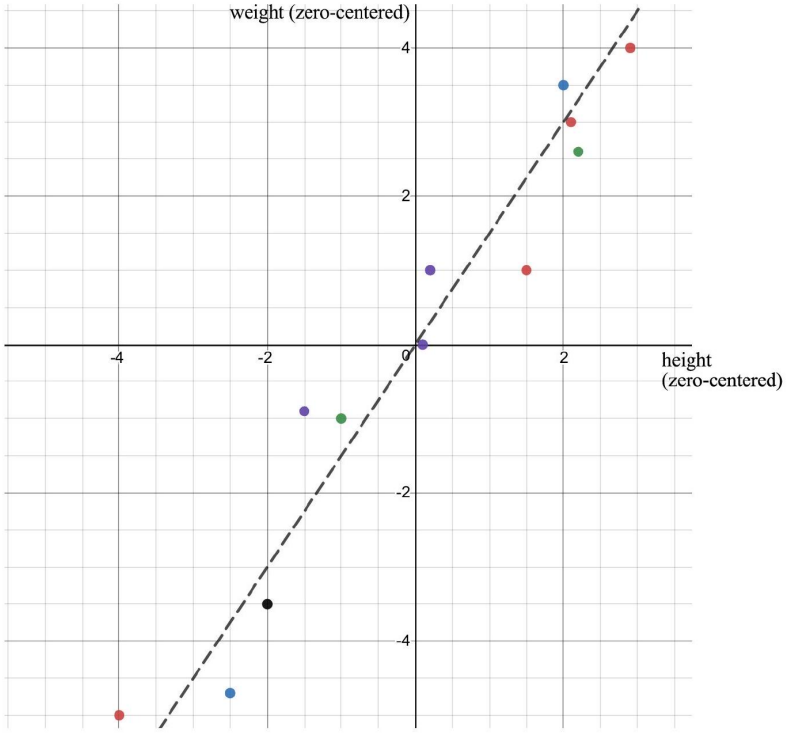
# Ch5 Lecture 4

## *SVD on matrices of data*

### *Example: Height and weight*

$$A^T = \begin{bmatrix} 2.9 & -1.5 & 0.1 & -1.0 & 2.1 & -4.0 & -2.0 & 2.2 & 0.2 & 2.0 & 1.5 & -2.5 \\ 4.0 & -0.9 & 0.0 & -1.0 & 3.0 & -5.0 & -3.5 & 2.6 & 1.0 & 3.5 & 1.0 & -4.7 \end{bmatrix}$$

. . .

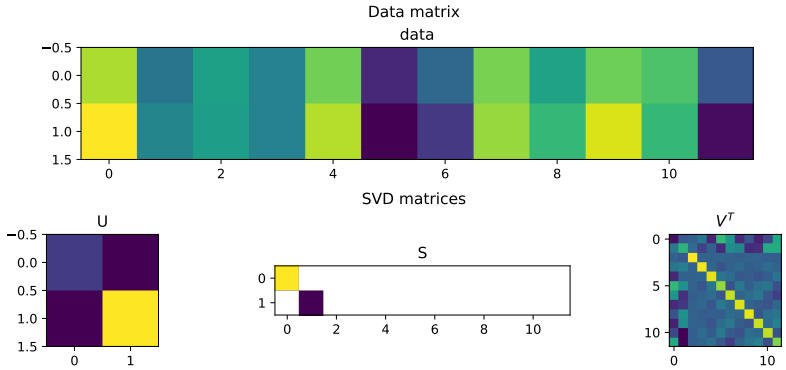


$$A^T = \begin{bmatrix} 2.9 & -1.5 & 0.1 & -1.0 & 2.1 & -4.0 & -2.0 & 2.2 & 0.2 & 2.0 & 1.5 & -2.5 \\ 4.0 & -0.9 & 0.0 & -1.0 & 3.0 & -5.0 & -3.5 & 2.6 & 1.0 & 3.5 & 1.0 & -4.7 \end{bmatrix}$$

. . .

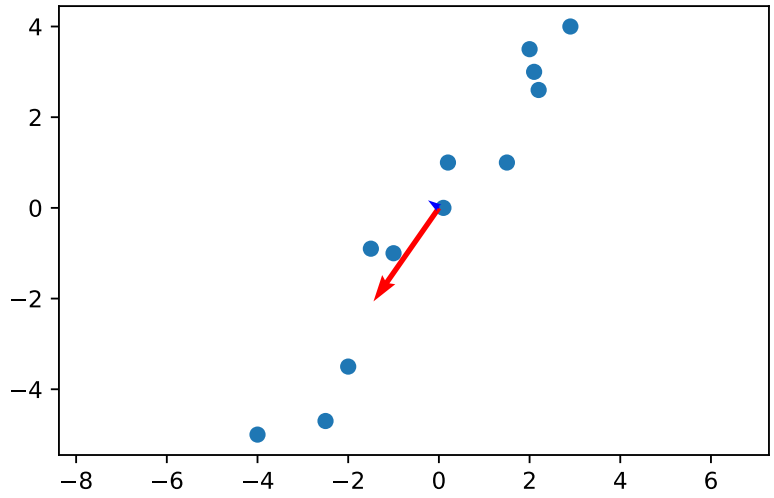
### *Plotting*

```
<Figure size 1650x1050 with 0 Axes>
```




The columns of the U matrix, graphically:

```
plt.clf()

plt.scatter(data.T[:,0],data.T[:,1])
plt.quiver(0,0,Ud[0,0]*Sd[0,0]/5,Ud[1,0]*Sd[0,0]/5,angles='xy',scale_units='xy',scale=1, color
plt.quiver(0,0,Ud[0,1]*Sd[1,1]/5,Ud[1,1]*Sd[1,1]/5,angles='xy',scale_units='xy',scale=1, color
plt.axis('equal')
plt.show()
```




. . .

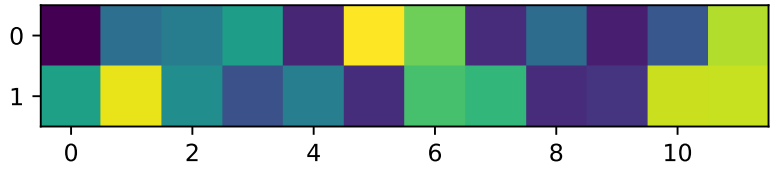
U captures relationships between the **rows** of the data matrix.

. . .

Since there are only two rows, only 2x2 matrix needed to capture all the relationships.

