



Computational Physics

Exercise Sheet 07

RENE-MARCEL LEHNER

Summer Term
2023

E0: Matrix diagonalization - Power
method

3

E1: Householder with QR Iteration....

9

ASSIGNMENT N°

07

E0: Matrix diagonalization - Power method

Task:

Many physical systems can be described by matrices whose eigenvalues correspond to the eigenfrequencies of these systems. Therefore, matrix diagonalization is an important tool in physics, and the power method is a simple and significant technique.

Given the symmetric 4×4 -matrix:

$$A = \begin{pmatrix} 1 & -2 & -3 & 4 \\ -2 & 2 & -1 & 7 \\ -3 & -1 & 3 & 6 \\ 4 & 7 & 6 & 4 \end{pmatrix}$$

- Determine the eigenvalues of the matrix using `numpy.linalg.eig` or the C++ library Eigen (<http://www.eigen.tuxfamily.org>).
- Determine the eigenvalues a second time by implementing the power method. Choose a suitable initial vector v_0 and an appropriate number of iterations. Compare your results with the ones from the first part of the task.

a) Ausführung der Methode `np.linalg.eig(A)` liefert

```
1 Numpy eigenvalues:
2 (array( [-9.27793969, 12.08505811,  4.59704881,  2.59583277]),
3
4 array([ [-0.45657804, -0.07467658,  0.64264911,  0.61070616],
5         [-0.49016739, -0.47399743,  0.24995413, -0.6874484 ],
6         [-0.44274934, -0.42870916, -0.70253377,  0.35584796],
7         [ 0.5960247 , -0.76547913,  0.17598679,  0.166808  ]]))
8
```

Die Eigenwerte der Matrix sind also:

$$\lambda_1 \approx -9.27793969$$

$$\lambda_2 \approx 12.08505811$$

$$\lambda_3 \approx 4.59704881$$

$$\lambda_4 \approx 2.59583277$$

und die zugehörigen Eigenvektoren sind:

$$v_1 \approx \begin{pmatrix} -0.45657804 \\ -0.49016739 \\ -0.44274934 \\ 0.5960247 \end{pmatrix}, v_2 \approx \begin{pmatrix} -0.07467658 \\ -0.47399743 \\ -0.42870916 \\ -0.76547913 \end{pmatrix}, v_3 \approx \begin{pmatrix} 0.64264911 \\ 0.24995413 \\ -0.70253377 \\ 0.17598679 \end{pmatrix}, v_4 \approx \begin{pmatrix} 0.61070616 \\ -0.6874484 \\ 0.35584796 \\ 0.166808 \end{pmatrix}$$

Man beachte, dass die **Spalten** der ausgegebenen Matrix (das 2. Array) die entsprechenden Eigenvektoren sind.

- b) Die Potenzmethode liefert unter bestimmten Bedingungen (Abschnitt Theorie TODO) den Eigenvektor zum **betragsmäßig größten Eigenwert**. Um weitere Eigenwerte zu erhalten, wird die Potenzmethode um das sogenannte **"Deflating"** erweitert.

Es wird der Einfluss des Eigenvektors (zum errechneten Eigenwert) von der ursprünglichen Matrix entfernt, sodass λ_2 der betragsmäßig größte Eigenwert wird. Die Potenzmethode liefert dann den Eigenvektor zu λ_2 . Auf diese Weise können alle Eigenwerte und -vektoren bestimmt werden.

Die gesamte Programmausführung wird also zusammengefasst zu

```
1 eigenvector_power = []
2 eigenvalue_power = []
3
4 for i in range(A.shape[0]):
5     eigenvector_power.append(power_iteration(A, 10000))
6     eigenvalue_power.append(eigenvalue(A, eigenvector_power[i]))
7
8     A = deflating(A, eigenvector_power[i], eigenvalue_power[i])
9
```

wobei A als quadratisch vorausgesetzt wird.

