MuPAD Computeralgebrapraktikum

Prof. Dr. Wolfram Koepf
Prof. Dr. Werner Seiler
Thomas Wassong

Frühstudium

- Alle Teilnehmer dieses Praktikums können sich zum Frühstudium anmelden.
- Bei erfolgreicher Teilnahme (mündliche Prüfung) erhalten Sie 4 ECTS-Credits im Rahmen der Schlüsselkompetenzen, die Ihnen bei einem späteren Studium anerkannt werden.

Frühstudium

- Hierzu müssen Sie
  - sich ein Anmeldeformular mitnehmen,
  - eine Bestätigung des Lehrers abzeichnen lassen,
  - und beides am nächsten Mittwoch wieder mitbringen.
- Dann werde ich den Kurs eintragen und die Formulare unterschrieben an die Universitätsverwaltung weiterreichen.
- Die Genehmigung für das Frühstudium gilt dann nur für diesen Kurs.

Zum Kurs

- Unser Kurs findet normalerweise im Computerraum 2421 statt.
- Normalerweise besteht der Kurs aus einem Wechsel zwischen Vorlesung und Übung.
- Ich rate Ihnen, das Wichtigste mitzuschreiben.
- Heute gibt es ‚nur' eine Vorführung mit dem Computeralgebrasystem MuPAD.

22.10.08 Heutige Themen Start

- Rechnen mit Dezimalzahlen
- Rechnen mit ganzen Zahlen
- Rechnen mit algebraischen Zahlen
- Rechnen mit Polynomen und rationalen Funktionen
- Rechnen mit Matrizen
- Lösen von Gleichungen
- Graphische Darstellungen
- Differential- und Integralrechnung

Zeitplan (Raum 2421)

29.10.08	Wassong
05.11.08	Koepf
12.-26.11.08	Seiler
03.12.08	Wassong
10.12.08	Koepf
17.12.08	Wassong
14.01.09	Koepf
21.01.-04.02.09	Seiler
11.02.09	Prüfungen

7.11.07 Programmiertechniken

- MuPAD besitzt wie alle General-Purpose-CAS eine eingebaute Programmiersprache.
- Diese enthält die üblichen Programmiertechniken, aber auch viele Hochsprachen-Konstrukte, die Schleifen z. T. unnötig machen.
- Wir beginnen mit der Fall-unterscheidung, dem if then else.
- MuPAD

Schleifen

- Will man die Fakultät $n! = 1 \cdots n$ berechnen, so geht dies z. B. mit einer Zählschleife (for):

- x:=1:
- for k from 1 to 100 do
  - x:=x*k
- end_for:
- x;

Schleifen

- Als vollständiges Programm sieht die Fakultätsfunktion dann so aus:

- Fak1:=proc(n)
- local x,k;
- begin
- x:=1;
- for k from 1 to n do
  - x:=x*k
- end_for;
- x
- end_proc:

Übungsaufgabe 1: Summen

- Programmieren Sie die Berechnung der Summe

$$S(n) := \sum_{k=1}^{n} k^2 = 1 + 4 + \cdots + n^2$$

- Lösung:
- S:=proc(n)
- local s,k;
- begin s:=0;
- for k from 1 to n do s:=s+k^2 end_for;
- s
- end_proc:

Übungsaufgabe 2: Modulo

- Programmieren Sie die Modulo-Funktion

  Mod(a,b):=a modulo b

  die in der letzten Woche behandelt wurde, durch sukzessives Abziehen.
- Benutzen Sie z. B. while.
- Berechnen Sie 1234567 mod 1234.
- MuPAD

Berechnung der Fakultät durch Hochsprachenkonstrukte

- _mult (Produkt), _plus (Summe), $ (Liste)
- product, sum (Formel gesucht!)
- fact bzw. ! (Hochsprachenfunktion)
- rekursiv: Die Fakultät ist eindeutig gegeben durch die Vorschriften

$$n! = n(n-1)! \quad \text{und} \quad 0! = 1\,.$$

- Zugehöriges Programm:
- Fak3:=proc(n) begin if n=0 then 1 else n*Fak3(n-1) end_if end_proc:

Übungsaufgabe 3: Euklidischer Algorithmus

- Programmieren Sie die Berechnung des größten gemeinsamen Teilers rekursiv:

- ggT(a,b):=ggT(b,a) wenn a<b
- ggT(a,0):=a
- ggT(a,b):=ggT(b,a mod b)

- Verschachteltes if mit elif.
- Berechnen Sie ggT(12345678,234).
- Berechnen Sie den ggT zweier 100-stelliger Dezimalzahlen.
- MuPAD

