UNIKASSEL VERSITÄT

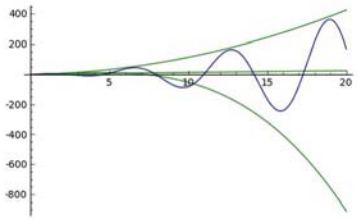
FRÜHSTUDIUM MATHEMATIK

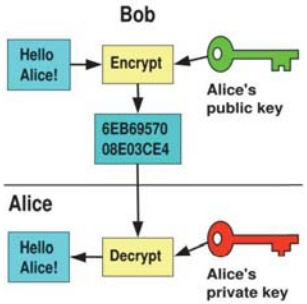
Computeralgebrapraktikum

Prof. Dr. W. Koepf und Prof. Dr. W. Seiler

Einige Themen

Modulares Rechnen
Programmieren mit sage
Kodierungstheorie
Kryptographie





START Mi., 5. November 2014 HPS

16:15 - 17:45 Uhr, Hörsaal 1409, ab 12. Nov. R. 2421

Wird als Frühstudium anerkannt

Anmeldung und Infos bei schaumburg@lg-ks.de

http://www.mathematik.uni-kassel.de/~seiler/AGCA.html

SAGE Computeralgebrapraktikum: Elementare Zahlentheorie und Anwendungen

Prof. Dr. Wolfram Koepf
Prof. Dr. Werner Seiler
WS 2014

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,
  - ein Empfehlungsschreiben des Lehrers besorgen,
  - und beides am nächsten Mittwoch mitbringen.
- Dann werde ich die Formulare unterschrieben an die Universitätsverwaltung weiterreichen.
- Die Genehmigung für das Frühstudium gilt dann nur für diesen Kurs.

Zum Kurs

- Unser Kurs findet im Computerraum 2421 statt.
- Der Kurs besteht aus einem Wechsel zwischen Vorlesung und Übung.
- Ich rate Ihnen, das Wichtigste mitzuschreiben.
- Außerdem sollten Sie unbedingt die Programmierübungen mit SAGE durchführen.

5.11.14 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

Vorläufiger Zeitplan (Raum 2421)

- 12.11.-03.12.14 Koepf
- 10.-17.12.14 Seiler
- 14.-21.01.15 Seiler
- 28.01.-13.02.15 Koepf
- 17./19.02.15 mündliche Prüfungen

Anmelden am Computer

- Bitte melden Sie sich mit einem der Accounts spwg1 bis spwg21 an.
- Die Accounts werden von mir der Teilnehmerliste entsprechend zugeteilt.

Benutzung von SAGE

- Um SAGE zu benutzen, gehen Sie mit einem Browser auf die vereinbarte Webseite.
- Alle Teilnehmer benutzen denselben Username seminar .
- Wenn Sie Ihre Ausarbeiten abspeichern wollen, verwenden Sie bitte einen Dateinamen der Form Nachname_Thema

Programmiertechniken

- SAGE 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.
- SAGE

Schleifen

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

- Als SAGE-Programm sieht dies dann so aus:

```
def fac1(n):
    res = 1
    for k in range(1,n+1):
        res = res * k
    return res
```

Übungsaufgabe 1: Summen

- Programmieren Sie die Berechnung der Summe

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

- SAGE-Programm:
- def Summe(n):
- s = 0
- for k in range(1,n+1):
- s = s + k^2
- return s

Berechnung der Fakultät durch Hochsprachenkonstrukte

- product, sum
- factorial (Hochsprachenfunktion)
- rekursiv: Die Fakultät ist eindeutig gegeben durch die Vorschriften

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

- SAGE-Programm:
- def fac3(n):
- if n == 0: return 1
- return n*fac3(n-1)

Fibonaccizahlen

- Die Fibonaccizahlen sind gegeben durch

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

- Das rekursive Programm ist sehr langsam, weil die Anzahl der Aufrufe exponentiell wächst.
- Merkt man sich aber die bereits berechneten Resultate, ist die Anzahl der Aufrufe linear in *n*.

- SAGE-Programm:
- memo = {0:0, 1:1}
- def fib2(n):
- if not n in memo: memo[n] = fib2(n-1)+fib2(n-2)
- return memo[n]

Übungsaufgabe 2: 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.
- Abfrage für „n ist gerade”: n%2 == 0
- Wir vergleichen die Rechenzeiten dieser Funktion mit der eingebauten Funktion fibonacci für n=100.000.

Übungsaufgabe 3: Modulo

- Programmieren Sie die Modulo-Funktion

  Mod(a,b):=a modulo b

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

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 next_prime.

Übungsaufgabe 4: 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)

- Ggfs. verschachteltes if mit elif.
- Berechnen Sie ggT(12345678,234).
- Berechnen Sie den ggT zweier 100- bzw. 1000-stelliger Dezimalzahlen.
- Verwenden Sie ZZ.random_element(10^100)

Übungsaufgabe 5: Schnelles Potenzieren

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

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

- Abfrage für n gerade: n%2==0
- Berechnen Sie $a^n \bmod p$ für drei hundertstellige Dezimalzahlen.

Schnelles Potenzieren iterativ

- Überlegen Sie sich, wie man die Funktion

  PowerMod(a,n,p):= a^n mod p

  iterativ statt rekursiv programmieren kann.
- Während das rekursive Programm top-down (Halbieren von n) verläuft, läuft das iterative Programm bottom-up (Verdoppeln).

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

$$n = n_L n_{L-1} \cdots n_2 n_1 n_0 = n_0 + 2n_1 + 4n_2 + \cdots + 2^L n_L$$

lässt sich die Potenz wie folgt darstellen:

$$a^n = a^{n_0} (a^{n_1})^2 \cdots (a^{n_{L-1}})^{2^{L-1}} (a^{n_L})^{2^L} \quad .$$

Schnelles Potenzieren: Iteratives Programm

- Erst müssen wir also die binären Ziffern bestimmen.
- Übungsaufgabe 6: Schreiben Sie ein Programm Ziffern(n,b) mittels Division mit Rest (quo_rem).
- 1234.quo_rem(10) -> (123,4)
- 1234.quo_rem(10)[0] -> 123
- Beginnend mit liste=[] fügt der Befehl liste.append(x) das Element x an die Liste an.
- Nun können wir iterativ multiplizieren: iterativepowermod