Algorithmen der Linearen Algebra

Thomas Markwig

Fachbereich Mathematik
Eberhard Karls Universität Tübingen

September 2022

Webseite der Vorlesung

http://www.math.uni-tuebingen.de/~keilen/Lehre/SS22/algss22de.ht

Ziel des Kurses

- Lerne grundlegende Strukturen von Programmiersprachen kennen
- Wende diese an, um mathematische Algorithmen in einer Programmiersprache zu implementieren
- Lerne, wie man Fehler in Programmen systematisch finden kann (Debugging)
- Methode: Learning by doing
  - Jeder bearbeitet die Aufgaben im eigenen Tempo.
  - Jeder bearbeitet soviele Aufgaben, wie er kann.
  - Keine Mindestanzahl an Aufgaben erforderlich.
  - Studienleistung = die im Kurses erstellte Bibliothek

Verwendete Software

- Computeralgebrasystem SINGULAR
- Spezialisiert auf Rechnungen in und über Polynomringen
- Open Source, d.h. u.a. Download und Nutzung kostenlos
- Plattformen: Linux, Windows, OSX
- Webseite: http://www.singular.uni-kl.de
- Anwendungsgebiete:
  - (Nicht-)Kommutative Algebra
  - Algebraische Geometrie
  - Singularitätentheorie

SINGULAR-Installation unter Windows (mit WSL)

1) Installiere WSL (Ubuntu) aus dem Microsoft Store
2) Starte anschließend Ubuntu (Icon auf dem Desktop) und folge den Anweisungen (z.B. Benutzerkonto anlegen)
3) Installiere SINGULAR auf Ubuntu
   - `sudo apt-get update`
   - `sudo apt-get install singular`
4) Zum Starten von `Singular`
   - Starte jeweils Ubuntu und gib `Singular` ein.
5) Einige Hinweise
   - Das Windows-Hauptverzeichnis findet man unter /mnt/c.
   - Das Ubuntu-Hauptverzeichnis findet man im Explorer durch Eingabe von \\wsl$\.

SINGULAR-Installation unter Windows (mit Cygwin)

1) Installiere Cygwin mit Standardeinstellung:
   `https://cygwin.com/setup-x86_64.exe`
   - Wähle beim Installieren `Full` aus und installiere `wget`
2) Speichere
   `ftp://jim.mathematik.uni-kl.de/repo/cygwin/x86/singular-4.3.1.tar.xz`
   in C:\cygwin
3) Starte Cygwin und gib folgendes im Terminal ein:
   - cd /
   - tar Jxf singular-4.3.1.tar.xz
4) Starte `setup-x86` (Cygwin-setup) erneut
   - Wähle `Full` aus und suche nach `emacs`
   - Installiere alle Emacs-Pakete nach

SINGULAR starten und beenden

- SINGULAR läuft grundsätzlich in einem Terminal (Windows: Ubuntu oder Cygwin-Terminal)
- Starten mit dem Befehl: `Singular`
- Beenden mit einem der Befehle:
  - `quit;`
  - `exit;`
  - `$`
- Jeder Befehl in SINGULAR schließt mit Semikolon ab.
- Neue Kommandozeile beginnt immer mit >.
- Nicht abgeschlossene Kommandozeile beginnt mit einem Punkt.

Einfaches SINGULAR-Beispiel

```
                        SINGULAR
 A Computer Algebra System for Polynomial Computations

 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann
FB Mathematik der Universitaet, D-67653 Kaiserslautern
> 2+3;
5
> 2+3+
. 7*5;
40
> $

$Bye.
```

Wie bekomme ich Hilfe?

- Webseite von SINGULAR http://www.singular.uni-kl.de
- Online Manual -> Index
- Online Manual -> Libraries

SINGULAR unterbrechen

- `CTRL-C` unterbricht SINGULAR
  - `r` : bricht die aktuelle Rechnung sofort ab
  - `c` : führt die aktuelle Rechnung weiter
  - `q` : bricht SINGULAR ab

Einfaches SINGULAR-Beispiel

```
                                SINGULAR
 A Computer Algebra System for Polynomial Computations

 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann
FB Mathematik der Universitaet, D-67653 Kaiserslautern
> for (int i=1;i<=10;i++){i=i-1;}


^C// ** Interrupt at cmd:`$INVALID$` in line:'i=i-1;'
abort after this command(a), abort immediately(r),
print backtrace(b), continue(c) or quit Singular(q) ?q

halt 2
```

Nach dem Starten – Ring festlegen

- Nach Start: Ring definieren, über dem gearbeitet werden soll.
- Damit legt man auch den Zahlbereich fest:
  - z.B. $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}_2$, $\mathbb{F}_8$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}[i]$, etc.
- Beispiele:
  - `ring r=0,t,lp;`

    Polynomring $\mathbb{Q}[t]$, Variable t, Koeffizienten in $\mathbb{Q}$
  - `ring r=(real,15),x,lp;`

    Polynomring $\mathbb{R}[x]$, Variable x, Koeffizienten in $\mathbb{R}$ - Rechnung mit 15 Nachkommastellen
  - `ring r=5,(x,y),lp;`

    Polynomring $\mathbb{F}_5[x, y]$, Variablen x, y, Koeffizienten in $\mathbb{F}_5$

Variablen in SINGULAR

- Vor Nutzung muss der Typ einer Variablen festgelegt werden.
- Syntax: `typ Variablenname (= Variablenwert);`
- Typen von Variablen:
  - `int i=1;` ganze Zahl(integer)
  - `string s="Hallo";` Zeichenkette
  - `intvec iv=1,2,3,4;` Integervektor
  - `intmat im[2][3]=1,2,3,4,5,6;` Integermatrix

Variablen in SINGULAR

- Vor Nutzung muss der Typ einer Variablen festgelegt werden.
- Syntax: `typ Variablenname (= Variablenwert);`
- Typen von Variablen:
  - `int i=1;` ganze Zahl(integer)
  - `string s="Hallo";` Zeichenkette
  - `intvec iv=1,2,3,4;` Integervektor
  - `intmat im[2][3]=1,2,3,4,5,6;` Integermatrix
  - `ring R=0,t,lp;` Ring
  - `number n=4/6;` Zahl $\frac{4}{6}$ im Zahlbereich
  - `list L=n,iv,s;` Liste
  - `matrix M[2][3]=1,2,3,4,5,6;` Matrix von Polynomen
  - `vector v=[1,2,3];` Vektor von Polynomen

Variablen in SINGULAR

- Vor Nutzung muss der Typ einer Variablen festgelegt werden.
- Syntax: `typ Variablenname (= Variablenwert);`
- Typen von Variablen:
  - `int i=1;` — ganze Zahl(integer)
  - `string s="Hallo";` — Zeichenkette
  - `intvec iv=1,2,3,4;` — Integervektor
  - `intmat im[2][3]=1,2,3,4,5,6;` — Integermatrix
  - `poly f=t2+5t+1;` — Polynom
  - `ideal I=f,t3;` — Ideal
  - `qring Q=std(I);` — Faktorring R/I
  - `map g=R,t+1;` — Abbildung $f : R \longrightarrow Q : t \mapsto t+1$
  - `proc` — Prozedur

