaps the indefinite summation of hypergeometric functions:

$$\sum_k F(n,k)$$

where *F(n,k)* is hypergeometric in n and k, that is *F(n+1,k)/F(n,k)* and *F(n,k+1)/F(n,k)* are both rational functions of n and k.The decision procedure found by Gosper enables one to compute such sums, like

$$\sum_k\binom{n+k}{k}$$

Symbolische Integration

Ralf Kraume

In diesem Artikel werden die grundsätzliche Vorgehensweise beim symbolischen Integrieren und die Teilalgorithmen in ihrer Wirkungsweise beschrieben.
Sei f zunächst eine rationale Funktion in der Unbestimmten x, mit der Eigenschaft $x' = 1$, über einem Konstantenkörper **C**. Für die Elemente des Konstantenkörpers ist die Ableitung 0. In einem Computeralgebra-System sind das im allgemeinen der Körper der rationalen Zahlen **Q**, algebraische Erweiterungen von **Q** und Körper, die durch Adjunktion konstanter transzendenter Elemente aus diesen entstehen. Wenn der Grad des Zählerpolynoms größer oder gleich dem Grad des Nennerpolynoms ist, kann durch Division ein Polynom abgespalten werden. Dies ist der polynomiale oder auch spezielle Anteil f_s von f. In dem restlichen Term, dem normalen Anteil f_n, ist dann der Grad des Zählerpolynoms kleiner als der Grad des Nennerpolynoms.
Beispiel: Es sei

$$f = \frac{x^5 + x^4 - 4x^3 - 4x^2 + 4x + 4}{x^4 - 4x^2 + 4} = x + 1 + \frac{x^3 - x^2 - 2x - 6}{(x^2 - 2)^2}$$

Die Hermite-Reduktion wird auf den normalen Anteil angewandt und liefert rationale Funktionen g und h so, daß $f = f_s + g' + h$, wobei der Nenner von h quadratfrei ist. Die rationale Funktion h wird auch logarithmischer Anteil genannt. Die Anwendung der Hermite-Reduktion auf f aus dem obigen Beispiel ergibt dann

$$f = x + 1 + \left(\frac{x + 2}{x^2 - 2}\right)' + \frac{x}{x^2 - 2}$$

wobei $g = (x + 2) / (x^2 - 2)$ und $h = x / (x^2 - 2)$ sind. Der logarithmische Anteil kann dann nach Partialbruchzerlegung sehr einfach integriert werden, wie das folgende Beispiel zeigt.

$$\int \frac{\sqrt{2}}{x^2 - 2} = \int \left(\frac{1/2}{x - \sqrt{2}} - \frac{1/2}{x + \sqrt{2}}\right) = \frac{1}{2}\ln(x - \sqrt{2}) - \frac{1}{2}\ln(x + \sqrt{2})$$

Wie jedoch die Integration des logarithmischen Anteils von f aus dem obigen Beispiel

$$\int h = \int \frac{x}{x^2 - 2} = \int \left(\frac{1/2}{x - \sqrt{2}} + \frac{1/2}{x + \sqrt{2}}\right) = \frac{1}{2}\ln(x - \sqrt{2}) + \frac{1}{2}\ln(x + \sqrt{2}) = \frac{1}{2}\ln(x^2 - 2)$$

zeigt, können Logarithmusfunktionen mit gleichen Faktoren zusammengefaßt werden. Ohne diese Zusammenfassung können im Integral algebraische Zahlen auftreten, die zur Darstellung des Integrals nicht notwendig sind. Auch ist es für Nennerpolynome vom Grad größer als 4 im allgemeinen nicht möglich, sie in Linearfaktoren zu zerlegen, also eine Partialbruchzerlegung mit linearen Nennern herzustellen. Der Rothstein-Trager Algorithmus liefert das Integral des logarithmischen Anteils und vermeidet von vornherein algebraische Zahlen, die zur Darstellung des Integrals nicht notwendig sind. Der spezielle Anteil ist für das obige Beispiel gliedweise zu integrieren und das gesamte Integral ist

$$\int f = \int f_s + g + \int h = \frac{1}{2}x^2 + x + \frac{x+2}{x^2-2} + \frac{1}{2}\ln\,(x^2-2)$$

Im Fall transzendenter Funktionen wie zum Beispiel

$$f = \frac{(x+2)\ (\exp(x)^3 + \exp(x)^2 - 1) - (x+1)\exp(x)}{\exp(x)^2 + \exp(x)} + \frac{x + (x+2)^2\,(\ln(x)+1)}{x\,(x+2)}$$

werden Logarithmus- und Exponentialfunktionen durch Unbestimmte ersetzt, die dieselben Ableitungseigenschaften haben. Mit Hilfe des Struktur-Theorems muß geprüft werden, ob diese Unbestimmten unabhängig voneinander sind. Sie heißen logarithmische beziehungsweise exponentielle Monome und werden dem Konstantenkörper **C** sukzessive adjungiert. Es sei x die letzte adjungierte Unbestimmte und $k(x)$ der durch die sukzessive Adjunktion entstandene Körper. Zähler und Nenner des Integranden sind Polynome in der Integrationsvariablen und den adjungierten Unbestimmten und werden als Polynome in der jeweils letzten Körpererweiterung behandelt.
Für das letzte Beispiel seien x transzendent über **Q** mit $x' = 1$, $x_1 = \ln(x)$ und $x_2 = \exp(x)$. Dann ist x_1 transzendent über $\mathbf{Q}(x)$ und x_2 ist transzendent über $\mathbf{Q}(x,x_1)$. Damit ist

$$f = \frac{(x^3 + 4x^2 + 4x)\,x_2^2 + (\,(x^2 + 4x + 4)\,x_1 + x^2 + 5x + 4)\,x_2 - x^3 - 4x^2 - 4x}{(x^2 + 2x)\,x_2} + \frac{1}{x_2 + 1}$$

Der Integrand f wird wie zuvor in einen speziellen Anteil und einen normalen Anteil zerlegt. Für jeden irreduziblen Faktor P_n des Nenners des normalen Anteils gilt $\gcd\,(P_n, P_n') = 1$, und der Grad des Zählerpolynoms ist kleiner als der Grad des Nennerpolynoms. Für jeden irreduziblen Faktor P_s des Nenners des speziellen Anteils gilt $\gcd\,(P_s, P_s') \neq 1$.
Der normale Anteil von f aus dem letzten Beispiel ist

$$f_n = \frac{1}{x_2 - 1}$$

Dann wird die Hermite-Reduktion angewendet und liefert rationale Funktionen g und h so, daß $f = f_s + g' + h$ ist. Es seien A und B teilerfremde Polynome in der letzten Körpererweiterung, und es sei $h = A/B$. Beim der Anwendung des Rothstein-Trager Algorithmus muß die Resultante

$$R\,(z) = \mathrm{res}_x\,(A - zB', B)$$

berechnet werden. Diese ist ein Polynom in der Unbestimmten z. Wenn dieses Polynom eine nicht konstante Nullstelle hat, ist f nicht elementar integrierbar.

In unserem Beispiel liefert der Rothstein-Trager Algorithmus, da der Nenner bereits quadratfrei ist,

$$\int f_n = x - \ln(x_2 + 1) = x - \ln(\exp(x) + 1)$$

Für den speziellen Anteil müssen je nach Typ der letzten adjungierten Unbestimmten verschiedene Verfahren angewendet werden. Ist dies die Integrationsvariable, so ist der spezielle Anteil wieder gliedweise zu integrieren und wir sind fertig. Wenn zuletzt ein logarithmisches Monom adjungiert worden ist, muß der Integrationsalgorithmus rekursiv angewendet werden. Die dann auftretenden Integranden sind Elemente von k. Wenn zuletzt ein exponentielles Monom adjungiert worden ist, muß eine Differentialgleichung gelöst werden, die Differentialgleichung von Risch. Zur Lösung dieser Differentialgleichung muß der Integrationsalgorithmus wie im vorherigen Fall auch wieder rekursiv angewendet werden.
Der spezielle Anteil von f aus dem letzten Beispiel ist

$$f_s = \frac{(x^3 + 4x^2 + 4x)\,x_2^2 + ((x^2 + 4x + 4)\,x_1 + x^2 + 5x + 4)\,x_2 - x^3 - 4x^2 - 4x}{(x^2 + 2x)\,x_2}$$
$$= (x + 2)\,x_2 + \frac{x + 2}{x}x_1 + \frac{x^2 + 5x + 4}{x^2 + 2x} - \frac{x + 2}{x_2}$$

Wenn die letzte Erweiterung ein exponentielles Monom ist, hat der spezielle Anteil $f_s \in k(x)$ die Form

$$f_s = \sum_{j=-t}^{r} p_j x^j$$

Nach dem Prinzip von Liouville hat, wenn f elementar integrierbar ist, f_s die Form

$$f_s = \left(\sum_{j=-t}^{r} p_j x^j\right)' + \sum_{i=1}^{m} c_i \frac{a_i'}{a_i} = \sum_{j=-t}^{r} (q_j' + ju'q_j)\,x^j + \sum_{i=1}^{m} c_i \frac{a_i'}{a_i}$$

wobei $u \in k$ mit $x' = u'x$, $q_{-t}, \ldots, q_r \in k$, $a_1, \ldots, a_m \in k$ und $c_1, \ldots, c_m \in \mathbf{C}$ sind. Durch Koeffizientenvergleich erhält man das Gleichungssystem

$$p_j = q_j' + ju'q_j \text{ für } r \geq j \geq 1 \text{ und } -1 \geq j \geq -t$$
$$p_0 = q_0' + \sum_{i=1}^{m} c_i \frac{a_i'}{a_i}$$

Dann ist f elementar integrierbar genau dann, wenn für $r \geq j \geq 1$ und $-1 \geq j \geq -t$ die Differentialgleichungen von Risch $p_j = q_j' + ju'q_j$ lösbar sind und p_0 elementar integrierbar ist.