Die berechneten Eigenwerte und -vektoren sind nach

```
1 Power method eigenvalues: [12.085058112675243, -9.27793969008451,
2 4.597048810634316, 2.5958327667749517]
3 Power method eigenvectors:
4 [array([0.07467658, 0.47399743, 0.42870916, 0.76547913]),
5 array([ 0.45657804,  0.49016739,  0.44274934, -0.5960247 ]),
6 array([-0.64264911, -0.24995413,  0.70253377, -0.17598679]),
7 array([ 0.61070616, -0.6874484 ,  0.35584796,  0.166808  ])]
```

Eigenwerte

$$\lambda_1 \approx -9.27793969008451$$

$$\lambda_2 \approx 12.085058112675243$$

$$\lambda_3 \approx 4.597048810634317$$

$$\lambda_4 \approx 2.595832766774952$$

Eigenvektoren

$$v_1 \approx \begin{pmatrix} 0.45657804 \\ 0.49016739 \\ 0.44274934 \\ -0.5960247 \end{pmatrix} v_2 \approx \begin{pmatrix} 0.07467658 \\ 0.47399743 \\ 0.42870916 \\ 0.76547913 \end{pmatrix} v_3 \approx \begin{pmatrix} -0.64264911 \\ -0.24995413 \\ 0.70253377 \\ -0.17598679 \end{pmatrix} v_4 \approx \begin{pmatrix} 0.61070616 \\ -0.6874484 \\ 0.35584796 \\ 0.166808 \end{pmatrix}$$

Die Eigenwerte stimmen mit denen von **Numpy exakt** überein, genauso wie die Eigenvektoren (bis auf ein Vielfaches von -1). Man beachte, dass die Eigenwerte **genauer** als die Werte von **Numpy** bestimmt sind und die **float**-Präzision vollständig ausschöpfen.

Die Aufgabe könnte hier enden.

Da jedoch einige Fragen offen bleiben und eine Untersuchung des Konvergenzverhaltens aussteht, wird die Aufgabe etwas erweitert.

1. Was ist das charakteristische Polynom und gibt es eine analytische Lösung für die Eigenwerte? Falls nicht, kann eine genauere Lösung berechnet werden, die als Vergleich mit den Werten der Potenzmethode dient?
 2. Wie gut konvergiert die Potenzmethode gegen die Eigenwerte? Wie viele Iterationen sind in etwa nötig, um einen genauen Eigenwert zu erhalten?
 3. Wie hängt die Größe der Matrix mit der Konvergenzgeschwindigkeit und -genauigkeit ab?
-

Theorie

Charakteristisches Polynom

Das Charakteristische Polynom lautet

$$\lambda^4 - 10\lambda^3 - 80\lambda^2 + 773\lambda - 1338 = 0 \quad .$$

In der Tat **gibt** es dazu eine analytische Lösung. Der **Satz von Abel-Ruffini** besagt, dass *keine geschlossene Formel für Gleichungen 5. Grades oder höher* gefunden werden kann. Im Umkehrschluss bedeutet dies, dass es für alle Gleichungen des Grades 4 oder kleiner geschlossene Formeln geben muss.

Eine geschlossene Formel für die vorliegende Quartische Gleichung wurde bereits vor vielen Jahrhunderten (*Lodovico Ferrari (1522–1565)*) gefunden. Da dort viele Wurzelterme auftreten, sind die Ergebnisse mit hoher Wahrscheinlichkeit **irrationale Zahlen**, weswegen auch die Ergebnisse der geschlossenen Formel die Maschinenpräzision vollständig ausschöpfen und nicht von den Ergebnissen einer geeigneten numerischen Methode abweichen würden.

Daher entspricht unser numerisches Ergebnis, das die maximale Präzision ausschöpft und vollständig konvergiert ist, der **Referenzlösung**, die für den relativen Fehler verwendet wird.

Potenzmethode

1. Wähle einen Startvektor v_0 .
2. Definiere eine Iteration durch $w_n = A \cdot v_{n-1}$ und Normierung $v_n = \frac{w_n}{|w_n|}$.

Bedingungen: Die Eigenwerte sind **nicht** entartet, A ist diagonalisierbar und v_0 ist **nicht orthogonal** zu dem Eigenvektor.

Es gilt auch, dass die Konvergenzgeschwindigkeit von dem **Verhältnis der zwei größten Eigenwerte** abhängt und **linear** ist:

$$\frac{w_1}{|w_1|} = \frac{A^n v_0}{|A^n v_0|} = x_1 + \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|\right)$$

