```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# """
# Created on Fri Jan  5 23:57:22 2018
#
# @author: christianehelzel
# """
```

1 12. Vorlesung: Singulärwertzerlegung,

1.1 Teil 2 Datenkompression in der Bildübertragung

```
import numpy as np
import numpy.linalg as npl
from matplotlib import pyplot as plt

image_color = plt.imread('globe.png')
image_color.shape
```

```
(2048, 2048, 3)
```

plt.imread(): Read an image from a file into an array. Return value is a numpy.array of size MxNx3 (for RGB image) or MxN for grayscale image

Die einzelnen Farbkanäle (Rot, Grün und Blau)

```
# image_color[:,:,0] : Red
# image_color[:,:,1] : Green
# image_color[:,:,2] : Blue
```

Zeichne das Bild (farbig)

```
plt.figure()
plt.imshow(image_color)
plt.axis('off')
plt.show()

# plt.imshow(): Displays the image on the axis
```

Erzeuge schwarz-weiss Bild

Hier benutzen wir nur den roten Farbkanal (weitere Varianten: siehe Übung)

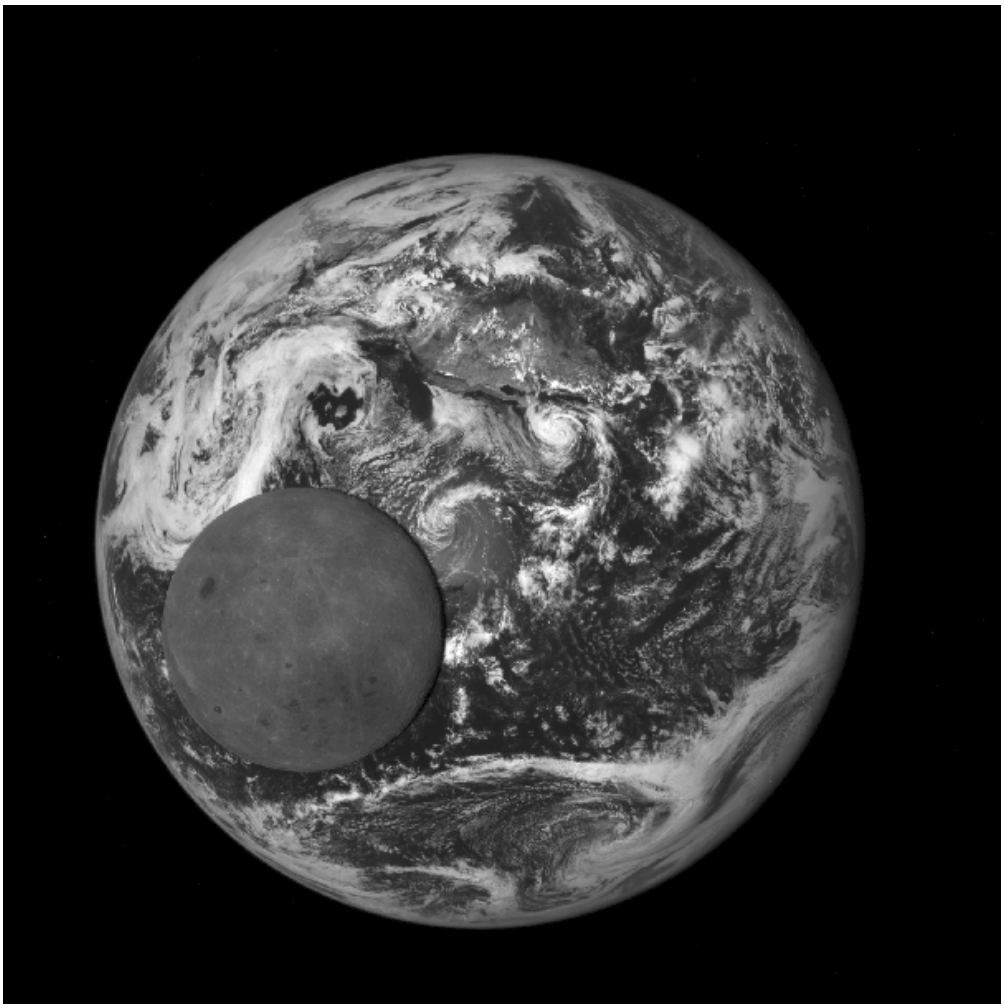
```
red_layer = image_color[:,:,0]
red_layer.shape
```

```
(2048, 2048)
```