Für unser Beispiel ergibt sich das Gleichungssystem

$$\begin{aligned} p_1 = {} & q_1' + q_1 && = && x+2 \\ p_0 = {} & q_0' + \sum_{i=1}^{m} c_i \frac{a_i'}{a_i} && = && \frac{x+2}{x} x_1 + \frac{x^2+5x+4}{x^2+2x} \\ p_{-1} = {} & q'_{-1} - q_{-1} && = && -x-2 \end{aligned}$$

Es sind also die Differentialgleichungen von Risch $q_1' + q_1 = x+2$ und $q'_{-1} - q_{-1} = -x-2$ zu lösen. Die Lösungen sind $q_1 = x+1$ und $q_{-1} = x+3$.
Weiter ist $p_0 = ((x+2)x_1)/x + (x^2+5x+4)/(x^2+2x)$ elementar zu integrieren. Der normale Anteil ist hier 0, es ist also nur noch der spezielle Anteil zu integrieren.
Wenn zuletzt ein logarithmisches Monom adjungiert worden ist, hat der spezielle Anteil $f_s \in k(x)$ die Form

$$f_s = \sum_{j=0}^{\bar{r}} \bar{p}_j x^j$$

Nach dem Prinzip von Liouville hat f_s die Form

$$f_s = \left(\sum_{j=0}^{\tilde{r}} \bar{q}_j x^j \right)' + \sum_{i=1}^{\bar{m}} \bar{c}_i \frac{\bar{a}_i'}{\bar{a}_i}$$

wobei $\bar{q}_{-t}, \ldots, \bar{q}_r \in k$, $\bar{a}_1, \ldots, \bar{a}_m \in k$ und $\bar{c}_1, \ldots, \bar{c}_m \in \mathbf{C}$ sind. Es gilt $\tilde{r} = \bar{r} + 1$ und es ergibt sich

$$\begin{aligned} \sum_{j=0}^{\bar{r}} \bar{p}_j x^j = {} & \left(\sum_{j=0}^{\bar{r}+1} \bar{q}_j x^j \right)' + \sum_{i=1}^{\bar{m}} \bar{c}_i \frac{\bar{a}_i'}{\bar{a}_i} \\ = {} & \sum_{j=0}^{\bar{r}+1} \bar{q}_j' x^j + \sum_{j=1}^{\bar{r}+1} j \bar{q}_j x^{j-1} x' + \sum_{i=1}^{\bar{m}} \bar{c}_i \frac{\bar{a}_i'}{\bar{a}_i} \\ = {} & \bar{q}'_{\bar{r}+1} x^{\bar{r}+1} + \sum_{j=0}^{\bar{r}} \left((j+1) \bar{q}_{j+1} x' + \bar{q}_j' \right) x^j + \sum_{i=1}^{\bar{m}} \bar{c}_i \frac{\bar{a}_i'}{\bar{a}_i} \end{aligned}$$

Durch Koeffizientenvergleich erhält man das Gleichungssystem

$$\begin{aligned} 0 &= \bar{q}'_{\bar{r}+1} \\ \bar{p}_{\bar{r}} &= (\bar{r}+1) \bar{q}_{\bar{r}+1} x' + \bar{q}_{\bar{r}}' \\ \bar{p}_{\bar{r}-1} &= \bar{r} \bar{q}_{\bar{r}} x' + \bar{q}'_{\bar{r}-1} \\ &\ldots \\ \bar{p}_1 &= 2 \bar{q}_2 x' + \bar{q}_1' \\ \bar{p}_0 &= \bar{q}_1 x' + \bar{q}_0' + \sum_{i=1}^{\bar{m}} \bar{c}_i \frac{\bar{a}_i'}{\bar{a}_i} \end{aligned}$$

Für $p_0 = ((x+2)x_1)/x + (x^2+5x+4)/(x^2+2x)$ aus dem letzten Beispiel ergibt sich das Gleichungssystem

$$\begin{aligned} 0 &= \bar{q}_2{}' \\ \bar{p}_1 &= 2\bar{q}_2 x' + \bar{q}_1{}' = \frac{x+2}{x} \\ \bar{p}_0 &= \bar{q}_1 x' + \bar{q}_0{}' + \sum_{i=1}^{\bar{m}} \bar{c}_i \frac{\bar{a}_i{}'}{\bar{a}_i} = \frac{x^2+5x+4}{x^2+2x} \end{aligned}$$

Aus der ersten Gleichung folgt, daß $\bar{q}_2$ eine Konstante ist. Die Integration von $(x+2)/x = 1+2/x$, wozu der Rothstein–Trager Algorithmus angewendet wird, ergibt $x+2\ln(x)$. Also ist $\bar{q}_2 = 1$ und $\bar{q}_1$ ist hierdurch bis auf eine Konstante c bestimmt. Es gilt also $\bar{q}_1 = x+c$. Es ist

$$\int \bar{p}_0 - xx'_1 = \int \frac{x^2+5x+4}{x^2+2x} - 1 \int \frac{3x+4}{x^2+2x} = 2\ln(x) + \ln(x+2)$$

Also ist $c = 2$ und somit $\bar{q}_1 = x+2$, $\bar{q}_0 = 0$ sowie $\bar{m} = 1$, $\bar{c}_1 = 1$ und $\bar{a}_0 = x+2$. Also ergibt sich

$$\begin{aligned} \int f &= \int f_n + q_1 x_2 + q_{-1} x_2^{-1} + \bar{q}_2 x_1^2 + \bar{q}_1 x_1 + \bar{q}_0 + \bar{c}_1 \ln(\bar{a}_1) \\ &= x - \ln(\exp(x)+1) + (x+1)\exp(x) + \frac{x+3}{\exp(x)} + \ln(x)^2 + (x+2)\ln(x) + \ln(x+2) \end{aligned}$$

Some Variations on the Mandelbrot Set

Sergei Ivanov

Computer realizations of Irwin Jungreis' algorithm for constructing the conformal mapping from the complement of the disc onto the complement of the Mandelbrot set of $z^2 + c$ are described. The realization is based on solving a large nonlinear system of algebraic equations by methods of computer algebra. Some special procedures for treating this problem were needed, in particular, a MuPAD procedure for graphical presentation of conformal mappings. Some aspects of using computer algebra systems for problems requiring a lot of symbolic computation are discussed.[1]

Introduction

As it is known the Mandelbrot set of iterations $z^2 + c$ is the set of values c in the complex plane for which the Julia set [1] is connected. In this article we describe a realization, by use of computer algebra, of Irwin Jungreis [2] algorithm for building the conformal mapping from the complement of the disc onto the complement of the Mandelbrot set.

There are three ways to compute a conformal mapping (by a Laurent series):

16 Take a lot of paper and compute it by hand.

17 Write a complicated program, in C or a similar language, to compute the conformal mapping.

18 Use a computer algebra system for writing a simple program that realizes Jungreis' algorithm.

In his article [2] I. Jungreis mentioned that he computed 4096 coefficients of this conformal mapping. I do not know what computation method he used, I will use here the, in my opinion, most simple method **3**.

To realize method **3** I used MuPAD 1.2 and Maple V on the following hardware: IBM AT 386/40 Mhz, 4 MB RAM, under Windows 3.1 and a Sun SPARCclassic 1.

On the IBM I have computed 25 coefficients of the conformal mapping with 4 iterations $z^2 + c$ in 944 seconds and on the Sun 50 coefficients with the same number of iterations in 2871 seconds. The problem requires high precision computation dealing with numbers more than 20 digits long. Such computations take a lot of memory, for instance on the Sun the computation of 50 coefficients took nearly 32 MB of RAM (Random Access Memory). Extending the RAM, it is possible to compute the conformal mapping up to a very good precision.

Basis of the Algorithm

At last, after long time of consulting with my father, I have understood the sense of Jungreis algorithm.

As far as I know the Mandelbrot set definition is:

The *Mandelbrot set M* is the subset of **C** defined as $\{c \in \mathbf{C} | J(f_c) \text{ is connected}\}$, where $J(f_c)$ is the Julia set of $f_c(z) = z^2 + c$.

In order to build the conformal mapping from the complement of the disc onto the complement of the Mandelbrot set, we first need one of Jungreis formulas [2].

---

1. Authors Address: Institute of Appl. Math. & Mech., Luxemburg st. 74, Donetsk 340114, Ukraine

Let p_n be the n^{th} iteration of $p_1(z) = z^2 + c$. The polynomials $f^n(c) = p_n(0)$ will play the main role in the computations for the corresponding algorithm. We easily find $f^1(c) = c$, $f^2(c) = c^2 + c$, $f^3(c) = (c^2 + c)^2 + c$, $f^4(c) = ((c^2 + c)^2 + c)^2 + c$. Using `fsolve` in Maple, or a similar procedure in MuPAD, it's possible to find quickly the zeros of the polynomial $f^n(c)$ in order to plot them in the complex plane. These zeros give a very good approximation for the boundary of the Mandelbrot set.
The desired conformal mapping can be found approximately in the following form [2]:

$$\Psi_k(z, N) = z + \sum_{j=0}^{N} \frac{b_j}{z^j}$$

I. Jungreis proved that

$$f^k\left(z + \sum_{j=0}^{\infty} \frac{b_j}{z^j}\right) = z^{2^{k-1}}$$

which means that approximation of the coefficients of the rational function $f^k(\Psi_k(z, N))$ leads to zero, except for the leading one. Here k is the number of iterations of $z^2 + c$. We proceed with $k = 4$. The corresponding system of $N + 1$ algebraic equations in $N + 1$ unknowns (some equations have more than 100 terms) can be solved, for example in Maple. By this the coefficients $b_0, b_1, \dots, b_N$ of the conformal mapping $\Psi_k(z, N)$ can be found. The number of iterations k and the number of coefficients N are responsible for the precision of the conformal mapping $\Psi_k(z, N)$. To increase this precision one highly depends on the hardware capabilities (size of RAM and CPU speed).
With growth of N the system of equations will increase considerably. For $N = 25$ and $k = 4$ some equations contain more than 100 terms. So, it seems completely impossible for me to use either method **1** or **2** as described in the introduction.
On an IBM AT I solved this system for $N = 25$ and on a Sun for 51 coefficients.
To make my program faster, I replaced the Maple procedure `coeff` by some own procedure `ratcoeff` which reduced the time of computation by 100 times or so. Unfortunately, Maple turned out neither to be very suitable for plotting the resulting conformal mapping nor for writing my own graphics procedure to do that. So, I wrote my own conformal mapping plot function in MuPAD which has outstanding graphic capabilities[1].

