

U N I K A S S E L V E R S I T Ä T

FRÜHSTUDIUM MATHEMATIK

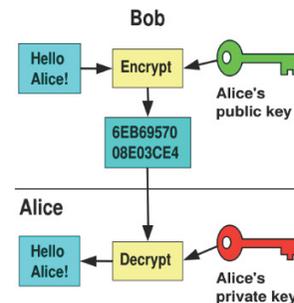
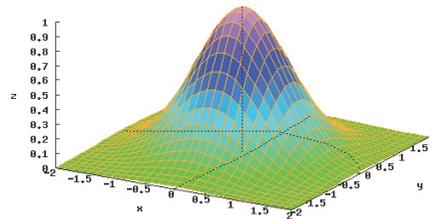
Computeralgebrapraktikum

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



Einige Themen

Modulares Rechnen
Programmieren mit Maxima
Kodierungstheorie
Kryptographie



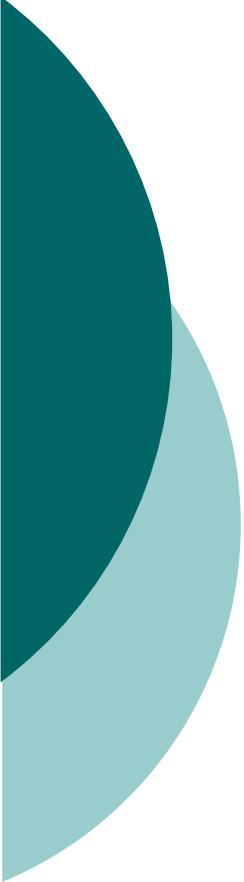
START Mi., 4. November 2015 HPS

16:15 - 17:45 Uhr, Hörsaal 1409, ab 11. Nov. R. 2421

Wird als Frühstudium anerkannt

Anmeldung und Infos bei schaumburg@lg-ks.de

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



Maxima Computeralgebra-Praktikum: Elementare Zahlentheorie und Anwendungen

Prof. Dr. Wolfram Koepf

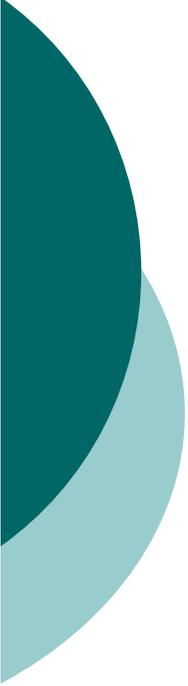
Prof. Dr. Werner Seiler

WS 2015/2016



Frühstudium

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



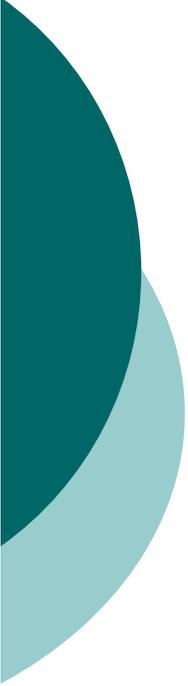
Frühstudium

- 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 [Maxima](#) durchführen.



4.11.15 Heutige Themen

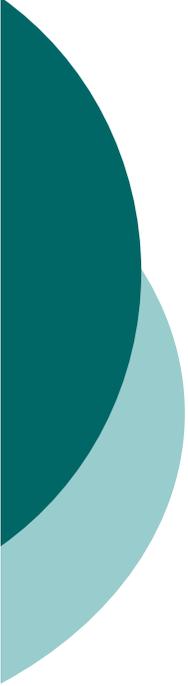
Start

- 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)

- 11.11.-02.12.15 Koepf
- 09.-16.12.15 Seiler
- 13.-20.01.16 Seiler
- 27.01.-10.02.16 Koepf
- danach 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.