Zeichne das Bild (schwarz-weiß)

'cmap' gibt an in welchen Farben das Graustufenbild gezeichnet werden soll

```
# cmap=None by default
plt.figure()
plt.imshow(red_layer,cmap="gray")
plt.axis('off')
plt.show()
```



Liste weiterer color-maps:

https://matplotlib.org/examples/color/colormaps_reference.html

1.2 Singulärwertzerlegung und Bilder

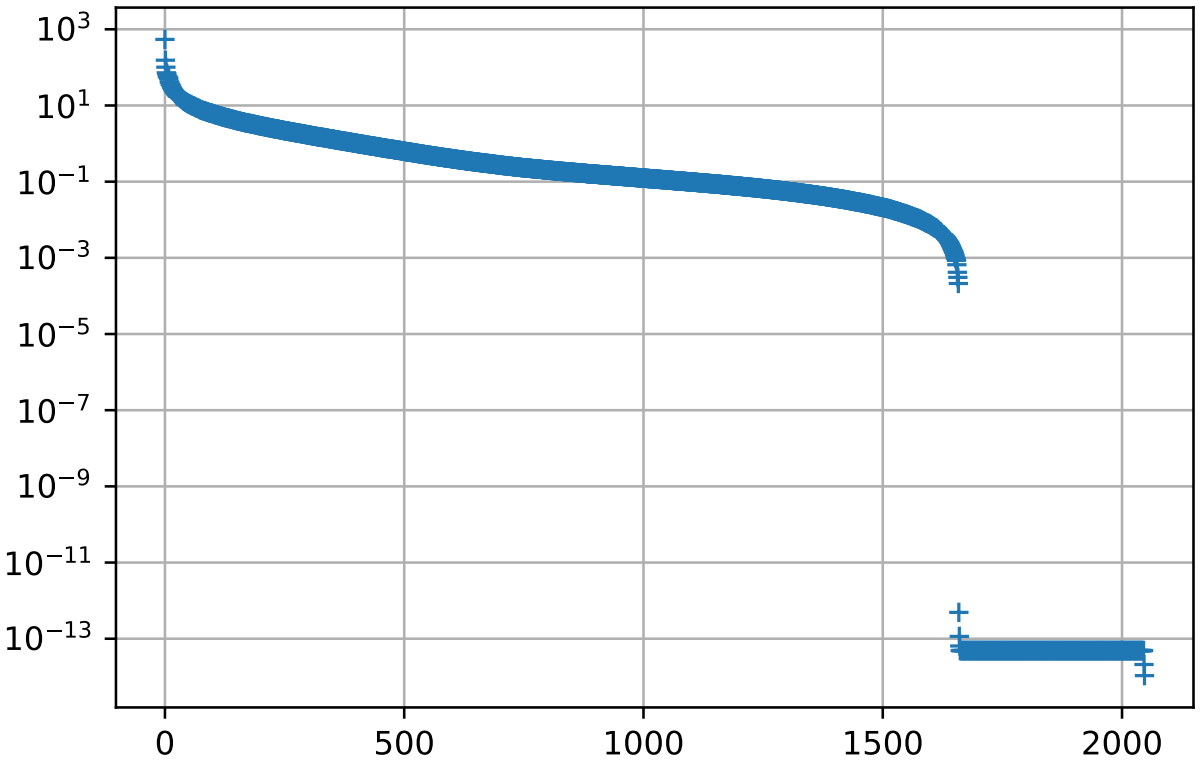
Berechne Singulärwertzerlegung (SVD)

```
U, sigma, Vh = npl.svd(red_layer);
```

Plot Singulärwerte auf logarithmischer Scala

Beachte: Wenige Singulärwerte (ca. 50) sind mindestens 100 x größer als die meisten anderen Singulärwerte

```
plt.figure()
plt.semilogy(sigma,'+')
plt.grid('on')
plt.show()
```



1.3 Datenkompression

```
k = 100;

U, sigma, Vh = npl.svd(red_layer);
```

Daten reduzieren: Benutze nur die ersten k Singulärwerte

```
sigma_k = sigma[:k];
U_k = U[:, :k];
Vh_k = Vh[:k, :];
```