Using the Algorithm

Well, after one gets through all these mathematical tricks, one can use computer algebra to get to the goal. In formula (1) let $k = 4$ and let $N = 5$ to be the number of coefficients of the Laurent series. Then we obtain the following system of six equations:

$4 + 8\, b_0 = 0,$

$28\, b_0 + 28\, b_0^2 + 6 + 8\, b_1 = 0,$

$84\, b_0^2 + 8\, b_2 + 56\, b_0^3 + 28\, b_1 + 56\, b_1 b_0 + 6 + 36\, b_0 = 0,$

$30\, b_0 + 56\, b_2 b_0 + 70\, b_0^4 + 90\, b_0^2 + 8\, b_3 + 36\, b_1 + 5 + 28\, b_1^2 + 168\, b_1 b_0^2 + 168\, b_1 b_0 + 140\, b_0^3 +$
$28\, b_2 = 0$

$56\, b_3 b_0 + 2 + 30\, b_1 + 168\, b_2 b_0 + 84\, b_1^2 + 28\, b_3 + 56\, b_1 b_2 + 120\, b_0^3 + 168\, b_0 b_1^2 + 36\, b_2 +$
$280\, b_1 b_0^3 + 60\, b_0^2 + 8\, b_4 + 56\, b_0^5 + 20\, b_0 + 420\, b_1 b_0^2 + 168\, b_2 b_0^2 + 180\, b_1 b_0 + 140\, b_0^4 = 0,$

1. Sergei wrote a complete library for graphic primitives not knowing that at the same time we were implementing these features in the MuPAD kernel.

$1 + 60\,b_0^3 + 90\,b_0^4 + 84\,b_0^5 + 420\,b_0^2 b_1^2 + 56\,b_1^3 + 280\,b_0^4 b_1 + 28\,b_0^6 + 20\,b_1 + 36\,b_3 + 30\,b_2 +$
$28\,b_4 + 8\,b_5 + 6\,b_0 + 28\,b_2^2 + 336\,b_1 b_0 b_2 + 120\,b_1 b_0 + 90\,b_1^2 + 168\,b_3 b_0 + 180\,b_2 b_0 + 168\,b_1 b_2 +$
$560\,b_1 b_0^3 + 360\,b_1 b_0^2 + 280\,b_2 b_0^3 + 420\,b_2 b_0^2 + 168\,b_3 b_0^2 + 30\,b_0^2 + 56\,b_4 b_0 + 420\,b_0 b_1^2 +$
$56\,b_1 b_3 = 0.$

Using Maples standard procedure `solve` we get the following solution in 20 seconds:

$$b_0 = -\frac{1}{2} \qquad b_1 = \frac{1}{8} \qquad b_2 = -\frac{1}{4} \qquad b_3 = \frac{15}{128} \qquad b_4 = 0 \qquad b_5 = -\frac{47}{1024}$$

The resulting approximation for the conformal mapping of the complement of the disc onto the complement of the Mandelbrot set then is

$$\Psi_4(z, 5) = z - \frac{1}{2} + \frac{1}{8}z^{-1} - \frac{1}{4}z^{-2} + \frac{15}{128}z^{-3} - \frac{47}{1024}z^{-5}$$

How all this was done

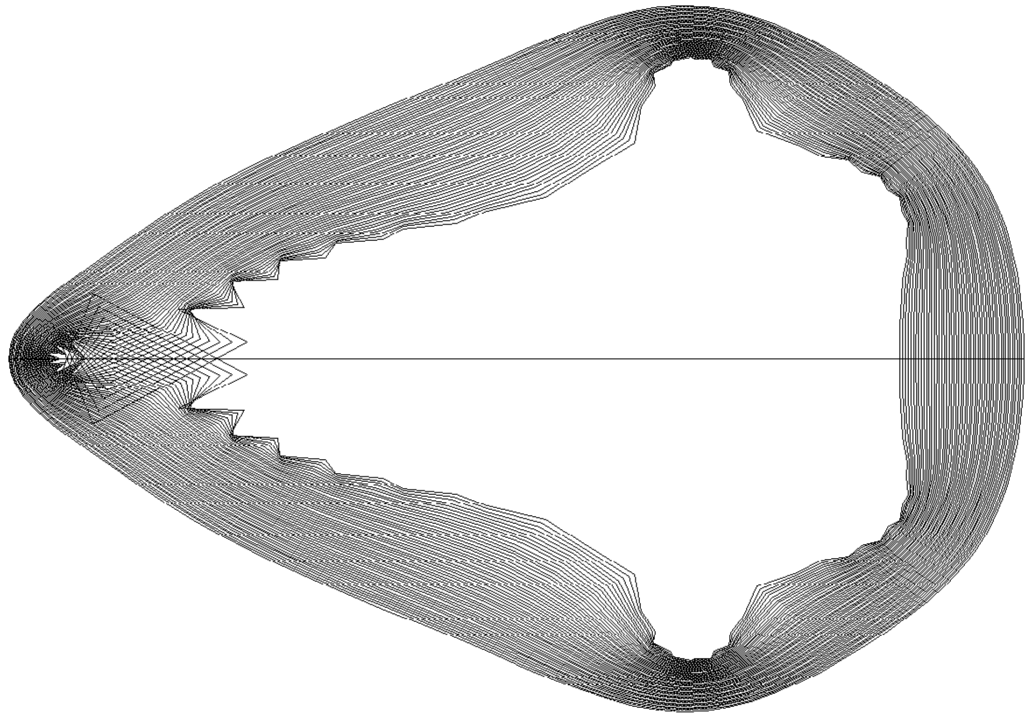
Of course, it is impossible to obtain a reasonable conformal mapping by using only the methods I described above; even using a computer algebra system it is very hard to do all these computations interactively. So, you need to make the process automatic.
For this aim I wrote a whole computer algebra package. There you simply specify the value of N (using a function `SetCoefficientsNumber(N)`, the number of iterations of $z^2 + c$ (using `SetIteration-sNumber(N)`. Then the coefficients are computed by `FindCoefficients()`, the result are the coefficients of the Laurent series. To get the points of the conformal mapping you call `InsertCoefficients()`. Then you can either pipe the computed values in some program to make a pretty plot of the conformal mapping (either in C or a similar language), or you can use my own MuPAD `conformal` function which I wrote to plot conformal mappings. Therein, due to the outstanding graphic capabilities of MuPAD [1], you can variate coloring, scaling, smoothness, plotting style and so on. This results in *very pretty* pictures. The codes (the Maple as well as the MuPAD programs) you can obtain from me by email using the address below.

Conclusion

For inexperienced users computer algebra systems (like MuPAD and Maple) are very simple to use, with these they can quickly solve bulky mathematical problems. Moreover using such systems is much simpler than to study some mathematical theories. Good graphic capabilities, systems like MuPAD make available, can serve for interactive graphic work and image processing. I've experimented a lot with *computer algebra* packages and sometimes it's more interesting than *computer games*.
You can reach me by e-mail address: `serg@iamm.ac.msk.su` and I will be very glad to present my packages for doing all the things I described above.
I was able to consult with many experts from the MuPAD group. I am grateful for their suggestions, comments, corrections and criticisms which greatly helped improve my first article.

Bibliography

[1] P. Blanchard, Complex analytic dynamics on the Riemann sphere. *Bulletin AMS, v. 1, N. 1, 1984, 85-141*
[2] I. Jungreis, The uniformization of the complement of the Mandelbrot set. *Duke Math. J., v.52, no. 4, 1985, 935-938*
[3] B. Fuchssteiner et. al., *MuPAD Tutorial* Birkhäuser, Basel, 1994



Conformal Mapping: The Complement of the Disc on the Complement of the Mandelbrot Set

Lindenmayer-Systeme in MuPAD

Klaus Drescher

Lindenmayer-Systeme sind formale Systeme, mit denen das Wachstum von Pflanzen modelliert werden kann. Weiter kann eine Vielzahl selbstähnlicher Kurven beschrieben werden.

L-Systeme

Ein *L-System* beschreibt – ähnlich wie eine Grammatik – eine Sprache. Ein L-System ist ein Tripel $L = (V, \omega, P)$ mit einem endlichen Alphabet V, einem Axiom $\omega \in V^+$ und einer Funktion $P : V \rightarrow V^*$. Hierbei sei V^* die Menge aller Worte über V und V^+ die Menge aller nichtleeren Worte. P nennt man auch die *Produktionen* von L.
Sei $u = a_1 ... a_n \in V^*$, dann heißt das Wort $v = P(a_1)...P(a_n)$ *aus u abgeleitet*, man schreibt dann $u \Rightarrow v$. Ein Wort $u \in V^*$ heißt *aus L erzeugt*, falls es eine Folge von Ableitungen

$$\omega \Rightarrow u_1 \Rightarrow u_2 \Rightarrow ... \Rightarrow u$$

gibt, falls also u in endlich vielen Schritten aus ω abgeleitet werden kann.
Ein Beispiel: Gegeben das Alphabet V = {**`L`**, **`R`**, **`+`**, **`-`**}, das Axiom ω = **`L`** und die Produktionen

```
L → L+R+
R → -L-R
+ → +
- → -
```

Dann können aus dem Axiom folgende Worte abgeleitet werden:

```
L
L+R+
L+R++-L-R+
L+R++-L-R++-L+R+--L-R+
...
```

Ein L-System hat viel Ähnlichkeit mit einer kontextfreien Grammatik, nur gibt es keine Terminalsymbole, weiter werden in einem Ableitungsschritt alle möglichen Ableitungen parallel durchgeführt.
Neben den oben beschriebenen kontextfreien, deterministischen L-Systemen betrachtet man noch kontextsensitive und nichtdeterministische L-System, siehe [1].

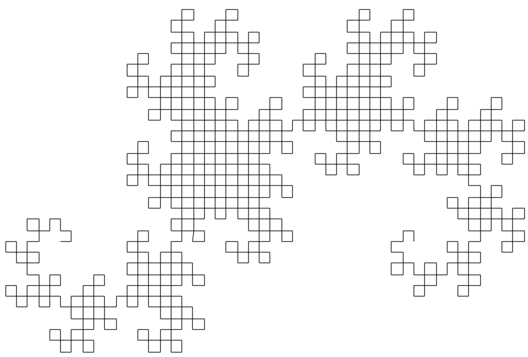
Interpretation als Kurve

Die von einem L-System erzeugten Worte können als Kurven interpretiert werden. Hierzu wird ein spezielles Grafik-Device – das *Turtle* – verwendet. Ein Turtle ist ein Geschöpf, welches in Flatland (einer 2-dimensionalen Welt) lebt. Es kann sich links oder rechts um seine Achse drehen und geradeaus kriechen und dabei eine Spur hinterlassen oder auch nicht. Weiter kann es seine Farbe ändern. Die Länge der Vorwärtsbewegung ist immer fest, genau wie der Drehwinkel. Der Zustand eines einfachen Turtles wird durch Position, Richtung und Farbe beschrieben. Fortgeschrittene Exemplare dieser Gattung können sich ihren Zustand merken und später diesen Zustand wieder annehmen.
Ein Turtle gehorcht folgenden Kommandos:

- **`line`**: vorwärts, dabei Linie ziehend,
- **`move`**: vorwärts ohne Linie,
- **`left`**: links herum drehen,
- **`right`**: rechts herum drehen,
- **`color`**$_c$: Farbe c annehmen,
- **`push`**: Zustand merken
- **`pop`**: alten Zustand annehmen

Jedem Zeichen des Alphabets eines L-Systems wird nun ein Kommando zugeordnet, z.B.