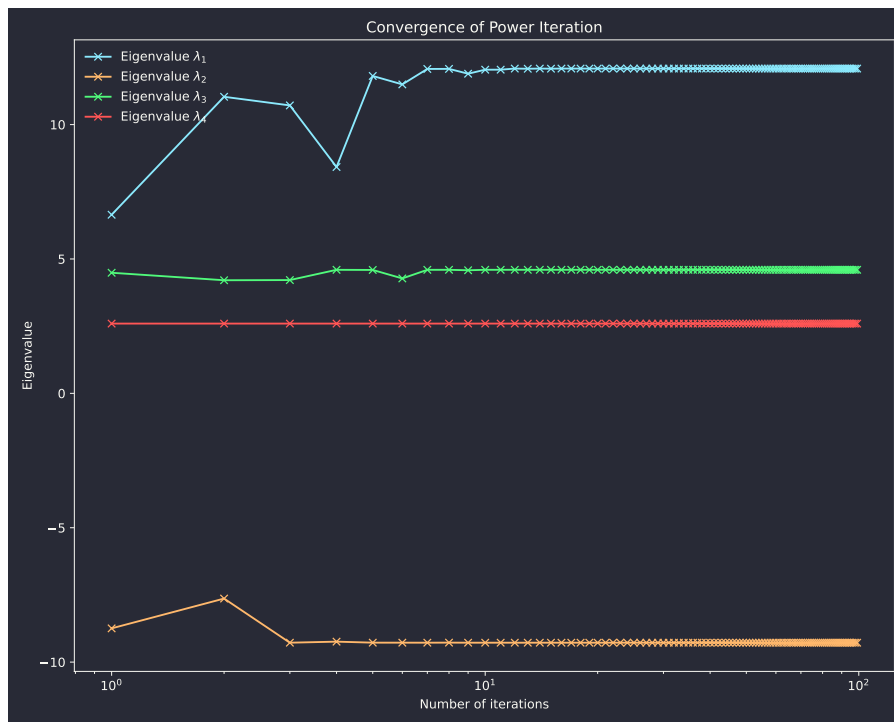


Figure 1.1: Approximierte Eigenwerte in Abhängigkeit der Iterationsschritte. Die Potenzmethode konvergiert schon für wenige Iterationsschritte.

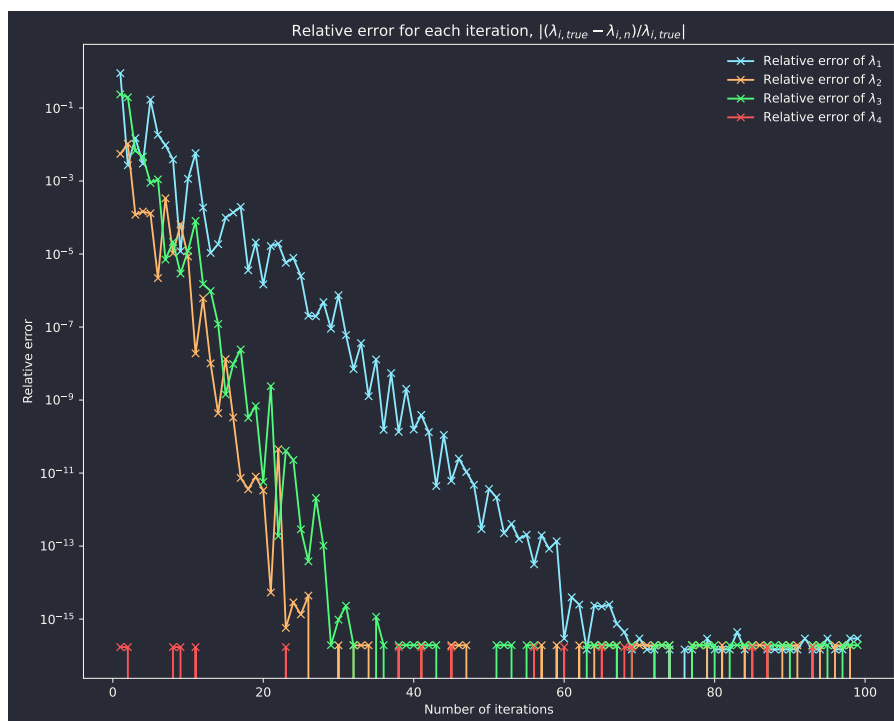


Figure 1.2: Relative Fehler in Abhängigkeit der Iterationsschritte. Die höchstmögliche Genauigkeit bzgl. der Maschinenpräzision wird in unter 100 Iterationsschritten erreicht.