Übungsaufgabe 4: Schnelles Potenzieren

- Programmieren Sie die Berechnung von a^n mod p rekursiv und effizient (Divide-and-Conquer):

- a^0 mod p:=1
- a^n mod p:=($a^{n/2}$ mod p)^2 mod p (n gerade)
- a^n mod p:=(a^{n-1} mod p)*a mod p (sonst)

- Abfrage: is(n,Type::Even)
- Berechnen Sie a^n mod p für drei hundertstellige Dezimalzahlen.
- MuPAD

Schnelles Potenzieren: iterativ

- Das rekursive Programm ist sehr einfach.
- Hier ist es schon etwas komplizierter, ein iteratives Programm zu erstellen.
- Mit der Binärdarstellung des Exponenten

$$n = n_L n_{L-1} \cdots n_2 n_1$$

  lässt sich die Potenz wie folgt darstellen:

$$a^n = a^{n_1} (a^{n_2})^2 \cdots (a^{n_{L-1}})^{2^{L-1}} (a^{n_L})^{2^L} \ .$$

Schnelles Potenzieren: Iteratives Programm

- Dies führt zu folgendem Programm:

- Erst müssen wir die binären Ziffern bestimmen: Ziffern

- Dann können wir iterativ multiplizieren: iterativepowermod

Freiwillige Hausaufgabe 1: Primzahlzwillinge

- Unter Primzahlzwillingen versteht man zwei Zahlen p und $p + 2$, die beide Primzahlen sind. So sind etwa 5 und 7 oder 101 und 103 Primzahlzwillinge.
- In dieser Aufgabe sollen Sie die kleinsten Primzahlzwillinge finden, die größer als 100.000 sind.
- Man verwende <> und nextprime.

Freiwillige Hausaufgabe 2: Listen

- Kehren Sie mit Hilfe rekursiver Programmierung den Inhalt einer Liste um. Testen Sie Ihr Programm mit einer beliebigen Liste mit 100 Elementen.
- Eine Liste: liste:=[a,b,...,z] kann man mit $ erzeugen.
- Mit op(liste,1) erhält man das erste Element, mit [op(liste,2..nops(liste))] die Restliste.
- Mit append fügt man Elemente an eine Liste an.

Fibonaccizahlen

- Die Fibonaccizahlen sind erklärt durch

$$F_n = F_{n-1} + F_{n-2} \quad \text{und} \quad F_0 = 0,\ F_1 = 1\,.$$

- Wir bestimmen die Fibonaccizahlen rekursiv. MuPAD
- Das Programm ist sehr langsam, weil die Anzahl der Aufrufe exponentiell wächst.
- Merkt man sich aber die bereits berechneten Resultate (im Speicher), dann ist die Anzahl der Aufrufe linear in *n*.
- MuPAD

Übungsaufgabe 5: Fibonaccizahlen mit Divide-and-Conquer

- Schreiben Sie ein Programm, welches die Fibonaccizahlen aus den Beziehungen

  $$F_{2n} = F_n(F_n + 2F_{n-1}) \text{ und } F_{2n+1} = F_{n+1}^2 + F_n^2$$

  durch sukzessives Halbieren berechnet.
- Vergleichen Sie die Rechenzeiten Ihrer Funktion mit der eingebauten Funktion numlib::fibonacci für n=1.000.000.
- MuPAD

Erweiterter Euklidischer Algorithmus

Algorithm 1 Euklidischer Algorithmus $\texttt{EA}$

Eingabe: zwei natürliche Zahlen $x, y \in \mathbb{N}$ mit $x \leq y$

Ausgabe: $d = \mathrm{ggT}\,(x, y)$

if $x \mid y$ **then**
    return x
else
    return $\texttt{EA}(y \bmod x, x)$
end if

Algorithm 2 Erweiterter Euklidischer Algorithmus $\texttt{EEA}$

Eingabe: $x, y \in \mathbb{N}$ mit $x \leq y$

Ausgabe: $s, t \in \mathbb{Z}$ mit $\mathrm{ggT}\,(x, y) = sx + ty$

if $x \mid y$ **then**
    return $(1, 0)$
else
    $(s', t') \leftarrow \texttt{EEA}(y \;\texttt{mod}\; x, x)$
    $s \leftarrow t' - s' \cdot (y \;\texttt{div}\; x); \quad t \leftarrow s'$
    return (s, t)
end if

Freiwillige Hausaufgabe 3: Erweiterter Euklidischer Algorithmus

- Programmieren Sie den erweiterten Euklidischer Algorithmus EEA(x,y) anhand des gegebenen Programms.
- Eingabe: x,y, wobei x nicht notwendig kleiner als y ist.
- Man benutze div und mod.
- Ausgabe: [g,s,t], wobei g=ggT(x,y) und s und t die zugehörigen Bézout-koeffizienten mit $g = s\,x + t\,y$ sind.
- Lösen Sie EEA(1234,56789).

