

Duale Hochschule Sachsen  
Staatliche Studienakademie Leipzig

# Hausarbeit zur Singulärwertzerlegung

im Rahmen des Studiengangs Bachelor of Science  
in der Studienrichtung Informatik  
im Modul Algorithmen und Datenstrukturen (5CS-TI2AD-30)

Eingereicht von: **Malte Luca Peukert**  
**Beethovenstraße 27, 06749 Bitterfeld-Wolfen**  
**Seminargruppe: CS24-1**  
**Matrikelnummer: 5002722**

Leipzig, 28. Dezember 2025

# Inhaltsverzeichnis

<b>1 Einleitung</b>	I
<b>2 Problemstellung</b>	I
<b>3 Mathematische Grundlagen der Singulärwertzerlegung</b>	II
3.1 Grundlagen der Matrixdarstellung von Bildern . . . . .	II
3.2 Definition der Singulärwertzerlegung . . . . .	II
3.3 Rang-k-Approximation . . . . .	III
3.4 Speicherersparnis durch Rang-k-Approximation . . . . .	III
<b>4 Programmbeschreibung und Auswertung</b>	III
4.1 Lösungsansatz und Wahl der Programmiersprache . . . . .	IV
4.2 Programmstruktur und Ablauf . . . . .	IV
4.3 Einlesen der Bildmatrix . . . . .	V
4.4 Berechnung der Singulärwertzerlegung . . . . .	V
4.5 Rekonstruktion der Rang-k-Approximation . . . . .	VI
4.6 Speicherung der Rekonstruktionen . . . . .	VI
4.7 Fehlerberechnung (Frobenius-Norm) . . . . .	VII
4.8 Energieerhaltung der Singulärwerte . . . . .	VII
4.9 Berechnung der Speicherersparnis . . . . .	VIII
4.10 Darstellung als Graustufenbild . . . . .	IX
4.11 Zentrale Verarbeitung - <code>process_file()</code> . . . . .	IX
4.12 Ergebnisse (CSV) . . . . .	X
4.13 Komandzeileninterface (CLI) . . . . .	X
<b>5 Auswertung der Ergebnisse</b>	X
5.1 Datenbasis . . . . .	XI
5.2 Detaillierte numerische Befunde und Interpretation . . . . .	XI
5.2.1 Frobenius-Fehler . . . . .	XI
5.2.2 Energieerhalt $E(k)$ . . . . .	XII
5.2.3 Speicherbedarf und relative Einsparung . . . . .	XII
5.3 Bestimmung des optimalen k-Werts . . . . .	XIV
5.4 Numerische Genauigkeit und Stabilität . . . . .	XV
<b>Literatur</b>	XV
<b>Anhang</b>	XVI
Selbstständigkeitserklärung . . . . .	XVI

# 1 Einleitung

In unserer digitalisierten Welt sind Bilder allgegenwärtig. Durch die Weiterentwicklung von Mobiltelefonen, hat so gut wie jeder eine hochauflösende Kamera in der Hosentasche. Mit dieser steigenden Auflösung, wächst der Bedarf an effizienter Speicherung und Übertragung. Dafür ist es notwendig wirksam und zuverlässig Bilddaten zu komprimieren. Klassische Verfahren wie zum Beispiel JPEG oder PNG nutzen mathematische Transformation, um redundante Informationen zu entfernen und damit den Speicherbedarf signifikant zu reduzieren. Eine besonderst effektives und effizientes mathematisches Verfahren ist dafür die Singulärwertzerlegung (Singular Value Decomposition, SVD). Sie hat große Bedeutung für die Bildverarbeitung, aber auch weitreichende Anwendungen in der Signalverarbeitung, Datenanalyse, Statistik und maschinellem Lernen[1].

Die SVD ist ein zentrales Konzept der linearen Algebra[2]. Besonders relevant für Bildkompression ist ihre Eigenschaft, Matrizen optimal durch niedrigere Ränge approximieren zu können. Diese Tatsache, die im sogenannten Eckart-Young-Theorem formalisiert ist und bis heute als Grundlage zahlreicher Kompressionsverfahren dient.