Einfaches Beispiel

```
                        SINGULAR
 A Computer Algebra System for Polynomial Computations

 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann
FB Mathematik der Universitaet, D-67653 Kaiserslautern
> ring R=0,t,lp;
> ideal I=t5+t;
> poly f=t2+1;
> qring Q=std(I);
> map g=R,t2;
> g(f);
t4+1
>
```

Einfaches Beispiel

```
> ring R=0,t,lp;
> poly f=t2+1;
> string S="Mathematik";
> int i=3;
> list L;
> L=f,S,i;
> L;
[1]:
   t2+1
[2]:
   Mathematik
[3]:
   3
>
```

Einfaches Beispiel

```
> intmat M[2][3]=1,2,3,4,5,6;
> intmat N[3][4]=1,0,1,0, 0,1,0,1, 1,1,1,1;
> print(M*N);
     4      5      4      5
    10     11     10     11
>
```

Zuweisungen

- Zuweisungsoperator ist das Gleichheitszeichen =
- Beispiel:

```
> int k=1;
> int j;
> j=k+4;
> j;
5
>
```

Schleifen

- `for`-Schleife

```
> int i,s;
> for (i=1;i<=10;i++)
. {
.     s=s+i;
. }
> s;
55
```

- `while`-Schleife

```
> int i,s;
> while (i<=10)
. {
.     s=s+i;
.     i++;
. }
> s;
55
```

- `break;` : bricht eine Schleife ab

Verzweigungen

- Verzweigung mittels der `if-else`-Anweisung:

```
> int i=10;
> int s=7;
> if (i<5 or s<10)
. {
.    s=5;
. }
. else
. {
.    s=0;
. }
> s;
5
```

- Der `else`-Anteil kann fehlen.

Vergleichsoperatoren

- `<` — ist echt kleiner als
- `>` — ist echt größer als
- `<=` — ist kleiner als oder gleich
- `>=` — ist größer als oder gleich
- `==` — ist gleich
- `!=` — ist ungleich
- `<>` — ist ungleich
- Das Ergebnis ist ein boolscher Ausdruck:
  - `wahr` hat den Wert `1` (vom Typ `int`)
  - `falsch` hat den Wert `0` (vom Typ `int`)

Prozeduren in SINGULAR

- Datentyp `proc` erlaubt, Funktionen selbst zu schreiben
- Syntax:

```
proc Prozedurname (Parameterliste)
{
   Prozedurkörper
}
```

Ein einfaches Beispiel

```
> proc permcol (matrix A, int c1, int c2)
. {
.   matrix B=A;
.   B[1..nrows(B),c1]=A[1..nrows(A),c2];
.   B[1..nrows(B),c2]=A[1..nrows(A),c1];
.   return(B);
. }
> ring r=0,(x),lp;
> matrix A[3][3]=1,2,3,4,5,6,7,8,9;
> print(A);
1,2,3,
4,5,6,
7,8,9
> print(permcol(A,1,2));
2,1,3,
5,4,6,
8,7,9
```

Bibliotheken in SINGULAR

- Bibliotheken erlauben, Prozeduren zu speichern und durch Laden der Bibliothek in SINGULAR aufrufbar zu machen
- Laden der Bibliothek in SINGULAR :

  `LIB "Bibliotheksname (ggf. mit Pfad)";`
- Eine Bibliothek ist eine reine Textdatei, in der die Prozeduren hintereinander aufgelistet sind.
- Bestimmte Syntax-Merkmale werden dabei empfohlen (siehe folgende Folien).
- Jeder Teilnehmer soll eine eigene Bibliothek anlegen, in die er alle Prozeduren des Kurses einträgt:

  `nachname-vorname.lib`

  Z.B.: `markwig-thomas.lib`

Bibliotheken in SINGULAR – Bibliothekskopf

```
////////////////////////////////////////////////////////////////
version="1.0";

info="
  LIBRARY: markwig.lib  Algorithmen der Linearen Algebra

  AUTHOR:  Thomas Markwig, thomas.markwig@uni-tuebingen.de

  PROCEDURES:
     permcol(matrix,int,int)  vertauscht Spalten der Matrix
     permrow(matrix,int,int)  vertauscht Zeilen der Matrix
";
////////////////////////////////////////////////////////////////
LIB "inout.lib";
////////////////////////////////////////////////////////////////
```

Bibliotheken in SINGULAR – Prozeduren

```
proc permcol (matrix A, int c1, int c2)
"USAGE:  permcol(A,c1,c2); A matrix, c1,c2 positive integers
RETURN:  matrix, A being modified by permuting column c1 and c2
NOTE:    Platz für wichtige Anmerkungen,
         auch über mehrere Zeilen gestreckt
EXAMPLE: example permcol; shows an example"
{
  matrix B=A;
  B[1..nrows(B),c1]=A[1..nrows(A),c2];
  B[1..nrows(B),c2]=A[1..nrows(A),c1];
  return(B);
}
example
{
  "EXAMPLE:";
  echo = 2;
  ring r=0,(x),lp;
  matrix A[3][3]=1,2,3,4,5,6,7,8,9;
  print(A);
  print(permcol(A,2,3));
}
```

Debugging in SINGULAR – die Tilde ~

- Beim Schreiben einer Prozedur macht man Fehler.
- SINGULAR zeigt dies, indem
  - die Bibliothek nicht mehr geladen wird,
  - die Prozedur beim Durchlauf eine Fehlermeldung liefert,
  - die Prozedur ein falsches Ergebnis liefert.
- In den beiden ersten Fällen gibt SINGULAR Hinweis auf die fehlerhafte Zeile.
- In den beiden letzten Fällen helfen `break points`:
  - Füge in der Prozedur eine Leerzeile mit einer Tilde (`~`) ein.
  - Beim Durchlauf hält SINGULAR an der Stelle an und man kann sich alle Variablenwerte anzeigen lassen.
  - Das hilft, den Fehler zu lokalisiern.

Probleme beim Debugging

- SINGULAR mittels ~ unterbrochen und dann ...
- ... eine fehlerhafte Abfrage eingegeben:

```
-- break point in keilen.lib::ONB --
-- called from keilen.lib::ONB --
-- called from keilen.lib::ONB --
-- called from STDIN --
B=M;
   ? ‘B‘ is undefined
   ? error occurred in or before keilen.lib::ONB line 2596
(keilen.lib::ONB,2598) >>;~<<
breakpoint 0 (press ? for list of commands)
>>
```

- Dann muss man den Befehl `q;` eingeben!

ESingular + emacs

- Statt `Singular` kann man auch `ESingular` starten.
- SINGULAR wird dann im Editor Emacs gestartet.
- Man kann SINGULAR dort ganz normal nutzen.
- Aber man hat zusätzlich einen Editor zum Schreiben der Bibiliothek.
- `emacs` bietet hilfreiches Highlighting und Einrücken!
- Automatisches Einrücken:
  - Region markieren
  - `Esc-x indent-region`

Editor `Visual Studio Code`

- Editor `Visual Studio Code` liefert Highlighting, Einrücken, etc.
- Für Linux, Windows und OSX:

  `https://code.visualstudio.com`

- Installieren und dann Syntax auf C++ stellen

  Fußleiste: `Plain Text` umstellen auf `C++`

Editor sublime text

- Editor sublime text liefert Highlighting, Einrücken, etc.
- Für Linux, Windows und OSX:

  https://www.sublimetext.com

- Installieren und dann Syntax auf C++ stellen

  ```
  Tools -> Command Palette -> Set Syntax C++
  ```

Erste Algorithmen und Prozeduren

Aufgabe

Formuliere einen Algorithmus, der das Minimimum zweier ganzer Zahlen berechnet.