The first two rows of the $V^T$ matrix

```
#plt.scatter(data[:,0],data[:,1])
plt.clf()
plt.imshow(Vd.T[:2,:])

plt.show()
```

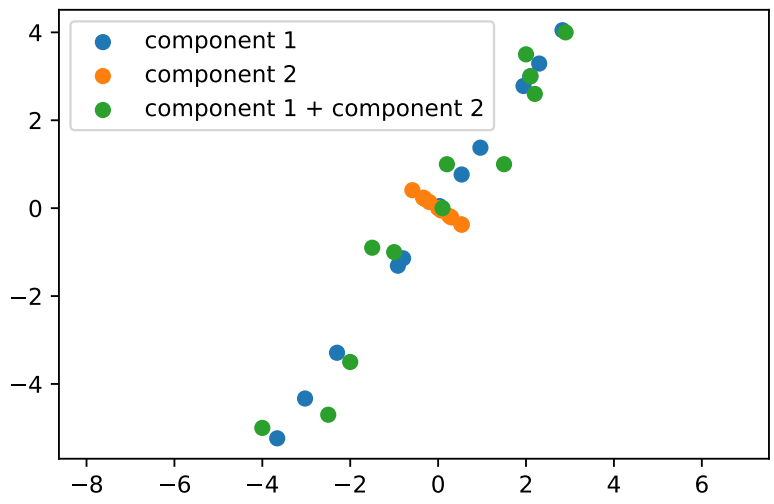


. . .

V captures relationships between the **columns** of the data matrix. 12x12 possible values, but only 12x2 needed to capture all the relationships.

The data from these first two rows of the $V^T$ matrix, after multiplication by the singular values and rotated by the columns of the U matrix:

```
#plt.scatter(data[:,0],data[:,1])
plt.clf()
component1 = np.outer(Vd.T[:,0],Sd[0,0]*Ud[0])
component2 = np.outer(Vd.T[:,1],Sd[1,1]*Ud[1])
plt.scatter(component1[:,0],component1[:,1])
plt.scatter(component2[:,0],component2[:,1])
combined = component1 + component2
plt.scatter(combined[:,0],combined[:,1])
plt.axis('equal')
# add a legend
plt.legend(["component 1", "component 2", "component 1 + component 2"])
plt.show()
```



. . .

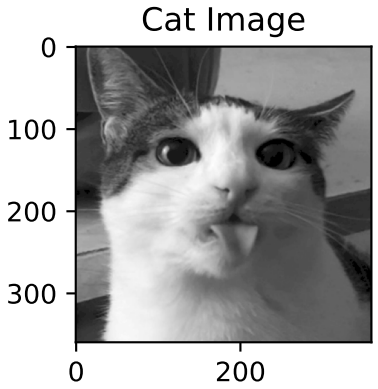
A reminder:

$$\mathbf{A} = \mathbf{USV}^\mathsf{T} = \sigma_1 u_1 v_1^\mathsf{T} + \cdots + \sigma_r u_r v_r^\mathsf{T}$$

## *SVD on images*

### *Motivation: a cat*

```
# display the path of the current python environment

import cv2

image = cv2.imread('test_cat.png', cv2.IMREAD_GRAYSCALE)
plt.figure(figsize=(2, 2))
plt.imshow(image, cmap='gray')
plt.title('Cat Image')
plt.show()
```




### *Dimensions of the decomposition*

What are the dimensions of the decompositions for an image?

```
U, S, Vt = np.linalg.svd(image, full_matrices=False)
print(f'The shape of U is {U.shape}, the shape of S is {S.shape}, the shape of V is {Vt.shape}')
```

```
The shape of U is (360, 360), the shape of S is (360,), the shape of V is (360, 360)
```

*pause*

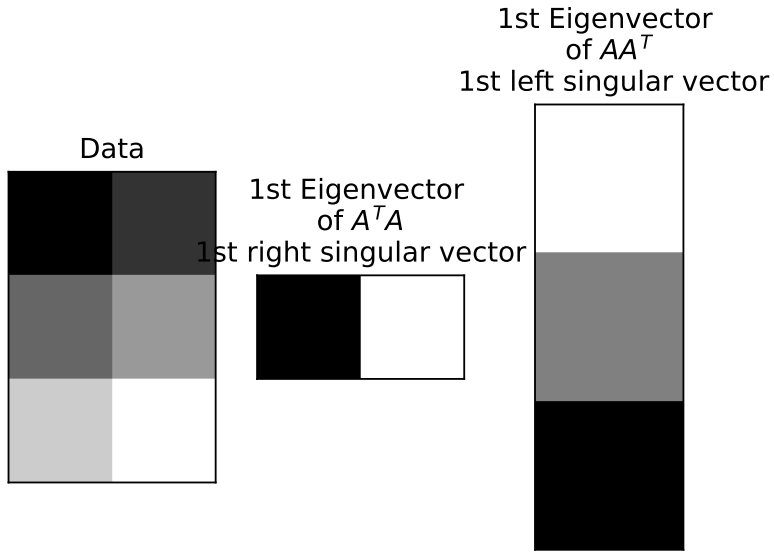
. . .

**Left singular values**, corresponding to U, are the eigenvalues of $AA^T$. For an image, $AA^T$ is the covariance matrix of the rows of $A$.

. . .

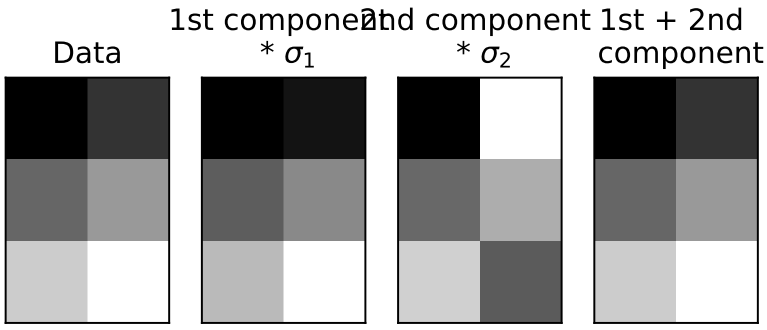
**Right singular values** are the eigenvalues of $A^TA$. For an image, $A^TA$ is the covariance matrix of the columns of $A$.