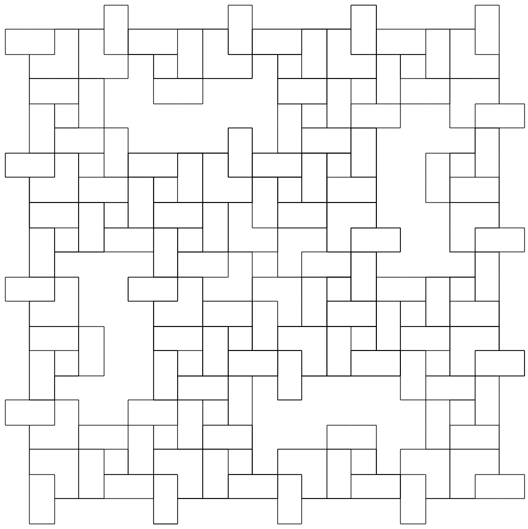
`L` →**`line`**
`R` →**`line`**
`+` →**`left`**
`-` →**`right`**.



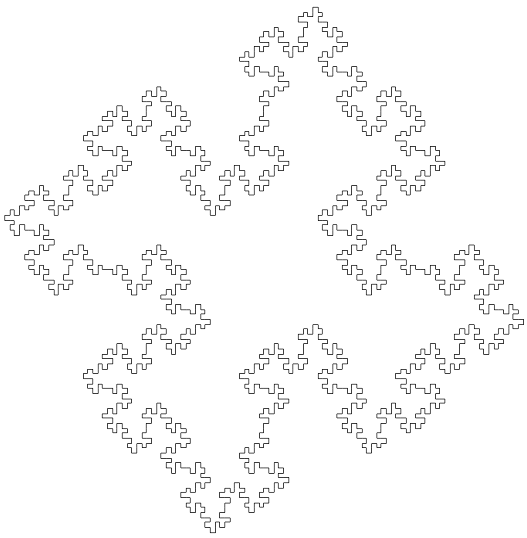
Ein Wort kann als Bewegung eines Turtles interpretiert werden, indem es als Anweisungsfolge für das Turtle verwendet wird, indem also das Wort **`L+R+`** in die Anweisungen **`line left line left`** übersetzt wird.
Die erste Abbildung zeigt die Kurve, die mit dieser Interpretation aus der 11ten Ableitung des Axioms des Beispiels gewonnen wurde.
Bei den folgenden Beispielen besteht das Alphabet immer aus $V = \{$**`L`**, **`R`**, **`+`**, **`-`**, **`[`**, **`]`**$\}$ und wird wie folgt interpretiert:

`L` $\rightarrow$ **`line`**
`R` $\rightarrow$ **`line`**
`+` $\rightarrow$ **`left`**
`-` $\rightarrow$ **`right`**
`[` $\rightarrow$ **`push`**
`]` $\rightarrow$ **`pop`**

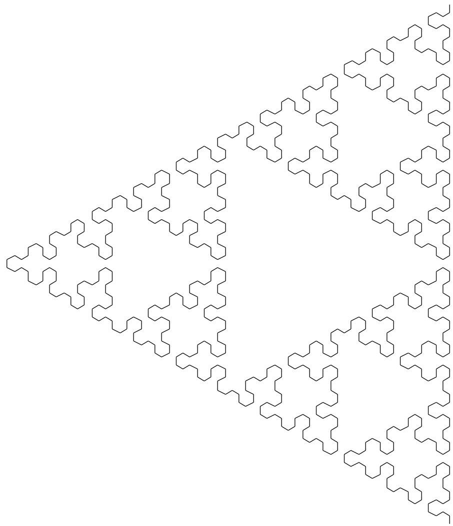
Wenn eine Produktion für ein Zeichen x nicht explizit angegeben wird, so wird immer die Regel $x \rightarrow x$ angenommen. Bei den Beispielen wird neben dem L-System auch der Drehwinkel des Turtles als Wert δ angegeben.



δ=90
`w = L-L-L-L`
`L` $\rightarrow$ **`LL-L+L-L-LL`**

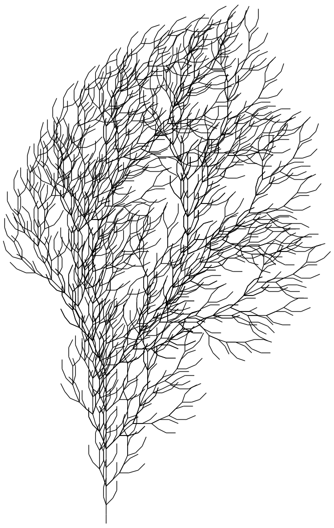


δ=90
`w = L-L-L-L`
`L` $\rightarrow$ **`L-L+L+LL-L-L+L`**

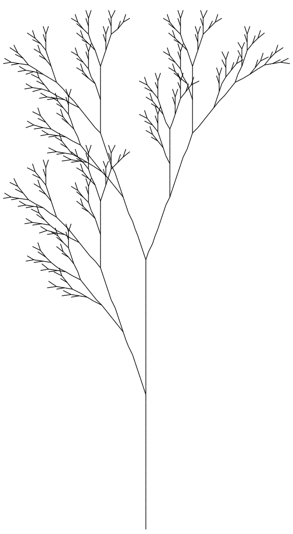


δ=60
`w = R`
`L` $\rightarrow$ **`R+L+R`**
`R` $\rightarrow$ **`L-R-L`**

Ohne die Kommandos **push** und **pop** wird immer nur ein durchgehender Pfad erzeugt. Verzweigende Strukturen, wie sie für Pflanzen typisch sind, können nur mit den Operationen **push** und **pop** erzeugt werden.



$\delta=22{,}5$
w = L
L $\rightarrow$ LL−[−L+L+L]+[+L−L−L]



$\delta=20$
w = L
L $\rightarrow$ R[+L]R[−L]+L
R $\rightarrow$ RR

In [1] werden neben 2-dimensionalen Turtles auch 3-dimensionale betrachtet, bei denen neben Drehungen auch Roll- und Kippbewegungen möglich sind. Damit können erstaunlich realistisch wirkende Bilder von Pflanzen erzeugt werden.

L-Systeme in MuPAD

L-Systeme und Turtles können in der neusten Version von MuPAD dank der neuen Grafik-Primitiven und der Domains leicht implementiert werden. Eine Implementation ist in der **share**-Library enthalten und kann mittels **loadlib("Lsys")** geladen werden. Sie wird im Dokument **Lsys.ps** im Verzeichnis **share/doc/extensions** der ftp-Distribution beschrieben. Diese Implementation enthält allerdings die Turtle-Kommandos **push** und **pop** noch nicht. Eine erweiterte Implementation kann der Autor auf Wunsch zusenden (**kg@uni-paderborn.de**).
Die obigen Bilder wurden mit der erweiterten Implementation erzeugt. Die erste Abbildung wurde z.B. wie folgt erzeugt:

```
>> loadlib("Lsys"):
>> l:= Lsys(90, "L", "L"="R+R+",
   "R"="-L-R", "L"=line, "R"=line):
>> Lsys::plot(l, 10):
```

Das erste Argument von **Lsys** ist der Drehwinkel δ, das zweite das Axiom ω, es folgen die Produktionen und die Interpretation der Zeichen als Turtle-Kommandos. Mit dem Kommando **Lsys::plot(l,10)** wird die 10te Generation des Axioms erzeugt und ausgegeben.

Literatur

[1] P. Prusinkiewicz, A. Lindenmayer: *The Algorithmic Beauty of Plants*, Springer 1990

New Features in MuPAD 1.2.1

Oliver Kluge

Dynamic Modules

Beside the conventional concept of libraries written in the language of the CA system, MuPAD offers another library concept, called dynamic modules. Users can implement algorithms in the C programming language (in the near future moreover in C++) to extend the MuPAD system by new functions. The advantages are:
- **Speed**. Functions written in the MuPAD programming language are executed by MuPAD's interpreter, whereas modules are compiled into machine code.
- Users can use the **flexibility** of C (C++) and have access to all internal functions and data structures of the MuPAD kernel.

Dynamic modules forces no interprocess communication, they are directly linked to the MuPAD kernel on demand at run time. It is also possible to unload modules at run time. This is why we speak of dynamic modules. The method used is called dynamic linking. Because the operating system must support dynamic linking, modules are available only for the following platforms:
- Sun SPARC : SunOS 4.1.x / SunOS 5.x
- Silicon Graphics: Irix 5.1 or later
- IBM RS/6000 : AIX 3.1 or later

A paper written in German is included in the ftp distribution. You will find a brief introduction in English in the on-line demo.

The Linear Algebra Package

This package is completely embedded in MuPAD's domain concept and therefore the package is based on the domains package (see below). The linalg package is also an example to show the user how to use the domains package and how to implement polymorphic algorithms. The linalg package has some predefined domains and categories: MatrixCat(R), SquareMatrixCat(R), Matrix(R), SquareMatrix(R) where R is the domain of the coefficients. The user can use standard domains, for example Integer or Rational, or can define his own domain for the coefficients. For example, the gaussian elimination works not only on fields, it works also on integral domains. Here comes a selection of functions provided by linalg: gaussian elimination, Gauss-Jordan elimination, solving systems of linear equations, inverting matrices, computing the norm, the rank and the determinant of a matrix, computing the Sylvester matrix, computing a basis for a vector space, QR factorization of regular matrices and many operations on matrices.
A paper written in German is included in the ftp distribution. You will find a brief introduction in English in the on-line demo.