Erste Algorithmen und Prozeduren

Aufgabe

Formuliere einen Algorithmus, der das Minimimum zweier ganzer Zahlen berechnet.

Algorithmus

Input m, n zwei ganze Zahlen

Output $\min\{m, n\}$

Instructions

Erste Algorithmen und Prozeduren

Aufgabe

Formuliere einen Algorithmus, der das Minimimum zweier ganzer Zahlen berechnet.

Algorithmus

Input m, n zwei ganze Zahlen

Output $\min\{m, n\}$

Instructions

- Wenn $m > n$:
  - Dann gib n zurück.
  - Sonst gib m zurück.

Erste Algorithmen und Prozeduren

Aufgabe

Schreibe eine SINGULAR-Prozedur, die zwei ganze Zahlen einliest und ihr Minimum ausgibt.

Algorithmus

Input m, n zwei ganze Zahlen

Output $\min\{m, n\}$

Instructions

- Wenn $m > n$:
  - Dann gib n zurück.
  - Sonst gib m zurück.

Erste Algorithmen und Prozeduren

Aufgabe

Schreibe eine SINGULAR-Prozedur, die zwei ganze Zahlen einliest und ihr Minimum ausgibt.

Algorithmus

Input m, n zwei ganze Zahlen

Output $\min\{m, n\}$

Instructions

- Wenn $m > n$:
  - Dann gib n zurück.
  - Sonst gib m zurück.

```
proc minimum (int m, int n)
{
  if (m > n)
  {
    return(n);
  }
  else
  {
    return(m);
  }
}
```

Quadratsumme

Aufgabe

Schreibe eine SINGULAR-Prozedur, die die Summe der ersten n Quadratzahlen berechnet.

Quadratsumme

Aufgabe

Schreibe eine SINGULAR-Prozedur, die die Summe der ersten n Quadratzahlen berechnet.

Algorithmus

Input n eine ganze Zahl

Output $\sum_{i=1}^{n} i^2$

Instructions

Quadratsumme

Aufgabe

Schreibe eine SINGULAR-Prozedur, die die Summe der ersten n Quadratzahlen berechnet.

Algorithmus

Input n eine ganze Zahl

Output $\sum_{i=1}^{n} i^2$

Instructions

- Setze $s = 0$.
- Für $i = 1$ bis n:
  - Setze $s = s + i \cdot i$.
- Gib s zurück

Quadratsumme

Aufgabe

Schreibe eine SINGULAR-Prozedur, die die Summe der ersten n Quadratzahlen berechnet.

Algorithmus

Input n eine ganze Zahl

Output $\sum_{i=1}^{n} i^2$

Instructions

- Setze $s = 0$.
- Für $i = 1$ bis n:
  - Setze $s = s + i \cdot i$.
- Gib s zurück

```
proc quadratsumme (int n)
{
  int s = 0;
  for (int i=1;i<=n;i++)
  {
    s = s + i*i;
  }
  return (s);
}
```

Rekursive Algorithmen

- Ein Algorithmus heißt rekursiv, wenn er sich selbst wieder aufruft.
- Oft lassen sich Algorithmen in rekursiver Form einfacher formulieren.

Quadratsumme (rekursiv)

Aufgabe

Schreibe eine rekursive SINGULAR-Prozedur, die die Summe der ersten n Quadratzahlen berechnet.

Algorithmus

Input n eine ganze Zahl

Output $\sum_{i=1}^{n} i^2$

Instructions

- Wenn $n = 1$:
  - Gib 1 zurück.
- Sonst:
  - Wende den Algorithmus auf $n - 1$ an, addiere n^2 und gib das Ergebnis zurück.

Quadratsumme (rekursiv)

Aufgabe

Schreibe eine rekursive SINGULAR-Prozedur, die die Summe der ersten n Quadratzahlen berechnet.

Algorithmus

Input n eine ganze Zahl

Output $\sum_{i=1}^{n} i^2$

Instructions

- Wenn $n = 1$:
  - Gib 1 zurück.
- Sonst:
  - Wende den Algorithmus auf $n - 1$ an, addiere n^2 und gib das Ergebnis zurück.

```
proc qs (int n)
{
  if (n == 1)
  {
    return(1);
  }
  else
  {
    return(n*n+qs(n-1);
  }
}
```

Binomialkoeffizienten

Algorithmus

Input n, k zwei ganze Zahlen

Output $\binom{n}{k}$

Instructions

Binomialkoeffizienten

Algorithmus

Input n, k zwei ganze Zahlen

Output $\binom{n}{k}$

Instructions

- Wenn $k < 0$ oder $k > n$:
  - Gib 0 zurück.
- Sonst:
  - Setze $N = 1$ und $Z = 1$.
  - Für i von $n - k + 1$ bis n:
    - Setze $Z = Z * i$.
  - Für i von 1 bis k:
    - Setze $N = N * i$.
  - Gib Z/N zurück.

Binomialkoeffizienten

Algorithmus

Input n, k zwei ganze Zahlen

Output $\binom{n}{k}$

Instructions

- Wenn $k < 0$ oder $k > n$:
  - Gib 0 zurück.
- Sonst:
  - Setze $N = 1$ und $Z = 1$.
  - Für i von $n - k + 1$ bis n:
    - Setze $Z = Z * i$.
  - Für i von 1 bis k:
    - Setze $N = N * i$.
  - Gib Z/N zurück.

```
proc binomi (int n, int k)
{
  if ((k < 0) or (k > n))
  {
    return(0);
  }
  else
  {
    int N,Z = 1,1;
    for (int i=n-k+1;i<=n;i++)
    {
      Z = Z * i;
    }
    for (i=1;i<=k;i++)
    {
      N = N * i;
    }
    return (Z div N);
}
```

Minimum in einem Vektor

Algorithmus

Input $v = (v_1, \ldots, v_k)$ Vektor ganzer Zahlen

Output $\min\{v_1, \ldots, v_k\}$

Instructions

- Setze $s = v_1$.
- Für i von 2 bis zur Länge des Vektors:
  - Wenn $v_i < s$:
    - Dann setze $s = v_i$.
- Gib s zurück.

Minimum in einem Vektor

Algorithmus

Input $v = (v_1, \ldots, v_k)$ Vektor ganzer Zahlen

Output $\min\{v_1, \ldots, v_k\}$

Instructions

- Setze $s = v_1$.
- Für i von 2 bis zur Länge des Vektors:
  - Wenn $v_i < s$:
    - Dann setze $s = v_i$.
- Gib s zurück.

```
proc minimum (intvec v)
{
   int s=v[1];
   for (int i=2;i<=size(v);i++)
   {
     if (v[i] < s)
     {
        s=v[i];
     }
   }
   return(s);
}
example
{
   "EXAMPLE:";
   echo=2;
   intvec v=3,2,5,2,1;
   minimum(v);
}
```

Zeilentausch

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, R)$ und $1 \leq k, l \leq m$.

Output A mit den Zeilen k und l vertauscht.