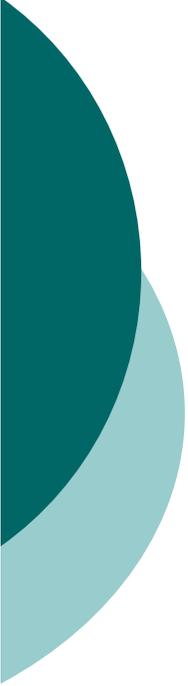
Benutzung von Maxima

- Um Maxima zu benutzen, verwenden Sie die Oberfläche [wxMaxima](#).
- Mit diesem Programm kann man Arbeitsblätter erstellen, abspeichern und wieder laden. Diese enthalten aber nur die Eingaben, nicht die Ausgaben.
- Wenn Sie Ihre Ausarbeiten abspeichern, verwenden Sie am besten einen Dateinamen der Form Nachname_Thema



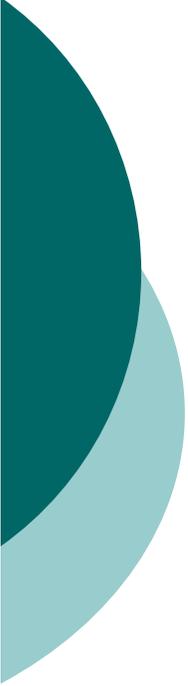
Programmiertechniken

- Maxima 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 Fallunterscheidung, dem **if then else**.
- Maxima



Schleifen

- Will man die Fakultät $n! = 1 \cdot \dots \cdot n$ berechnen, so geht dies z. B. mit einer Schleife (**for**).
- Als Maxima-Programm sieht dies dann so aus:
- ```
Fak1(n):=block([x,k],
 x: 1,
 for k: 1 thru n do
 x: k*x,
 x)$
```



# Übungsaufgabe 1: Summen

---

- Programmieren Sie die Berechnung der Summe

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

- Maxima-Programm:
- ```
Summe(n):=block([k,s],
  s: 0,
  for k from 1 thru n do
    s: s+k^2,
  s
)$
```

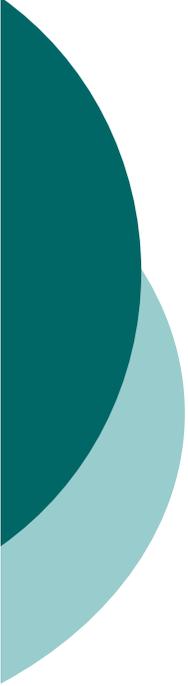


Berechnung der Fakultät durch Hochsprachenkonstrukte

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

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

- Maxima-Programm:
- $\text{Fak4}(n) := \text{if } n = 0 \text{ then } 1$
 $\text{else } n * \text{Fak4}(n-1) \$$



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 .
- Maxima-Programm:
 - `Fib2[0]: 0;`
 - `Fib2[1]: 1;`
 - `Fib2[n] := Fib2[n-1] + Fib2[n-2];`



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“: **evenp(n)**
- Wir vergleichen die Rechenzeiten dieser Funktion mit der eingebauten Funktion **fib** für $n=100.000$.
- Lösung



Übungsaufgabe 3: Modulo

- Programmieren Sie die Modulo-Funktion

$\text{Mod}(a,b) := a \text{ modulo } b$

die in der Vorlesung behandelt wurde, durch sukzessives Abziehen.

- Benutzen Sie z. B. **while**.
- Berechnen Sie $1234567 \text{ mod } 1234$.



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` und `primep`.
- [Lösung](#)



Übungsaufgabe 4: Euklidischer Algorithmus

- Programmieren Sie die Berechnung des größten gemeinsamen Teilers rekursiv:
- $ggT(a,b) := ggT(b,a)$ wenn $a < b$
- $ggT(a,0) := a$
- $ggT(a,b) := ggT(b, a \bmod b)$

- Ggfs. verschachteltes if mit **else if**.
- Berechnen Sie $ggT(12345678, 234)$.
- Berechnen Sie den ggT zweier 1000-stelliger Dezimalzahlen.
- Verwenden Sie **random(10¹⁰⁰⁰)**.
- Lösung



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: **evenp(n)**
- Berechnen Sie $a^n \bmod p$ für drei zehnstellige Dezimalzahlen.
- Lösung



