Die Berechnung der Jordanschen Normalform reeller Matrizen über ℂ in C

Dokumentation zum Numerikpraktikum von Sascha Richter[1], Matrikelnummer 9980878
Universität Leipzig im Sommersemester 2007

Zusammenfassung
Es wird ein Algorithmus und ein C-Programm zur Bestimmung der Jordanschen Normalform einer reellen Matrix, die als Matrix über ℂ aufgefasst wird, vorgestellt.

Stichworte: Jordansche Normalform, Numerische Lineare Algebra, Eigenwertproblem, QR-Zerlegung, Rangbestimmung

Inhalt

0. Einleitung ... 2
1. Ein Algorithmus – kommentiert ... 3
1. Berechne die Eigenwerte λ_j von A ... 4
2. Ermittle die algebraischen Vielfachheiten der Eigenwerte ... 5
3. Ermittle die geometrischen Vielfachheiten der Eigenwerte ... 5
4. „Algorithmus von Bosch “ ... 6
5. „Zusammenpuzzeln“ ... 7
2. Der Algorithmus – Kurzfassung in Pseudocode ... 7
3. Andere mögliche Algorithmen ... 8
4. Design- und Implementierungsaspekte ... 8
5. Programmerstellung und -benutzung ... 9
Literaturverzeichnis ... 10
Anhang 1 ... 11
Liste der selbst implementierten Funktionen ... 11
Anhang 2 ... 11
Darstellung der Abhängigkeit einzelner Funktionen voneinander ... 11
Anhang 3 ... 12
Instabilität der Berechnung der JNF am Beispiel von Mathematica ... 12
Anhang 4 ... 13
Dokumentation zu den mitgelieferten Beispielen ... 13

[1] Max-Metzger-Strasse 7, 04157 Leipzig, tiruvarur@gmx.de

0. Einleitung[2]

Nicht jede Matrix ist diagonalisierbar, aber über jedem algebraisch abgeschlossenen Körper (also insbesondere über ℂ) gibt zu jeder Matrix A Matrizen Q und J, so dass[3] $A=Q^{-1}JQ$. Dabei ist J – die sogenannte Jordansche[4] Normalform (JNF) von A – von folgender Gestalt:

$$J=\begin{pmatrix} J_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & J_\mu \end{pmatrix},$$ die J_i wiederum haben die Bauart $J_{i_k}=\begin{pmatrix} \lambda_i & 1 & 0 & \dots & 0 \\ 0 & \lambda_i & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & \lambda_i \end{pmatrix}$ und heissen

„Jordankästchen“ oder „Jordanblöcke“. Bei diesen steht in der Diagonalen ein und derselbe Eigenwert von A und in der rechten[5] benachbarten Diagonalen stehen Einsen, ansonsten stehen überall Nullen. Die Anzahl der Spalten bzw. Zeilen der quadratischen Matrix J_{i_k} nennen wir *Grösse* des Jordanblocks.

Beispiele:

(λ)	$\begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}$	$\begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}$
Jordankästchen der Grösse 1	Jordankästchen der Grösse 2	kein Jordankästchen (bzw. 2 Jordankästchen der Grösse 1)

Im Falle einer diagonalisierbaren Matrix haben alle Jordankästchen die Grösse 1 und J ist eine Diagonalmatrix mit den Eigenwerten als Einträge in der Diagonalen[6].

Unter den *Eigenwerten* einer Matrix A versteht man die Lösungen $\{\lambda_i\}_{i=1\dots r}$ der Gleichung $\det(A-\lambda E)=0$, wobei E eine geeignet dimensionierte Einheitsmatrix ist. Wenn man diese (paarweise verschiedenen) Lösungen bereits hat, dann ist $\det(A-\lambda E)=(\lambda-\lambda_1)^{n_1}(\lambda-\lambda_2)^{n_2}\dots.(\lambda-\lambda_k)^{n_r}$. Dabei heisst n_j die *algebraische Vielfachheit* des Eigenwertes λ_j.

Satz[7]: Der i-te Eigenwert steht genau n_i-mal in der Hauptdiagonalen der JNF (ggf. auf mehrere Jordanblöcke verteilt).

Unter der *geometrischen Vielfachheit* des Eigenwertes λ_j versteht man die Dimension des von den *Eigenvektoren* zu diesem Eigenwert aufgespannten Vektorraums, d.h. den Vektoren x, die die Gleichung $Ax=\lambda_j x$ erfüllen.

Schreibweise: Geometrische Vielfachheit $(\lambda_j) = \dim_{\mathbb{C}}\ker(A-\lambda_j E)$

Dem sogenannten Dimensionssatz zufolge gilt #Zeilen(A)=#Spalten(A)= $\dim_{\mathbb{C}}\ker(A) + \mathrm{rg}(A)$.

(Dabei bezeichnet rg(A) den *Rang* von A = die Anzahl der linear unabhängigen Zeilen von A).

Satz: Die Anzahl der Jordankästchen zu einem Eigenwert λ ist gleich $\dim_{\mathbb{C}}\ker(A-\lambda E)$.

Beispiel:

r	1							
0	r							
		r	1	0	0	0	0	0
		0	r	1	0	0	0	0
		0	0	r	1	0	0	0
		0	0	0	r	1	0	0
		0	0	0	0	r	1	0
		0	0	0	0	0	r	1
		0	0	0	0	0	0	r

r	1	0						
0	r	1						
0	0	r						
			r	1	0	0	0	0
			0	r	1	0	0	0
			0	0	r	1	0	0
			0	0	0	r	1	0
			0	0	0	0	r	1
			0	0	0	0	0	r

[2] Zur Theorie der JNF (als Objekt der Linearen Algebra) siehe etwa Bosch, S.: Lineare Algebra, Springer 2006

[3] Matrizen A, B, für die es eine invertierbare Matrix Q gibt, so dass $A=Q^{-1}BQ$, heissen *ähnlich*. Ähnliche Matrizen haben z.B. dieselben Eigenwerte, dieselbe Determinante und denselben Rang.

[4] Nach Camille Jordan, 1838-1922: Traité des substitutions et des équations algébriques, 1870

[5] Das ist eine Konvention, es kann auch die linke Nachbardiagonale sein. Bei den Nachbardiagonalen spricht man auch von „Super“- und „Sub“diagonale (für „über“ bzw. „unter“).