### *Simple example*



### *Reconstructing our matrix*

```
U, S, Vt = np.linalg.svd(A, full_matrices=False)
ax1 = plt.subplot(141)
ax2 = plt.subplot(142)
ax3 = plt.subplot(143)
ax4 = plt.subplot(144)
ax1.imshow(A, cmap='gray')
ax2.imshow(np.outer(U[:,0], Vt[0,:])*S[0], cmap='gray')
ax3.imshow(np.outer(U[:,1], Vt[1,:])*S[1], cmap='gray')
ax4.imshow(np.outer(U[:,0], Vt[0,:])*S[0]+np.outer(U[:,1], Vt[1,:])*S[1], cmap='gray')
for ax in [ax1, ax2, ax3,ax4]:
  ax.axes.xaxis.set_ticks([])
  ax.axes.yaxis.set_ticks([])
ax1.set_title('Data')
ax2.set_title('1st component \n * $\sigma_1$')
ax3.set_title('2nd component \n * $\sigma_2$')
ax4.set_title('1st + 2nd \n component')
plt.show()
```

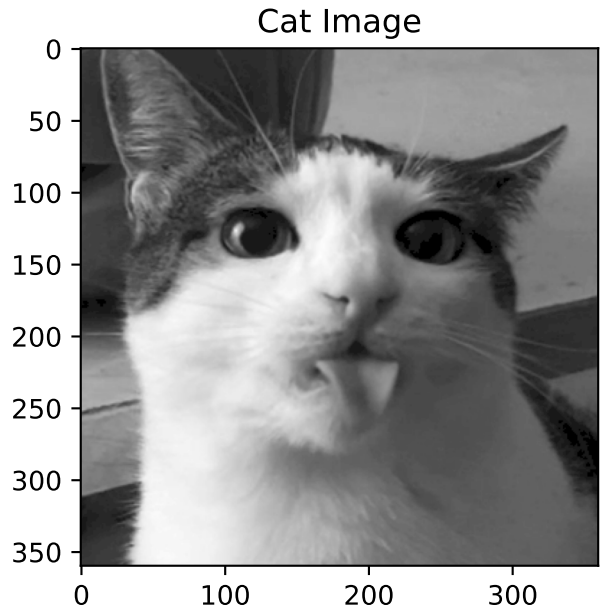



How much of the variance is captured by the first two components?

The variance captured by the each component is the sum of the squares of the singular values divided by the sum of the squares of all the singular values.

***Back to the cat***

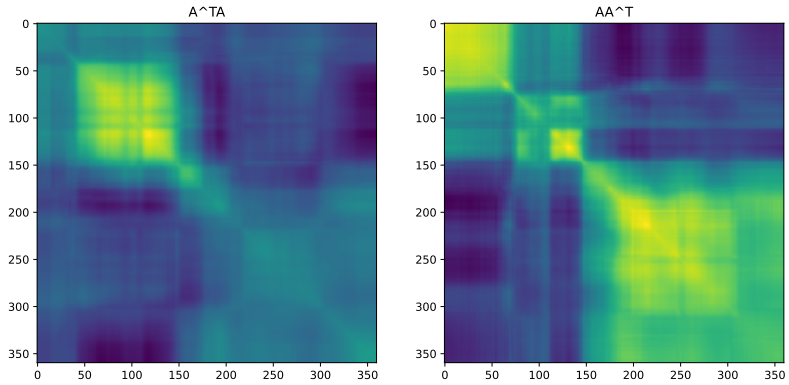
```
# !pip3 install opencv-python

plt.imshow(image, cmap='gray')
plt.title('Cat Image')
plt.show()
```



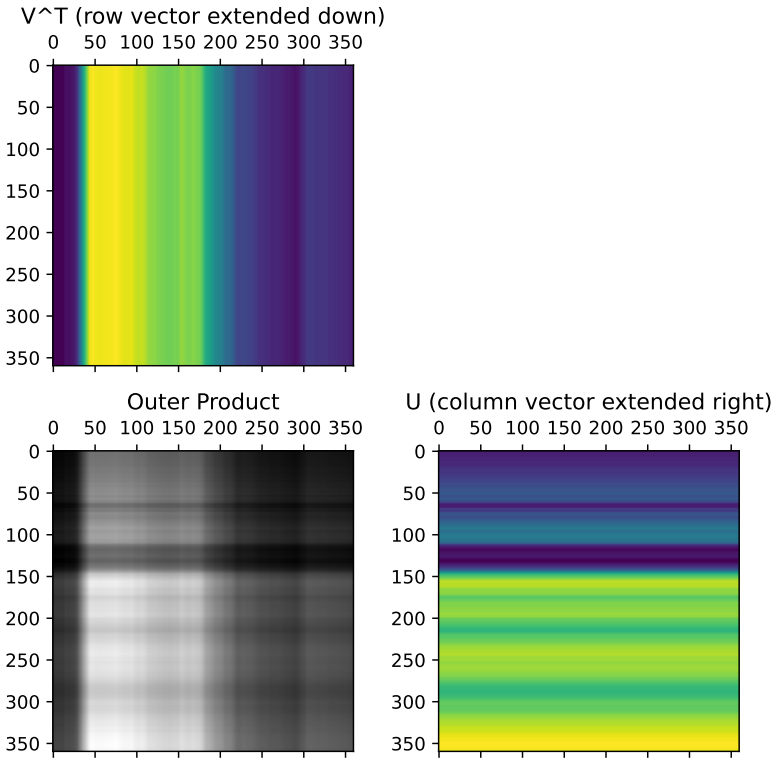

```
U, S, Vt = np.linalg.svd(image, full_matrices=False)
```

```
fig = plt.figure(figsize=(12, 6))
imsub = image-np.mean(image,axis=0)
imsub = imsub - np.mean(imsub,axis=1)
aat=imsub@imsub.T
ata=imsub.T@imsub
ax1=fig.add_subplot(121)
ax1.imshow(ata)


ax2=fig.add_subplot(122)
```

```
ax2.imshow(aat)
ax2.set_title("AA^T")
ax1.set_title("A^TA")
plt.show()
```