Chinesischer Restsatz

- Das Restproblem

$$x \equiv l_1 \mod m_1$$
$$x \equiv l_2 \mod m_2$$
$$\vdots$$
$$x \equiv l_k \mod m_k$$

hat eine eindeutige Lösung modulo $m := m_1 m_2 \cdots m_k$, sofern die Moduli m_j paarweise teilerfremd sind.

- Ein diesbezüglicher Algorithmus wurde von Prof. Werner Seiler angegeben.

Chinesischer Restsatz: Algorithmus

- Eingabe: $[l_1,\ldots,l_k]$ und $[m_1,\ldots,m_k]$
- Zwischenergebnisse:

$$\mu_i = \frac{m}{m_i} = m_1 \cdots m_{i-1} m_{i+1} \cdots m_k \in \mathbb{N}$$

$$s_i, t_i \in \mathbb{Z} \text{ mit } s_i\mu_i + t_i m_i = 1.$$

$$\hat{x} = \ell_1 s_1 \mu_1 + \ell_2 s_2 \mu_2 + \cdots + \ell_k s_k \mu_k$$

- Ausgabe: $x = \hat{x} \bmod m$.

Freiwillige Hausaufgabe 4: Chinesischer Restsatz

- Programmieren Sie den chinesischen Restsatz.
- Verwenden Sie den angegebenen Algorithmus.
- Bestimmen Sie die Lösung des Problems

$$x \equiv 2 \mod 3$$

$$x \equiv 3 \mod 4$$

$$x \equiv 1 \mod 7$$

Kleiner Satz von Fermat

- Für $n \in \mathbb{Z}$ und $p \in \mathbb{P}$ gilt die Gleichung

$$A(n): \qquad n^p \equiv n \mod p \,.$$

- Wir testen diese Gleichung mit MuPAD.
- Beweis durch vollständige Induktion. Man kann eine Aussage $A(n)$ für die natürlichen Zahlen beweisen, indem man $A(0)$ beweist und zeigt, dass aus $A(n)$ die Aussage $A(n+1)$ folgt.
- Induktionsanfang: Offenbar ist $A(0)$ korrekt.

Der binomische Lehrsatz

- Genauso wie die binomischen Formeln

$$(n+1)^2 = n^2 + 2n + 1 \quad \text{und}$$

$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

gelten, ist für beliebige Exponenten *p*

$$(n+1)^p = n^p + \binom{p}{1} n^{p-1} + \binom{p}{2} n^{p-2} + \cdots \binom{p}{p-1} n^1 + 1$$

mit

$$\binom{p}{k} = \frac{p(p-1)\cdots(p-k+1)}{k(k-1)\cdots 1} = p\frac{(p-1)!}{k!(p-k)!} \quad .$$

Kleiner Satz von Fermat

- Induktionsschluss: Gilt der Satz für ein *n*, so folgt

$$(n+1)^p \equiv n+1 \mod p \ ,$$

  also $A(n+1)$, da alle anderen Binomial-koeffizienten *p* als Teiler besitzen.
- Damit ist der *Kleine Satz von Fermat* durch vollständige Induktion bewiesen.
- Für ggT(*n*,*p*)=1 gilt nach Division durch *n*

$$n^{p-1} \equiv 1 \mod p \ .$$

Anwendungen der modularen Arithmetik in der Codierungstheorie und Kryptographie

- Wir beginnen mit einigen Prüfzeichenverfahren.
- Die 10-stellige ISBN (Internationale Standard-Buch-Nummer)

ISBN

- Die zehnstellige ISBN besteht aus einer neunstelligen Dezimalzahl $a_1 a_2 \cdots a_9$ und einer zehnten Prüfziffer a_{10}, welche aus der Formel

$$a_1 + 2a_2 + \cdots + 9a_9 + 10a_{10} \equiv 0 \bmod 11$$

berechnet wird.
- Ist $a_{10} = 10$, so wird $a_{10} = X$ gesetzt.

Übungsaufgabe 6: ISBN

- Programmieren Sie eine Prozedur ISBNPruefziffer, welche die ISBN-Prüfziffer berechnet.
- Bestimmen Sie die Prüfziffer der ISBN meines *Computeralgebra*-Buchs 3-540-29894-?
- Ist die ISBN 3-528-06752-7 meines Schulbuchs *DERIVE für den Mathematikunterricht* korrekt?