[6] „Diagonalisierbarkeit“ bedeutet gerade „Ähnlichkeit zu einer Diagonalmatrix“

[7] Je nach dem, wie man seine Theorie aufbaut, kann dies auch eine Definition sein.

Das seien die JNFen zu 2 Matrizen. Man sieht: diese Matrizen haben jeweils genau einen Eigenwert r mit der algebraischen Vielfachheit 10 und der geometrischen Vielfachheit 2, da es zu diesem Eigenwert jeweils zwei Jordankästchen gibt. Man sieht ferner: die Anzahl der Einsen in der Nebendiagonalen ist durch die geometrische Vielfachheit bereits festgelegt, aber die Grössen der Jordanblöcke sind es nicht.

Wenn man die Eigenwerte, ihre algebraischen und die geometrischen Vielfachheiten und die Anzahlen k_{ij}, mit welcher die Jordanblöcke der Grösse j zum i-ten Eigenwert vorkommen, kennt, dann hat man alle Informationen, die in der JNF codiert sind[8].

Im Prinzip kann man für jede feste Zeilenzahl n der Ausgangsmatrix A alle Formen auflisten, die die JNF von A haben kann. Beispielsweise für n=2:

$\begin{pmatrix} \lambda & 0 \\ 0 & \rho \end{pmatrix}$	$\begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}$	$\begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}$
2 verschiedene Eigenwerte	Ein Eigenwert, geom.Vielf. = 2	Ein Eigenwert, geom. Vielf. = 1

Die JNF einer Matrix ist eindeutig bis auf die Reihenfolge der Jordankästchen.

Bemerkung: $\mathbb{R}$ ist nicht algebraisch abgeschlossen, dennoch kann man zu einer Matrix mit Einträgen aus $\mathbb{R}$ eine sogenannte *reelle JNF* definieren, bei der J und die Transformationsmatrix nur reelle Einräge aufweisen. J hat dabei im Allgemeinen von Null verschiedene Einträge in *drei* benachbarten Diagonalen.

Aus der JNF einer Matrix A kann man relativ einfach folgende Informationen gewinnen:

- Eigenwerte von einer Matrix: stehen in der Diagonalen der JNF
- die algebraischen Vielfachheiten der Eigenwerte: Anzahl des Vorkommens eines Eigenwertes in der Diagonale der JNF
- die geometrischen Vielfachheiten der Eigenwerte: Anzahl der Jordankästchen zu dem jeweiligen Eigenwert
- Diagonalisierbarkeit von A: ist gegeben, wenn die JNF eine Diagonalmatrix ist
- det A= Produkt der Hauptdiagonalelemente der JNF
- Rang A = Rang JNF; letzterer ist einfach zu bestimmen, da die JNF bereits in Dreiecksform ist
  Speziell: Singularität der Matrix liegt vor, wenn 0 in der Hauptdiagonalen der JNF vorkommt, also wenn 0 ein Eigenwert der Matrix ist.
- Ähnliche Matrizen haben dieselbe[9] JNF; insofern liefert die JNF ein Ähnlichkeitskriterium

Die JNF ist eignet sich gut als Repräsentant für einen Endomorphismus eines Vektorraums. Mit ihr kann man relativ einfach hohe Potenzen von Matrizen berechnen, denn $A^m = (Q^{-1}JQ)^m = Q^{-1}J^m Q$.

Die JNF spielt eine Rolle bei der Lösung des Systems von Differentialgleichungen $Ay(x)=\frac{d}{dx}y(x)$.

1. Ein Algorithmus – kommentiert

Mit dem in der Einleitung Gesagten wäre „die Berechnung der JNF“ dadurch möglich, dass man die Transformationsmatrix Q berechnet, um durch Umstellung der Gleichung $A=Q^{-1}JQ$ nach J dieses J zu ermitteln. Manchmal[10] versteht man unter „Berechnung der JNF“ tatsächlich die Berechnung des Paares (Q, J). Demgegenüber will ich nur die JNF ermitteln.
Der folgende Algorithmus berechnet die[11] JNF einer Matrix[12] $A \in \mathbb{R}^{n \times n}$.

---

[8] Zur Bestimmung der k_{ij} siehe Kapitel 1.4.
[9] Bis auf Permutation der Jordankästchen.
[10] Z.B. liefert Mathematica mit JordanDecomposition[m] neben der JNF von m auch die Transformationsmatrix. Allerdings muss man den Ergebnissen misstrauen, siehe Anhang 3 !
[11] bis auf Reihenfolge der Jordankästchen eindeutige
[12] $A \in \mathbb{R}^{n \times n}$ bedeutet: wir lassen als Eingabe nur solche Matrizen $A \in \mathbb{C}^{n \times n}$ zu, deren Imaginärteil =0 ist.

1. Berechne die Eigenwerte λ_j von A

Die Eigenwerte $\{\lambda\}$ von $A \in \mathbb{R}^{n \times n}$ kann man grundsätzlich durch auflösen der Gleichung $\det(A-\lambda E)=0$ nach λ ermitteln. Für $n \leq 4$ ist dies stets exakt[13] möglich. Für $n \geq 5$ ist dies bekanntlich möglich, wenn die Galoisgruppe des charakteristischen Polynoms auflösbar ist; diese „Auflösbarkeit" wiederum ist algorithmisch feststellbar[14]. Man weiss auch, dass sich Lösungen von Gleichungen 5ten Grades stets „in terms of Jacobi theta functions"[15] und die Lösungen von Polynomgleichungen höheren Grades stets „in terms of Siegel modular functions"[16] ausdrücken lassen[17].
Ein numerischer Ansatz zur Lösung von $\det(A-\lambda E)=0$ bestünde darin, eine Interpolation der linken Seite zu suchen[18].

Spezielle Matrizen haben spezielle Eigenwerte. So haben z.B. reelle symmetrische Matrizen nur reelle Eigenwerte und schiefsymmetrische Matrizen haben ausser 0 keinen reellen Eigenwert. Wenn man nun a priori oder nach programminterner Überprüfung der Eingabematrix weiss, dass man eine spezielle Matrix vorliegen hat, dann kann man entweder spezielle Algorithmen zur Eigenwertermittlung verwenden oder, wenn man Algorithmen verwendet, die die Eigenwerte allgemeiner Matrizen ermitteln, deren Output einer Plausibilitätsprüfung unterziehen oder dieses verwerfen oder korrigieren. Wenn man bspw. den QR.-Algorithmus (s.u.) auf eine reelle symmetrische Matrix angewendet und nicht-verschwindende Imaginärteile für einige Eigenwerte erhalten hat, dann könnte man diese im einfachsten Fall gleich 0 setzen.