Domains Package

With version 1.2 the data type `domain' was introduced in MuPAD. A domain represents a user-defined abstract data type, the operations of such a type are defined as `methods' of the domain. These methods overload (redefine) the operators and system functions of the MuPAD kernel such that the user-defined methods are applied to the domain elements instead of the "default" system functions.
A domain is used to represent an algebraic structure, for example an integer residue class ring. The `domains' library package supports the construction of new algebraic structures. It allows an easy and consistent definition of new domains, which may depend on parameters. New domains are created by so-called `domain constructors'. One may for example define a domain constructor which creates all integer residue class rings $\mathbf{Z}_n$ dependent upon the modulus n.
Domains which have a similar mathematical structure may be members of a `category'. A category adds a level of abstraction because it postulates conditions which must hold for a domain in order to become a valid member of

the category. Operations may be defined for all members of a category based on the assumptions and basic operations of that category. The residue class rings for example are members of the category `Ring', in addition they are members of the category `Field' if the modulus is a prime. Categories may also depend on parameters and are created by `category constructors'.
Attributes of domains and categories are defined in terms of so-called `axioms'. Axioms state properties of domains or categories. They are defined by `axiom constructors'.
The categories defined so far for the most part follow the conventions of modern algebra. There are some properties of the categories which differ from the "classical" non-constructive theory of modern algebra because these properties are not constructive or can't be constructed efficiently. The category hierarchy of the domains package is quite similar to (part of) the category hierarchy of AXIOM.
Two papers are included in the ftp distribution. There is also a brief introduction in English in the on-line demo.

Integration

The procedure int(f,x) integrates the argument f with respect to the variable x.If the tower of field extensions is a transcendental elementary extension of the rational numbers then it is possible to decide whether the argument f is elementary integrable or not and to determine the integral.
At the moment MuPAD does not provide any arithmetic for algebraic numbers and also algebraic relationships between transcendental numbers are not implemented.Hence, if the field of constants contains algebraic or transcendental numbers problems might occur.
The integration algorithm is also used if the tower of field extensions contains algebraic dependent extensions. But then the argument f is not necessary not elementary integrable if the procedure call is returned.
Furthermore, two other new procedures are available: simplify([f_1,...],x) simplifies a list of expressions respectively represents the expressions with respect to a tower of elementary transcendental extensions. And equal([f_1,...],x) tests expressions for equality.

Number Theory

This package contains about 30 functions. A short selection of functions provided by this package: chinese remainder; hensel lifting; computing the Carmichael and Fibonacci numbers, the Jacobi and Legendre symbol; solving linear congruences; computing roots. The corresponding helppages are included in the ftp distribution.

Miscellaneous

Indefinite summation of rational functions, finding integer roots of a polynomial over Q[x], a very comfortable possibility to check the type of an expression, Lindenmayer-Systems and 2D-Turtle graphics are included.
Informations about new developments, automatic access to the ftp server, demonstrations, interactive license form and automatic processing of license agreements are now available via the WWW service. The document URL of the MuPAD - Home Page is:

`http://math-www.uni-paderborn.de/~cube/`

MuPAD is now also available for DECstation / Ultrix 4.2 or later, PC / FreeBSD 1.0 or later, and Silicon Graphics / Irix 5.1.1 or later.

FORTHCOMING ATTRACTIONS

- Reference manual in English published by Birkhäuser in spring 1995. On-line version will be available at the same time.
- Motif version combined with a new tool communication mechanism and protocol.
- Parallel version for Sun multiprocessor machines.

Neues in der MuPAD-Grafik

Thorsten Schulze

In diesem Artikel werden die Neuerungen der Grafikschnittstelle des Computeralgebrasystems MuPAD vorgestellt. Diese Neuerungen umfassen einerseits die Möglichkeit, Listen von Grafikprimitiven darzustellen. Andererseits wird dem Benutzer die Möglichkeit gegeben, grafische Objekte mit Hilfe einer selbstdefinierten Prozedur einzufärben.

Grafikprimitive

In bisherigen Versionen des Computeralgebrasystems MuPAD konnten nur parametrisierte zwei- und dreidimensionale Kurven sowie dreidimensionale Flächen grafisch dargestellt werden. Dieses hat sich mit der Freigabe der aktuellen MuPAD-Version geändert, denn nun können beliebige Listen von Grafikprimitiven geplottet werden. Die Grundeinheit bei den Grafikprimitiven wird durch die Funktion `point()` zur Verfügung gestellt, die dazu dient einen zwei- oder dreidimensionalen Punkt zu spezifizieren. Je nach Dimension der Szene enthält die Grundstruktur eines Punktes die x-, y- und gegebenenfalls die z-Koordinate und eine Option, welche die Farbe dieses Punktes beschreibt. Die beiden folgenden Befehle spezifizieren zwei Punkte, von denen dem letzteren eine Farbe zugewiesen wird.

```
a := point(1,1):
b := point(1,2,3,Color=[0,1,0]):
```

Mit Hilfe von Punkten können die höheren Grafikprimitiven, die Polygone, erzeugt werden. Polygone werden durch die Systemfunktion `polygon()` definiert und setzen sich aus einer beliebigen Anzahl von Punkten zusammen. Zudem gibt es optionale Parameter, mit denen man einerseits die Farbe des Polygons angeben kann und andererseits bestimmen kann, ob das Polygon geschlossen und/oder gefüllt sein soll. Die beiden letztgenannten Optionen stehen zur Zeit jedoch nur für Dreiecke zur Verfügung, da höhere Polygone vor der grafischen Darstellung erst trianguliert werden müßten, und die dazu benötigten Algorithmen in der aktuellen Version noch nicht implementiert sind. Die folgenden Befehle dienen dazu, ein gefülltes, dreidimensionales Dreieck zu spezifizieren:

```
a := point(-1,0,0):
b := point(1,0,0):
c := point(1,1,2):
triangle := polygon(a,b,c,
      Closed = TRUE, Filled = TRUE);
```

Beispiele

In diesem Abschnitt werden einige Beispiele für Listen von Grafikprimitiven angegeben. Das folgende Beispiel erzeugt ein sogenanntes Sierpinski-Dreieck:

```
#
| Calculates the point in the middle
| of the line connecting a and b.
#
mid_point := proc(a, b)
   local x, y;
begin
   x := op(a,1)+(op(b,1)-op(a,1))/2:
   y := op(a,2)+(op(b,2)-op(a,2))/2:
   point(x,y):
end_proc:


#
| Recursive Sierpinski-triangle
#
sier := proc(a, b, c, n)
   local a1, b1, c1;
begin
   if n = 0 then
      polygon(a, b, c, Closed = TRUE):
   else
      a1 := mid_point(a, b):
      b1 := mid_point(b, c):
      c1 := mid_point(c, a):
      sier(a, a1, c1, n-1),
      sier(a1, b, b1, n-1),
      sier(b1, c, c1, n-1):
   end_if:
end_proc:
#
```

```
| Plot command
#
a := point(-1, 0):
b := point(1, 0):
c := point(0, sqrt(2.0)):

plot2d(Axes = None,[Mode = List,
        [sier(a, b, c, 5)]]):
```

Würden diese Befehle in MuPAD eingegeben, so würde die in Abbildung 21 gezeigte Grafik erzeugt werden.

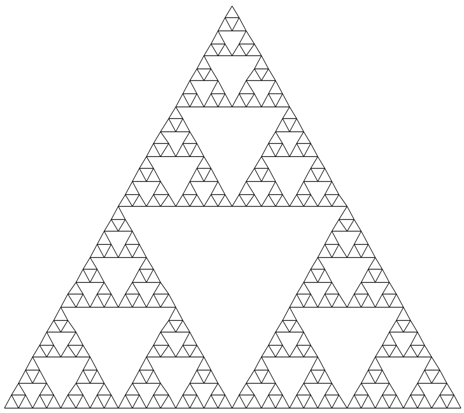


Abbildung 21: Sierpinski-Dreieck der Stufe 5

Ein weiteres Beispiel für benutzerdefinierte Punktelisten ist auf der Titelseite zu sehen, hierbei handelt es sich um ein Sierpinski-Tetraeder. Den Eckpunkten des ursprünglichen Tetraeders wurden die Farben Rot (`[1,0,0]`), Grün (`[0,1,0]`), Blau (`[0,0,1]`) und Weiß (`[1,1,1]`) zugeordnet. Die Farben der Eckpunkte der verfeinerten Tetraeder ergeben sich dann als arithmetisches Mittel der Farben der ursprünglichen Eckpunkte. Eine weitere Anwendung der Grafikprimitiven sind die sogenannten Lindenmayer-Systeme, siehe dazu auch den Artikel von K. Drescher.

Benutzerdefinierte Farbfunktionen

In diesem Abschnitt werden die benutzerdefinierten Farbfunktionen erläutert und einige Beispielgrafiken gezeigt.

In den bisherigen MuPAD-Versionen konnte man Objekte in einer einzigen Farbe darstellen (`Color=[Flat]`), oder die Farben in Abhängigkeit von der Höhe der Stützstellen verteilen (`Color=[Height]`). In der aktuellen Version von MuPAD ist diese Farbfunktionalität erweitert worden. Dem Benutzer wird nun die Möglichkeit gegeben, MuPAD-Prozeduren zu implementieren, welche die Farben eines Objektes bestimmen. Diese Farbfunktionen werden an den Stützstellen des zugehörigen Objektes ausgewertet. Wie die Argumente und das Ergebnis einer solchen Farbfunktion aussehen sollten, wird anhand einer parametrisierten, zweidimensionalen Kurve erläutert. In MuPAD wird eine solche Kurve dargestellt, indem man die Funktionen angibt, welche die x- und y-Koordinaten der Kurve beschreiben. Bei diesen Funktionen, sagen wir `x(u)` und `y(u)`, handelt es sich zumeist um Funktionen in einer Veränderlichen `u`. Zusätzlich muß dann noch der Bereich, den die unabhängige Variable durchläuft, angegeben werden. Bei der Berechnung der Stützstellen wird nun der gegebene Bereich in äquidistante Teilintervalle aufgeteilt (abhängig von `Grid` und `Smoothness`), und die beiden Funktionen an den Endpunkten dieser Teilintervalle evaluiert. Da, wie oben bereits erwähnt, die Farbfunktion an diesen Stützstellen evaluiert wird, hat man verschiedene Möglichkeiten: Die Farbfunktion kann einerseits in Abhängigkeit vom aktuellen Wert `u_val` der unabhängigen Variablen `u` formuliert werden. Andererseits kann man die Farbfunktion auch in Abhängigkeit vom aktuellen Wert der x-Funktion an dieser Stelle formulieren, d.h. man verwendet gerade die x-Koordinate `x_coord = x(u_val)` zur Berechnung des Farbwertes. Gleiches gilt dann natürlich auch für die y-Koordinate `y_coord = y(u_val)`. Die Farbfunktion für eine zweidimensionale Kurve kann also in Abhängigkeit von obigen drei Parametern `x_coord`, `y_coord` und `u_val` formuliert werden, welches auch die Argumente einer solchen Farbfunktion sind. Ähnliche Überlegungen führen dann zu unterschiedlichen Vereinbarungen von Farbfunktionen für die verschiedenen Objektmodi und können im MuPAD Benutzerhandbuch nachgelesen werden.