Bild aus den reduzierten Daten wiederherstellen

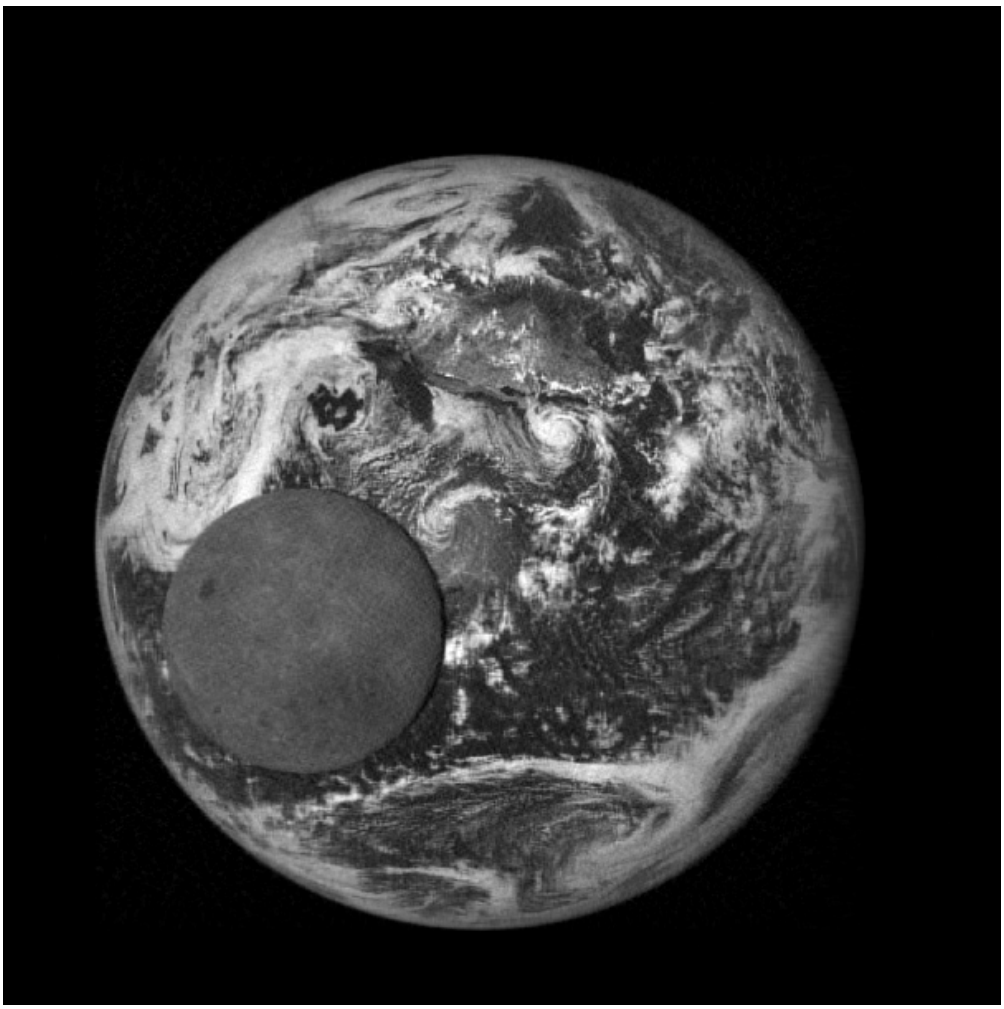
```
bild_compressed = U_k @ np.diag(sigma_k) @ Vh_k;
```

Wichtig: Die Einträge in arrays müssen im Intervall [0,1] sein, damit Matplotlib das Bild anzeigen kann. Durch die Komprimierung kann es sein, dass das wiederhergestellte Bild Einträge <0 oder >1 hat.

```
bild_compressed=np.clip(bild_compressed,0,1)
# np.clip() : clip(limit) the values in an array.
# Given an interval, values outside the interval are clipped to the interval edges.
```

Zeichne das aus den reduzierten Matrizen rekonstruierte Bild

```
plt.figure()
plt.imshow(bild_compressed,cmap="gray")
plt.axis('off')
plt.show()
```