Ein perfekter Algorithmus zur Bestimmung der Eigenwerte sollte – für unsere Zwecke – all dies berücksichtigen und versuchen, durch Ausnutzung der speziellen Gestalt des charakteristischen Polynoms bzw. der Eingabematrix die Eigenwerte „so exakt wie möglich" zu bestimmen, und nur, wenn all dies erfolglos geblieben ist, zu „numerischen" Algorithmen im engeren Sinne greifen.
Die numerischen Algorithmen zur Eigenwertbestimmung lassen sich danach klassifizieren

- ob sie gewisse Forderungen an A wie Symmetrie oder Definitheit stellen und
- ob sie *alle* Eigenwerte oder nur *bestimmte* (wie z.B. den Betragsgrössten) Eigenwerte berechnen[19].

Unseren Zweck verfolgend müssen wir zu einem Verfahren greifen, dass alle Eigenwerte einer reellen Matrix berechnet, über die nichts vorausgesetzt wird.
Es wird das QR-Verfahren verwendet[20]. Idee: Zerlege A multiplikativ in eine orthogonale Matrix Q und eine obere Dreiecksmatrix R: A=QR. Die Matrix R enthält in ihrer Diagonale die Eigenwerte von A.
Den *Numerical Recipes in C* [21] folgend, werden dazu folgende Operationen ausgeführt:

- „Balancing"[22] von A

---

[13] genauer: durch – ggf. iterierte – Wurzelausdrücke („Radikale")

[14] Distler, A.:Ein Algorithmus zum Lösen einer Polynomgleichung durch Radikale. Diplomarbeit, TU Braunschweig, 2005. http://www.icm.tu-bs.de/ag_algebra/software/distler/Diplom.pdf.

[15] Hermite, C. "Sulla risoluzione delle equazioni del quinto grado." Annali di math. pura ed appl. 1, 256-259, 1858. zitiert in: http://mathworld.wolfram.com/Polynomial.html

[16] Umemura, H.: Solution of algebraic equations in terms of theta constants. In D. Mumford, Tata Lectures in Theta II, Progress in Mathematics 43, Birkhäuser, Boston, 1984. zitiert in: http://en.wikipedia.org/wiki/Polynomial#Solving_polynomial_equations

[17] Das muss aber nicht immer so kompliziert sein: so kann man die Lösungsmenge der Gleichung $\lambda^{100}=1$ auch ohne Abiturkenntnisse ermitteln.

[18] *Mathematica* geht diesen Weg in bestimmten Fällen; Mathematica-Dokumentation A 9.4

[19] Dieses Kriterium ist relativ willkürlich, denn über den sogenannten Spektralsatz kann man, wenn man einen Eigenwert hat, den nächsten und somit alle Eigenwerte bestimmen.

[20] Siehe dazu bspw. Engeln-Müllges / Reutter: Numerik-Algorithmen mit ANSI C-Programmen. BI-Wissenschaftsverlag 1993, Seiten 107ff und S.175 ff oder
Quarteroni, Sacco, Saleri: Numerische Mathematik 1, Springer 2002, S.214-235

[21] Press, W. et al.: Numerical Recipes in C, Cambridge University Press 1992, Kapitel 11.5 und 11.6 (online: http://www.nrbook.com/a/bookcpdf.php)

[22] Numerical Recipes Kap.11.5: S. 483: „The sensitivity of eigenvalues to rounding errors during the execution of some algorithms can be reduced by the procedure of *balancing*."
Das Balancing wird auch erklärt und empfohlen in Faires / Burden: Numerische Methoden, Spektrum 1994,

- Bringen von A auf die „obere (reelle) Hessenberg-Form“[23]
- Anwenden des QR-Algorithmus auf die Hessenberg-Form[24].

2. Ermittle die algebraischen Vielfachheiten der Eigenwerte

In der Diagonalen von R aus dem letzten Schritt stehen die n Eigenwerte, die aber ggf. mehrfach auftauchen: diese Anzahlen sind die algebraische Vielfachheiten.
Wir müssen also folgende Transformation durchführen:

$$(\sigma \quad \sigma \quad \tau \quad \sigma \quad .... \quad \rho) \mapsto \begin{pmatrix} \sigma & \tau & \rho & .... \\ n_\sigma & n_\tau & n_\rho & .... \end{pmatrix}$$

Dabei hat der Eingabevektor n Spalten und die Ergebnismatrix hat r Spalten (= Anzahl verschiedener Eigenwerte) und n_* ist die algebraische Vielfachheit des Eigenwertes $*$.
Dieser Schritt stellt eine numerische Herausforderung dar: insofern wir mit endlicher Genauigkeit rechnen wollen, werden in der Regel *alle* Eigenwerte verschieden sein ! Man muss sich also eine Logik überlegen, unter welcher Bedingung man (numerisch verschiedene) Eigenwerte als gleich ansehen will. In unserem Programm wird der naive Ansatz implementiert

wenn $|\sigma_i - \sigma_j| < \epsilon$, dann sind σ_i und σ_j gleich.

Dabei orientiert sich die Wahl von ϵ an der Anzahl der Nachkommastellen, die durch den verwendeten Floatingpoint-Datentyp höchstens dargestellt werden kann.

Kågström und Ruhe[25] sagen „[Die Separation der verschiedenen Eigenwerte] is the central problem in all attempts to determine the Jordan normal form from inaccurate numerical data.“ und empfehlen zur Lösung dieses Problems in ihrem Algorithmus sogenannte Gerschgorin-Kreise[26].
Damit verhält es sich wie folgt[27]. Sei $A \in \mathbb{C}^{n \times n}$ mit n (nicht notwendig verschiedenen) Eigenwerten.
Der i-te Gerschgorin-Kreis ist dann die Menge

$K_i = \{ z \in \mathbb{C} \mid \|z - a_{ii}\| \leqq \sum_{j=1, j \neq i}^{n} \|a_{ij}\| \}$ für i=1..n. Es gilt dann, dass alle Eigenwerte von A in $\bigcup_{i=1}^{n} K_i$ liegen.