Instructions

- Definiere eine Hilfsvariable b.
- Für i von 1 bis zur Anzahl der Spalten von A:
  - Setze $b = a_{ki}$.
  - Setze $a_{ki} = a_{li}$.
  - Setze $a_{li} = b$.
- Gib A zurück.

Zeilentausch

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, R)$ und $1 \leq k, l \leq m$.

Output A mit den Zeilen k und l vertauscht.

Instructions

- Definiere eine Hilfsvariable b.
- Für i von 1 bis zur Anzahl der Spalten von A:
  - Setze $b = a_{ki}$.
  - Setze $a_{ki} = a_{li}$.
  - Setze $a_{li} = b$.
- Gib A zurück.

```
proc zeilentausch (matrix A,
                   int k, int l)
{
  poly b;
  for (int i=1;i<=ncols(A);i++)
  {
    b=A[k,i];
    A[k,i]=A[l,i];
    A[l,i]=b;
  }
  return(A);
}
```

Zeilentausch – Alternative

```
proc zeilentausch (matrix A,
                   int k, int l)
{
  poly b;
  for (int i=1;i<=ncols(A);i++)
  {
    b=A[k,i];
    A[k,i]=A[l,i];
    A[l,i]=b;
  }
  return(A);
}
```

```
proc zeilentausch (matrix A,
                   int k, int l)
{
  int n=ncols(A);
  matrix B[1][n]=A[k,1..n];
  A[k,1..n] = A[l,1..n];
  A[l,1..n]=B;
  return(A);
}
```

Zeilenvielfaches

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, R)$ und $1 \leq k \leq m$ und $\lambda \in R$.

Output A mit den Zeile k mit λ multipliziert.

Instructions

- Für i von 1 bis zur Anzahl der Spalten von A:
  - Setze $a_{ki} = \lambda \cdot a_{ki}$.
- Gib A zurück.

Zeilenvielfaches

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, R)$ und $1 \leq k \leq m$ und $\lambda \in R$.

Output A mit den Zeile k mit λ multipliziert.

Instructions

- Für i von 1 bis zur Anzahl der Spalten von A:
  - Setze $a_{ki} = \lambda \cdot a_{ki}$.
- Gib A zurück.

```
proc zeilenvielfaches (matrix A,
                       int k, poly p)
{
  for (int i=1;i<=ncols(A);i++)
  {
    A[k,i]=p*A[k,i];
  }
  return(A);
}
```

Elementare Zeilenoperation

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, R)$ und $1 \leq k, l \leq m$ und $\lambda \in R$.

Output A mit λ-faches von Zeile k zu Zeile l addiert.

Instructions

- Für i von 1 bis zur Anzahl der Spalten von A:
  - Setze $a_{li} = a_{li} + \lambda \cdot a_{ki}$.
- Gib A zurück.

Elementare Zeilenoperation

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, R)$ und $1 \leq k, l \leq m$ und $\lambda \in R$.

Output A mit λ-faches von Zeile k zu Zeile l addiert.

Instructions

- Für i von 1 bis zur Anzahl der Spalten von A:
  - Setze $a_{li} = a_{li} + \lambda \cdot a_{ki}$.
- Gib A zurück.

```
proc elementarezeilenoperation
(matrix A, int k, poly p, int l)
{
  for (int i=1;i<=ncols(A);i++)
  {
    A[l,i]=A[l,i]+p*A[k,i];
  }
  return(A);
}
```

Der Gauß-Algorithmus (rekursiv – [LA-Skript, Alg. 7.10])

Notation

- A_{11} = Teilmatrix von A ohne 1. Zeile und Spalte
- A_1 = Teilmatrix von A ohne 1. Spalte

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, K)$.

Output Eine Zeilen-Stufen-Form von A.

Instructions

- Durchlaufe die erste Spalte von oben nach unten, bis ein Element $a_{i1} \neq 0$ gefunden wurde oder das Ende der Spalte erreicht ist.
- Wenn ein $a_{i1} \neq 0$ gefunden wurde:
  - Vertausche die Zeilen a_1 und a_i.
  - Für $k = 2, \ldots, m$ addiere zur k-ten Zeile das $-\frac{a_{k1}}{a_{11}}$-fache der ersten.
  - Bilde dann die Untermatrix A_{11} von A.
- Sonst bilde eine Untermatrix A_1.
- Wende den Algorithmus auf die gebildete Untermatrix von A an und ersetze diese in A durch das Ergebnis.
- Gib A zurück.

Der Gauß-Algorithmus (rekursiv – [LA-Skript, Alg. 7.10])

Notation

- A_{11} = Teilmatrix von A ohne 1. Zeile und Spalte
- A_1 = Teilmatrix von A ohne 1. Spalte

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, K)$.

Output Eine Zeilen-Stufen-Form von A.

Instructions

- Setze $i = 1$.
- Solange $a_{i1} = 0$ und $i < m$:
  - Setze $i = i + 1$.
- Wenn $a_{i1} \neq 0$:
  - Tausche in A die Zeilen 1 und i.
  - Für $k = 2$ bis m:
    - Addiere in A das $-\frac{a_{k1}}{a_{11}}$-fache der ersten Zeile zur k-ten Zeile.
- Wenn $m > 1$ und $n > 1$:
  - Wenn $a_{11} \neq 0$:
    - Wende den Algorithmus auf A_{11} an und ersetze A_{11} in A durch das Ergebnis.
  - Sonst:
    - Wende den Algorithmus auf A_1 an und ersetze A_1 in A durch das Ergebnis.
- Gib A zurück.

Reduzierte Zeilen-Stufen-Form ([LA-Skript, Alg. 7.10])

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, K)$.

Output Die reduzierte Zeilen-Stufen-Form von A.

Instructions

- Ersetze A durch eine Zeilen-Stufen-Form von A.
- Durchlaufe alle Zeilen von A beginnend mit der letzten (Laufindex i):
  - Finde in der i-ten Zeile das erste Element a_{ij} ungleich 0.
  - Teile die i-te Zeile durch a_{ij}.
  - Für k von 1 bis $i-1$ addiere dann das $-a_{kj}$-fache der i-ten Zeile zur k-ten Zeile.
- Gib A zurück.

Reduzierte Zeilen-Stufen-Form ([LA-Skript, Alg. 7.10])

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, K)$.

Output Die reduzierte Zeilen-Stufen-Form von A.

Instructions

- Ersetze A durch eine Zeilen-Stufen-Form von A.
- Für i von m bis 1:
  - Setze $j = 1$.
  - Solange $a_{ij} = 0$ und $j < n$:
    - Setze $j = j + 1$.
  - Wenn $a_{ij} \neq 0$:
    - Multipliziere die i-te Zeile mit $\frac{1}{a_{ij}}$.
    - Für k von 1 bis $i - 1$:
      - Addiere das $-a_{kj}$-fache der i-ten Zeile zur k-ten Zeile.
- Gib A zurück.

Rang einer Matrix ([LA-Skript, Alg. 7.12])

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, K)$.

Output $\mathrm{rang}(A)$

Instructions

- Überführe A in ZSF.
- Zähle die Anzahl r der Nicht-Nullzeilen in der ZSF.
- Gib r zurück.

Inverses einer Matrix ([LA-Skript, Alg. 7.15])

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}_n(K)$.