Diese Hausarbeit untersucht die Anwendung der SVD zur Speicherung eines vereinfachten Bildes und zeigt wie, mithilfe der Rang-k-Approximation Speicherplatz reduziert werden kann, ohne die wesentlichen Bildinformationen zu verlieren

## 2 Problemstellung

Hochauflösende digitale Bilder bestehen aus einer großen Anzahl von Pixeln, die üblicherweise als eine zweidimensionale Matrix gespeichert werden. Dementsprechend wächst der Speicherbedarf eines unkomprimierten Bildes proportional zur Anzahl der Pixel. Bei einer Größe von 15x25 enthält die Matrix bereits 375 einzelne Werte, während ein Bild im HD-Format bereits über zwei Millionen Pixel umfasst.

Die zugrunde liegende Aufgabe dieser Arbeit besteht darin,

1. die Matrixdarstellung des Buchstabens „F“ aus einer Textdatei einzulesen,
2. die vollständige SVD dieser Matrix berechnen,
3. sukzessive Rang-k-Approximation zu erzeugen und analysieren,
4. die Veränderung der Bildqualität bei zunehmendem k zu untersuchen,

5. die erzielten Speicherersparnisse gegenüber der Originalmatrix zu bestimmen.

Das zu Grunde liegende Ziel ist es zu verstehen, wie stark ein Bild komprimiert werden kann, ohne an Erkennbarkeit oder Informationswert zu verlieren.

## 3 Mathematische Grundlagen der Singulärwertzerlegung

### 3.1 Grundlagen der Matrixdarstellung von Bildern

Ein Graustufenbild lässt sich mathematisch als Matrix

$$A \in \mathbb{R}^{m \times n} \quad (3.1)$$

darstellen. Jeder Eintrag  $a_{ij}$  entspricht dabei den Helligkeitswert eines Pixels dar. Binäre Bilder, wie das in dieser Hausarbeit benutzte „F“, enthalten nur die Werte 0 und 1. Sie eignen sich trotzdem hervorragend zu Untersuchung von Strukturzerlegung, da sie klare Kanten und Zusammenhänge besitzen.

### 3.2 Definition der Singulärwertzerlegung

Die SVD zerlegt eine Matrix  $A$  in drei Komponenten:

$$A = U\Sigma V^T \quad (3.2)$$

Dabei gelten folgende Eigenschaften:

- $U \in \mathbb{R}^{m \times m}$  enthält die orthogonalen Basisvektoren der Zeilenstruktur.
- $\Sigma \in \mathbb{R}^{m \times n}$  enthält die Singulärwerte  $\sigma_1 \geq \sigma_2 \geq \dots$
- $V \in \mathbb{R}^{n \times n}$  enthält die orthogonalen Basisvektoren der Spaltenstruktur

Jeder Singulärwert  $\sigma_i$  in  $\Sigma$  gibt die Bedeutung des zugehörigen Basisvektors an. Größere Werte deuten auf wichtigere Strukturen hin, während kleinere Werte oft Rauschen oder weniger relevante Details repräsentieren.

### 3.3 Rang-k-Approximation

Die entscheidende Eigenschaft der SVD für die Bildkompression ist die Möglichkeit, eine Matrix  $A$  durch eine Rang-k-Approximation  $A_k$  zu nähern:

$$A_k = U_k \Sigma_k V_k^T \quad (3.3)$$

Hierbei werden nur die ersten  $k$  Singulärwerte und die zugehörigen Spalten von  $U$  und  $V$  verwendet. Diese Approximation minimiert den Frobenius-Norm-Fehler zwischen  $A$  und  $A_k$ , was bedeutet, dass  $A_k$  die beste mögliche Annäherung an  $A$  mit Rang  $k$  ist.

