

Einführung in Sage - Einheit 5

Datencontainer, Lineare Abbildungen, Eigenwert und
Eigenvektoren

Jochen Schulz

Georg-August Universität Göttingen 

- 1 Datencontainer in Sage
- 2 Lineare Abbildungen
- 3 Eigenwerte und Eigenvektoren

- 1 **Datencontainer in Sage**
- 2 Lineare Abbildungen
- 3 Eigenwerte und Eigenvektoren

- Folgen sind ein grundlegender Typ. Zum Beispiel werden Listen und Mengen aus Folgen aufgebaut.
- Folgen sind Sage-Ausdrücke vom Typ `tuple`.
- Eine Folge ist eine Aneinanderreihung beliebiger Sage-Objekte, welche durch Kommata getrennt sind.
- Wir kennen Folgen bereits durch die Eingabe mehrerer Befehle in einer Zeile.

Konstruktion von Folgen I

- Einfache Definition

```
t=var('a,b,c,d,e'); Folge1 = a,b,c; Folge2 = c,d,e;  
Folge1 ; Folge2
```

(a, b, c)

(c, d, e)

- Verbinden von Folgen

```
Folge3 = Folge1+Folge2; Folge3
```

(a, b, c, c, d, e)

Konstruktion von Folgen II

- Konstruktion durch `tuple()`: Der Aufruf `tuple(range(a,b)+)`, für $a, b \in \mathbb{Z}$ erzeugt die Folge $a, a+1, \dots, b-1$:

```
tuple(range(-1,7))
```

`(-1, 0, 1, 2, 3, 4, 5, 6)`

- Der Aufruf `tuple(Objekt(i) for i in range(m,n))` erzeugt die Folge `Objekt(m), Objekt(m+1), ... ,Objekt(n-1)`

```
tuple(i^2 for i in range(2,8))
```

`(4, 9, 16, 25, 36, 49)`

```
tuple(x^i for i in range(4,8))
```

`(x^4, x^5, x^6, x^7)`

Konstruktion von Folgen III

- Erzeugen von n identischen Objekten

```
tuple(sin(x) for i in range(0,5))
```

```
(sin(x), sin(x), sin(x), sin(x), sin(x))
```

- Erzeugen von n funktionalen Objekten

```
x = tuple(sin(i) for i in range(1,5)); x
```

```
(sin(1), sin(2), sin(3), sin(4))
```

Konstruktion und Zugriff

- Erzeugen einer leeren Folge

```
folge = (); folge2 = folge,2,3; folge2
```

((), 2, 3)

- Zugriff auf Elemente

```
x = 1,2,3,4; x[2]
```

3

```
x[2]; x[0:2]
```

3

(1, 2)

- Eine Liste ist eine geordnete Folge beliebiger Sage Objekte.
- In Sage ist eine Liste in eckigen Klammern eingeschlossen.
- Listen haben den Datentyp `list`.
- Matrizen werden als geschachtelte Liste definiert.
- Sie baut auf den eben beschriebenen Folgen auf.

Konstruktion von Listen

- Konstruktion 'per Hand'

```
Liste = [1,[1,2], Set([1,2,3]),x]; Liste
```

```
[1, [1, 2], {1, 2, 3}, (1, 2, 3, 4)]
```

- Listen können leer sein:

```
Liste = []; Liste
```

```
[]
```

- Erzeugen von Listen mit funktionalen Objekten

```
Liste = [2i for i in range(1,9)]; Liste
```

```
[2, 4, 8, 16, 32, 64, 128, 256]
```

Mit Hilfe der Funktion `map(<f>,<Liste>)` kann eine Funktion `f` auf alle Elemente der Liste `<Liste>` angewendet werden. Erwartet eine Funktion mehrere Argumente, muss diese Funktion erst gekapselt werden.

```
map(sin,[x,1,0,pi,0.3])
```

```
[sin(x), sin(1), 0, 0, 0.295520206661340]
```

```
map(is_prime,[2,3,4,5,6,7])
```

```
[True, True, False, True, False, True]
```

map_threaded()