Das Ergebnis einer benutzerdefinierten Farbfunktion muß, ebenso wie bei den anderen Optionen `Flat` und `Height` zur Farbbestimmung, eine Liste aus RGB-Werten sein. Eine solche Liste umfaßt drei Einträge mit reellen Werten zwischen `0.0` und `1.0`, welche die Rot-,

Grün- und Blau-Anteile der jeweiligen Farbe beschreiben
In einem Plot-Befehl gibt man eine benutzerdefinierte Farbfunktion dabei mit Hilfe der Option `Color = [Function, function_name]` an, wobei mit `function_name` gerade der Name der zu verwendenden MuPAD-Prozedur gemeint ist.

Beispiele

In diesem Abschnitt werden einige Beispiele zu benutzerdefinierten Farbfunktionen gegeben. Das erste Beispiel färbt dabei eine Kurve in Abhängigkeit von der x-Koordinate ein.

```
#
| Colour function for a curve
#
cfun:= proc(x_coord,y_coord,u_val)
   local grey;
begin
   grey:= (x_coord-XMIN)/(XMAX-XMIN):
   [grey, grey, grey]:
end_proc:

#
| Plot command
#
XMIN:= -1.0: XMAX:= 1.0:

plot2d(Axes = None, LineWidth = 18,
         [Mode = Curve,
         [cos(u), sin(u)],
         u = [-PI, PI], Grid = [150],
         Color = [Function, cfun]]);
```

Diese Befehle würden die in Abbildung 22 dargestellte Grafik erzeugen:

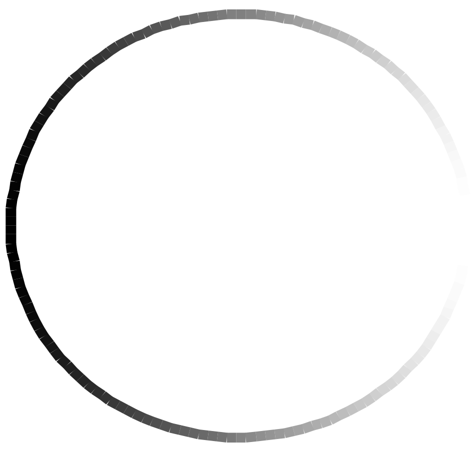


Abbildung 22: Farbfunktion für eine 2D-Kurve

Im folgenden Beispiel wird eine Kugel mit einem Schachbrettmuster überzogen. Die dazu notwendigen Befehle sind:

```
#
| Colour function for a surface
#
cfun:= proc(x_c,y_c,z_c,u_val,v_val)
   local u, v;
begin
   u := round((u_val-UMIN+DU/2)/DU:
   v := round((v_val-VMIN+DV/2)/DV:
   if (u+v) mod 2 = 0 then
      [0.5,0.5,0.5]:
   else
      [1.0,1.0,1.0]:
   end_if:
end_proc:

#
| Plot command
#
UGRID := 31: VGRID := 31:
UMIN := 0.0: VMIN := -float(PI):
UMAX := float(PI): VMAX := float(PI):
DU := (UMAX-UMIN)/UGRID:
DV := (VMAX-VMIN)/VGRID:

plot3d(Axes = None, [Mode = Surface,
         [sin(u)*cos(v), sin(u)*sin(v),
         cos(u)], u = [0, PI],
         v = [-PI, PI],
         Grid = [UGRID, VGRID],
         Style = [ColorPatches,Andmesh],
         Color = [Function, cfun]]):
```

Diese Befehle erzeugen die in Abbildung 23 dargestellte Grafik.

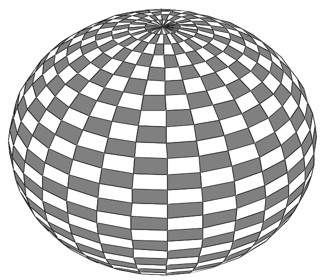


Abbildung 23: Farbfunktion für eine Oberfläche

Das Lineare Algebra Paket „linalg“

Frank Postel

Ab MuPAD Version 1.2.1 stehen dem Benutzer spezielle Datenstrukturen und Algorithmen für Matrizen und Vektoren sowie das Lineare Algebra Paket **`linalg`** zur Verfügung.

Einführung

Als Datenstruktur für Matrizen liegen die Domains **`Matrix(R)`** zugrunde, die Matrizen mit Koeffizienten aus einem Rng **`R`** (Ring ohne Eins) repräsentieren. Solche Domains werden mit dem Domain-Konstruktor **`Matrix`** erzeugt. Vektoren basieren ebenfalls auf dieser Datenstruktur und werden als 1 x n-Matrizen (Zeilenvektoren) oder als m x 1-Matrizen (Spaltenvektoren) aufgefaßt. Als weitere Datenstruktur für Matrizen stehen Domains **`SquareMatrix (n,R)`** zur Verfügung, die mit dem Domain-Konstruktor **`SquareMatrix`** erzeugt werden. Ein solches Domain repräsentiert den Ring der n-reihigen quadratischen Matrizen mit Koeffizienten aus einem Rng **`R`** (Ring ohne Eins). Das Lineare Algebra Paket **`linalg`** basiert auf den Domain-Konstruktoren **`Matrix`** und **`SquareMatrix`**. Es bietet eine Vielzahl an Funktionen wie z.B. die Lösung von linearen Gleichungssystemen, die Bestimmung einer Basis eines Vektorraumes sowie eines Orthogonal- oder Orthonormalsystems aus einer Menge von linear unabhängigen Vektoren und die Berechnung der QR-Faktorisierung einer Matrix.

Definition und Arithmetik von Matrizen und Vektoren

Nach dem Befehl

```
>> loadlib("domains"):
```

stehen dem Benutzer u.a. die oben erwähnten Datenstrukturen für Matrizen zur Verfügung. Das Domain **`MZ`**, das durch

```
>> MZ := Matrix( Integer ):
```

erzeugt wird, repräsentiert Matrizen mit Koeffizienten aus dem Ring der ganzen Zahlen. Es existieren nun verschiedenen Möglichkeiten, Matrizen bzw. Vektoren als Elemente aus dem Domain **`MZ`** zu definieren. Einige Beispiele:

```
>> A := MZ( 2,2,[ [1,2],[2,3] ] );
 +-      -+
 | 1 , 2 |
 |        |
 | 2 , 3 |
 +-      -+
>> B := MZ( 2,2,1,Diagonal );
 +-      -+
 | 1 , 0 |
 |        |
 | 0 , 1 |
 +-      -+
>> v := MZ( 2,1,[1,2] );
 +- -+
 | 1 |
 |   |
 | 2 |
 +- -+
```

Summen, Produkte und Potenzen von Matrizen werden über die gewöhnlichen arithmetischen Operatoren **`+`**, **`-`**, **`*`** und **`^`** in MuPAD berechnet:

```
>> A + B;
 +-      -+
 | 2 , 2 |
 |        |
 | 2 , 4 |
 +-      -+
>> A * v;
 +- -+
 | 5 |
 |   |
 | 8 |
 +- -+
>> A^10;
 +-                  -+
 | 514229 ,  832040  |
 |                    |
 | 832040 , 1346269  |
 +-                  -+
```

Handelt es sich bei dem Koeffizientenring **`R`** der Matrizen um einen Integritätsbereich, so existieren zudem eine Funktion zur Invertierung regulärer Matrizen und ein Gauss-Algorithmus, der eine obere Dreiecksform sowie die Determinante und den Rang einer Matrix als Liste liefert:

```
>> A^(-1);
 +-       -+
 | - 3 , 2 |
 |         |
 | 2 , - 1 |
 +-       -+

>> A::gaussElim( A );
 -- +-        -+        --
 |  | 1 ,   2 |          |
 |  |         |, - 1, 2  |
 |  | 0 , - 1 |          |
 -- +-        -+        --
```

Weiterhin bieten die erwähnten Domain-Konstruktoren verschiedene Matrizenmanipulationsfunktionen wie z.B. Transponieren von Matrizen, Löschen oder Vertauschen von Spalten bzw. Zeilen von Matrizen und vertikales oder horizontales Zusammenfügen von Matrizen. Das Domain **SM**, das durch

```
>> SM := SquareMatrix( 2,Rational ):
```

erzeugt wird, repräsentiert den Ring der 2-reihigen quadratischen Matrizen mit Koeffizienten aus dem Körper der rationalen Zahlen. Es unterscheidet sich zu Domains, die mit dem Domain-Konstruktor **Matrix** erzeugt werden, hinsichtlich der algebraischen Struktur:

```
>> MZ::hasProp( Ring );
 FALSE

>> SM::hasProp( Ring );
 TRUE
```

Sämtliche Operationen, die für den Domain-Konstruktor **Matrix** definiert sind (Manipulationsfunktionen, Gauss-Algorithmus etc.) stehen dem Domain-Konstruktor **SquareMatrix** ebenfalls zur Verfügung. Auch deren interne Repräsentation von Matrizen ist identisch. Es handelt sich hierbei um eine dichte Darstellung in Form eines Feldes vom Typ **DOM_ARRAY**. Das Feld ist im dritten Operanden einer Matrix abgelegt, während die ersten beiden Operanden die Dimension der Matrix wiedergeben.