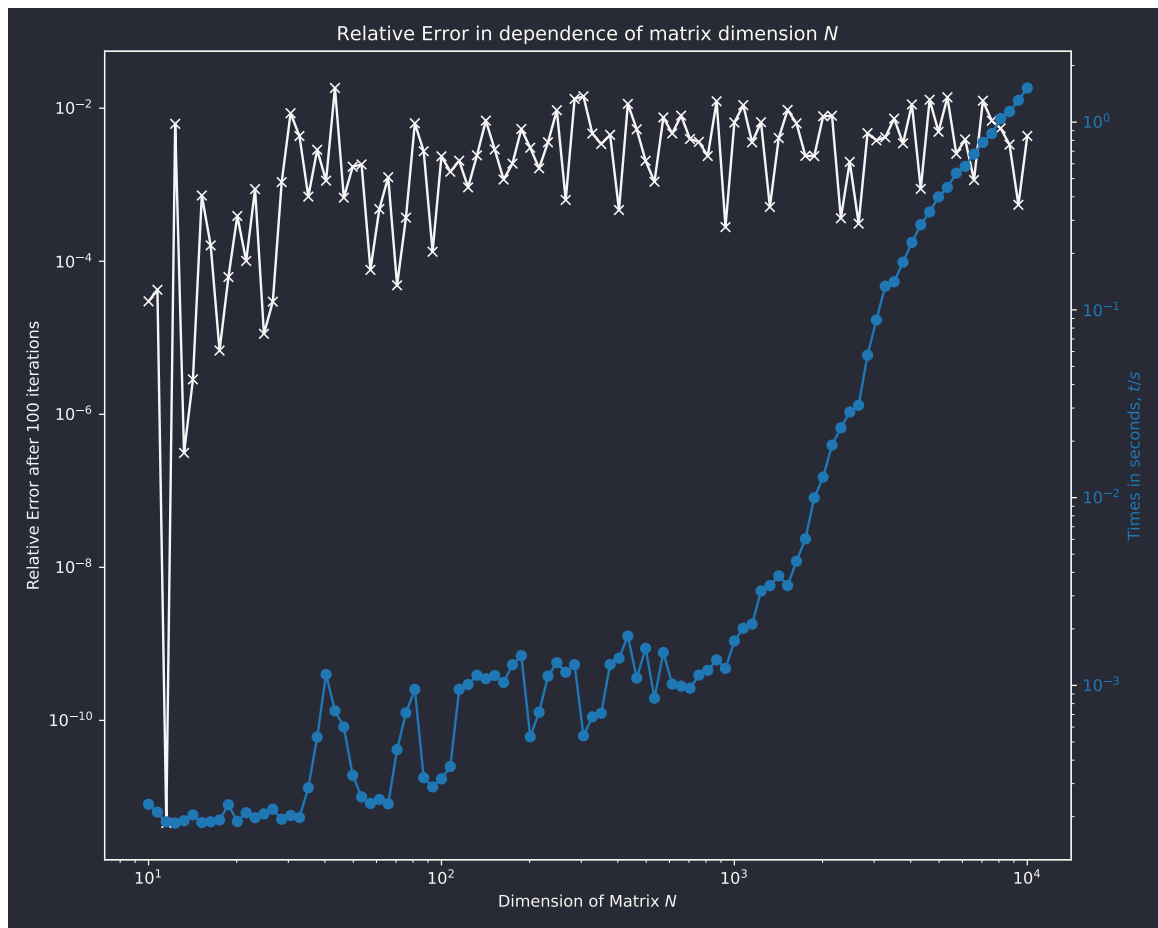


Figure 1.3: 100 Iterationsschritte für alle Matrixdimensionen. Der relative Fehler pendelt sich sehr schnell bei etwa 10^{-2} ein, unabhängig von der Größe der Matrix. Die Präzision aus Abbildung 1.2 wird bis auf einen Ausreißer nicht erreicht. Trotz der nicht steigenden Genauigkeit wird aber ein **exponentieller** Anstieg in der Laufzeit beobachtet, was ohne weitere Untersuchung auf die quadratisch wachsende Natur der Werte ($N \times N$) zurückgeführt wird.