Weiterhin enthält jede Vereinigung von m dieser Kreise, welche die übrigen (n–m) Kreise nicht schneiden, genau m Eigenwerte. Es ist klar, dass diese Aussage eine grobe Approximation der Eigenwerte liefert, aber ich habe nicht näher untersucht, wie man das in einen Algorithmus umsetzen kann.

3. Ermittle die geometrischen Vielfachheiten der Eigenwerte

Die geometrischen Vielfachheiten werden nicht gesondert, sondern innerhalb des „Algorithmus von Bosch“ (siehe dort) ermittelt. An dieser Stelle folgende Erläuterung dazu:
Wie in der Einleitung vermerkt, gilt: $n = \dim_{\mathbb{C}} \ker(A - \lambda_j E) + rg(A - \lambda_j E)$, also

$\dim_{\mathbb{C}} \ker(A - \lambda_j E) = n - rg(A - \lambda_j E)$.

Der Rang von A wird mittels Gaussschem Algorithmus mit totaler Pivotisierung bestimmt. Auch die Rangbestimmung einer Matrix stellt eine Herausforderung an die Numerik dar: man muss ja irgendwie entscheiden, ob alle Einträge einer Zeile „gleich Null“ sind. Aufgrund der endlichen Genauigkeit wird das in der Regel nie der Fall sein. Man spricht vom „numerischen Rang“ einer Matrix in Unterscheidung zum theoretischen Rang.
Wenn man a priori oder durch programminterne Logik weiss, dass die Eingabematrix nur Einträge aus $\mathbb{Z}$ und dass deren Eigenwerte nur aus den ganzen Gausschen Zahlen sind, dann kann man zur Rangbestimmung eine spezielle Form des Gausschen Algorithmus verwenden, die ohne Division auskommt.

S. 408 sowie verwendet von Kågström/Ruhe, siehe Fussnote 25, S. 407

[23] Numerical Recipes Kap.11.5; S. 485: „Recommended, but not required, is that this routine be preceded by balanc.“ Trsf. auf obere Hessenbergform findet sich auch bei Kågström/Ruhe, siehe Fussnote 25, S. 407

[24] Numerical Recipes Kap.11.6

[25] Kågström, B., Ruhe, A.: An Algorithm for Numerical Computation of the Jordan Normal Form of a Complex Matrix. in: ACM Transactions on Mathematical Software, Volume 6, Issue 3, S. 398 - 419

[26] Sie verweisen dabei auf: Ruhe, A.: An algorithm for numerical determination of the structure of a general matrix. BIT 10 (1970), 196-216

[27] Die folgenden Ausführungen nach Faires / Burden S. 378.

Als Alternative zu Gauss wird manchmal empfohlen, die SVD[28] zu verwenden. Idee: Jede Matrix A besitzt eine Zerlegung $A=U\Sigma V^*$, wobei U, V unitäre Matrizen aus $\mathbb{C}^{n\,x\,n}$ sind und $\Sigma\in\mathbb{R}^{n\,x\,n}$. Dabei ist $\Sigma=\mathrm{diag}(\sigma_1, \sigma_2,..,\sigma_r,0,0,..,0)$ mit r = rg(A). Die σ_i sind gleich den Quadratwurzeln aus den Eigenwerten von $\overline{A}^T A$, man müsste also wiederum ein Eigenwertproblem – für komplexe Matrizen – lösen.

Eine weitere – aufwendige – Methode zur Rangbestimmung liefert das *Rangkriterium von Frobenius*[29]: Für $A\in\mathbb{R}^{n\,x\,n}$ und für $1\leqq s\leqq n$ ist eine *(sxs)-Untermatrix* von A eine (sxs)-Matrix, die aus A durch Streichen von (n – s) Zeilen und Spalten entsteht. Die Determinante einer (sxs)-Untermatrix heisst eine *s-reihige Unterdeterminante* von A. Es ist nun $\mathrm{rang}(A)= r \geq 1 \Leftrightarrow$

1. A besitzt eine r-reihige Unterdeterminante $\neq 0$. und
2. Jede (r+1)-reihige Unterdeterminante von A ist gleich 0.

Wenn man diese Determinanten ohne Gauss sondern z.B. nach dem Laplacschen Entwicklungssatz berechnet, dann kommt man bei dieser Rangbestimmung vorteilhafterweise ohne Division aus. Während die Determinantenberechnung mit Laplace sehr aufwändig ist, gibt es neuerer Algorithmen, welche die Determinante – ohne Division – in $O(n^4)$ berechnen[30].

Für neueste Entwicklungen zur Rangbestimmung siehe etwa die Homepage von Zhonggang Zeng [31].

4. „Algorithmus von Bosch[32] “

Wir kennen nun die Diagonale der JNF und die Anzahl der Jordankästchen zu jedem Eigenwert, aber folgende Grössen sind noch unbekannt:

k_{ij}: die Anzahl , mit der das Jordankästchen der Grösse j zum Eigenwert i vorkommt.

i=1...r (r: Anzahl verschiedener Eigenwerte)

$j=1...n_i$. (n_i: algebraische Vielfachheit des i-ten Eigenwertes)

Nach S. Bosch gelten folgende Relationen:

$$k_{i,n_i} = \mathrm{rg}(A-\lambda_i E)^{n_i-1} - n + n_i$$
$$k_{i,n_i-1} = \mathrm{rg}(A-\lambda_i E)^{n_i-2} - n + n_i - 2k_{i,n_i}$$
$$k_{i,n_i-2} = \mathrm{rg}(A-\lambda_i E)^{n_i-3} - n + n_i - 3k_{i,n_i} - 2k_{i,n_i-1}$$
$$k_{i,n_i-3} = \mathrm{rg}(A-\lambda_i E)^{n_i-4} - n + n_i - 4k_{i,n_i} - 3k_{i,n_i-1} - 2k_{i,n_i-2}$$
$$\vdots$$
$$k_{i,2} = \mathrm{rg}(A-\lambda_i E)^{1} - n + n_i - (n_i-1)k_{i,n_i} - \ldots\ldots - 2k_{i,3}$$
$$k_{i,1} = \mathrm{rg}(A-\lambda_i E)^{0} - n + n_i - n_i\, k_{i,n_i} - \ldots\ldots - 2k_{i,2}$$

(In der letzten Zeile bedeutet $(A-\lambda_i E)^0 \equiv E$, also $\mathrm{rg}(A-\lambda_i E)^0 = n$.)