Das Lineare Algebra Paket „linalg“

Das Lineare Algebra Paket **linalg** wird mit

```
>> loadlib( "linalg" );
```

geladen. Zur Zeit stehen folgende Funktionen zur Verfügung, die sich mit der Funktion **info** anzeigen lassen:

```
>> info(linalg);
linalg::addColumn
linalg::addRow
linalg::adjoint
linalg::angle
linalg::ColumnOf
linalg::charMatrix
linalg::charPolynomial
linalg::concatMatrix
linalg::crossProduct
linalg::deleteColumn
linalg::deleteRow
linalg::det
linalg::dimen
linalg::expr2Matrix
linalg::factorQR
linalg::freeSet
linalg::gauss
linalg::gaussElim
linalg::getColumn
linalg::getRow
linalg::innerProduct
linalg::linearSolve
linalg::multiplyColumn
linalg::multiplyRow
linalg::nonZeros
linalg::ncols
linalg::norm
linalg::nrows
linalg::nullSpace
linalg::orthogonalSet
linalg::orthonormalSet
linalg::RowOf
linalg::rank
linalg::scalarMult
linalg::setColumn
linalg::setRow
linalg::stackMatrix
linalg::swapColumn
linalg::swapRow
linalg::sylvester
linalg::trace
linalg::transpose
linalg::VectorOf
linalg::vectorDimen
```

Da ein Vorstellen aller Funktionen der Library **linalg** den Rahmen dieses Artikels sprengen würde, sollen im folgenden Beispiele für einige wenige Funktionen gegeben werden. Zuvor sei definiert:

```
>> MQ := Matrix( Rational ):
>> A := MQ( 2,4,[ [1,-3,7,1],
                  [1,1,3,0] ] ):
>> b := MQ( 2,1,[1,-1]):
```

Die Funktion **linalg::linearSolve** liefert als Lösung des linearen Gleichungssystem **A*x = b** eine Liste, deren erstes Element eine spezielle Lösung des Gleichungssystems ist und deren zweites Element aus einer Liste mit einer Basis des Lösungsraumes des zugehörigen homogenen Gleichungssystems besteht:

```
>> linalg::linearSolve( A,b );
 -- +-      -+ -- +-   -+ +-      -+ -- --
 |  | - 1/2 | |  | - 4 | | - 1/4 |  |  |
 |  |       | |  |     | |       |  |  |
 |  | - 1/2 | |  |  1  | |  1/4  |  |  |
 |  |       |,|  |     |,|       |  |  |
 |  |   0   | |  |  1  | |   0   |  |  |
 |  |       | |  |     | |       |  |  |
 |  |   0   | |  |  0  | |   1   |  |  |
 -- +-      -+ -- +-   -+ +-      -+ -- --
```

Die allgemeine Lösung des Gleichungssystems kann auch als Linearkombination mit einem freien Bezeichner ausgegeben werden:

```
>> linalg::linearSolve( A,b,c );
 [ - 4 c[1] - 1/4 c[2] - 1/2,
   c[1] + 1/4 c[2] - 1/2, c[1], c[2] ]
```

In vielen Situationen liegen lineare Gleichungen vor, die zu lösen sind. Mit der Funktion **linalg::expr2Matrix** lassen sich die zu diesen Gleichungen zugehörige Koeffizientenmatrix und die rechte Seite bestimmen:

```
>> linalg::expr2Matrix( [x+2 = -5*y,
             2*x = 4-y],[x,y],Rational );
 -- +-      -+  +-   -+ --
 |  | 1 , 5 |  | - 2 |  |
 |  |       |, |     |  |
 |  | 2 , 1 |  |  4  |  |
 -- +-      -+  +-   -+ --

>> linalg::linearSolve( op(%) );
 +-     -+
 |  22/9 |
 |       |
 | - 8/9 |
 +-     -+
```

Mit der Funktion **linalg::freeSet** läßt sich aus einer Menge von Vektoren eine Basis des Vektorraumes bestimmen, der von den angegebenen Vektoren aufgespannt wird:

```
>> linalg::freeSet( [MQ(1,2,[2,1]),
      MQ(1,2,[-3,5]),
      MQ(1,2,[9,18])] );
 [[2 , 1], [- 3 , 5]]
```

Als abschließendes Beispiel soll die Funktion **linalg::orthogonalSet** vorgestellt werden, die ein Orthogonal-System aus linear unabhängigen Vektoren über einem Körper bestimmt:

```
>> linalg::orthogonalSet( [
      MQ(1,2,[2,1]),
      MQ(1,2,[-3,5])] );
 [[2 , 1], [- 13/5 , 26/5]]
```

Ausblick

Das Lineare Algebra Paket **linalg** präsentiert sich in seiner ersten Version. Was die Weiterentwicklung dieser Library betrifft, so sind zwei wesentliche Aspekte zu nennen. Zum einen fehlt zu diesem Zeitpunkt eine Funktion zur Bestimmung von Eigenwerten und Eigenvektoren von Matrizen, der Funktionen zur Berechnung von Normalformen von Matrizen folgen werden. Zum anderen ist eine weitere Datenstruktur für Matrizen vorgesehen, die eine dünne Darstellung von Matrizen repräsentiert. In einer solchen Datenstruktur werden lediglich die Einträge einer Matrix gespeichert, die nicht dem Nullelement des Koeffizientenringes entsprechen. Als interne Repräsentation einer Matrix wird dabei das RR(C)O-Format gewählt, was für „Row-wise Representation Complete and Ordered“ steht. Aufbauend auf dieser Datenstruktur sollen – in Anlehnung an den Domain-Konstruktor **Matrix** – verschiedene Matrizen-manipulationsfunktionen und ein Gauss-Algorithmus zur Verfügung gestellt werden.

Factorization in MuPAD

Jian Yuan

From MuPAD version 1.2 on, an experimental package `faclib` is available for polynomial factorization. In this short article, we explain the usage of this library `faclib` by showing several examples and give the references to the algorithms we used in this package.

Let's begin with a very simple example. If you would like to factor the polynomial

```
2*x^2-2*y^2
```

you just need to enter

```
>> factor(2*x^2-2*y^2);
```

then MuPAD will give the output:

```
[2, x-y, 1, x+y, 1]
```

which represents the expression

```
2*(x-y)^1*(x+y)^1
```

Of course, everyone knows that the polynomial `2*x^2-2*y^2` can be factored as `2*(x-y)*(x+y)` even without using a computer. However, when you would like to factor much more complicated polynomials, `faclib` will help you a lot.

The function `factor` in the above simple example is provided by the package `faclib` of MuPAD, and it can be used as `factor(a)`. This function performs a complete factorization of polynomials in one or more variables over the rational numbers. The input polynomial `a` should be a multivariate polynomial over the rational numbers or a univariate polynomial over the integers modulo a prime integer. It may be in the form of a polynomial created by `poly` as well as in the usual expression form. All the names in the input are viewed as variables. The output is of the form `[u,f_1,e_1,...,f_n,e_n]` such that `a = u * f_1^e_1 *... * f_n^e_n`, where `u` is the constant content and each `f_i` is a primitive irreducible polynomial which is in the same form as that of the input polynomial. This kind of data structure is more suitable for programming purposes.

Now let's see a few more complicated examples. Set

```
>> a1:= poly(2*x^4-8*y^4-
       8*x*y^3+8*x^3*y+
       6*x^2*y^2, [x,y]):
```

```
>> a2:= -3*x^5/2+2*y^5-
       17*x^4*y/2-23*x^2*y^3^2-
       33*x^3*y^2/2:
```

```
>> a3:= poly(-x-x^2+2*x^3-
       2*x^4+x^5+x^6-3,
       [x],IntMod(7)):
```

MuPAD gives the results as follows:

```
>> factor(a1);
[2, poly(x - y, [x, y]), 1,
poly(x + y, [x, y]), 1,
poly(x + 2 y, [x, y]), 2]
```

```
>> factor(a2);
[- 1/2, 3 x - y, 1, x + y, 2,
x + 2 y, 2]
```

```
>> factor(a3);
[1, poly(x + 2, [x], IntMod(7)), 2,
poly(x + 1, [x], IntMod(7)), 4]
```

Sometimes one only needs to get the repeated factors of a polynomial, which is the square free factorization and, in fact, is the first main step of factorization. The package `faclib` also provides a function called `sqrfree`, used as `sqrfree(a)`, for this purpose. The function `sqrfree` computes the square free factorization of a multivariate polynomial over the rational numbers or a univariate polynomial over the integers modulo a prime integer. It gets all the repeated factors of the polynomial. Its input may have the same form as that of the function `factor`, and its output is also of the form `[u, f_1, e_1,..., f_n, e_n]` such that `a = u * f_1^e_1 *...* f_n^e_n`, but here each `f_i` is primitive and `gcd(f_i, f_j) = 1` for all `i <> j`. The following are the results for the same examples given above:

```
>>sqrfree(a1);
[2, poly(x^2 - y^2, [x, y]), 1,
poly(x + y*2, [x, y]), 2]
>>sqrfree(a2);
[-1/2, x*3 - y, 1,
x*y*3 + x^2 + y^2*2, 2]
>>sqrfree(a3);
[1, poly(x + 2, [x], IntMod(7)), 2,
poly(x + 1, [x], IntMod(7)), 4]
```

At present, only these two functions can be directly used in general. For experienced user, there are still many functions in the **`faclib`** library that may be very useful. For instance, the functions for distinct degree factorization and splitting equal degree factors for univariate polynomials, etc. However, one should carefully read the comments in the procedures before using them.

The basic computational ideas of this library are based on the book written by K. O. Geddes et al [1]. Namely, for multivariate polynomials, we first use homomorphism techniques to reduce multivariate factoring problems to univariate factoring problems modulo a prime, then use Hensel lifting to obtain a factorization in the larger domain; for univariate polynomials, we use the „modulo p“ factorization methods, along with construction techniques for lifting a „modulo p“ factorization over the integer.

Acknowledgements

The author would like to thank Prof. Dr. Benno Fuchssteiner for giving her the opportunity to work on the MuPAD library, the people of the MuPAD group for useful discussions, and the secretaries and computer managers in the Dept. of Math. & Info. of Univ. Paderborn for constant facilities.

Bibliography

[1] Keith O. Geddes, Stephen R. Czapor and George Labahn, *Algorithms for computer algebra*, Chaps. 6-8, Kluwer Academic Publishers, 1992.