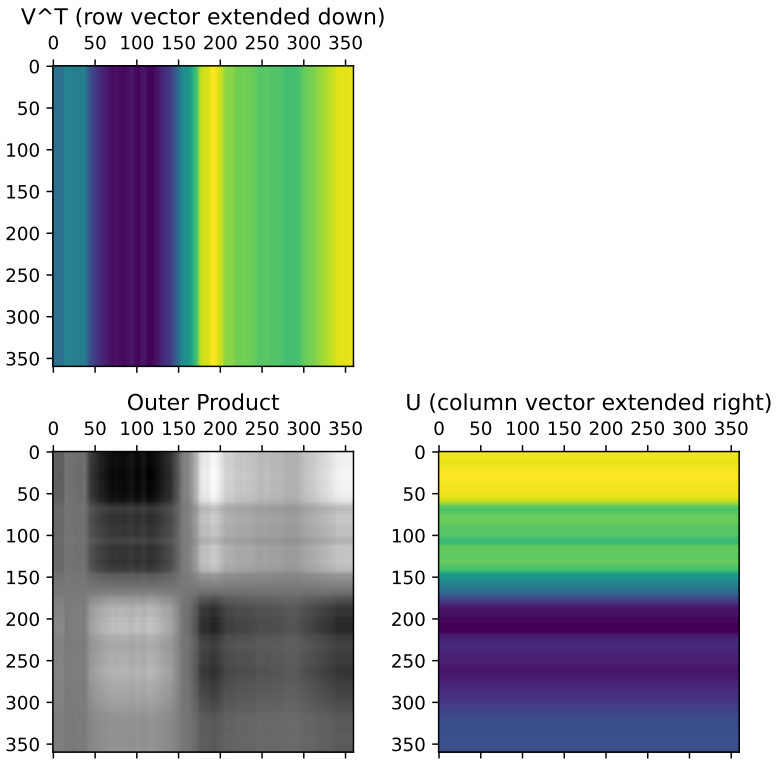


### *First Singular Value*



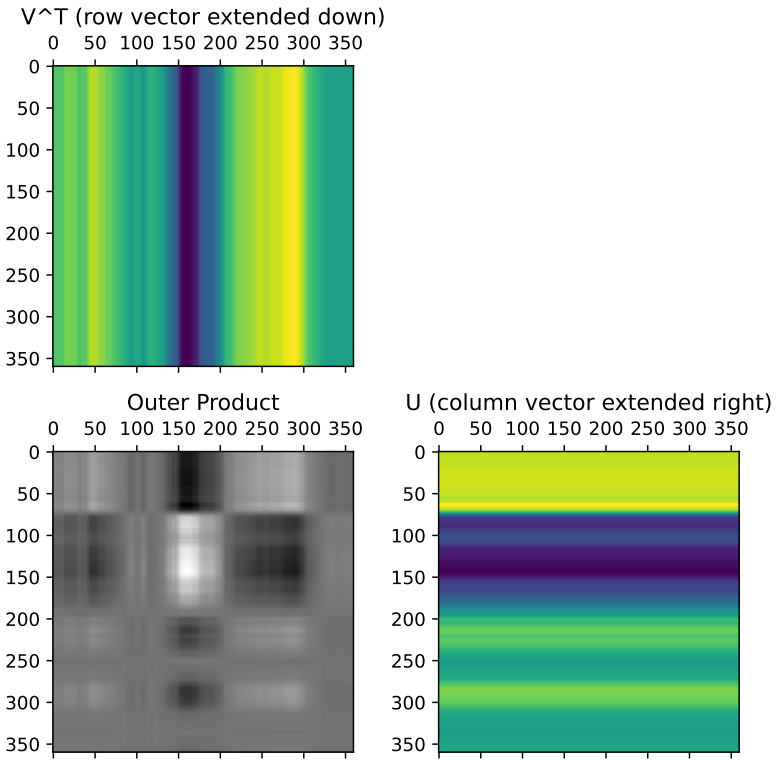

### *Second Singular Value*

```
plot_uv(1)
```



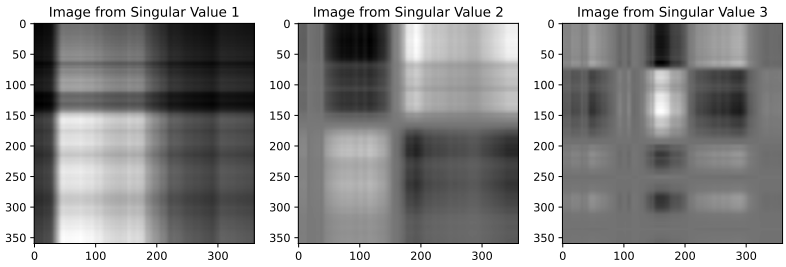

### *Third Singular Value*

```
plot_uv(2)
```



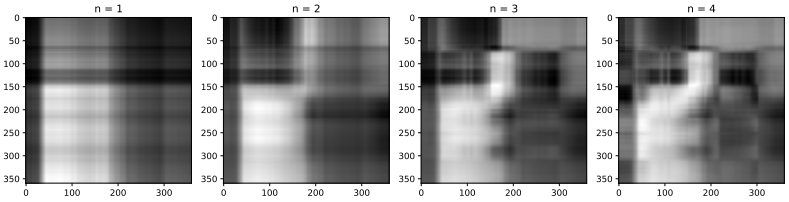

### *Adding them up*

```
fig = plt.figure(figsize=(12, 6))
for i in range(3):
    reconstructed_image = np.matrix(U[:,i:i+1]) * np.diag(S[i:i+1]) * np.matrix(Vt[i:i+1,:])
    ax1 = fig.add_subplot(131+i)
    ax1.imshow(reconstructed_image, cmap='gray')
    ax1.set_title(f'Image from Singular Value {i+1}')
plt.show()
```



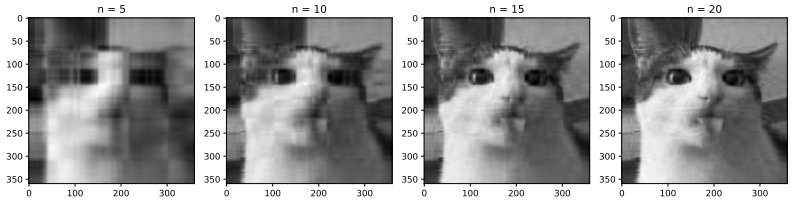

. . .

```
plt.figure(figsize=(16,4))

# start, end, step = 5, 25, 5
start, end, step = 1, 5, 1
for i in range(start, end, step):
    plt.subplot(1, (end - start) // step + 1, (i - start) // step + 1)
    reconstructed = np.matrix(U[:, :i]) * np.diag(S[:i]) * np.matrix(Vt[:i, :])
    plt.imshow(reconstructed, cmap='gray')
    plt.title('n = %s' % i)

plt.tight_layout()
plt.show()
```

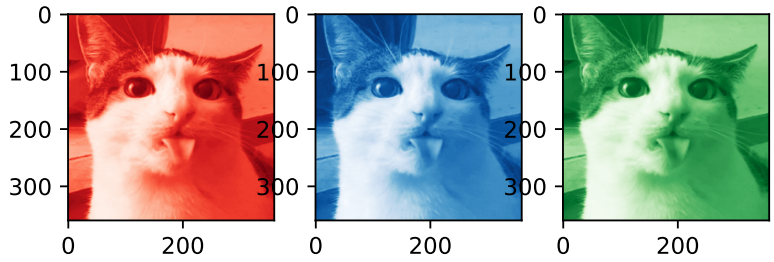



```
plt.figure(figsize=(16,4))

start, end, step = 5, 25, 5
#start, end, step = 1, 5, 1
for i in range(start, end, step):
    plt.subplot(1, (end - start) // step + 1, (i - start) // step + 1)
```

```
    reconstructed = np.matrix(U[:, :i]) * np.diag(S[:i]) * np.matrix(Vt[:i, :])
    plt.imshow(reconstructed, cmap='gray')
    plt.title('n = %s' % i)

plt.tight_layout()
plt.show()
```