Daraus kann man die folgende Rekursion herleiten:

$$\text{mit } s_j = s_{j+1} + k_{i,j+1} + \sum_{m=j+1}^{n_i} k_{im} \qquad (s_{n_i} = 0) \text{ ergibt sich}$$
$$k_{i,j} = \mathrm{rg}(A-\lambda_i E)^{j-1} - n + n_i - s_j \qquad i=1..r, \quad j=(n_i-1)...1$$

Daraus wiederum kann man folgendes Programm in Pseudocode ableiten:

```
for ( j=1..r)
      k_{jn_j}= rg(A-λ_jE)^{n_j-1} -n+n_j
for ( i=1..r)
```

[28] Singular Value Decomposition

[29] nach: Brehmer, S.,Belkner, H.: Analytische Geometrie und lineare Algebra. VEB Deutscher Verlag der Wissenschaften, 1972

[30] Rote, G.: Division-Free Algorithms for the Determinant and the Pfaffian: Algebraic and Combinatorial Approaches. http://page.inf.fu-berlin.de/~rote/Papers/pdf/Division-free+algorithms.pdf

[31] Dort (http://www.neiu.edu/~zzeng/) v.a. „A rank-revealing method with updating, downdating and applications“. Im Erscheinen.

[32] Bosch, S.: Lineare Algebra, S. 234ff

```
s_{n_i} = 0
for(j=(n_i-1)..1)
      s_j  = s_{j+1} + k_{i,j+1}
      for(m=(j+1)..n_i)
            s_j  = s_j + k_{im}
      k_{ij}= rg(A-λ_iE)^{j-1} -n+n_i - s_j
```

In der Wikipedia[33] finden sich Angaben, die mit obigen Bezeichnungen zu folgendem Algorithmus führen:

```
for ( j=1..r)
      for(j=0..n_i)       a_{ij}=n-rg(A-λ_iE)^j
      for(j=1..(n_i-1))   k_{ij}=2a_{ij}-a_{i,j-1} -a_{i,j+1}
```

Dieser Algorithmus rechnet – zumindest in der von mir aufgestellten Form – manchmal falsch.

Es sei angemerkt, dass Bosch noch ein „viel effektiveres Verfahren“[34] vorschlägt. Dieses basiert auf der Berechnung von Elementarteilern und des Minimalpolynoms sowie auf der Anwendung des Chinesischen Restsatzes. Diesen Weg habe ich allerdings nicht weiter verfolgt.

5. „Zusammenpuzzeln“

Wir haben jetzt alle Informationen zusammengetragen, die man dem Programmbenutzer nun irgendwie mitteilen muss. Ein Algorithmus der folgenden Art ist nötig, wenn man die JNF wirklich als Matrix (im Sinne einer Datenstruktur für dichtbesetzte Matrizen) haben will:

```
for(i,j=1...n)
      jnf_{ij}=1
 zc=1   // "Zeilencount"
for(i=1...r)
      for(j... n_i)
            // Füge k_{ij}-mal einen Jordanblock mit Eigenwert λ_i
            //                der  Grösse j an die jnf an
            for(v=1...k_{ij})
                  for(m...j)
                        jnf_{zc,zc}=λ_i
                        if(m≠j) jnf_{zc,zc+1}=1
                        zc++;
```

Im Anhang 1 findet sich eine Liste aller Funktionen, die das Programm verwendet.
Im Anhang 2 findet sich ein Diagramm, das die Abhängigkeit der einzelnen Programmfunktionen voneinander darstellt.

2. Der Algorithmus – Kurzfassung in Pseudocode

Code	Kommentar
ermittle die Eigenwerte	Balancing, Transf. auf obere Hessenbergform, QR
ermitt. die alg. Vielfht.	naiv; siehe Kapitel 1.2
ermittle die k_{ij}	nach Bosch, siehe Kap. 1.4, Rangbestimmung mit Gauss
baue die JNF	siehe Kap. 1.5

Man vergleiche zum Überblick auch mit Anhang 2.

[33] de.wikipedia.org/wiki/Jordansche_Normalform, abgerufen am 10.08.2007; Etwas Ähnliches habe ich auch in einem Paper oder Buch gesehen, weiss aber leider nicht mehr, wo.
[34] S. 238

3. Andere mögliche Algorithmen

Sofern man nicht gerade die JNF haben will, sollte man nicht versuchen, sie mit Mitteln der Numerik zu bestimmen[35]. Dennoch gibt es Algorithmen, die genau dies zu tun beanspruchen. Erwähnenswert sind insbesondere

- der Algorithmus von Kågström und Ruhe[36] und
- der Algorithmus von Zurmühl und Falk[37].

Beide Algorithmen berechnen nicht nur die JNF sondern auch die Transformationsmatrix.

4. Design- und Implementierungsaspekte

- Das vorliegende Programm ist „minimal“ in dem Sinne, dass es nur Funktionen und Konstrukte benutzt, die in jedem C-Compiler-System vorhanden sind[38].
- Geschwindigkeit und Effizienz waren *keine* Designziele.
- Das Programm ist durch die Definition von Funktionen möglichst modular aufgebaut. D.h., wenn z.B. jemand eine bessere als die vorliegende(n) Funktion(en) zur Berechnung der Eigenwerte hat, dann muss er nicht das komplette Programm sondern nur bestimmte Stellen verändern resp. bestimmte Funktionen ersetzen.
  Dem Fluss der Information durch das Programm folgt man am besten, indem man `main()` und die zentrale Funktion `jnf()` sequentiell durchgeht.
- Die Grössen sämtlicher Vektoren und Matrizen (also (ggf. mehrdimensionale[39]) Arrays) werden zur Laufzeit, durch die Dimension der Eingabematrix, festgelegt. In ISO/ANSI–C erfordert dies Konstrukte wie:
  ```
  int  *mein_int_vektor= (int *)malloc((n+1)*sizeof(int));
  ```
  im Gegensatz etwa zu dem Konstrukt
  ```
  int mein_int_vektor[n+1];
  ```
  bei welchem n bereits zur Compilezeit bekannt sein muss.
  Für `malloc` (und `free`) ist die Einbindung der stdlib erforderlich.
- Vektoren und Matrizen werden wie in der Mathematik üblich mit 1 beginnend indiziert; da in C die Indizes von 0 beginnend gezählt werden, muss man also, wenn man einen Vektor mit n Komponenten haben will, ein Feld mit (n+1) Einträgen deklarieren.
- Der Basisdatentyp („BASETYPE“) für alle Floatingpointberechnungen ist der von C bereitgestellte Typ **`double`**. Eine Alternative wäre vielleicht, einen „arbitrary precision arithmetic“-Datentyp entweder selbst zu bauen oder einen solchen aus einer Bibliothek zu übernehmen[40].
- Es wird ein selbstgebauter Datentyp KOMPLEX zur Darstellung komplexer Zahlen verwendet:
  ```
  typedef struct
  {
        BASETYPE re;
        BASETYPE im;
  } KOMPLEX;
  ```
  Diese Designentscheidung lag nahe, als es darum ging, Matrizen mit komplexen Einträgen zu modellieren. Ohne diesen Datentyp würde eine solche zur Laufzeit initialisierte Matrix etwa so zu deklarieren sein:
  ```
  BASETYPE ***x=(BASETYPE ***)malloc((n+1)*sizeof(BASETYPE **));
      for(i=1;i<=n;i++)
            x[i]=(BASETYPE**)malloc((n+1)*sizeof(BASETYPE *));
                  for(i=1;i<=n;i++)
                    for(j=1;j<=n;j++)
  ```