Output A^{-1}

Instructions

- Erweitere die Matrix A um $\mathbb{1}_n$ zu $C = (A, \mathbb{1}_n) \in \mathrm{Mat}(n \times 2n, K)$.
- Überführe C in reduzierte ZSF $C' = \big(A', B\big)$.
- Falls rang $\big(A'\big) = n$, dann gib B zurück, sonst eine Fehlermeldung.

Normalform einer Matrix ([LA-Skript, Alg. 7.18])

Algorithmus

Input $A = (a_{ij}) \in \mathrm{Mat}(m \times n, K)$.

Output Normalform $\mathrm{NF}(A)$ von A bezüglich Äquivalenz sowie die zugehörigen Transformationsmatrizen $S \in \mathrm{Gl}_m(K)$ und $T \in \mathrm{Gl}_n(K)$

Instructions

- Überführe A durch Zeilenoperationen in reduzierte ZSF und überführe $\mathbb{1}_m$ durch die selben Zeilenoperationen in eine Matrix S.
- Überführe A durch Spaltenoperationen in Normalform und überführe $\mathbb{1}_n$ durch die selben Spaltenoperationen in eine Matrix T.
- Gib die Normalform von A sowie die Matrizen S und T zurück.

Basis eines Unterraums von K^n ([LA-Skript, Alg. 7.20])

Algorithmus

Input Ein Erzeugendensystem F des Unterraums $U \subseteq K^n$.

Output Eine Basis von U.

Instructions

- Schreibe die Vektoren von F als Zeilen in eine Matrix A und überführe A in Zeilen-Stufen-Form.
- Gib die ersten $\operatorname{rang}(A)$ Zeilen als Vektoren zurück.

Injektivität / Surjektivität ([LA-Skript, Alg. 7.23])

Algorithmus

Input $A \in \mathrm{Mat}(m \times n, K)$.

Output Meldung, ob f_A injektiv, surjektiv oder bijektiv ist.

Instructions

- Bestimme den Rang r von A.
- Ist $r = m = n$, gib "f_A ist bijektiv" zurück. Ist $r = m < n$, gib "f_A ist surjektiv" zurück. Ist $r = n < m$, gib "f_A ist injektiv" zurück.

Summe von Unterräumen des K^n ([LA-Skript, Alg. 7.25])

Algorithmus

Input Erzeugendensysteme F und G von zwei Unterräumen U und U' des K^n.

Output Eine Basis von $U + U'$.

Instructions

- Bilde aus F und G ein Erzeugendensystem und berechne mittels des Basis-Algorithmus' eine Basis von $U + U' = \langle F \cup G \rangle$.
- Gib diese Basis zurück.

Lineare Unabhängigkeit ([LA-Skript, Alg. 7.26])

Algorithmus

Input Eine Familie F von m Vektoren in K^n.

Output Eins, falls F linear unabhängig ist, Null sonst.

Instructions

- Ist F leer, gib Eins zurück, sonst schreibe die Vektoren in F als Spalten in eine Matrix A.
- Ist $\text{rang}(A) = m$, so gib Eins zurück, sonst Null.

Bild von f_A ([LA-Skript, Alg. 7.27])

Algorithmus

Input $A \in \mathrm{Mat}(m \times n, K)$.

Output Eine Basis von $\mathrm{Im}(f_A)$.

Instructions

- Transponiere A und überführe die Transponierte in ZSF.
- Transponiere das Ergebnis wieder und gib die ersten $\mathrm{rang}(A)$ Spaltenvektoren zurück.

Lösung eines LGS ([LA-Skript, Alg. 8.10])

Algorithmus

Input Die erweiterte Koeffizientenmatrix $(A \mid b) \in \mathrm{Mat}(m \times (n+1), K)$ eines LGS $Ax = b$.

Output Eine spezielle Lösung c von $Ax = b$ und eine Basis B von $\mathrm{Lös}(A, 0)$, sofern das Gleichungssystem lösbar ist.

Instructions

- Berechne eine reduzierte Zeilen-Stufen-Form $(A' \mid b')$ von $(A \mid b)$ mit $r = \mathrm{rang}(A')$.
- Ist $\mathrm{rang}(A) < \mathrm{rang}(A' \mid b')$, dann ist das LGS nicht lösbar.
- Überführe $(A' \mid b')$ in eine $n \times (n+1)$-Matrix $(A'' \mid b'')$ durch Einfügen und Streichen von Nullzeilen, so daß die Pivotelemente anschließend auf der Diagonale der Matrix A'' stehen.
- Ersetze jede Null auf der Diagonale von A'' durch -1.
- Die spezielle Lösung ist $c := b''$ und die Spalten von A'', die eine -1 auf der Diagonale haben, sind eine Basis von $\mathrm{Lös}(A, 0)$.

Kern von f_A ([LA-Skript, Alg. 8.12])

Algorithmus

Input $A \in \mathrm{Mat}(m \times n, K)$.

Output Eine Basis von $\mathrm{Ker}(f_A)$.

Instructions

- Bestimme eine Lösung (c, B) von $Ax = 0$.
- Gib B als Basis zurück.

Transformationsmatrix $T_{B'}^{B}$ ([LA-Skript, Alg. 8.14])

Algorithmus

Input Zwei Basen $B = (b_1, \ldots, b_n)$ und $B' = (b'_1, \ldots, b'_n)$ im K^n.

Output Die Transformationsmatrix $T_{B'}^{B}$.

Instructions

- Schreibe die Vektoren $b'_1, \ldots, b'_n, b_1, \ldots, b_n$ in dieser Reihenfolge als Spalten in eine Matrix A.
- Bringe A auf reduzierte ZSF.
- Die letzten n Spalten von rZSF(A) sind $T_{B'}^{B}$.

Matrixdarstellung $M_D^B(f_A)$ ([LA-Skript, Alg. 8.16])

Algorithmus

Input Eine Matrix $\in \mathrm{Mat}(m \times n, K)$, eine Basis $B = (b_1, \ldots, b_n)$ von K^n und eine Basis $D = (d_1, \ldots, d_m)$ im K^m.

Output Die Matrixdarstellung $M_D^B(f_A)$.

Instructions

- Schreibe die Vektoren $d_1, \ldots, d_m, A \circ b_1, \ldots, A \circ b_n$ in dieser Reihenfolge als Spalten in eine Matrix M.
- Bringe M auf reduzierte ZSF.
- Die letzten n Spalten von $\mathrm{rZSF}(M)$ sind $M_D^B(f_A)$.

Steinitz ([LA-Skript, Alg. 8.19])

Algorithmus

Input Eine Basis $B = (x_1, \ldots, x_n)$ und eine linear unabhängige Familie $F = (y_1, \ldots, y_r)$ von Vektoren in $V = \mathrm{Lin}(B) \subseteq K^n$.

Output Eine Basis B' von V, die F und Vektoren aus B enthält.

Instructions

- Für $i = 1, \ldots, r$ tue:
  - Schreibe die Vektoren in B als Spalten in eine Matrix A.
  - Bilde die erweiterte Matrix (A, y_i).
  - Überführe (A, y_i) in reduzierte Zeilen-Stufen-Form und suche in der letzten Spalte den ersten Eintrag ungleich Null.
  - Streiche den entsprechenden Vektor aus B und füge y_i als letzten Vektor in B ein.
- Gib B zurück.