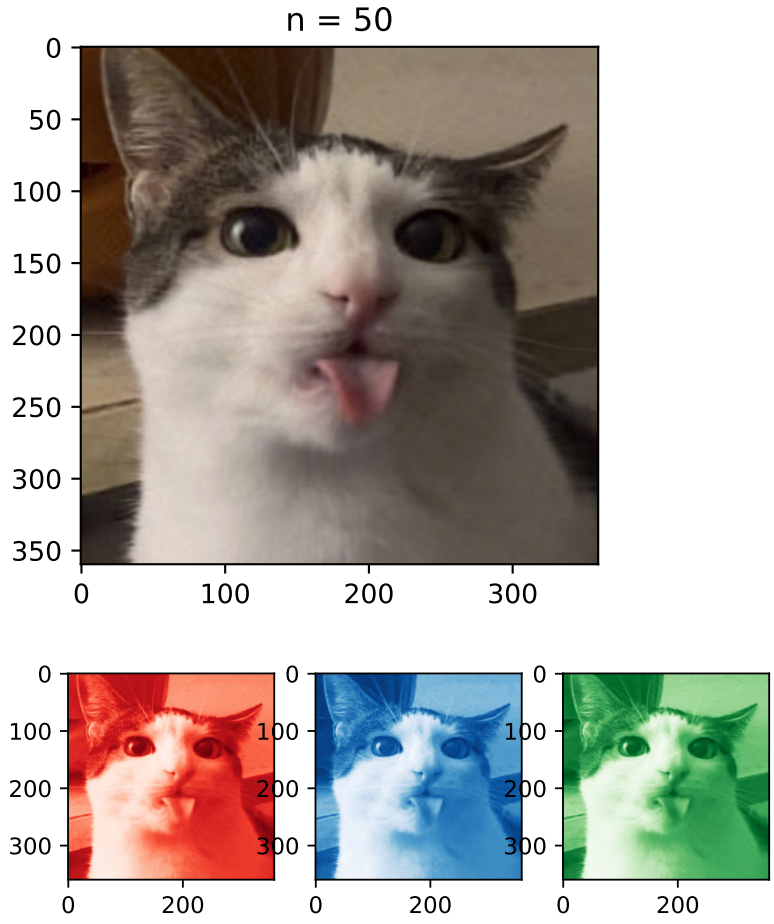



### *Color images*



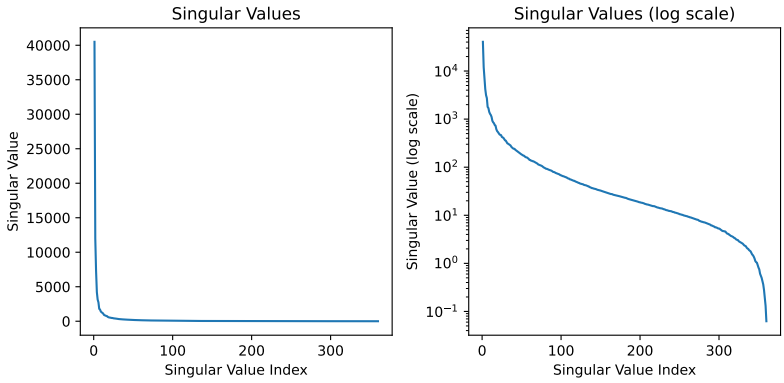
```
# SVD for each channel
U_R, S_R, Vt_R = np.linalg.svd(R, full_matrices=False)
U_G, S_G, Vt_G = np.linalg.svd(G, full_matrices=False)
U_B, S_B, Vt_B = np.linalg.svd(B, full_matrices=False)

n = 50  # rank approximation parameter
R_compressed = np.matrix(U_R[:, :n]) * np.diag(S_R[:n]) * np.matrix(Vt_R[:n, :])
G_compressed = np.matrix(U_G[:, :n]) * np.diag(S_G[:n]) * np.matrix(Vt_G[:n, :])
B_compressed = np.matrix(U_B[:, :n]) * np.diag(S_B[:n]) * np.matrix(Vt_B[:n, :])

# Combining the compressed channels
compressed_image = cv2.merge([np.clip(R_compressed, 1, 255), np.clip(G_compressed, 1, 255), np
compressed_image = compressed_image.astype(np.uint8)
plt.imshow(compressed_image)
```

```
plt.title('n = %s' % n)
plt.show()

# Plotting the compressed RGB channels
plt.subplot(1, 3, 1)
plt.imshow(R_compressed, cmap='Reds_r')
plt.subplot(1, 3, 2)
plt.imshow(B_compressed, cmap='Blues_r')
plt.subplot(1, 3, 3)
plt.imshow(G_compressed, cmap='Greens_r')
plt.show()
```



### *How many singular values to keep?*

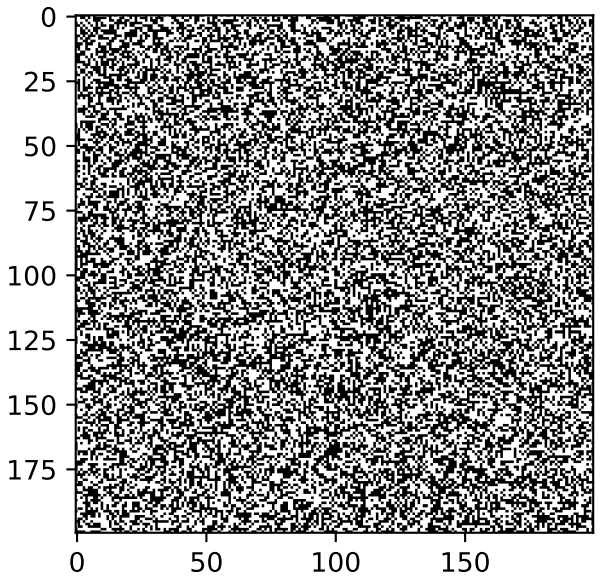
```
# Plotting the singular values
plt.figure(figsize=(8,4))

plt.subplot(1, 2, 1)
plt.plot(range(1, len(S) + 1), S)
plt.xlabel('Singular Value Index')
plt.ylabel('Singular Value')
plt.title('Singular Values')

plt.subplot(1, 2, 2)
plt.plot(range(1, len(S) + 1), S)
plt.xlabel('Singular Value Index')
plt.ylabel('Singular Value (log scale)')
plt.title('Singular Values (log scale)')
plt.yscale('log')

plt.tight_layout()
plt.show()
```



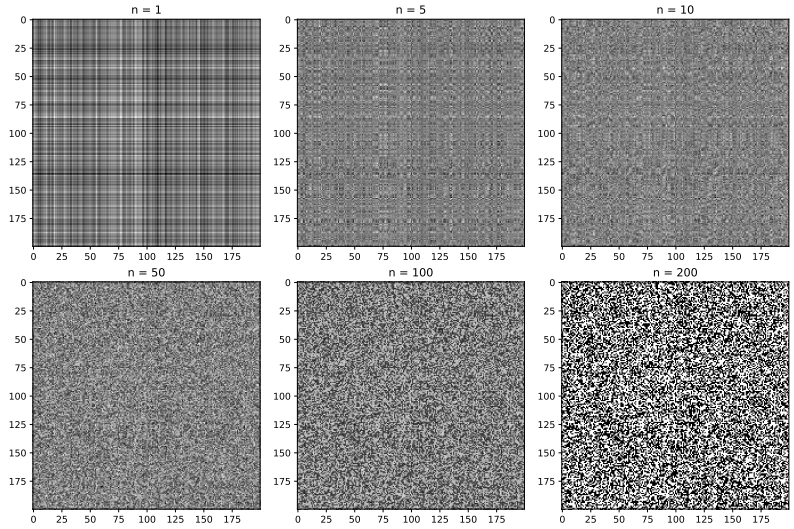

### *Different sorts of images*

Just plain noise:

```
noise = np.random.randint(0,2,size=(200,200))
plt.imshow(noise, cmap='gray')
```