### 3.4 Speicherersparnis durch Rang-k-Approximation

Originalmatrix:

$$\text{Speicherbedarf von } O = m \times n \quad (3.4)$$

Die Speicherersparnis ergibt sich aus der Reduktion der benötigten Elemente zur Speicherung der Matrix. Anstatt die gesamte  $m \times n$  Matrix  $A$  zu speichern, müssen nur noch die ersten  $k$  Spalten von  $U$ , die ersten  $k$  Zeilen und Spalten von  $\Sigma$  sowie die ersten  $k$  Spalten von  $V$  gespeichert werden. Die Gesamtanzahl der zu speichernden Elemente reduziert sich somit auf:

$$S(k) = k(m + n + 1) \quad (3.5)$$

Dies führt zu einer erheblichen Reduktion des Speicherbedarfs, insbesondere wenn  $k$  deutlich kleiner ist als sowohl  $m$  als auch  $n$ .

Relative Speicherersparnis:

$$\text{Relative Speicherersparnis } E(k) = 1 - \frac{S(k)}{O} = 1 - \frac{k(m + n + 1)}{m \times n} \quad (3.6)$$

## 4 Programmbeschreibung und Auswertung

In diesem Kapitel wird das entwickelte Python-Programm `svd_image_compression.py` detailliert beschrieben. Das Programm dient der praktischen Umsetzung der Singulärwertzerlegung zur Bildkompression und erfüllt alle Anforderungen der Aufgabenstellung. Es ermöglicht das Einlesen einer Bildmatrix, die Berechnung der reduzierten

SVD, die Rekonstruktion für verschiedene Rangwerte, die grafische Darstellung der Ergebnisse, die Berechnung von Fehlermaßen, der Energieerhaltung sowie der Speicherersparnis. Zusätzlich wird eine übersichtliche CSV-Datei zur Dokumentation der Ergebnisse erzeugt.

## 4.1 Lösungsansatz und Wahl der Programmiersprache

Zur praktischen Umsetzung wird Python in Verbindung mit der Bibliothek NumPy und Matplotlib verwendet. NumPy stellt eine hocheffiziente Implementierung der SVD bereit, die intern auf der LAPACK-Bibliothek basiert und Matplotlib ermöglicht die anschauliche Visualisierung der Ergebnisse.

Python bietet darüber hinaus:

- hohe Lesbarkeit und einfache Syntax,
- einfache Datenverarbeitung,
- Plattformunabhängigkeit,
- schnelle Umsetzung von mathematischen Algorithmen.

## 4.2 Programmstruktur und Ablauf

Das Programm ist klar modular aufgebaut und besteht aus drei Hauptbereichen:

1. **Hilfsfunktionen** (Matrix-Einlesen, SVD, Rekonstruktion, Speicherersparnis, Fehlerberechnung).
2. **Verarbeitungsfunktion** `process_file` als zentrale Pipeline
3. **Komandzeileninterface (CLI)** mit `argparse`

Der grundsätzliche Ablauf des Programms ist wie folgt:

1. Einlesen der Bildmatrix aus `SVD_F.txt`.
2. Berechnung der reduzierten Singulärwertzerlegung.
3. Rekonstruktion der Bildmatrix für verschiedene Rangwerte  $k$ .
4. Speicherung der rekonstruierten Matrizen als PNG-Dateien.
5. Optionale Speicherung als Textdateien.

6. Berechnung und Ausgabe von Fehlermaßen, Energieerhaltung und Speicherersparnis.
7. Erzeugung einer CSV-Datei mit den Ergebnissen.

## 4.3 Einlesen der Bildmatrix

Das Einlesen der Bilddaten erfolgt über die Funktion:

```
def load_matrix(path: str):
```

Die Matrix wird aus einer Textdatei eingelesen:

```
A = np.loadtxt(path)
```

Die Datei SVD\_F.txt enthält die binäre Matrix des Buchstabens „F“. Jeder Eintrag entspricht einem Pixelwert (0 oder 1). Nach dem Einlesen wird die Matrix in ein NumPy-Array vom Typ `float` umgewandelt, das für die weitere Verarbeitung verwendet wird.

## 4.4 Berechnung der Singulärwertzerlegung

Die Funktion:

```
def compute_svd(A: np.ndarray):
```

berechnet die reduzierte SVD der Matrix  $A$  mittels NumPy:

```
U, S, Vt = np.linalg.svd(A, full_matrices=False)
```

Die Zerlegung erfolgt gemäß der Definition:

$$A = U\Sigma V^T \quad (4.1)$$

Dabei enthalten:

- $U$ : linke Singulärvektoren,
- $S$ : Singulärwerte (als 1D-Array),
- $V^T$ : transponierte rechte Singulärvektoren.