Gleichungen eines Unterraums ([LA-Skript, Alg. 8.22])

Algorithmus

Input Eine Familie $F = (x_1, \ldots, x_m)$ von Vektoren im K^n.

Output Eine Matrix $A \in \mathrm{Mat}(k \times n, K)$ mit $\mathrm{Lös}(A, 0) = \mathrm{Lin}(F)$.

Instructions

- Schreibe die Vektoren aus F als Zeilen in eine Matrix $B \in \mathrm{Mat}(m \times n, K)$ und bestimme eine Basis $(y_1, \ldots, y_k)$ von $\mathrm{Ker}(f_B) = \mathrm{Lös}(B, 0)$.
- Schreibe die $y_1, \ldots, y_k$ als Zeilenvektoren in eine Matrix A.
- Gib A zurück.

Durchschnitt zweier Unterräume ([LA-Skript, Alg. 8.24])

Algorithmus

Input Zwei Familien F und G von Vektoren in K^n.

Output Eine Basis des Schnitts von $\mathrm{Lin}(F)$ und $\mathrm{Lin}(G)$.

Instructions

- Bestimme Matrizen A und A', so daß $\mathrm{Lin}(F) = \mathrm{Lös}(A, 0)$ und $\mathrm{Lin}(G) = \mathrm{Lös}\left(A', 0\right)$.
- Bilde aus den Zeilen von A und A' eine gemeinsame Matrix A''.
- Bestimme eine Basis B von $\mathrm{Ker}(f_{A''}) = \mathrm{Lös}\left(A'', 0\right)$ und gib B zurück.

Determinante einer Matrix (Gauß) ([LA-Skript, Alg. 10.15])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$.

Output $\det(A)$.

Instructions

- Setze $d = 1$.
- Überführe A mittels Gauß-Algorithmus in nicht-reduzierte ZSF. Jedesmal, wenn dabei zwei Zeilen vertauscht werden, ersetze d durch $-d$. - Wird bei der Gaußreduktion ein Pivotelement zu Null, gib Null zurück und brich ab.
- Gib das Produkt von d mit den Diagonalelementen der ZSF zurück.

Determinante einer Matrix (Laplace) ([LA-Skript, Alg. 10.35])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$.

Output $\det(A)$.

Instructions

- Initialisiere det auf Null.
- Falls $n = 1$, setze det $= a_{11}$ und gehe zu Schritt 3. Sonst tue für $i = 1, \ldots, n$:
  - Bilde eine Hilfsmatrix B durch Streichen der ersten Spalte und der i-ten Zeile von A.
  - Rufe den Algorithmus mit B auf und merke Dir das Ergebnis in einer Hilfsvariablen x.
  - Addiere zu det die Zahl $(-1)^{i+1} \cdot a_{i1} \cdot x$.
- Gib det zurück.

Minimalpolynom einer Matrix ([LA-Skript, Alg. 13.15])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$.

Output μ_A.

Instructions

- Falls A nicht quadratisch ist, gib 0 zurück.
- Bilde die Potenzen $A^0, \ldots, A^n$ und schreibe die Matrizen in Form von Spaltenvektoren der Länge n^2 in eine Matrix $X \in \mathrm{Mat}\left(n^2 \times (n+1), K\right)$.
- Berechne eine Basis von $\mathrm{Lös}(X, 0)$.
- Verwende die Negativen der Koeffizienten des ersten Basisvektors als Koeffizienten eines Polynoms und gib dieses zurück.

Diagonalisierbarkeit einer rationalen Matrix über $\mathbb{C}$

Algorithmus

Input $A \in \mathrm{Mat}_n(\mathbb{Q})$.

Output 1, falls A diagonalisierbar ist, sonst 0.

Instructions

- Berechne das Minimalpolynom μ_A von A.
- Faktorisiere das Minimalpolynom μ_A.
- Wenn μ_A einen mehrfachen Faktor hat, gib 0 zurück, sonst 1.

Bemerkung

- A ist genau dann diagonalisierbar, wenn μ_A keine mehrfache Nullstelle hat.
- Die Algebra lehrt uns, daß das genau dann der Fall ist, wenn μ_A keinen mehrfachen irredziblen Faktor hat.
- Denn: die irreduziblen Faktoren sind separabel und die Minimalpolynome ihrer Nullstellen.

Diagonalisierung einer Matrix ([LA-Skript, Alg. 13.24])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$.

Output T, falls A über K diagonaliserbar ist, wobei T eine invertierbare Matrix ist, so daß $T^{-1} \circ A \circ T$ eine Diagonalmatrix ist.

Instructions

- Berechne das charakteristische Polynom von A.
- Faktorisiere das charakteristische Polynom über K. Ist einer der Faktoren nicht linear, ist A nicht diagonalisierbar (nicht einmal trigonalisierbar) und man gebe 0 zurück. Sind alle Faktoren linear, so liefert die Faktorisierung die Eigenwerte $\lambda_1, \ldots, \lambda_r$ sowie ihre algebraischen Vielfachheiten $n_1, \ldots, n_r$.
- Bestimme für jeden Eigenwert λ_i eine Basis des Eigenraums $\mathrm{Eig}(A, \lambda_i)$ als $\mathrm{Lös}(A - \lambda_i \mathbb{1}_n, 0)$ sowie seine Dimension, d. h. die geometrische Vielfachheit von λ_i.
- Stimmt für jeden Eigenwert die algebraische Vielfachheit mit der geometrischen überein, so schreibe man die im 3. Schritt bestimmten Basen als Spalten in eine Matrix und erhält so T. Ferner erhält man D, indem man die Eigenwerte $\lambda_1, \ldots, \lambda_r$ entsprechend ihren algebraischen Vielfachheiten in der Diagonalen einer Nullmatrix einträgt.

Elementarteiler ([LA-Skript, Alg. 14.17+Auf. 14.20])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$.

Output Liste mit den Eigenwerten von A und den Elementarteilern.

Instructions

- Bestimme das Minimalpolynom μ_A von A und faktorisiere es.
- Wenn μ_A nicht in Linearfaktoren zerfällt, gib eine Fehlermeldung zurück.
- Für jeden Eigenwert λ_i mit $\mathrm{mult}(\mu_A, \lambda_i) = m_i$ bestimme man für $j = 0, \ldots, m_i + 1$ die Zahlen $r_{ij} = \mathrm{rang}\left((A - \lambda_i \mathbb{1})^j\right)$ und berechne daraus den Vektor der Elementarteiler $(t_{i1}, \ldots, t_{im_i})$ mit

$$t_{ij} = r_{ij-1} - 2 \cdot r_{ij} + r_{ij+1}.$$

  Den Eigenwert und den Vektor der Elementarteiler speichere man als i-ten Eintrag in einer Liste `nf`.
- Man gebe die Liste `nf` zurück.

Jordansche Normalform einer Matrix ([LA-Skript, Alg. 14.15])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$.

Output J_A und T mit $T^{-1} \circ A \circ T = J_A$.

