

Jordan_Normal_Form_5x5

June 25, 2025

In diesem Jupyter-Notebook wollen wir die Berechnung einer Jordan'schen Normalform einmal genau unter die Lupe nehmen. Wir machen das an einem konkreten Beispiel, und gehen dabei Schritt-für-Schritt vor, so wie man es von Hand machen würde. Allerdings nutzen wir den Computer für alle Rechnungen – zum Üben, bzw. für die Klausurvorbereitung ist das eine sinnvolle Hilfe. Natürlich können Eigenwerte, JNF, etc. mit Software auch direkt berechnet werden; die entsprechenden high-level Befehle finden Sie ganz am Ende von diesem Dokument. Zur Information: die hier verwendete Software ist Python bzw. die Bibliothek Sympy.

Ein paar Erklärungen zur Programmiersprache: - `eye(n)` ist die $(n \times n)$ -Einheitsmatrix

```
[1]: from sympy import Matrix, symbols, eye, init_printing, simplify
init_printing()
x = symbols('x')
```

Hier eine Matrix A. Gesucht ist eine Matrix J in JNF und eine invertierbare Matrix S sodass gilt: $S^{-1} A S = J$

```
[50]: A = Matrix([
    [5, 4, 2, 1, 0],
    [0, 5, 1, 0, 0],
    [0, 0, 5, 1, 0],
    [0, 0, 0, 5, 1],
    [0, 0, 0, 0, 5]
])
A
```

```
[50]: 
$$\begin{bmatrix} 5 & 4 & 2 & 1 & 0 \\ 0 & 5 & 1 & 0 & 0 \\ 0 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

```

Schritt 1: Wir brauchen die Eigenwerte mit Vielfachheit. Diese Informationen können wir aus dem charakteristischen Polynom ablesen. Also berechnen wir das.

```
[24]: Ax = x * eye(5) - A
Ax
```

```
[24]:
```

$$\begin{bmatrix} x-5 & -4 & -2 & -1 & 0 \\ 0 & x-5 & -1 & 0 & 0 \\ 0 & 0 & x-5 & -1 & 0 \\ 0 & 0 & 0 & x-5 & -1 \\ 0 & 0 & 0 & 0 & x-5 \end{bmatrix}$$

```
[25]: char_poly = Ax.det()
char_poly
```

```
[25]: (x - 5)5
```

```
[26]: char_poly.expand()
```

```
[26]: x5 - 25x4 + 250x3 - 1250x2 + 3125x - 3125
```

```
[51]: char_poly.factor()
```

```
[51]: (x - 5)5
```

Wir sehen hier folgende Dinge: - das charakteristische Polynom zerfällt sogar über RR in Linearfaktoren. Also existiert die JNF mit Transformation über RR. - 5 ist der einzige Eigenwert und dieser hat Vielfachheit 5 - Die JNF sieht so aus: $\begin{bmatrix} 5 & * & 0 & 0 & 0 \\ 0 & 5 & * & 0 & 0 \\ 0 & 0 & 5 & * & 0 \\ 0 & 0 & 0 & 5 & * \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$

Schritt 2: Wir gehen jetzt jeden der Eigenwerte durch und untersuchen wo in J 1en bzw. 0en auf der Nebendiagonalen stehen. Am Ende werden wir J vollständig kennen.

Wir fangen an mit $\lambda = 5$ und schauen uns die Matrix $A - 5I$ an. Genauer geht es um die Kerne von $(A - 5I)^k$ für wachsendes k . Die Kerne berechnen wir jeweils mit dem Gauß-Algorithmus (in Python Befehl "rref()" = reduced row echelon form = reduzierte Zeilenstufenform).

```
[28]: # Nullspace of (A - I)
lamb = 5
Alamb = (A - lamb * eye(5))
Alamb
```

```
[28]:  $\begin{bmatrix} 0 & 4 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ 
```

```
[35]: N = Alamb.rref()
# rref() gibt eine Liste mit zwei Einträgen: die reduzierte Zeilenstufenform
# und die Liste der Pivot Indizes. Wir brauchen nur den ersten Eintrag:
N = N[0]
N
```

```
[35]:
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Wir sehen: der Rang ist 4, also der Kern 1-dimensional. Das Young-Diagramm hat also nur eine Spalte. Entsprechend gibt es nur ein Jordankästchen und damit kennen wir die JNF von A jetzt schon: fünf 5en auf der Diagonale und vier 1en auf der Nebendiagonalen.

Schritt 3: wir bestimmen die Transformationsmatrix. Da wir die Form des Young-Diagramms schon kennen ist klar: $\ker((A - \lambda I)^k)$ wird stationär sobald es der ganze $\mathbb{R}^4$ ist und das passiert bei $k = 5$. Außerdem gilt $\dim \ker((A - \lambda I)^k) = k$. Wir prüfen das mal eben nach:

- (“Matrix ** 3” ist Matrix potenzieren, hier also $\text{Matrix}^3 = \text{Matrix} \cdot \text{Matrix} \cdot \text{Matrix}$)
- `nullspace()` berechnet eine Basis des Kerns
- `len()` ist Länge einer Liste, hier also Anzahl der Basisvektoren in einer Basis, also Dimension.