Schnelles Potenzieren iterativ

- Überlegen Sie sich, wie man die Funktion

$$\text{PowerMod}(a,n,p) := a^n \bmod p$$

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:

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



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.
- `mod(1234,10) -> 4`
- `(1234-mod(1234,10))/10 -> 123`
- `part(divide(1234,10),1) -> 123`
- Beginnend mit `l:[]` fügt der Befehl `append(l,x)` das Element `x` an die Liste an.
- Nun können wir iterativ multiplizieren:
[iterativepowermod](#)



Erweiterter Euklidischer Algorithmus

Algorithm 1 Euklidischer Algorithmus EA

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

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

```
1: if  $x \mid y$  then  
2:   return  $x$   
3: else  
4:   return EA( $y \bmod x, x$ )  
5: end if
```

Algorithm 2 Erweiterter Euklidischer Algorithmus EEA

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

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

```
1: if  $x \mid y$  then  
2:   return  $(1, 0)$   
3: else  
4:    $(s', t') \leftarrow$  EEA( $y \bmod x, x$ )  
5:    $s \leftarrow t' - s' \cdot (y \text{ div } x); \quad t \leftarrow s'$   
6:   return  $(s, t)$   
7: end if
```



Übungsaufgabe 6: 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 **mod** und **divide**.
- Ausgabe: $[s,t]$ (oder sogar $[g,s,t]$), wobei $g = \text{ggT}(x,y)$ und s und t die zugehörigen Bézoutkoeffizienten mit $g = s x + t y$ sind.
- Lösen Sie $EEA(1234,56789)$ und vergleichen Sie (nach **load(gcdex)**) mit **igcdex**.
- Maxima



Chinesischer Restsatz

- Das Restproblem

$$x \equiv l_1 \pmod{m_1}$$

$$x \equiv l_2 \pmod{m_2}$$

$$\vdots$$

$$x \equiv l_k \pmod{m_k}$$

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

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

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

$$\hat{x} = l_1 s_1 \mu_1 + l_2 s_2 \mu_2 + \cdots + l_k s_k \mu_k$$

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



Freiwillige Hausaufgabe: Chinesischer Restsatz

- Programmieren Sie den chinesischen Restsatz als `chrem(l1ist,m1ist)`.
- Verwenden Sie den angegebenen Algorithmus.
- Bestimmen Sie die Lösung des Problems

$$x \equiv 2 \pmod{3}$$

$$x \equiv 3 \pmod{4}$$

$$x \equiv 1 \pmod{7}$$



Kleiner Satz von Fermat

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

$$A(n): \quad n^p \equiv n \pmod{p} .$$

- Wir testen diese Gleichung mit [Maxima](#).
- 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

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

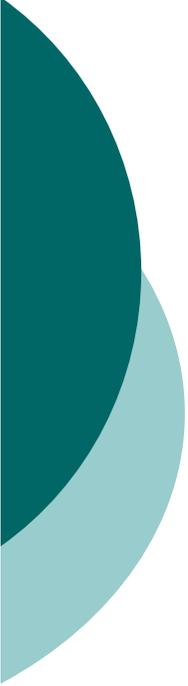
$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

gelten, ist für beliebige Exponenten p

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

mit

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



Kleiner Satz von Fermat

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

$$(n+1)^p \equiv n+1 \pmod{p},$$

also $A(n+1)$, da alle anderen Binomialkoeffizienten p als Teiler besitzen.

- Damit ist der *Kleine Satz von Fermat* durch vollständige Induktion bewiesen.
- Für $\text{ggT}(n,p)=1$ gilt nach Division durch n

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

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 $a_1a_2 \cdots a_9$ und einer zehnten Prüfziffer a_{10} , welche aus der Formel

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

berechnet wird.