[35] So Z.B. Stewart, G.W.: Matrix Algorithms. SIAM 1998: „[die JNF ist] uncomputable.“ Vgl .auch Anhang 3.
[36] Siehe Fussnote 25
[37] Zurmühl, R., Falk, S.: Matrizen und ihre Anwendungen, Springer 1984, S. 253ff
[38] Man beachte aber die optionale Nutzung einer „matrix.h“, wie in Kapitel 5 beschrieben.
[39] Man kann eine Matrix auch in einem eindimensionalen Array ablegen, muss dann aber bei der Indizierung aufpassen.
[40] Java-Programmierer haben es da leichter: dort gehören die „arbitrary precision arithmetic“-Datentypen `BigInteger` und `BigDecimal` zur Standard-Bibliothek dazu.

```
x[i][j]=(BASETYPE )malloc(2*sizeof(BASETYPE));
```

was vielleicht doch etwas umständlich ist.

5. Programmerstellung und -benutzung

Das Programm besteht aus 2 C-Dateien:

jnf_v02.c
jnf_mymathlib_v02.c.

Die Binary erhält man durch Kompilieren und Linken der ersten Datei, welche die zweite Datei einbindet. Die erste Datei enthält eine Implementierung des Algorithmus´, der in den Kapiteln 1 bzw. 2 beschrieben wurde.

Die Eigenwerte werden durch das Programm noch auf 2 bis 3 weitere Weisen ermittelt:

„mymathlib.c“ enthält einige C-Funktionen, die ein unbekannter Autor im Internet[41] veröffentlicht hat. (Ich habe nur eine Funktion davon selbst implementiert und im Wesentlichen nur Funktionen aus verschiedenen Dateien zusammengeführt.)
Auch in der mymathlib werden die Eigenwerte nach Transformation auf die obere Hessenbergform über die QR-Zerlegung ermittelt; dennoch kommt der Algorithmus gelegentlich zu mehr oder weniger abweichenden Eigenwerten.

Wenn man den Compiler „lccwin32“ benutzt, dann ermittelt das Programm die Eigenwerte noch unter Verwendung von Funktionen der in diesem System implementierten „Meschach-Library[42]“. Das wird bewerkstelligt durch Inkludieren der „matrix.h“. Explizit werden von meinem Programm nur die Datentypen `MAT, VEC` und `MNULL` und die Funktionen `m_get(),v_get(), m_copy(), schur()` und `schur_evals()` verwendet (die aber ihrerseits weitere Konstrukte und Funktionen aus der Bibliothek verwenden).
Wenn man den LCC *nicht* hat, dann kommentiert man einfach die Präprozessordirektive **`#define LCC32`**, die u.a. das Inkludieren der „matrix.h“ bewirkt, aus. Das sollte man auch wirklich tun, denn vermutlich stellen auch andere Compilersysteme eine Datei unter diesem Namen zur Verfügung.
Man könnte aber auch noch folgendes tun:

- Einbinden der kompletten Meschach-Library oder
- untersuchen, ob der eigene Compiler oben genannte Kontrukte und Funktionen unter denselben Namen und mit denselben Funktionalitäten zur Verfügung stellt.

Die Meschach-Library verwendet zur Berechnung der Eigenwerte die Schur-Zerlegung $A = QU\overline{Q}^T$, wobei Q eine unitäre und U eine Dreiecksmatrix ist, in deren Haupdiagonale[43] die Eigenwerte von A, das bei Meschach auch komplex sein darf, stehen. `schur()` scheint sehr zuverlässig zu funktionieren, aber `schur_evals()`, das eigentlich nur eine Matrix auslesen muss, versagt manchmal.

Das Programm nimmt bei seinem Aufruf ein Argument entgegen, das den Namen einer ASCII-Datei enthält, in der die Eingabe-Matrix A gespeichert ist. In dieser müssen, durch Kommata getrennt, zeilenweise die Einträge der Matrix angegeben werden. Das „mathematische Komma“ ist der Punkt (.), der auch dann gesetzt werden muss, wenn der Eintrag eine ganze Zahl ist.
Beispiel:

Datei: beispiel.txt
Inhalt:

```
25.,-16.,30.,-44.,-12.
13.,-7.,18.,-26.,-6.
-18.,12.,-21.,36.,12.
-9.,6.,-12.,21.,6.
11.,-8.,15.,-22.,-3.
```

[41] http://mymathlib.webtrellis.net/matrices.html
[42] http://www.netlib.org/c/meschach/
[43] Bei komplexen Eigenwerten stehen die Imaginärteile in der Sub- und Superdiagonalen.

Sodann gibt das Programm die Matrix aus der Datei zur Kontrolle und die 2 bzw. 3 Eigenwertmengen aus, die von den 2 bzw. 3 Eigenwertalgorithmen ermittelt wurden.
Nun wird der Programmnutzer gefragt, mit welchen Eigenwerten er weiterrechnen möchte.
Daraufhin gibt das Programm Informationen zu den k_{ij} aus, und bricht bei Inkonsistenzen ab. „Inkonsistenzen“ bedeutet an dieser Stelle: Probleme bei der Rangbestimmung.
Das Programm gibt im Falle des Nicht-Abbruchs dann die Haupt- und die Superdiagonale der JNF der übergebenen Matrix aus.
Der Programmbenutzer kann davon ausgehen, dass die Hauptdiagonale einigermassen korrekt ist, aber der Superdiagonale sollte er nicht blind vertrauen. Tests zeigten, dass, wenn das Programm überhaupt zu Ende rechnet, dann die Superdiagonale meistens korrekt ist.