Die Europäische Artikelnummer (EAN)

- Die 13-stellige EAN wird beim Einscannen an der Ladenkasse benutzt. Es gilt

$$a_1 + 3a_2 + a_3 + \cdots + a_{11} + 3a_{12} + a_{13} \equiv 0 \bmod 10$$

ISBN 3-540-29894-0

9 783540 298946

Übungsaufgabe 7: EAN

- Programmieren Sie eine Prozedur EANPruefziffer, welche die EAN-Prüfziffer berechnet.
- Bestimmen Sie die Prüfziffer der EAN meines *Computeralgebra*-Buchs 978354029894-?
- Prüfen Sie im Internet: Ist die neue dreizehnstellige ISBN eine EAN?

Fehlerkorrigierende Codes

- Benutzt man ein Prüfzeichen, das einer Gleichung genügt, kann man die Größe eines Fehlers *entdecken*.
- Benutzt man zwei Prüfzeichen, welche zwei simultanen Gleichungen genügen, kann man ggfs. die Größe eines Fehlers und simultan die Position des Fehlers berechnen.
- Dann kann man einen Fehler korrigieren.

Multiplikatives Inverses

- Um die Position des Fehlers aufzuspüren, muss man für ggT(*e*,*p*)=1 eine Gleichung der Form

$$x \cdot e \equiv 1 \mod p$$

nach $x = e^{-1} \bmod p$ auflösen.

- Dies macht man mit dem erweiterten Euklidischen Algorithmus, angewandt auf (*e*,*p*).
- Die Lösung ergibt sich zu $e^{-1} \bmod p = s \mod p$.
MuPAD

Übungsaufgabe 8: modulares Inverses

- Programmieren Sie eine Funktion modinv(e,p), welche das modulare Inverse von e modulo p bestimmt
- Benutzen Sie igcdex.
- Bestimmen Sie das modulare Inverse von 1234 modulo 56789.
- Finden Sie heraus, wie dies auch mit powermod geht.

Reed-Solomon-Code

- Nehmen wir an, wir wollen das Wort „WORT" verschlüsseln, so dass bei der Übertragung ein Fehler repariert werden kann.
- Im ersten Schritt schreiben wir für jeden Buchstaben seine Nummer im Alphabet:
- „WORT" $\to$ {23, 15, 18, 20}
- MuPAD
- Für das Alphabet und das Leerzeichen reichen 30 Buchstaben.

Reed-Solomon-Code

- Nun fügen wir zwei weitere Elemente a_0 und a_1 zu $\{a_2,a_3,a_4,a_5\}$ an, welche folgenden Gleichungen genügen:

$$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 \equiv 0 \mod 31,$$

$$a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 \equiv 0 \mod 31.$$

- In unserem Fall liefert dies „WORT" $\rightarrow$ $\{1, 16, 23, 15, 18, 20\}$

Reed-Solomon-Code

- Nehmen wir an, es wird versehentlich „WIRT" $\rightarrow \{1, 16, 23, 9, 18, 20\}$ übertragen.
- Unter der Prämisse, dass höchstens ein Fehler aufgetreten ist, müssen wir herausfinden,
  - dass der Fehler den Abstand -6 hat
  - und dass er an der Position $x=3$ aufgetreten ist.
- Dann können wir den Fehler reparieren.

Reed-Solomon-Code

- Wir berechnen die Fehler

$$e := a_0 + a_1 + a_2 + a_3 + a_4 + a_5$$
$$\equiv 1 + 16 + 23 + 9 + 18 + 20 \mod 31$$

(also ist mind. ein Fehler aufgetreten) und

$$s := a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5$$
$$\equiv 1 \cdot 16 + 2 \cdot 23 + 3 \cdot 9 + 4 \cdot 18 + 5 \cdot 20 \mod 31$$

und erhalten $e=25$ sowie $s=13$.

Reed-Solomon-Code

- Wie berechnen wir die Stelle *x*, an er der Fehler auftrat?
- Der Fehler *e* produziert in der zweiten Summe den Fehler *xe* mod 31.
- Also ist

$$s \equiv x \cdot e \mod 31 \text{ oder}$$

$$x \equiv s \cdot e^{-1} \mod 31$$

  und in unserem Fall *x*=3.
- Dies alles kann leicht in MuPAD programmiert werden.

Fehlerkorrigierende Codes

- Read-Solomon-Codes werden beim Lesen einer Musik-CD extensiv genutzt.
- Ohne fehlerkorrigierende Codes gäbe es bei der CD keinerlei Musikgenuss.
- Eine zerkratzte CD kann Hundert-tausende von Fehlern enthalten!
- Bei einer CD-ROM darf es (nach der Fehlerkorrektur!) überhaupt keine Lesefehler mehr geben!