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



Übungsaufgabe 7: 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?
- [Maxima](#)

Die Europäische Artikelnummer (EAN)

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

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





Übungsaufgabe 8: 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?
- [Maxima](#)



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 $\text{ggT}(e,p)=1$ eine Gleichung der Form

$$x \cdot e \equiv 1 \pmod{p}$$

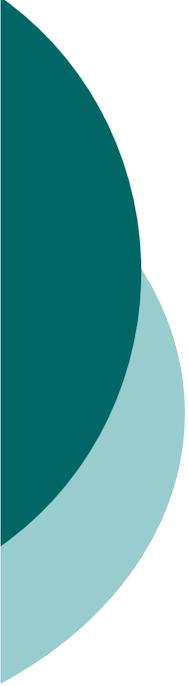
nach $x = e^{-1} \pmod{p}$ auflösen.

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



Übungsaufgabe 9: modulares Inverses

- Programmieren Sie eine Funktion $\text{modinv}(e,p)$, 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.
- [Maxima](#)



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“ \rightarrow {23, 15, 18, 20}
- Maxima
- 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:

$$a_0 + a_1 + a_2 + a_3 + a_4 + a_5 \equiv 0 \pmod{31},$$

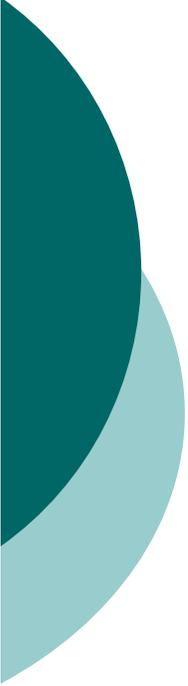
$$a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 \equiv 0 \pmod{31}.$$

- 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

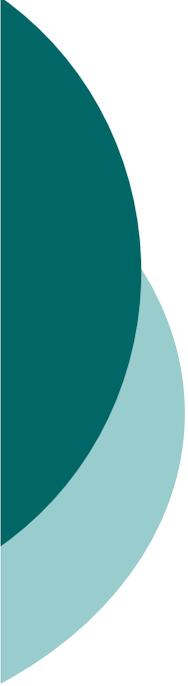
- Wir berechnen die Fehler

$$\begin{aligned} e &:= a_0 + a_1 + a_2 + a_3 + a_4 + a_5 \\ &\equiv 1 + 16 + 23 + 9 + 18 + 20 \pmod{31} \end{aligned}$$

(also ist mind. ein Fehler aufgetreten) und

$$\begin{aligned} s &:= a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 \\ &\equiv 1 \cdot 16 + 2 \cdot 23 + 3 \cdot 9 + 4 \cdot 18 + 5 \cdot 20 \pmod{31} \end{aligned}$$

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



Reed-Solomon-Code

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

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

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

und in unserem Fall $x=3$.

- Dies alles kann leicht in [Maxima](#) 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 Hunderttausende von Fehlern enthalten!
- Bei einer CD-ROM darf es (nach der Fehlerkorrektur!) überhaupt keine Lesefehler mehr geben!



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* Verschlüsselungsverfahrens.
- 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 Verschlüsselungsverfahrens ein, ist also zu zeigen

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



Warum funktioniert RSA?

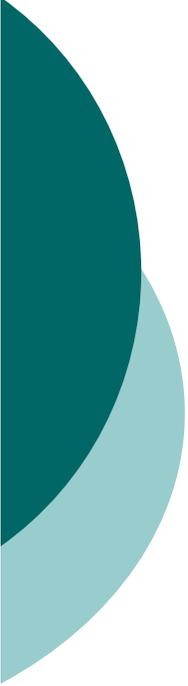
- Wegen $e \cdot d = 1 \pmod{\varphi}$ ist also $e \cdot d = 1 + k \cdot \varphi$ für eine ganze Zahl k .
- Also ist zu zeigen, dass

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

- 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.



Übungsaufgabe 10

- Schreiben Sie eine MuPAD-Prozedur **InitialisiereRSA**, die das RSA-Verfahren initialisiert:
 - Bestimme $m = p \cdot q$ mit 200-stelligen Primzahlen p und q .
 - Berechne $\varphi = (p - 1)(q - 1)$.
 - Bestimme einen öffentlichen Schlüssel e , der keinen gemeinsamen Teiler mit φ haben darf.
 - Berechne den privaten Schlüssel d mit der Eigenschaft $e \cdot d = 1 \pmod{\varphi}$.
 - Maxima