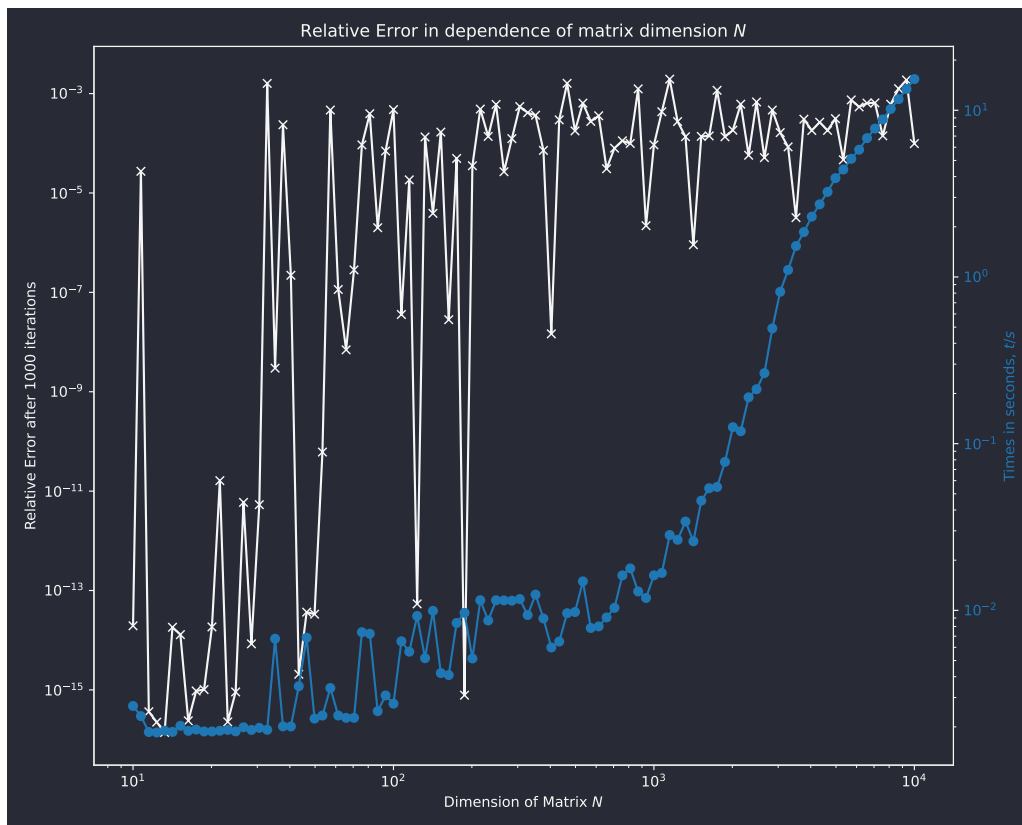


Figure 1.4: 1000 Iterationsschritte für alle Matrixdimensionen. Der relative Fehler pendelt sich wieder sehr schnell, dieses mal aber bei etwa 10^{-3} ein, unabhängig von der Größe der Matrix. Die Präzision aus Abbildung 1.2 wird (bis auf ein paar Ausreißer) noch bis 50×50 -Matrizen erreicht. Auch hier wird ein exponentieller Anstieg der Laufzeit beobachtet. Unter grober Beobachtung wird auch festgestellt, dass die Laufzeit linear mit den Iterationsschritten ansteigt, was zu erwarten ist, da der Rechenaufwand pro Iteration unverändert bleibt.

Fazit

Die Potenzmethode konvergiert schnell und recht zuverlässig gegen die gesuchten Eigenwerte und -vektoren.

Auf den ersten Blick kann der relative Fehler einer hochdimensionalen Matrix reziprok mit einer Vergrößerung der Iterationsschritte und damit einer Verlängerung der Laufzeit abgefangen werden (eine Zehnfachsteigerung der Iterationsschritte, und somit auch der Laufzeit, resultiert in einer Zehntelreduzierung des relativen Fehlers.).

Es müsste aber weiter untersucht werden, unter welchen Bedingungen und wie weitreichend diese Aussage gültig ist.

Es muss vor allem beachtet werden, dass die Konvergenzgeschwindigkeit des relativen Fehlers auch sehr stark von dem Verhältnis der beiden betragsmäßig größten Eigenwerte abhängig ist. Würde dieser Faktor mit einberechnet werden, könnte man eine einheitlichere Aussage über die Konvergenz- und Laufzeitabhängigkeiten treffen.



E1: Householder with QR Iteration

Task:

In the following, you are asked to become familiar with the Householder algorithm and the QR iteration. Using this method, it is possible to efficiently diagonalize even large matrices.

Given the symmetric $N \times N$ matrix:

$$A_{k,l} = k + l + k\delta_{k,l} \quad 0 \leq k, l < N.$$

- (a) Use Householder transformations to transform matrix A into tridiagonal form.
- (b) Determine the eigenvalues of A using QR iteration.
- (c) Diagonalize matrix A also using the power method that you implemented in Task 1. Compare the results and runtimes.

Do this for $N = 10$, $N = 100$ and $N = 1000$.