Die reduzierte Form (`full_matrices=False`) sorgt dafür, dass nur die notwendigen Komponenten berechnet werden, was Speicher spart.

## 4.5 Rekonstruktion der Rang-k-Approximation

Die Rang-k-Approximation wird durch die Funktion:

```
def reconstruct_rank_k(U: np.ndarray, S: np.ndarray, Vt: np.ndarray, k: int)
```

berechnet. Die mathematische Umsetzung lautet:

$$A_k = U_k \Sigma_k V_k^T \quad (4.2)$$

Dabei werden nur die ersten  $k$  Spalten von  $U$ , die ersten  $k$  Singulärwerte und die ersten  $k$  Zeilen von  $V^T$  verwendet. Die Implementierung erfolgt direkt als Matrixprodukt:

```
return U[:, :k] @ np.diag(S[:k]) @ Vt[:k, :]
```

Durch die vektorisierte Form ist die Rekonstruktion sowohl numerisch stabil als auch effizient. Fals ein ungültiger Wert für  $k$  übergeben wird, wird eine `ValueError` ausgelöst:

```
if k <= 0:  
    raise ValueError("k muss >= 1 sein")
```

## 4.6 Speicherung der Rekonstruktionen

Die rekonstruierten Matrizen werden als PNG-Bilder gespeichert. Dies erfolgt in der Funktion:

```
def save_reconstruction(Ak: np.ndarray, filename: str)
```

als Textdateien gespeichert:

```
np.savetxt(filename, Ak, fmt=".3f")
```

Die Speicherung mit drei Nachkommastellen ermöglicht eine kompakte Dokumentation der Näherungsmatrix, mit Werten wie 0.000, 0.333, 0.667 und 1.000, in einer Textdatei entsprechend des jeweiligen  $k$ , wie z.B. `reconstruction_k_1.txt`.

Zusätzlich werden alle Rekonstruktionen als Graustufenbilder gespeichert:

```
plot_matrix(Ak, title=f'Rang-{k}', savepath=png_name)
```

Damit entsteht eine visuelle Darstellung der Qualität der Approximation für jedes  $k$ .

## 4.7 Fehlerberechnung (Frobenius-Norm)

Die Rekonstruktionsqualität wird durch die Frobenius-Norm des Fehlers mit der Funktion gemessen:

```
def reconstruction_error(A: np.ndarray, Ak: np.ndarray)
```

Mathematisch wird der Fehler wie folgt definiert:

$$\|A - A_k\|_F \quad (4.3)$$

und wird implementiert mit:

```
return float(np.linalg.norm(A - Ak, ord='fro'))
```

Diese Norm misst die quadratische Abweichung aller Matrixeinträge und ist ein gängiges Maß für die Rekonstruktionsqualität.

## 4.8 Energieerhaltung der Singulärwerte

Der Anteil der durch die ersten  $k$  Singulärwerte erklärten Energie wird durch:

```
def energy_retained(S: np.ndarray, k: int)
```

berechnet. Die mathematische Definition lautet:

$$E(k) = \frac{\sum_{i=1}^k \sigma_i^2}{\sum_{i=1}^r \sigma_i^2} \quad (4.4)$$

Diese Kennzahl gibt an, wie viel der Gesamtinformation der Originalmatrix durch die Rang-k-Approximation erhalten bleibt. Die Implementierung erfolgt durch:

```
k = min(k, len(S))
total = np.sum(S ** 2)
if total == 0:
    return 0.0
return np.sum(S[:k] ** 2) / total
```

## 4.9 Berechnung der Speicherersparnis

Die Speicherersparnis wird durch die Funktion:

```
def storage_savings(m: int, n: int, k: int)
```

berechnet.