Literaturverzeichnis

Bosch, S.: Lineare Algebra, Springer 2006
Brehmer, S., Belkner, H.: Einführung in die analytische Geometrie und lineare Algebra, Dt. Verlag d. Wissenschaften, Berlin 1968.
Engeln-Müllges / Reutter: Numerik-Algorithmen mit ANSI C-Programmen. BI-Wissenschaftsverlag 1993
Faires / Burden: Numerische Methoden, Spektrum 1994
Kågström, B., Ruhe, A.: An Algorithm for Numerical Computation of the Jordan Normal Form of a Complex Matrix. in: ACM Transactions on Mathematical Software, Volume 6, Issue 3, S. 398 – 419
Press, W. et al.: Numerical Recipes in C, Cambridge University Press 1992 (online: http://www.nrbook.com/a/bookcpdf.php)
Quarteroni, Sacco, Saleri: Numerische Mathematik 1, Springer 2002
Zurmühl, R., Falk, S.: Matrizen und ihre Anwendungen, Springer 1984

Zur Implementierung in C leistet gute Dienste:
Wolf, Jürgen: C von A bis Z.Galileo Press 2006. online: www.pronix.de/pronix-4.html

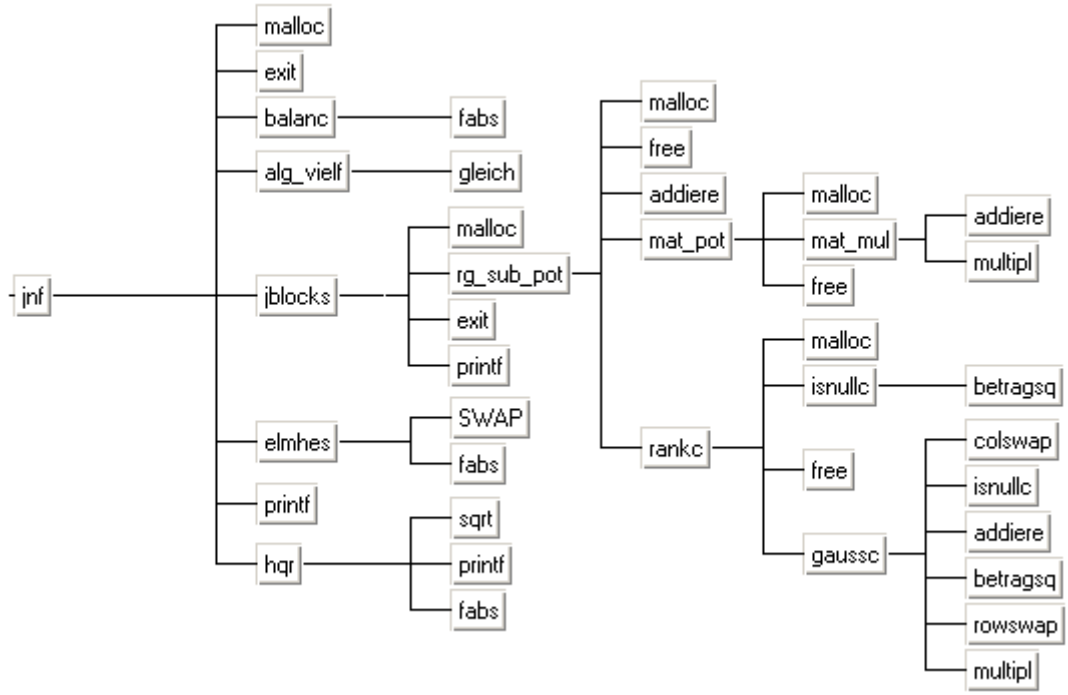
Anhang 1

Liste der selbst implementierten Funktionen

getlines	Ermittelt die Anzahl der Zeilen einer ASCII-Datei; dient zur Ermittlung der Dimension von A
datei2matrix	Stellt die Daten aus der externen Datei dem Programm zur Verfügung
balanc	optionale Hilfstransformation für elmhes
elmhes	bringt eine reelle Matrix auf obere Hessenbergform
hqr	führt eine QR-Zerlegung einer Matrix in oberer Hessenbergform durch
addiere	addiert 2 komplexe Zahlen
multipl	multipliziert 2 komplexe Zahlen
betragsq	ermittelt das Quadrat des Betrages einer komplexen Zahl
rowswap	tauscht 2 Zeilen einer komplexen Matrix
isnullc	prüft, ob der Betrag einer komplexen Zahl kleiner als ein gewisses Epsilon ist
gaussc	bringt eine komplexe Matrix in Dreiecks- bzw. Stufenform
rankc	ermittelt den Rang einer komplexen Matrix
mat_mul	multipliziert 2 komplexe Matrizen
mat_pot	potenziert eine komplexe Matrix
rg_sub_pot	ermittelt $rg(A-zE)^k$ für eine komplexe Matrix A
gleich	ermittelt, ob sich 2 komlexe Zahlen nur um ein gewisses Epsilon unterscheiden
alg_vielf	ermittelt, wie oft jede komplexe Zahl in einer gegebenen Menge vorkommt
jblocks	ermittelt, wie oft ein Jordanblock in der JNF vorkommt
jnf	ermittelt die JNF