Kryptographie

- Am 10. Januar und am 14. November 2007 war Verborgene Welten das Thema der Sendung Alles Wissen im dritten Fernsehprogramm des HR.
- Für einen Beitrag zu dieser Sendung wurde auch ich interviewt, und zwar zum Thema Kryptologie.
- Als kurzen Einblick in dieses aktuelle Forschungsgebiet sehen wir uns den fünfminütigen Beitrag über Kryptologie an.
- Filmstart

Kryptographie

- Bei einem Verschlüsselungsverfahren wird eine Nachricht *N* mit Hilfe einer Funktion *E* und eines Schlüssels *e* verschlüsselt:

$$K = E_e(N)\ .$$

- Die Dekodierung erfolgt mit der Funktion *D* und dem Schlüssel *d*:

$$N = D_d(K) = D_d(E_e(N))\ .$$

- Die Funktionen *E* und *D* sollten effizient berechnet werden können.
- Ein Problem ist die Schlüsselübergabe.

Asymmetrische Kryptographie

- Das RSA-Verfahren ist ein Beispiel eines *asymmetrischen* Verschlüsselungs-verfahrens.
- Solche Verfahren wurden 1976 von Diffie und Hellman eingeführt.
- Hierbei verwenden Sender und Empfänger jeweils eigene Schlüssel *e* und *d*.
- Der Schlüssel *e* wird jeweils öffentlich bekannt gegeben, während der Schlüssel *d* geheim bleibt.
- Ein Schlüsselaustausch des persönlichen Dekodierungsschlüssels *d* ist demnach nicht erforderlich.

Kryptographisches Protokoll des RSA-Verfahrens (1978)

- Der Empfänger und Teilnehmer beim RSA-Verfahren
  - besorgt sich eine 400-stellige Dezimalzahl $m = p \cdot q$ mit 200-stelligen Primzahlen $p, q \in \mathbb{P}$,
  - berechnet $\varphi = (p - 1)(q - 1)$,
  - bestimmt und veröffentlicht einen öffentlichen Schlüssel e, der keinen gemeinsamen Teiler mit φ haben darf,
  - und berechnet seinen privaten Schlüssel d mit der Eigenschaft $e \cdot d = 1 \pmod{\varphi}$.
  - Verschlüsselung und Entschlüsselung sind gegeben durch

$$K = E_e(N) = N^e \pmod{m} \quad \text{und} \quad D_d(K) = K^d \pmod{m}\ .$$

Was brauchen wir also für RSA?

- Bestimmung großer Primzahlen: isprime, nextprime
- Wir müssen möglichst effizient modulare Potenzen $N^e \pmod{m}$ berechnen: powermod
- Effiziente Bestimmung des modularen Inversen $d = e^{-1} \pmod{\varphi}$: powermod
- Außerdem: Mit geeigneten Hilfsfunktionen wandeln wir unsere Nachrichten zuerst in Zahlen um und transformieren diese am Ende wieder zurück.

Warum funktioniert RSA?

- Der Funktionsmechanismus des RSA-Verfahrens beruht auf dem kleinen Satz von Fermat.
- Hierfür müssen wir zeigen, dass

$$D_d(E_e(N)) = N \ .$$

- Setzt man die Formeln des Verschlüsselungsverfahren ein, ist also zu zeigen

$$(N^e)^d \equiv N^{ed} \equiv N \pmod{m} \ .$$

Warum funktioniert RSA?

- Wegen $e \cdot d = 1 \pmod{\varphi}$ ist also $e \cdot d = 1 + k \cdot \varphi$ für eine ganze Zahl k.
- Also ist zu zeigen, dass

$$N^{ed} \equiv N^{1+k(p-1)(q-1)} \equiv N \pmod{m} .$$

- Wir rechnen zunächst modulo p und zeigen mit vollständiger Induktion:

$$N^{1+K(p-1)} \equiv N \pmod{p} .$$

- Da dieselbe Argumentation modulo q gilt, bekommen wir das Resultat schließlich modulo $p \cdot q = m$.

Warum funktioniert RSA?

- Induktionsanfang: K=0 ist klar.
- Induktionsschluss:

$$
\begin{aligned}
N^{1+(K+1)(p-1)} &\equiv N^{p} N^{K(p-1)} \\
&\equiv N \cdot N^{K(p-1)} \\
&\equiv N^{1+K(p-1)} \equiv N \pmod{p} .
\end{aligned}
$$

- Damit ist gezeigt, dass das RSA-Verfahren korrekt ist.