```
[52]: N2 = (Alamb**2).nullspace()
      N3 = (Alamb**3).nullspace()
      N4 = (Alamb**4).nullspace()
      N5 = (Alamb**5).nullspace()

      print("Dimension von ker((A - I)^k) für k=1..5:")
      print(f"k=1: {len(N1)}")
      print(f"k=2: {len(N2)}")
      print(f"k=3: {len(N3)}")
      print(f"k=4: {len(N4)}")
      print(f"k=5: {len(N5)}")
```

Dimension von $\ker((A - I)^k)$ für $k=1..5$:

```
k=1: 1
k=2: 2
k=3: 3
k=4: 4
k=5: 5
```

Wir brauchen jetzt einen Vektor v , welcher eine Basis von N_4 zu einer Basis von $\mathbb{R}^5$ ergänzt.

```
[53]: N4
```

```
[53]:  $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ 
```

```
[54]: v = Matrix([
      [0],[0],[0],[0],[1]
    ])
```

```
v
```

```
[54]:  $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ 
```

Das ist die letzte Spalte der Transformationsmatrix S! Die übrigen Spalten ergeben sich jetzt von rechts nach links als:

```
[55]: v2 = A*lambda*v  
v2
```

```
[55]:  $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ 
```

```
[56]: v3 = A*lambda*v2  
v3
```

```
[56]:  $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ 
```

```
[58]: v4 = A*lambda*v3  
v4
```

```
[58]:  $\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 
```

```
[59]: v5 = A*lambda*v4  
v5
```

```
[59]:  $\begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 
```

```
[81]: S = Matrix.hstack(v5, v4, v3, v2, v) #hstack schreibt mehrere Matrizen  
↳nebeneinander in eine große Matrix
```

```
S
```

```
[81]: 
$$\begin{bmatrix} 4 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```

Wir prüfen nach: $\det(S)$ ist nicht 0, also ist S invertierbar.

```
[75]: S.det() != 0
```

```
[75]: True
```

Wir bestimmen die Inverse von S ganz klassisch mit Gauß-Algorithmus und dabei eine Einheitsmatrix nebendran mitnehmen.

```
[77]: B = Matrix.hstack(S, eye(5))
      B = B.rref()[0]
      Sinv = B[:, 5:] #der Befehl nimmt die rechte 5x5-Teilmatrix aus B heraus
      Sinv
```

```
[77]: 
$$\begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & -\frac{1}{4} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```

Hier der Nachweis, dass Sinv invers zu S ist:

```
[79]: S*Sinv
```

```
[79]: 
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```

```
[80]: Sinv*S
```

```
[80]: 
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```

Hier also die Früchte unserer Arbeit: S transformiert A auf die vorhergesagt JNF:

```
[78]: Sinv*A*S
```

```
[78]:
```

$$\begin{bmatrix} 5 & 1 & 0 & 0 & 0 \\ 0 & 5 & 1 & 0 & 0 \\ 0 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Abschließende Bemerkungen: - Natürlich kann man J und S auch mit einer eingebauten Funktion “jordan_form()” berechnen. - Matrix invertieren geht mit der Funktion inv() auch direkt. - Eigenwerte und deren Vielfachheit findet man eigenvals().

```
[65]: J = A.jordan_form()
      J
```

```
[65]:  $\left( \begin{bmatrix} 4 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 5 & 1 & 0 & 0 & 0 \\ 0 & 5 & 1 & 0 & 0 \\ 0 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix} \right)$ 
```

```
[83]: S.inv()
```

```
[83]:  $\begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & -\frac{1}{4} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ 
```

```
[84]: eigenvals = A.eigenvals()
      eigenvals
```

```
[84]: {5: 5}
```

```
[ ]:
```