Anhang 2

Darstellung der Abhängigkeit einzelner Funktionen voneinander



Anhang 3

Instabilität der Berechnung der JNF am Beispiel von Mathematica

Man betrachte die folgenden beiden Outputs des Computeralgebrasystems *Mathematica 5.2*.:

$$\left\{\begin{pmatrix} 25 & -16 & 30 & -44 & -12 \\ 13 & -7 & 18 & -26 & -6 \\ -18 & 12 & -21 & 36 & 12 \\ -9 & 6 & -12 & 21 & 6 \\ 11 & -8 & 15 & -22 & -3 \end{pmatrix}, \begin{pmatrix} 0 & -8 & 0 & -12 & 0 \\ 1 & -8 & 0 & -6 & 0 \\ 0 & 0 & 0 & 12 & 0 \\ -\frac{1}{2} & 0 & 1 & 6 & 0 \\ \frac{1}{2} & -4 & -3 & -6 & 1 \end{pmatrix}, \begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix}\right\}$$

und

$$\left\{\begin{pmatrix} 25. & -16. & 30. & -44. & -12. \\ 13. & -7. & 18. & -26. & -6. \\ -18. & 12. & -21. & 36. & 12. \\ -9. & 6. & -12. & 21. & 6. \\ 11. & -8. & 15. & -22. & -3. \end{pmatrix}, \begin{pmatrix} 0.640716 & -0.390333 & 0.149661 & 0.390333 & -0.640716 \\ 0.378605 & -0.662567 & 0.773619 & 0.662567 & -0.378605 \\ -0.524222 & -0.544467 & 0.553847 & 0.544467 & 0.524222 \\ -0.262111 & -0.272234 & 0.103406 & 0.272234 & 0.262111 \\ 0.320358 & -0.195166 & 0.248348 & 0.195166 & -0.320358 \end{pmatrix}, \begin{pmatrix} 3. & 0 & 0 & 0 & 0 \\ 0 & 3. & 0 & 0 & 0 \\ 0 & 0 & 3. & 0 & 0 \\ 0 & 0 & 0 & 3. & 0 \\ 0 & 0 & 0 & 0 & 3. \end{pmatrix}\right\}$$

Die erste Matrix zeigt jeweils die Eingabematrix, die zweite und dritte das Output von `JordanDecompostion[Eingabematrix]`, nämlich die Transformationsmatrix und die JNF.
Wie man sieht, unterscheiden sich die Eingabematrizen nur dadurch, dass bei der ersteren ein *Dezimalpunkt* gesetzt wurde, aber das Output unterscheidet sich erheblich; im zweiten Fall wurde die JNF falsch berechnet !

Zusätzlich verwirrend ist, dass die Probe (bei entsprechenden Bezeichnungen: `q.j.Inverse[q]`) die Resultate *beider* Rechnungen zu bestätigen scheint; so erhält man durch die Probe für den „Fall mit Dezimalpunkt“ das Resultat,

$$\begin{pmatrix} 25. & -16. & 30. & -44. & -12. \\ 13. & -7. & 18. & -26. & -6. \\ -18. & 12. & -21. & 36. & 12. \\ -9. & 6. & -12. & 21. & 6. \\ 11. & -8. & 15. & -22. & -3. \end{pmatrix}$$

was ja zu erwarten wäre.
Eine genauere Betrachtung zeigt aber, dass beispielsweise die „25.“ oben links in Wahrheit den Wert 25.000000059604645 hat. Auch die Dreien in der Hauptdiagonalen, die als Eigenwerte ermittelt wurden, sind keine exakten Werte sondern 3.000000302808814, 3.0000000286658794 etc. Offenbar sieht *Mathematica* diese Werte alle als verschieden an, so dass es zu der obigen JNF kommt.

Der Support von *Mathematica* sagt dazu[44]:

> In general (not just in Mathematica), computing the Jordan canonical form numerically is known to be an ill-conditioned problem, especially when the eigenvalues of the matrix are repeated. With floating point arithmetic, one might obtain inexact eigenvalues that are very slightly (numerically) different, even though they should be the same in theory. It isn't possible to automatically determine when the eigenvalues are truly distinct, and when the differences are an artifact of numerical error. Thus in the machine arithmetic the Jordan form will almost always appear to be diagonal.

Das freie Computeralgebrasystem *Maxima 5.12* zeigt das beschriebene Verhalten nicht, sondern berechnet – zumindest für das obige Beispiel – die JNF korrekt.
Das freie Computeralgebrasystem *XCAS 0.6* berechnet im „Fall ohne Dezimalpunkt“ die JNF für obiges Beispiel korrekt, aber produziert dafür im „Fall mit Dezimalpunkt“ ein Output, das – zumindest für mich – völlig unverständlich ist.

---

[44] private Korrespondenz

Anhang 4

Dokumentation zu den mitgelieferten Beispielen

Das Programm wird mit 15 Beispielmatrizen in Form der Dateien `Beispiel01.txt` bis `Beispiel15.txt` ausgeliefert.

Zunächst ein Überblick, wie das Programm damit verfährt:

Matrix	Dim.	wahre Eigenwerte	Kommentar[45]
01	4×4	in ℝ	rechnet korrekt
02	3×3	in ℝ	rechnet korrekt
03	3×3	in ℝ	rechnet nicht korrekt
04	4×4	in ℝ	rechnet korrekt
05	5×5	in ℝ	rechnet korrekt
06	4×4	in ℝ	rechnet korrekt
07	3×3	in ℝ	rechnet korrekt
08	5×5	in ℝ	rechnet nicht korrekt; gibt Warnmeldung aus
09	4×4	in ℝ	rechnet korrekt
10	3×3	in ℂ	rechnet korrekt
11	7×7	in ℂ	rechnet korrekt
12	7×7	in ℂ	rechnet korrekt
13	7×7	in ℂ	rechnet korrekt
14	8×8	in ℂ	rechnet nicht korrekt; gibt Warnmeldung aus
15	13×13	in ℂ	bricht nach Warnmeldung ab

Detailierter nun noch die wahren[46] JNFs:

Beispiel01

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Beispiel02

$$\begin{pmatrix} -3 & 1 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Beispiel03

$$\begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

Beispiel04

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

Beispiel05

$$\begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

Beispiel06

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Beispiel07

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

Beispiel08

$$\begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Beispiel09

$$\begin{pmatrix} -8 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix}$$

Beispiel10

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2-i & 0 \\ 0 & 0 & 2+i \end{pmatrix}$$

Beispiel11

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1-3i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1+3i \end{pmatrix}$$

Beispiel12

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1-3i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1+3i \end{pmatrix}$$

[45] Das Kriterium der „Korrektheit“ bezieht sich auf die JNF; die Eigenwerte werden von meinem Programm stets – innerhalb gewisser Grenzen – „korrekt“ berechnet.

[46] Eindeutig bis auf die Reihenfolge der Jordankästchen.

Beispiel13

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1-i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1+i \end{pmatrix}$$

Beispiel14

$$\begin{pmatrix} -100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1-i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1+i \end{pmatrix}$$

Beispiel15

$$\begin{pmatrix} -176 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -176 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -154 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 110 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 110 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 110 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 110 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 110 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 110 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -11i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2200-22i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2200+22i \end{pmatrix}$$