Eine erweiterte Funktion `map_threaded(<f>,<Liste>)` führt die gegebene Funktion `<f>` rekursiv auf alle Elemente in der Liste an.

```
map_threaded(sin,[x,[1,0],[pi,[0.3]]])
```

```
[sin(x), [sin(1), 0], [0, [0.295520206661340]]]
```

```
map(is_prime,[2,3,4,5,6,7])
```

Zugriff auf Listen

Der Zugriff funktioniert genau wie bei Folgen. Beispiele:

```
Liste = [(x,i) for x in range(1,4) for i in range(0,x)];  
Liste
```

```
[(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), (3, 2)]
```

```
Liste[3], Liste[5]
```

```
((3, 0), (3, 2))
```

```
Liste[5] = 42; Liste
```

```
[(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), 42]
```

Weitere Befehle für Listen I

- Entfernen eines Elements aus der Liste

```
Liste = [a,b,c]; del Liste[1]; Liste
```

```
[a, c]
```

- Anhängen von Elementen mittels append

```
Liste.append([3,4,5]); print Liste
```

```
[a, c, [3, 4, 5]]
```

Weitere Befehle für Listen II

- Zusammenfügen von Listen mit dem `+` und `*`-Operator.

```
Liste2 = Liste+[3,4,5]; Liste2
```

```
[a, c, 3, 4, 5]
```

```
Liste*2
```

```
[a, c, [3, 4, 5], a, c, [3, 4, 5]]
```

- Sortieren von Listen mit `sort`

```
Liste = [4,23,1,3]; Liste.sort(); Liste
```

```
[1, 3, 4, 23]
```

Mit `<Objekt> in <Liste>` kann geprüft werden, ob ein Objekt `<Objekt>` in der Liste `<Liste>` enthalten ist.

```
Liste = [x+1,a,x+1,sin(b)]  
x+1 in Liste
```

True

Mit `<Liste>.index(<Objekt>)` erhält man die Position des Objektes in der Liste. Ist es nicht vorhanden, bekommt man eine entsprechende Meldung.

```
Liste.index(sin(b))
```


- Mittels `filter(Liste, Funktion)` können Objekte mit bestimmten Eigenschaften aus einer Liste ausgewählt werden.
- Dabei fungiert eine Funktion `Funktion` als Auswahlkriterium. Sie muss Boolesche Werte zurückliefern.
- Der Befehl `filter` funktioniert auch für Mengen, Dictionaries oder Ausdrücken.

Beispiele für filter

```
filter(bool,[1==1, 1==2, 3==3, 4==5, 7==7])
```

```
[True, True, True]
```

```
var('x,y,z,a')  
def f(x): return a in x.operands()  
filter(f,[a+2,x,y,z,sin(a)])
```

```
[a + 2, sin(a)]
```

Mittels `zip(<Liste1>, <Liste2>)` werden die Elemente zweier Listen paarweise zu einer neuen Liste verknüpft.

```
Liste1 = [a,b,c]; Liste2 = [e,f,g]  
zip(Liste1,Liste2)
```

```
[(a, e), (b, f), (c, g)]
```

Wörterbücher (Dictionaries)

- Eine Dictionary besteht aus einer Ansammlung von ‚Assoziationen‘ der Form `<Index>:<Wert>`. Indizes und Werte können dabei (fast) beliebige Sage Objekte sein.
- Dictionaries haben den Typ `dict`.
- Dictionaries sind Datenstrukturen, die für das Speichern großer Datenmengen gut geeignet sind.
- In Sage ist der indizierte Zugriff auf einzelne Elemente sehr schnell, da intern nicht die gesamte Datenstruktur durchsucht wird.
- der Index ist eindeutig