Originalmatrix:

$$O = m \times n \quad (4.5)$$

Speicherbedarf der Rang-k-Approximation:

$$S(k) = k(m + n + 1) \quad (4.6)$$

Relative Speicherersparnis:

$$E(k) = 1 - \frac{S(k)}{O} = 1 - \frac{k(m + n + 1)}{m \times n} \quad (4.7)$$

Die Implementierung erfolgt durch:

```
original = m * n
svd = k * (m + n + 1)
savings = 1 - svd / original
```

## 4.10 Darstellung als Graustufenbild

Letztlich werden die rekonstruierten Matrizen als Graustufenbilder visualisiert. Dies geschieht in der Funktion:

```
def plot_matrix(A: np.ndarray, title: str, savepath: str)
```

## 4.11 Zentrale Verarbeitung - process\_file()

Die Funktion:

```
def process_file(  
    input_path: str,  
    output_dir: str,  
    k_values: List[int],  
    save_text_recons: bool = True,  
    show_plots: bool = False  
):
```

dient als zentrale Pipeline für die gesamte Verarbeitung:

1. Erzeugung des Ausgabeordners.
2. Einlesen der Matrix.
3. Berechnung der SVD.
4. Iterative Rekonstruktion für jedes  $k$ .
5. Speicherung der PNG-Grafiken.
6. (Optionale) Speicherung der Textdateien.
7. Berechnung und Ausgabe der Fehlermaße, Energieerhaltung und Speicherersparnis.
8. Erstellung einer CSV-Datei mit den Ergebnissen.
9. (Optionale) Anzeige der Grafiken.

Für jeden  $k$ -Wert wird ein vollständiger Datensatz erzeugt.

## 4.12 Ergebnisse (CSV)

Die Datei `svd_results.csv` wird im Ausgabeordner gespeichert und enthält eine tabellarische Übersicht aller berechneten Werte für jedes  $k$ . Sie enthält folgende Spalten:

Spalte	Beschreibung
<code>k</code>	Rang der Approximation
<code>frobenius_error</code>	Frobenius-Norm des Rekonstruktionsfehlers
<code>energy_retained</code>	Anteil der durch die ersten $k$ Singulärwerte erklärten Energie
<code>original_values</code>	Speicherbedarf der Originalmatrix
<code>svd_values</code>	Speicherbedarf der Rang- $k$ -Approximation
<code>savings_relative</code>	Relative Speicherersparnis

Die Datei ermöglicht eine objektive Bewertung der Kompressionsleistung für verschiedene Rangwerte.

## 4.13 Komandzeileninterface (CLI)

Das Programm wird über ein einfaches CLI gesteuert, das mit der Bibliothek `argparse` implementiert ist. Es ermöglicht die Angabe folgender Parameter:

- `-input`: Pfad zur Eingabedatei (Standard: `SVD_F.txt`).
- `-out`: Verzeichnis für die Ausgabe (Standard: `svd_results`).
- `-ks`: Liste der  $k$ -Werte für die Rekonstruktion (Standard: 1,2,3,5,8,10).
- `-no-text`: Deaktiviert die Speicherung der Textdateien.
- `-show`: Aktiviert die Anzeige der Grafiken nach der Verarbeitung.

# 5 Auswertung der Ergebnisse

In diesem Kapitel werden die numerischen Ergebnisse der durchgeführten SVD-Experimente systematisch ausgewertet und interpretiert. Grundlage ist die Datei `svd_results.csv`, die für verschiedene Rangwerte  $k$  die folgenden Kennzahlen enthält: Frobenius-Fehler, Energieerhalt, Original- und SVD-Speicherbedarf sowie die relative Speicherersparnis.

Ziel der Auswertung ist es, (i) die numerischen Messwerte mathematisch einzuordnen, (ii) den besten Kompromiss zwischen Speicherersparnis und Rekonstruktionsqualität zu identifizieren und (iii) Empfehlungen für die Interpretation und Anwendung der SVD-Kompression abzuleiten.