Wieviel Platz spart man?

```
print("Originale Bildgröße: {0}".format(red_layer.shape));
print("Speicherbedarf:      {0}".format(red_layer.size));

print("Größe der Matrizen:  U = {0}, V^H = {1}, sigma = {2}".\
      format(U.shape, Vh.shape, sigma.shape));
print("Speicherbedarf:      {0:d}".format(U.size + Vh.size + sigma.size));


print("komprimierte Größen: U = {0}, V^H = {1}, sigma = {2}".\
      format(U_k.shape, Vh_k.shape, sigma_k.shape));
print("Speicherbedarf:      {0:d}".format(U_k.size + Vh_k.size + sigma.size));
# Bemerkung: Die Syntax der print-Anweisung wurde in VL3 eingeführt
```

```
print("Größe des komprimierten Bildes:  {0}".format(bild_compressed.shape))
```

```
Originale Bildgröße: (2048, 2048)
Speicherbedarf:      4194304
Größe der Matrizen:  U = (2048, 2048), V^H = (2048, 2048), sigma =
(2048,)
Speicherbedarf:      8390656
komprimierte Größen: U = (2048, 100), V^H = (100, 2048), sigma =
(100,)
Speicherbedarf:      411648
Größe des komprimierten Bildes:  (2048, 2048)
```

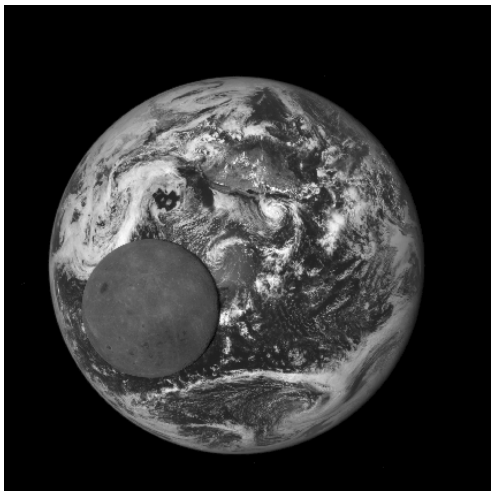
Beachte: Die Datenübertragung ist effizienter, das endgültige Bild hat aber die gleiche Größe

Zeichne beide Bilder im Vergleich nebeneinander

```
plt.figure()
plt.subplot(1, 2, 1);
plt.imshow(red_layer, cmap = "gray");
plt.axis("off");

plt.subplot(1, 2, 2);
plt.imshow(bild_compressed, cmap = "gray");
bild_compressed=np.clip(bild_compressed,0,1)
plt.axis("off");
```

```
(-0.5, 2047.5, 2047.5, -0.5)
```



1.4 Datenkompression für farbige Bilder

1. Ansatz führe SVD für 3*2048 x 2048 Matrix durch

```
big_image_color = np.vstack((image_color[:,:,0],\
                             image_color[:,:,1],image_color[:,:,2]))
U, sigma, Vh = npl.svd(big_image_color);
```

```
k=100

sigma_k = sigma[:k];
U_k = U[:, :k];
Vh_k = Vh[:k, :];
big_image_compressed = U_k @ np.diag(sigma_k) @ Vh_k;

big_image_compressed = np.clip(big_image_compressed,0,1)


[m,n,l] = image_color.shape

big_image_result = np.empty((m,n,3))
big_image_result[:,:,0] = big_image_compressed[0:m,0:n]
big_image_result[:,:,1] = big_image_compressed[m:2*m,0:n]
big_image_result[:,:,2] = big_image_compressed[2*m:,0:n]

plt.figure()
plt.imshow(big_image_result)
plt.axis('off')
plt.show()
```

Beide Bilder im Vergleich

```
plt.figure()
plt.subplot(1, 2, 1);
plt.imshow(image_color);
plt.axis("off");

plt.subplot(1, 2, 2);
plt.imshow(big_image_result);
plt.axis("off");
```

```
(-0.5, 2047.5, 2047.5, -0.5)
```

2. Ansatz: Führe SVD für jede Farbkomponente separat durch

```
[m,n,l] = image_color.shape
image_compressed = np.empty((m,n,3))
krgb = [40,40,40]
for i in range(3):
    k = krgb[i]
    image_part = image_color[:,:,i]
    U, sigma, Vh = npl.svd(image_part);

    sigma_k = sigma[:k];
    U_k = U[:, :k];
    Vh_k = Vh[:k, :];
    image_compressed[:,:,i] = U_k @ np.diag(sigma_k) @ Vh_k;


image_compressed=np.clip(image_compressed,0,1)
plt.figure()
plt.imshow(image_compressed)
plt.axis('off')
plt.show()
```

Beide Bilder im Vergleich

```
plt.figure()
plt.subplot(1, 2, 1);
plt.imshow(image_color);
plt.axis("off");

plt.subplot(1, 2, 2);
plt.imshow(image_compressed);
plt.axis("off");
```

```
(-0.5, 2047.5, 2047.5, -0.5)
```