Instructions

- Bestimme das Minimalpolynom μ_A von A und faktorisiere es.
- Wenn μ_A nicht in Linearfaktoren zerfällt, gebe man eine Fehlermeldung zurück, andernfalls gilt $\mu_A = \prod_{i=1}^{r}(t-\lambda_i)^{m_i}$.
- Für $i = 1, \ldots, r$ bilde man die Matrix $A_i = A - \lambda_i \mathbb{1}$ und führe folgende Schritte aus:
  - Berechne die Partition $P = (k_1, \ldots, k_{m_i})$ von $n - \mathrm{rang}(A_i^{m_i})$ mit $k_j = \mathrm{rang}(A_i^{j-1}) - \mathrm{rang}(A_i^j)$ gemäß sowie das zugehörige Young-Diagramm.
  - Bestimme eine Basis B_{m_i} von $\mathrm{Lös}\left(A_i^{m_i}, 0\right)$ sowie eine Basis B_{m_i-1} von $\mathrm{Lös}\left(A_i^{m_i-1}, 0\right)$.
  - Tausche B_{m_i-1} mittels des Satzes von Steinitz in B_{m_i} hinein und bestimme die in B_{m_i} verbliebenen Vektoren $x_1^{m_i}, \ldots, x_{k_{m_i}}^{m_i}$.
  - Dann fülle man die ersten k_{m_i} Spalten des Young-Diagramms von P durch die Vektoren $A_i^{m_i-1}x_l^{m_i}, \ldots, A_i^0 x_l^{m_i}$ auf, $l = 1, \ldots, k_{m_i}$.
  - Für $j = m_i - 1, \ldots, 1$ führe man folgendes aus:
    - bestimme eine Basis B_{j-1} von $\mathrm{Lös}\left(A_i^{j-1}, 0\right)$;
    - tausche B_{j-1} sowie die auf der j-ten Ebene des Young-Diagramms bereits eingetragenen Vektoren mittels des Satzes von Steinitz in B_j hinein;
    - bestimme die in B_j verbliebenen Vektoren $x_{k_{j+1}+1}^j, \ldots, x_{k_j}^j$;
    - für $l = k_{j+1}+1, \ldots, k_j$ fülle die Spalten des Young-Diagramms von P mit den Vektoren $A_i^{j-1}x_l^j, \ldots, A_i^0 x_l^j$.
  - Füge die Vektoren aus dem Young-Diagramm als Spalten in die Matrix T ein, beginnend in der linken oberen Ecke und die Spalten des Young-Diagramms von oben nach unten nacheinander durchlaufend.
- Gib $T^{-1} \circ A \circ T$ und T zurück.

Euklidischer Algorithmus ([AS-Skript, Alg. 7.31])

Algorithmus

Input R ein euklidischer Ring mit euklidischer Funktion ν sowie $a, b \in R \setminus \{0\}$.

Output $g \in \mathrm{ggT}(a, b)$ ein größter gemeinsamer Teiler von a und b.

Instructions

- Setze $r_0 = a$, $r_1 = b$ und $k = 2$.
- Solange $r_{k-1} \neq 0$:
  - Wähle $r_k \in R$ und $q_{k-1} \in R$ mit

    $$r_{k-2} = q_{k-1} \cdot r_{k-1} + r_k \quad \text{und} \quad (r_k = 0 \text{ oder } \nu(r_k) < \nu(r_{k-1})).$$

  - Setze $k = k + 1$.
- Gib r_{k-1} zurück.

Erweiterter Euklidischer Algorithmus ([AS-Skript, Bem. 7.55])

Algorithmus

Input R ein euklidischer Ring mit euklidischer Funktion ν sowie $a, b \in R \setminus \{0\}$.

Output $g \in \mathrm{ggT}(a, b)$ ein größter gemeinsamer Teiler von a und b sowie $x, y \in R$ mit $g = x \cdot a + y \cdot b$.

Instructions

- Wähle $q, r \in R$ mit $a = q \cdot b + r$ und $r = 0$ oder $\nu(r) < \nu(b)$.
- Wenn $r = 0$:
  - Gib $g = b$, $x = 0$ und $y = 1$ zurück.
- Sonst:
  - Wende den Algorithmus auf b und r an und berechne so $g, u, v \in R$ mit $g \in \mathrm{ggT}(b, r)$ und $g = u \cdot b + v \cdot r$.
  - Gib g sowie $x = v$ und $y = u - q \cdot v$ zurück.

Chinesischer Restsatz ([AS-Skript, Satz 7.72])

Algorithmus

Input $a = (a_1, \ldots, a_k)$ und $n = (n_1, \ldots, n_k)$ zwei Vektoren ganzer Zahlen, so daß die n_i paarweise teilerfremd sind.

Output $x \in \mathbb{Z}$ mit $0 \leq x < n_1 \cdot \ldots \cdot n_k$ und $x \equiv a_i \pmod{n_i}$ für alle $i = 1, \ldots, k$.

Instructions

- Für i von 1 bis k:
  - Berechne ein $x_i \in \mathbb{Z}$ mit $\overline{x_i} = \overline{N_i}^{-1} \in \mathbb{Z}_{n_i}$ für $N_i = \frac{n_1 \cdot \ldots \cdot n_k}{n_i}$.
- Berechne die Summe $x' = \sum_{i=1}^{k} x_i \cdot a_i \cdot N_i$.
- Berechne ein $0 \leq x < n_1 \cdot \ldots \cdot n_k$ mit $x \equiv x' \pmod{n_1 \cdot \ldots \cdot n_k}$.
- Gib x zurück.

Quadratwurzel ([GdM-Skript, Bsp. 11.23])

Algorithmus

Input $c \in \mathbb{Q}_{>0}$ und $n \in \mathbb{Z}_{>0}$.

Output $q \in \mathbb{Q}$, so daß q^2 bis auf n Dezimalstellen mit c übereinstimmt.

Instructions

- Setze $q = 1$.
- Solange q^2 mit c noch nicht bis auf n Dezimalstellen übereinstimmt:
  - Setze $q = \frac{1}{2} \cdot \left(q + \frac{c}{q}\right)$.
- Gib q zurück.

Positive Definitheit ([LA-Skript, Satz 16.36])

Algorithmus

Input $A \in \mathrm{Mat}_n(\mathbb{R})$.

Output 1, wenn A symmetrisch und positiv definit, 0 sonst.

Instructions

- Wenn $A = A^t$:
  - Überprüfe, ob alle Hauptminoren positiv sind.
  - Wenn ja:
    - Gib 1 zurück
  - Sonst:
    - Gib 0 zurück.

OGB mittels Gram-Schmidt ([LA-Skript, Bem. 15.22])

Bemerkung

- OBG = Basis, deren Vektoren paarweise senkrecht aufeinander stehen
- $b_A(x, y) = x^t \circ A \circ y$ definiert Skalarprodukt, wenn A positiv definit

Algorithmus

Input M eine Familie von Vektoren in $\mathbb{R}^n$ und ein $A \in \mathrm{Mat}_n(\mathbb{R})$ eine positive definite symmetrische Matrix.

Output Orthogonalbasis B von $\mathrm{Lin}(M)$ bezüglich des durch A definierten Skalarproduktes b_A.

Instructions

- Bestimme eine Basis $B = (x_1, \ldots, x_r)$ von $\mathrm{Lin}(M)$.
- Für i von 1 bis r:
  - Berechne die Summe $y_i = x_i - \sum_{j=1}^{i-1} \frac{b_A(y_j, x_i)}{b_A(y_j, y_j)} \cdot y_j$;
- Gib die veränderte Basis $(y_1, \ldots, y_r)$ zurück.