## 5.1 Datenbasis

k	Frobenius-Fehler	Energieerhalt	Original	SVD	Relative Einsparung
1	4.1923	79.08%	375	41	89.07%
2	1.8278	96.02%	375	82	78.13%
3	$4 \times 10^{-15}$	100.00%	375	123	67.20%
5	$4 \times 10^{-15}$	100.00%	375	205	45.33%
8	$4 \times 10^{-15}$	100.00%	375	328	12.53%
10	$4 \times 10^{-15}$	100.00%	375	410	-9.33%

## 5.2 Detaillierte numerische Befunde und Interpretation

### 5.2.1 Frobenius-Fehler

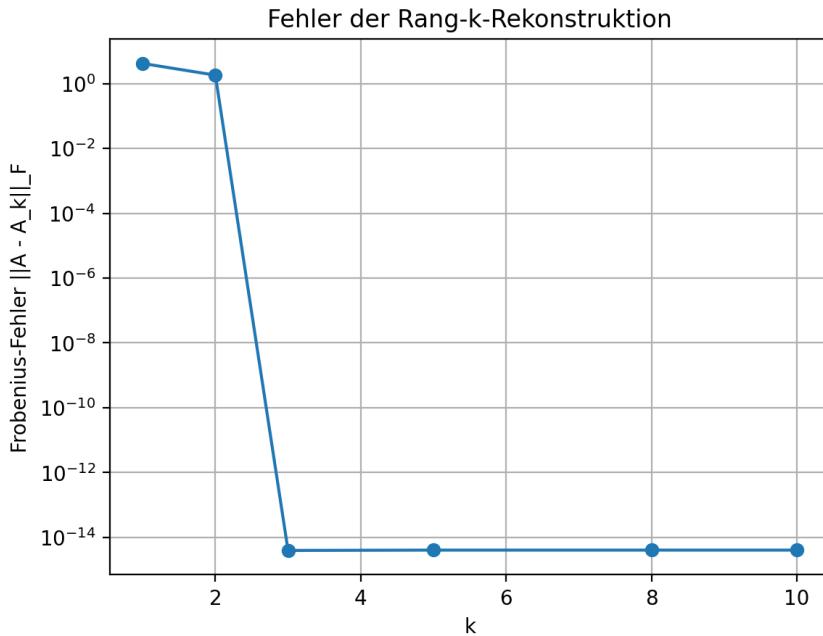
Aus den Messdaten:

$$k = 1 : \|A - A_1\|_F \approx 4.1923$$

$$k = 2 : \|A - A_2\|_F \approx 1.8278$$

$$k \geq 3 : \|A - A_3\|_F = 4 \times 10^{-15} \text{ (numerisch 0)}$$

**Interpretation:** Die deutliche Fehlerreduktion von  $k = 1$  zu  $k = 2$  zeigt, dass die zweite Singulärkomponente einen erheblichen Teil der Bildinformationen wiedergibt. Ab  $k = 3$  ist der Fehler praktisch null, was auf eine perfekte Rekonstruktion hinweist. Dies deutet darauf hin, dass die Matrix von SVD\_F.txt einen Rang von 3 besitzt. Anderer gesagt bedeutet das, dass die Originalmatrix exakt als Linearkombination von drei äußeren Produkten  $\sigma_i u_i v_i^T$  dargestellt werden kann.



**Abbildung 5.1:** Frobenius-Fehler in Abhangigkeit von k

### 5.2.2 Energieerhalt $E(k)$

Aus den Messdaten:

$$k = 1 : E(1) \approx 79.08\%$$

$$k = 2 : E(2) \approx 96.02\%$$

$$k \geq 3 : E(3) = 100.00\%$$

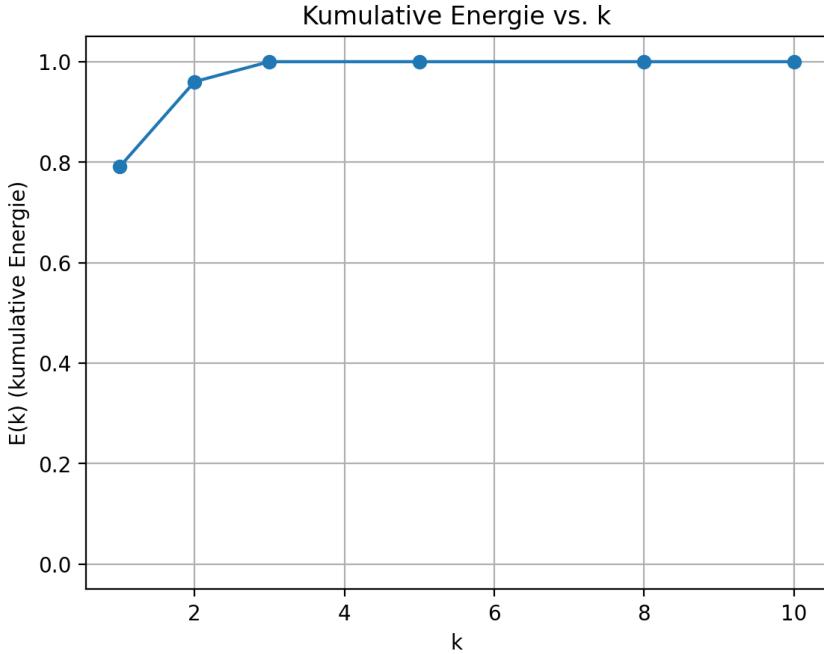
**Interpretation:** Der erste Singularwert tragt bereits fast 80% der Gesamtenergie, was bei binaren Bildern mit stark konzentrierten Informationen typisch ist. Beim zweiten Wert steigt der Energieerhalt auf uber 96%, damit sind die beiden groten Singularwerte bereits nahezu vollstandig. Ab  $k = 3$  wird die gesamte Energie erfasst, was im Einklang mit der Rang-3-Eigenschaft der Matrix steht.

**Folgerung:** Fur die vorliegende Matrix ist die Auswahl von  $k \geq 3$  aquivalent zur vollstandigen Rekonstruktion, wahrend  $k = 2$  bereits eine gute Nherung bieten.

### 5.2.3 Speicherbedarf und relative Einsparung

Aus den Messdaten:

Originalmatrix:  $O = 15 \times 25 = 375$



**Abbildung 5.2:** Energieerhalt in Abhangigkeit von k

$$k = 1 : S(1) = 41, \quad E(1) \approx 89.07\%$$

$$k = 2 : S(2) = 82, \quad E(2) \approx 78.13\%$$

$$k = 3 : S(3) = 123, \quad E(3) \approx 67.20\%$$

$$k = 5 : S(5) = 205, \quad E(5) \approx 45.33\%$$

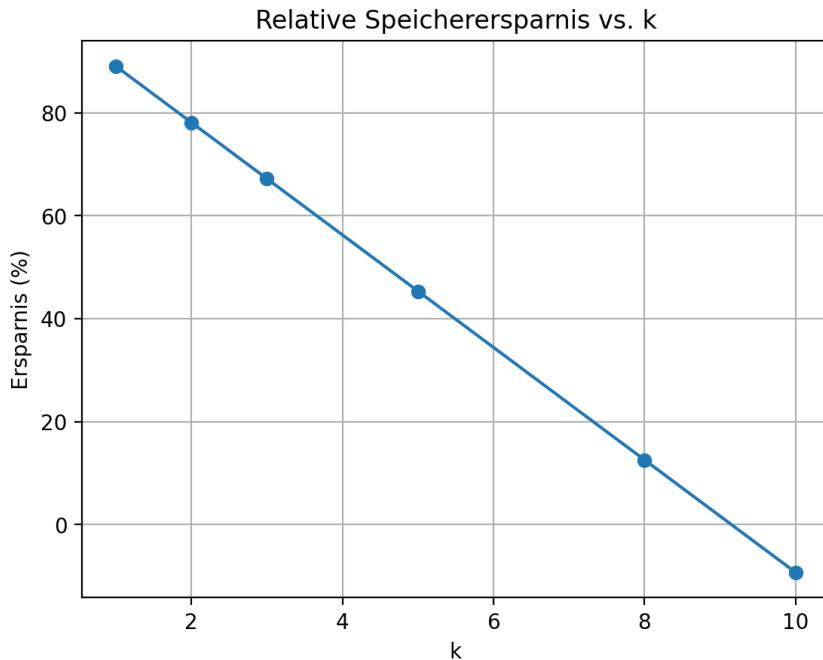
$$k = 8 : S(8) = 328, \quad E(8) \approx 12.53\%$$