Aufgelesen

Erholung

Apple konnte im vierten Quartal des Geschäftsjahres 94, das am 30. September endete, den höchsten Umsatz in der Geschichte des Unternehmens verbuchen. Mit fast 2,5 Milliarden Dollar lag das Ergebnis um rund 16 Prozent über dem Ergebnis des Vorjahresquartals.
Auch der Gewinn entwickelte sich positiv, so konnte dieser im vierten Quartal auf 114,7 Millionen Dollar gegenüber 2,4 Millionen Dollar im vierten Quartal des Vorjahres gesteigert werden.

MAC UP 12/ 94

SNI steigt bei Escom ein

Die Siemens Nixdorf Informationssysteme (SNI), Paderborn, erwirbt eine zehnprozentige Beteiligung an Escom. Siemens Nixdorf und Escom vereinbarten auf dem Gebiet der Produktbereitstellung und PC-Technologie eine Zusammenarbeit. Diese sieht neben dem gemeinsamen Einkauf von PC-Komponenten auch die Fertigung von Escom-Rechnern bei SNI vor.

Die Computer Zeitung 25/40, 6. Oktober 1994

Commodore GmbH muß aufgeben

Aus für Commodore Deutschland: Die Frankfurter Tochter der in Liquidation befindlichen Commodore International Ltd. muß Konkurs beantragen. Damit verschwindet ein Unterbenehmen aus der PC-Szene, das vor allem Anfang der 80er Jahre für Furore sorgte: Der „Volkscomputer“ C64 wurde zum Verkaufshit schlechthin.

Die Computer Zeitung 25/38, 22. September 1994

Peacock peilt die Umsatzmilliarde an

Für den unmittelbar bevorstehenden Abschluß des Geschäftsjahres 1993/94 rechnet Peacock mit einem Umsatz von rund 850 Millionen DM und einer Gewinnrendite vor Steuern von 2,5 Prozent.
Im Geschäftsjahr 1994/95 will Peacock seinen Umsatz um mindestens 40 Prozent auf über eine Milliarde Mark steigern.

Die Computer Zeitung 25/39, 29. September 1994

Mac Clones lizenziert

PC Pionier Apple will in den nächsten drei Jahren seinen PC Weltmarktanteil von derzeit 10 auf 16 Prozent steigern.
Dazu werden die Kalifornier ihr Betriebssystem Mac OS sowie Elemente der Power PC Architektur an andere Hardware Hersteller lizensieren und damit die Produktion von Mac Clones ermöglichen. Die ersten Mac Clones, die auf der Power-Mac Architektur basieren werden, sollen im Laufe des nächsten Jahres auf den Markt kommen.

Die Computer Zeitung 25/40, 6. Oktober 1994

China: PC Markt wächst

Der PC Markt in der Volksrepublik kommt langsam in Fahrt. Im zweiten Quartal 1994 wurden in China insgesamt 128.000 PC's verkauft. Drei Viertel davon waren Importgeräte

Die Computer Zeitung 25/40, 6. Oktober 1994

„ATM Low Speed“ für den Desktop

Obwohl ATM auf einer Übertragungsgeschwindigkeit von 155 Megabit pro Sekunde basiert, wollen IBM, LSI Logic, Fujitsu und 22 weitere Unternehmen einen Standard für ein sogenanntes „ATM Low Speed“ für Desktop-Computer entwickeln.
Grundlage ist eine Übertragungsleistung von 25 Megabit pro Sekunde.

Die Computer Zeitung 25/40, 6. Oktober 1994

Intel blockiert PCI-Erweiterung

Apple und IBM, Verbündete bei der Entwicklung des PowerPC, wollen in ihren Rechnern eine schnellere Variante des PCI-Busses einführen. PCI-Schöpfer Intel zieht jedoch nicht mit.
Prinzipiell ist ein schnellerer PCI-Bus - der Vorschlag des PowerPC-Konsortiums sieht eine Beschleunigung von derzeit 33 auf 66 Megahertz vor - für alle Beteiligten wünschenswert.
Zum Ende des Jahres wollen Apple und IBM nun PowerPC Rechner auf den Markt bringen, die erstmals speziell auf den PCI-Bus ausgelegt sind.

Die Computer Zeitung 25/38, 22. September 1994

SMCC Introduces hyperSPARC SPARCstation

Sun Microsystems Computer Company (SMCC) continued to enhance the industry's most successful high-end workstation family, the SPARCstation(TM) 20, by introducing a new model, the SPARCstation 20 Model HS11. The new workstation is aimed at compute-intensive technical applications where the system's floating-point performance and cache memory architecture are an advantage. Among these applications are simulation, electronic design, modelling and analysis.
SMCC's SPARCstation 20 Model HS11 uses the new 100 MHz hyperSPARC(TM) processor from ROSS Technology, Inc. The new SPARCstation 20 Model HS11 is rated at 104.5 SPECint92 and 127.6 SPECfp92.

SunFlash 71.20

Schnellere Power-Macs seit November

Apple stattet den Power-Mac 8100 künftig mit einem schnelleren PowerPC Prozessor aus. Seit dem 3. November taktet der PowerPC 601 des 8100 mit 110 statt 80 Megahertz.

MAC UP 12/ 94

Code nicht geknackt

Zum ersten Mal hat eine Bank, nämlich die Kreditkartengesellschaft VISA, ein Geldgeschäft über das Internet akzeptiert.
Die Verschlüsselungstechnologie der noch jungen Firma Netmarket in New Hampshire erfüllte dabei die Anforderungen der amerikanischen National Security Agency.

Die Computer Zeitung 25/37, 15. September 1994

IBM und Motorola stellen PowerPC 620 vor

IBM und Motorola präsentierten kürzlich den „PowerPC 620“, der mit einem einer Datenbus-Breite von 64 Bit einen weiteren Leistungssprung markiert. Der PPC 620 besitzt Abwärtskompatibilität zu den bisherigen 601-, 603- und 604-Prozessoren.

MAC UP 12/ 94

PRESS RELEASE of the NATIONAL ACADEMY of SCIENCE of UKRAINE DONETSK-KIEV

GLOBAL COOPERATION AGREEMENT BETWEEN NATIONAL ACADEMY OF SCIENCE OF UKRAINE AND MuPAD GROUP HAS BEEN SIGNED
The first Cooperation Agreement between the National Academy of Science of Ukraine and MuPAD has been signed. The cooperation was agreed upon in order to enhance development, research and use of computer algebra and other modern information technology, based on Mathematical and Engineering expert systems, in Ukraine. MuPAD is a general-purpose computer algebra system based on parallel, as well as sequential processing. MuPAD will be extended to a system which allows computer algebra as well as the creation of Interactive books and texts including „active parts“ giving access to expert systems of different kind in Mathematics, Engineering and Science. The agreement covers the system in its present state of development, the agreement will be automatically extended to future developments. MuPAD is equally attractive for students, teachers, and scientists, and available for Unix workstations as well as Macintosh computers, a version of MuPAD for MS Windows 3.1 will be presented in beginning of 1995. Both parties will contribute to create a favourable environment for further development and research in computer algebra as a service to the scientific community. The MuPAD group by this cooperation agreement will make available for nearly 90 000 Ukrainian scientists, schools (more than 2 000 000 students of which nearly 1 000 000 are attending high school) and industry a development which represents a considerable investment. MuPAD will be available to the Ukrainian institutions free of charge. Both parties are willing to examine after some time the question if the present cooperation can be extended to other areas of information technology. The Agreement was signed on October 18, 1994.

NATIONAL ACADEMY of SCIENCE of UKRAINE DONETSK-KIEV, November 15, 1994

Kalender

POSSO Workshop on Software
27 Feb.–3 März 1995, Paris
♦ Daniel Lazard, LITP, Universite' Paris VI, Case 168, 4 pl. Jussieu, F-75252 Paris Cedex 05, France
Fax: 33-1-44 27 40 42, Tel: 33-1-44 27 62 40, E-Mail: **lazard@posso.ibp.fr**

Conference on Symbolic Rewriting Techniques
1–5 Mai 1995, Monte Verita, Ticino, Schweiz
♦ Sekretariat, Institut fuer Wiss. Rechnen, ETH-Zentrum, CH-8092 Zürich
Tel: +41 1 632 7471, Fax: +41 1 632 1172, E-Mail: **sekwr@inf.ethz.ch**

HPCN Europe '95: The International Conference and Exhibition on High-Performance Computing and Networking
2–5 Mai 1995, Mailand
♦ HPCN Europe 1995, Conference Secretariat, Dr. Maria Luisa, Origoni, FAST , P.le R. Morand 2i, 20121 Milano, Italy

IMACS Conference on Applications of Computer Algebra
16–20 Mai 1995, University of New Mexico Albuquerque
♦ E-Mail: **aca@math.unm.edu**, WWW: **http://math.unm.edu/aca.html**

ED-MEDIA 95: World Conference on Educational Multimedia and Hypermedia
18–21 Juni 1995, Graz
♦ ED-MEDIA 95/AACE, P.O. Box 2966, Charlottesville, VA 22902 USA
E-Mail: **AACE@virginia.edu**, Tel: 804-973-3987, Fax: 804-978-7449

ICIAM 95: The Third International Congress on Industrial and Applied Mathematics
3–7 Juli 1995, Hamburg
♦ GAMM-office, University of Regensburg, NWF I - Mathematik, D-93053 Regensburg
Tel: 0941-943-4918, Fax: 0941-943-4005,
E-Mail: **iciam95@vax1.rz.uni-regensburd.d400.de**

AAECC 11: The Eleventh International Symposium on Applied Algebra, Algebraic Algorithms and Error Correcting Codes
17-21 Juli 1995, Paris
♦ Marc Giusti, GAGE, Ecole Polytechnique, F-91228, Palaiseau Cedex
Tel: +33-1-69334585, Fax: +33-1-69333050,
E-Mail: **giusti@ariana.polytechnique.fr**, WWW:
http://medicis.polytechnique.fr/inter-congres/aaecc.html

ESEC '95: fifth European Software Engineering Conference
25–28 September 1995, Sitges, Barcelona
♦ Victor Obach, DIFINSA, P. Lesseps 31, 08023 Barcelona
Tel: +34 3 415 4141, Fax: +34 3 415 5556, E-Mail: **esec95@fib.upc.es**,
WWW: **http://amoto.upc.es/Congressos/ESEC95.html**

SCAN '95: IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics
26–29 September 1995, Bergische Universität Wuppertal
♦ A. Frommer, B. Lang
E-Mail: **scan95@math.uni-wuppertal.de**