ONB mittels Gram-Schmidt ([LA-Skript, Alg. 15.21])

Algorithmus

Input M eine Familie von Vektoren in $\mathbb{R}^n$ und ein $A \in \mathrm{Mat}_n(\mathbb{R})$ eine positive definite symmetrische Matrix.

Output Orthonormalbasis B von $\mathrm{Lin}(M)$ bezüglich des durch A definierten Skalarproduktes b_A.

Instructions

- Bestimme eine OGB $B = (x_1, \ldots, x_r)$ von $\mathrm{Lin}(M)$ bez. b_A.
- Für i von 1 bis r:
  - Setze $y_i = \frac{1}{\sqrt{b_A(x_i, x_i)}}$.
- Gib $(y_1, \ldots, y_r)$ zurück.

Diagonalisierung normaler Matrizen ([LA-Skript, Alg. 16.12])

Algorithmus

Input $A \in \mathrm{Mat}_n(\mathbb{Q})$ normal, so dass χ_A über $\mathbb{Q}$ zerfällt.

Output Diagonalmatrix D und orthogonale Matrix T mit $D = T^{-1} \circ A \circ T$.

Instructions

- Berechne eine Diagonalmatrix D und eine invertierbare Matrix S mit $D = S^{-1} \circ A \circ S$ mit dem Diagonalisierungsalgorithmus.
- Für jeden Eigenwert von A berechne für die von den zugehörigen Spalten in S erzeugten Unterraum eine ONB und ersetze die Spalten in S durch diese.
- Gib S zurück.

Symmetrischer Gaußalgorithmus ([LA-Skript, Alg. 18.26])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$ symmetrisch und $T \in \mathrm{Gl}_r(K)$ mit $r \geq n$ wobei $\mathrm{char}(K) \neq 2$.

Output $T \in \mathrm{Gl}_r(K)$ so, daß $\tilde{T}^t \circ A \circ \tilde{T}$ eine Diagonalmatrix ist, wobei $\tilde{T}$ durch Streichen der ersten $r - n$ Spalten und Zeilen aus T entsteht.

Instructions

- Setze $m = r - n$.
- Man suche in der ersten Spalte von A den ersten Eintrag, der nicht Null ist. Existiert ein solcher, merke man sich die Zeilennummer z, sonst gehe man zu Schritt 5.
- Ist $z \neq 1$, so addiere die z-te Zeile von A zur ersten und die z-te Spalte zur ersten. Addiere ferner die $z + m$-te Spalte von T zur $m + 1$-ten Spalte.
- Für $k = 2, \ldots, n$ addiere man das $-A[1,k]/A[1,1]$-fache der ersten Zeile von A zur k-ten und das $-A[1,k]/A[1,1]$-fache der ersten Spalte zur k-ten. Sodann addiere man das $-A[1,k]/A[1,1]$-fache der $1 + m$-ten Spalte von T zur $k + m$-ten.
- Falls $n > 1$, dann erzeuge man eine Matrix B, indem man aus A die erste Zeile und die erste Spalte streicht. Sodann rufe man die Prozedur mit den Parametern B und T auf und speichere das Ergebnis in T.
- Man gebe T zurück.

Duale Basis ([LA-Skript, Alg. 19.10])

Algorithmus

Input $A \in \mathrm{Mat}_n(K)$, so daß die Spalten eine Basis des K^n sind.

Output $B \in \mathrm{Mat}_n(K)$, deren Spalten die zu den Spalten von A duale Basis sind.

Instructions

- Berechne die Inverse $C = A^{-1}$ von A.
- Gib $B = C^t$ zurück.

Reine Tensoren ([LA-Skript, Satz. 20.29])

Algorithmus

Input $A \in \mathrm{Mat}(m \times n, K)$ vom Rang $\mathrm{rang}(A) = 1$.

Output Vektoren $u \in K^m$ und $v \in K^n$, so daß $A = u \circ v^t$.

Instructions

- Berechne invertierbare Matrizen S und T, so daß $S \circ A \circ T$ in Normalform bezüglich Äquivalenz ist.
- Berechne $u = S^{-1} \circ e_1$ und $v = (T^{-1})^t \circ f_1$, wenn e_1 und f_1 die ersten kanonischen Basisvektoren im K^m bzw. im K^n sind.

Minimale Tensorzerlegung ([LA-Skript, Alg. 20.34])

Algorithmus

Input $A \in \mathrm{Mat}(m \times n, K)$.

Output $r = \mathrm{rang}(A)$ Matrizen $A_1, \ldots, A_r$ vom Rang 1, so dass $A = A_1 + \ldots + A_r$.

Instructions

- Berechne invertierbare Matrizen S und T, so daß $S \circ A \circ T$ in Normalform bezüglich Äquivalenz ist.
- Für i von 1 bis $r = \mathrm{rang}(A)$:
  - Setze $A_i = S^{-1} \circ E_{ii} \circ T^{-1}$, wobei $E_{ii} \in \mathrm{Mat}(m \times n, K)$ an der Stellen (i, i) eine Eins als Eintrag hat und sonst nur Nullen.
- Gib $(A_1, \ldots, A_r)$ zurück.

Minimalpolynom

- Sei $K \subseteq L$ eine Körpererweiterung.
- $\alpha \in L$ heißt algebraisch über K, wenn der Einsetzhomomorphismus

$$\phi_\alpha : K[t] \longrightarrow L : f \mapsto f(\alpha)$$

  nicht injektiv ist.
- Im Hauptidealring $K[t]$ wird der Kern von ϕ_α dann von einem eindeutig bestimmten normierten Polynom μ_α erzeugt, das das Minimalpolynom von α genannt wird.
- Ziel:
  - $\alpha \in L$ algebraisch über K mit Minimalpolynom $\mu_\alpha \in K[t]$.
  - $\beta \in L$ algebraisch über $K[\alpha]$ mit Minimalpolynom $\mu_\beta \in K[\alpha][s]$.
  - Finde das Minimalpolynom von $\alpha + \beta$ über K.
  - D.h. berechne den Kern von $\phi_{\alpha+\beta}$.
- SINGULAR kann Kerne von Ringhomomorphismen berechnen.

Minimalpolynom von $\sqrt{2}+\sqrt{3}$ über $\mathbb{Q}$ in SINGULAR

```
                            SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations             /
                                                                 0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann          \
FB Mathematik der Universitaet, D-67653 Kaiserslautern             \
> ring r=0,x,lp;
> ring R=0,(s,t),lp;
> qring Q=std(ideal(t2-2,s2-3));
> map phi=r,s+t;
> setring r;
> kernel(Q,phi);
_[1]=x4-10x2+1
>
```

- Frage: Wie geht so das?
- Antwort: Gröbnerbasen!
- Das lernt man in einer Vorlesung zur Computeralgebra!

Literaturverzeichnis

- Thomas Keilen: Kurzeinführung in SINGULAR . Skript 1999.
- Thomas Markwig: Algebraische Strukturen. Vorlesungsskript 2020.
- Thomas Markwig: Grundlagen der Mathematik. Vorlesungsskript 2015.
- Thomas Markwig: Lineare Algebra. Vorlesungsskript 2020.