Konstruktion von Dictionaries I

- Konstruktion: (`<Index>:<Wert>`, ... `<Index>:<Wert>`, ...)

```
T = {a: b, c: d}; T
```

```
{c: d, a: b}
```

Konstruktion von Dictionaries II

- Einträge können durch Zuweisungen der Form
`<Dict>[<Index>] = <Wert>` erzeugt oder verändert werden.

```
T[f(x)] = sin(x); T[1,2] = 5  
T[1,2,3] = [a,b,c]; T[a] = d;  
T
```

`{(1, 2): 5, f: sin(x), c: d, a: d, (1, 2, 3): [a, b, c]}`

- Erzeugen eines leeren Dictionaries

```
T1 = {}; T1
```

`{}`

Konstruktion und Zugriff

- Zugriff auf ein Dictionary

```
T[a], T[1, 2], T[c]
```

```
(d, 5, d)
```

```
float(T[f](4))
```

```
-0.7568024953079282
```

Wird ein Index nicht gefunden, gibt es eine Fehlermeldung

- Löschen von Einträgen

```
del T[a]
```

Befehle wie bei Listen

<code>in</code>	Es wird überprüft, ob ein Index in einem Dictionary vorkommt.
<code>filter</code>	Filtert ein Dictionary nach Kriterien. Prüft sowohl Werte als auch Indizes.
<code>map</code>	Wendet eine Funktion auf die Werte an.

Beispiele

`in` prüft ob a oder b als Index in einem Dictionary auftaucht.

```
Z = {}; Z[a] = b; Z[c] = d; Z[x] = b  
a in Z, b in Z
```

```
(True, False)
```

a taucht als Index auf, b nur als Wert.

`filter` prüft ebenfalls nur den Index.

```
f(y) = y == c; filter(f,Z)
```

```
[c]
```

```
f(y) = y == d; filter(f,Z)
```

```
[]
```

- Dictionaries können aneinander gehängt werden:

```
T={1:a,2:b}; S={3:c,4:d}  
T.update(S); T
```

{1: a, 2: b, 3: c, 4: d}

Vorsicht: Doppelt auftretenden Indizes werden überschrieben!

- 1 Datencontainer in Sage
- 2 Lineare Abbildungen**
- 3 Eigenwerte und Eigenvektoren

Lineare Abbildungen

Seien K -Vektorräume V und W gegeben. Eine Abbildung

$$F: V \rightarrow W$$

heißt **linear**, falls für $v, w \in V$ und $\lambda \in K$ gilt:

(L1) $F(v + w) = F(v) + F(w)$

(L2) $F(\lambda \cdot v) = \lambda \cdot F(v)$

Ist F bijektiv, so heißt F **Isomorphismus**.

Gilt $V = W$, so spricht man von einem **Endomorphismus**. Im Falle von $V = W$ und Bijektivität spricht man von einem **Automorphismus**.

Bemerkungen

- Sei $(v_i)_{i \in I}$ eine Basis in V und $(w_i)_{i \in I}$ seien Vektoren in W . Dann gibt es genau eine lineare Abbildung $F: V \rightarrow W$ mit $F(v_i) = w_i$ für alle $i \in I$.
- Das **Bild** von F ist $\text{Im}(F) = F(V) := \{F(v), v \in V\}$.
- Der **Kern** von F ist $\text{Ker}(F) := \{v \in V \mid F(v) = 0\}$
- Kern und Bild sind Untervektorräume.
- Dimensionsformel:

$$\dim V = \dim F(V) + \dim \text{Ker}(F)$$

- Die Menge der linearen Abbildungen von V nach W wird mit $\text{Hom}_K(V, W)$ bezeichnet. Sie ist ein Vektorraum durch punktweise Addition und Skalarmultiplikation.

- Jeder Matrix $A \in K^{m \times n}$ läßt sich durch

$$L_A : K^n \rightarrow K^m, \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \mapsto A \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

eine lineare Abbildung zuordnen.