$$k = 10 : S(10) = 410, \quad E(10) \approx -9.33\%$$

**Interpretation:** Die Speicherersparnis ist bei niedrigen  $k$ -Werten am groten. Bereits bei  $k = 1$  wird fast 90% des Speicherplatzes eingespart, wahrend bei  $k = 2$  immer noch 78% Einsparung erzielt werden. Ab  $k = 3$  sinkt die Einsparung auf etwa 67%, was immer noch signifikant ist. Mit zunehmendem  $k$  verringert sich die Einsparung rapide, und ab  $k = 10$  ubersteigt der Speicherbedarf der SVD-Rekonstruktion den der Originalmatrix.

Daraus lasst sich ableiten das SVD-Kompression nur sind voll ist wenn  $S(k) < O$ . Daraus ergibt sich eine Hochstgrenze  $k_{max}$  fur sinnvolle Kompression aus

$$k_{max} < \frac{O}{m + n + 1} = \frac{m \times n}{m + n + 1} \tag{5.1}$$



**Abbildung 5.3:** Relative Speicherersparnis in Abhängigkeit von  $k$

Für die vorliegende Matrix ( $m = 15, n = 25$ ) ergibt sich:

$$k_{max} < \frac{375}{41} \approx 9.15 \quad (5.2)$$

Was die Ergebnisse bestätigen, da bei  $k = 10$  die Einsparung negativ wird.

### 5.3 Bestimmung des optimalen $k$ -Werts

Die Antwort auf die Frage nach dem optimalen  $k$ -Wert hängt von den spezifischen Anforderungen an Speicherplatz und Rekonstruktionsqualität ab. Basierend auf den vorliegenden Ergebnissen lassen sich folgende Empfehlungen ableiten:

- Für Anwendungen, die eine nahezu perfekte Rekonstruktion erfordern, ist  $k = 3$  optimal, da hier die Originalmatrix exakt wiederhergestellt wird und dennoch eine Speicherersparnis von über 67% erzielt wird.
- Für Szenarien, in denen Speicherplatz eine kritische Rolle spielt und eine leichte Qualitätsminderung akzeptabel ist, bietet  $k = 2$  einen hervorragenden Kompromiss mit über 78% Speicherersparnis und einem Frobenius-Fehler von nur etwa 1.8278.
- $k = 1$  kann in sehr speicherbeschränkten Umgebungen verwendet werden, wobei jedoch ein deutlicher Qualitätsverlust in Kauf genommen werden muss.

Rekonstruktionen mit  $k > 3$  sind in diesem Fall nicht sinnvoll, da sie keinen Qualitätsgewinn bringen und den Speicherbedarf unnötig erhöhen.

## 5.4 Numerische Genauigkeit und Stabilität

Die bei  $k \geq 3$  beobachteten numerischen Fehler von  $4 \times 10^{-15}$  sind auf die begrenzte Genauigkeit der Gleitkommadarstellung in Computern zurückzuführen. Diese Werte sind vernachlässigbar und bestätigen die Stabilität der SVD-Methode für die vorliegende Matrix.

# Literatur

1. GOLUB, Gene H.; VAN LOAN, Charles F. *Matrix Computations*. 4. Aufl. Baltimore, MD: Johns Hopkins University Press, 2013.
2. STRANG, Gilbert. *Linear Algebra (MIT OpenCourseWare)* [<https://ocw.mit.edu/courses/18-06-linear-algebra-spring-2010/>]. 2010. Accessed: 2025-11-25.

# **Anhang**

## **Selbstständigkeitserklärung**

Ich versichere, dass ich die vorliegende Hausarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Bestandteile der Arbeit, die mittels Künstlicher Intelligenz entstanden sind, wurden ausdrücklich gekennzeichnet. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht noch einer anderen Prüfungsbehörde vorgelegt.

Leipzig, 10. Oktober 2025

---

Malte Luca Peukert