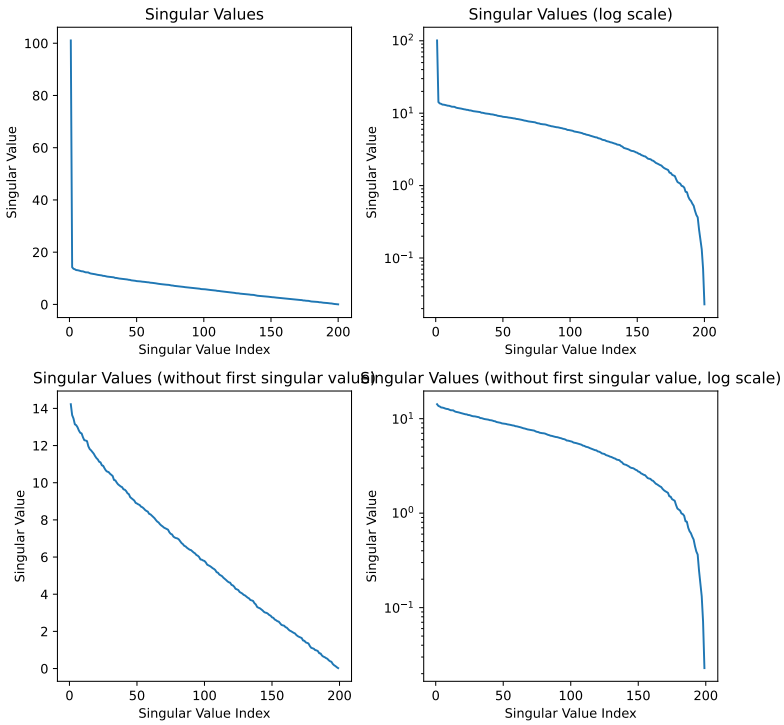

```
U_N, S_N, Vt_N = np.linalg.svd(noise, full_matrices=False)

# Plotting the compressed noise for different values of n
components = [1, 5, 10, 50, 100, 200]

fig = plt.figure(figsize=(12,8))

for i in range(len(components)):
    plt.subplot(2, 3, i+1)
    noise_compressed = np.matrix(U_N[:, :components[i]]) * np.diag(S_N[:components[i]]) * np.ma
    plt.imshow(noise_compressed, cmap='gray')
    plt.title('n = %s' % components[i])

plt.tight_layout()
plt.show()
```




*pause*

```
def plot_singular_values(S, title):
    plt.plot(range(1, len(S) + 1), S)
    plt.xlabel('Singular Value Index')
    plt.ylabel('Singular Value')
    plt.title(title)

plt.figure(figsize=(8, 8))

plt.subplot(2, 2, 1)
plot_singular_values(S_N, 'Singular Values')

plt.subplot(2, 2, 2)
plot_singular_values(S_N, 'Singular Values (log scale)')
plt.yscale('log')

plt.subplot(2, 2, 3)
plot_singular_values(S_N[1:], 'Singular Values (without first singular value)')

plt.subplot(2, 2, 4)
```

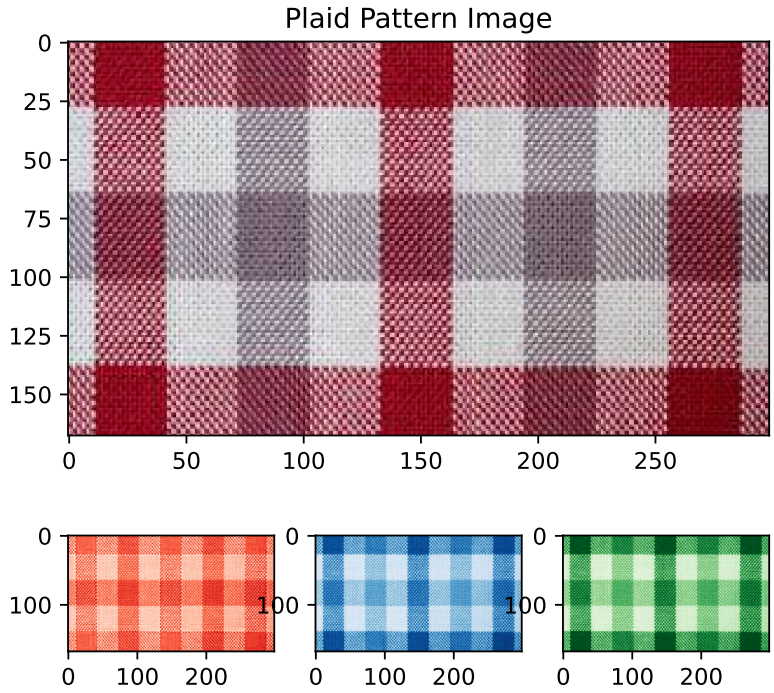
```
plot_singular_values(S_N[1:], 'Singular Values (without first singular value, log scale)')
plt.yscale('log')

plt.tight_layout()
plt.show()
```




### *Plaid shirt*

```
# Show plaid pattern image
plaid_image = cv2.imread('plaid_pattern.jpg')
plt.imshow(plaid_image[:,:,::-1])
plt.title('Plaid Pattern Image')
plt.show()

# Split the image into R, G, and B color channels
B, G, R = cv2.split(plaid_image)
plt.subplot(1, 3, 1)
```

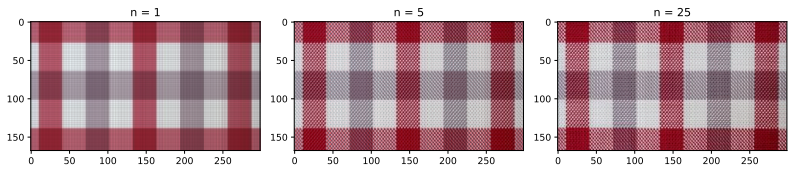
```
plt.imshow(R, cmap='Reds_r')
plt.subplot(1, 3, 2)
plt.imshow(B, cmap='Blues_r')
plt.subplot(1, 3, 3)
plt.imshow(G, cmap='Greens_r')
plt.show()
```