- Es gilt $\dim(L_A(K^n)) = \text{Rang}(A)$.

Sei V ein K -Vektorraum mit Basis $\mathcal{V} = (v_1, \dots, v_n)$.

- Die lineare Abbildung $\Phi_{\mathcal{V}} : K^n \rightarrow V$ mit

$$\Phi_{\mathcal{V}}(x_1, \dots, x_n) = x_1 v_1 + \dots + x_n v_n$$

ist ein Isomorphismus. Man nennt $\Phi_{\mathcal{V}}$ ein **Koordinatensystem** in V und $x = (x_1, \dots, x_n) = \Phi_{\mathcal{V}}^{-1}(v)$ den **Koordinatenvektor** zu $v \in V$.

- Ist \mathcal{Z} eine weitere Basis in V , so erhält man die Basiswechselabbildung von \mathcal{V} nach \mathcal{Z} durch $T := \Phi_{\mathcal{Z}}^{-1} \circ \Phi_{\mathcal{V}}$.

Seien K -Vektorräume V und W mit Basen $\mathcal{V} = (v_1, \dots, v_n)$ und $\mathcal{W} = (w_1, \dots, w_m)$ gegeben.

Für eine Matrix $A \in K^{m \times n}$ wird durch

$$\begin{aligned} F(v_1) &:= a_{11}w_1 + \cdots + a_{m1}w_m \\ &\vdots \\ F(v_n) &:= a_{1n}w_1 + \cdots + a_{mn}w_m \end{aligned}$$

eine lineare Abbildung F definiert. Dies ergibt einen Isomorphismus

$$L_{\mathcal{W}}^{\mathcal{V}} : K^{m \times n} \rightarrow \operatorname{Hom}_K(V, W), \quad A \mapsto F.$$

Kanonisches Beispiel

Seien K^n und K^m mit den kanonischen Basen \mathcal{K}_n und \mathcal{K}_m versehen.

- Die Abbildungen $\Phi_{\mathcal{K}_n}$ und $\Phi_{\mathcal{K}_m}$ sind Identitäten.
- Die Abbildung $L_{\mathcal{K}_m}^{\mathcal{K}_n}$ ist gegeben durch

$$L_{\mathcal{K}_m}^{\mathcal{K}_n}(A)(x) = Ax, \quad x \in K^n.$$

- Die Spaltenvektoren von A sind die Bilder der Einheitsvektoren unter der Abbildung $L_{\mathcal{K}_m}^{\mathcal{K}_n}(A)$.

Kommutierendes Diagramm

Seien K -Vektorräume V und W mit Basen $\mathcal{V} = (v_1, \dots, v_n)$ und $\mathcal{W} = (w_1, \dots, w_m)$ und eine lineare Abbildung F gegeben. Dann gilt das folgende kommutierende Diagramm:

$$\begin{array}{ccc} V & \xrightarrow{F} & W \\ \Phi_V \uparrow & & \uparrow \Phi_W \\ K^n & \xrightarrow{(L_{\mathcal{W}}^{\mathcal{V}})^{-1}(F)} & K^m \end{array}$$

Drehung und Spiegelung I

Drehung um den Winkel α - Drehmatrix G :

$$G(\alpha) := \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

```
var('a,b'); A = matrix([[cos(a),-sin(a)],[sin(a),cos(a)]])  
A(a=pi/2)*vector([1,1])
```

$(-1, 1)$

Drehung und Spiegelung II

Spiegelung bezüglich der Ebene

$$H(a) := \{x \in \mathbb{R}^3 | x^T a = 0\}, \|a\| = 1$$

durch

$$S(a) := I - 2aa^T.$$

```
a = matrix(3,1,[1,2,3])  
a = a/norm(a)  
I_n = identity_matrix(3)  
S = I_n - 2*a*a.transpose()  
norm(S*S-I_n)
```

0.0

- 1 Datencontainer in Sage
- 2 Lineare Abbildungen
- 3 Eigenwerte und Eigenvektoren**

Eigenwerte und Eigenvektoren

Sei $A \in K^{n \times n}$. Ein Element $\lambda \in K$ heißt **Eigenwert** von A , wenn ein $x \in K^n \setminus \{0\}$ existiert,

$$Ax = \lambda x$$

gilt. Der Vektor $x \in K^n$ heißt **Eigenvektor** zum Eigenwert λ .

- Die Eigenwerte sind die Nullstellen des **charakteristischen Polynoms**

$$p(t) := \det(A - t I_n).$$

- Es gibt höchstens n Eigenwerte.

- Eigenvektoren zu paarweise verschiedenen Eigenwerten sind linear unabhängig.
- Gibt es eine Basis aus Eigenvektoren, so ist A **diagonalisierbar**, d.h. man kann die Abbildung L_A bei geeigneter Basiswahl durch eine Diagonalmatrix repräsentieren.
- Jeder Endomorphismus eines komplexen Vektorraums läßt sich durch eine Matrix in **Jordanscher Normalform** darstellen.

- Bestimmung von Eigenwerten

```
_ = var('a1'); A = matrix([[cos(a1), sin(a1)], [sin(a1), -cos(a1)]])  
[ m.full_simplify() for m in A.eigenvalues()]
```

`[-1, 1]`

- Bestimmung von Eigenvektoren

```
A.eigenvectors_right()
```


- Bestimmung des charakteristischen Polynoms

```
E = identity_matrix(2)
p = (A-x*E).det(); p
```

```
(x - cos(a1))*(x + cos(a1)) - sin(a1)^2
```

```
[m.full_simplify() for m in solve(p==0,x)]
```

```
[x == -1, x == 1]
```

Lineare Gleichungssysteme (LGS)

Sei $A \in K^{m \times n}$ und $b \in K^m$. Gesucht ist die Menge der Lösungen (Lösungsraum):

$$\{x \in K^n \mid Ax = b\}$$

- Ist $b = 0$, so spricht man von einem **homogenen System**. Ansonsten spricht man von einem **inhomogenen System**.
- Der Lösungsraum W des homogenen Systems bildet einen Untervektorraum des K^n . Die Dimension ist

$$\dim(W) = n - \text{rang}(A).$$

Struktur des Lösungsraums

- Die Lösungen des inhomogenen Systems ($b \neq 0$) bilden einen affinen Unterraum des K^n . $X \subset K^n$ heißt **affiner Unterraum**, wenn ein Unterraum W von K^n und ein $v \in K^n$ existiert, so dass

$$X = v + W$$

gilt. Der Unterraum W ist durch X eindeutig bestimmt, v kann jeder Vektor aus X sein.

- Ist W der Lösungsraum des homogenen Systems und $v \in K^n$ eine beliebige Lösung von $Ax = b$, dann ist der Lösungsraum X von $Ax = b$ gegeben durch $X = v + W$.
- Zwei Lösungen des inhomogenen Systems unterscheiden sich durch eine Lösung des homogenen Systems.

- Das inhomogene System ist genau dann für alle b lösbar, wenn $\text{rang}(A) = m$ gilt.
- Das homogene bzw. das inhomogene System besitzt höchstens eine Lösung, genau dann wenn $\text{rang}(A) = n$ gilt.
- Der Lösungsraum des inhomogenen Systems ist genau dann nicht leer, wenn $\text{rang}(A) = \text{rang}(A, b)$ gilt.
- Praktisch kann ein LGS mit dem **Gausschen Eliminationsverfahren** gelöst werden.

Berechnung der Lösungen von $Ax = b$:

```
A = matrix([[1,2,3],[4,5,6],[7,8,9]])  
b1 = vector([0,0,0])  
b2 = vector([1,0,0])  
b3 = A*b2  
print A\b1  
print A\b3
```