*pause*

```
def rgb_approximation(R, G, B, n):
    U_R, S_R, Vt_R = np.linalg.svd(R, full_matrices=False)
    U_G, S_G, Vt_G = np.linalg.svd(G, full_matrices=False)
    U_B, S_B, Vt_B = np.linalg.svd(B, full_matrices=False)

    R_compressed = np.matrix(U_R[:, :n]) * np.diag(S_R[:n]) * np.matrix(Vt_R[:n, :])
    G_compressed = np.matrix(U_G[:, :n]) * np.diag(S_G[:n]) * np.matrix(Vt_G[:n, :])
```

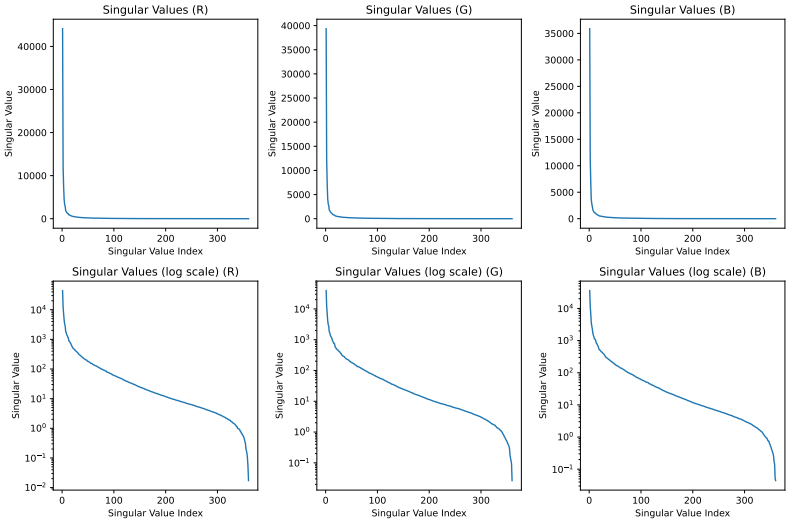
```
    B_compressed = np.matrix(U_B[:, :n]) * np.diag(S_B[:n]) * np.matrix(Vt_B[:n, :])

    compressed_image = cv2.merge([np.clip(R_compressed, 1, 255), np.clip(G_compressed, 1, 255)
    compressed_image = compressed_image.astype(np.uint8)

    return compressed_image

n_values = [1, 5, 25]

plt.figure(figsize=(12, 6))
for i, n in enumerate(n_values):
    plt.subplot(1, len(n_values), i+1)
    plt.imshow(rgb_approximation(R, G, B, n))
    plt.title('n = %s' % n)

plt.tight_layout()
plt.show()
```




### *Singular values*

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1)
plot_singular_values(S_R, 'Singular Values (R)')

plt.subplot(2, 3, 2)
plot_singular_values(S_G, 'Singular Values (G)')

plt.subplot(2, 3, 3)
plot_singular_values(S_B, 'Singular Values (B)')

plt.subplot(2, 3, 4)
```

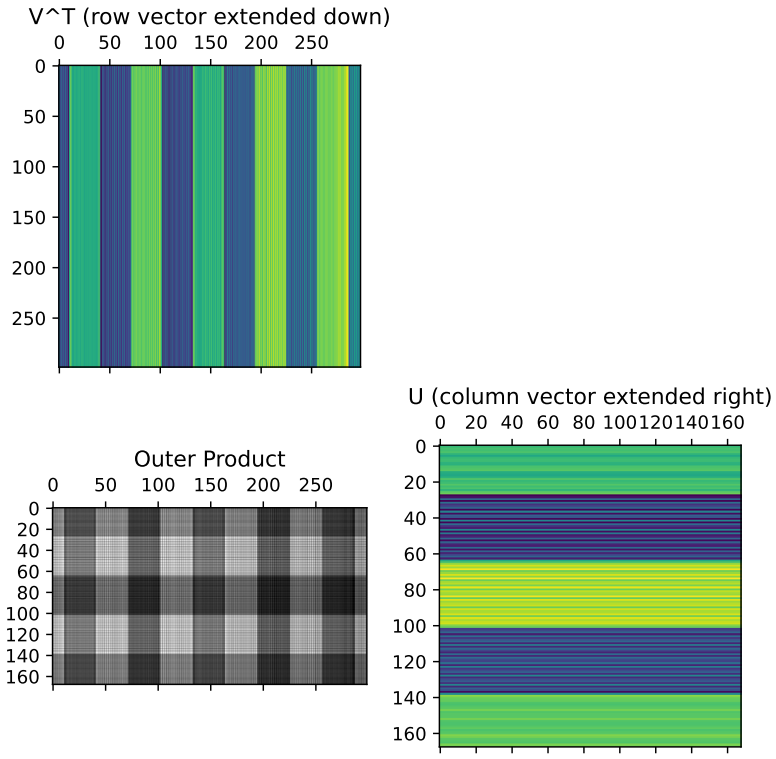
```
plot_singular_values(S_R, 'Singular Values (log scale) (R)')
plt.yscale('log')

plt.subplot(2, 3, 5)
plot_singular_values(S_G, 'Singular Values (log scale) (G)')
plt.yscale('log')

plt.subplot(2, 3, 6)
plot_singular_values(S_B, 'Singular Values (log scale) (B)')
plt.yscale('log')

plt.tight_layout()
plt.show()
```




### *Individual components*

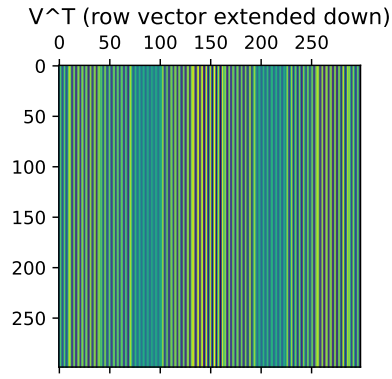
First component:

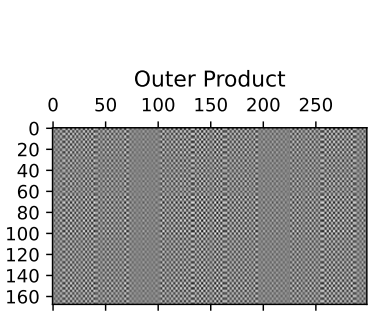
```
 U_R, S_R, Vt_R = np.linalg.svd(R, full_matrices=False)
plot_uv(0, U=U_R, S=S_R, Vt=Vt_R)
```

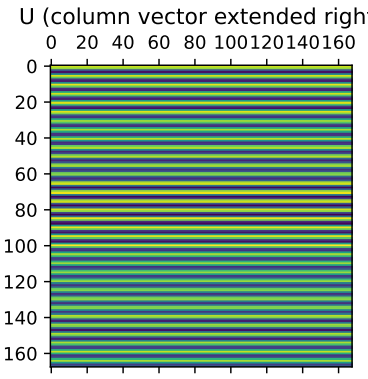



Second component:

```
plot_uv(1, U=U_R, S=S_R, Vt=Vt_R)
```



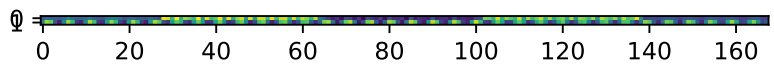




### *Using "PCA" from sklearn*

This is just an easier way to implement taking these first few components...

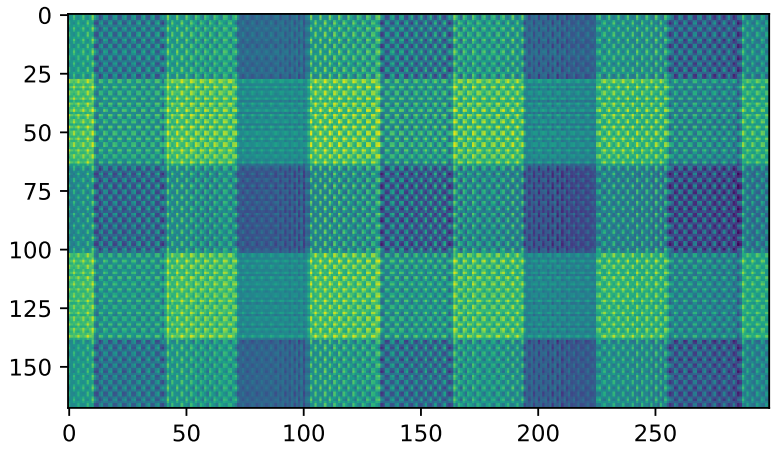
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(R) # fit the model -- compute the matrices
transformed = pca.transform(R) # transform the data
print(f'The shape of the image is {R.shape}, and the shape of the compressed image is {transfo
plt.imshow(transformed.T)
```

```
The shape of the image is (168, 299), and the shape of the compressed image is (168, 2)
```



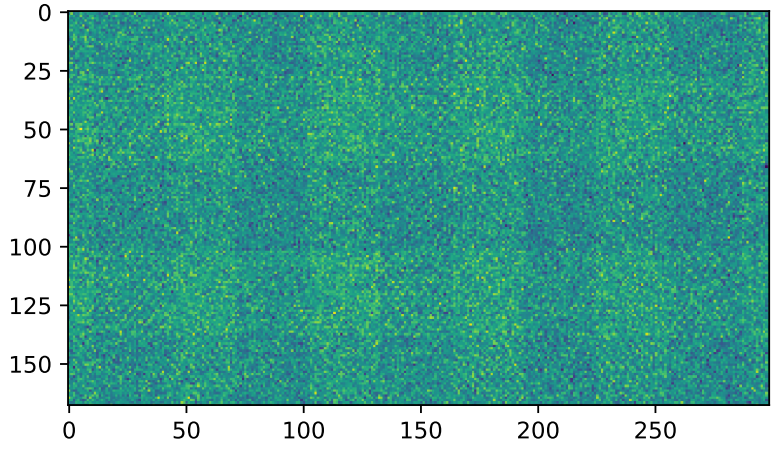
. . .

```
plt.imshow(pca.inverse_transform(transformed))
```



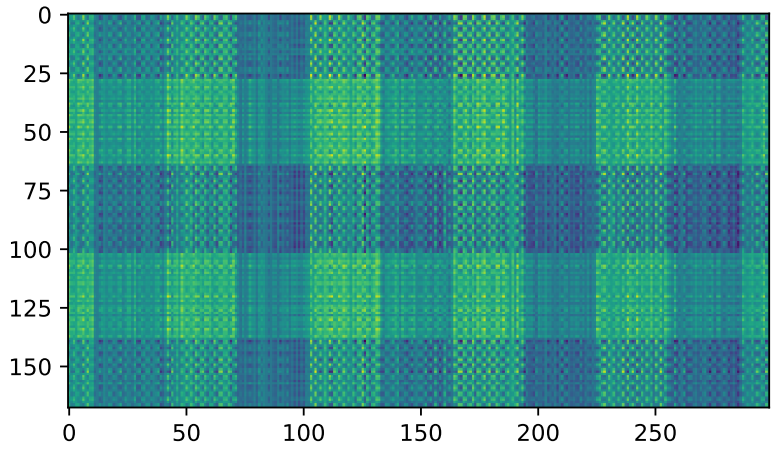

Try adding noise...

```
alpha = 10
R_noisy = R + np.random.normal(0, 10, R.shape)*alpha
plt.imshow(R_noisy)
```



Now clean it up with PCA:

```
pca = PCA(n_components=2)
pca.fit(R_noisy)
plt.imshow(pca.inverse_transform(pca.transform(R_noisy)))
```




## *SVD in higher dimensions*

### *Faces*

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)


# display a few of the faces, along with their names
fig, ax = plt.subplots(3, 4)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])

print(f'The shape of the faces dataset is {faces.images.shape}')
```

```
The shape of the faces dataset is (1348, 62, 47)
```


*pause*

### *PCA on faces*

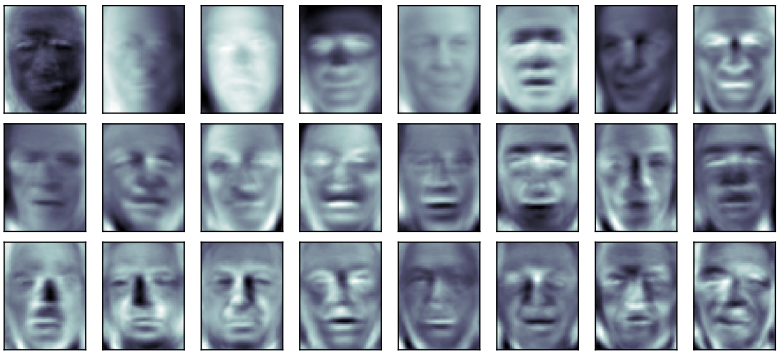
```
pca = PCA(150, svd_solver='randomized', random_state=42)
pca_small = PCA(10, svd_solver='randomized', random_state=42)
pca_very_small = PCA(2, svd_solver='randomized', random_state=42)
pca.fit(faces.data)
pca_small.fit(faces.data)
pca_very_small.fit(faces.data)
```

```
PCA(n_components=2, random_state=42, svd_solver='randomized')
```

. . .

This treatment from here

```
fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(pca.components_[i].reshape(62, 47), cmap='bone')
```



### *Reconstructions*

```
# Compute the components and projected faces
pca = pca.fit(faces.data)
components = pca.transform(faces.data)
projected = pca.inverse_transform(components)

components_small = pca_small.transform(faces.data)
projected_small = pca_small.inverse_transform(components_small)


components_very_small = pca_very_small.transform(faces.data)
projected_very_small = pca_very_small.inverse_transform(components_very_small)
```

```
# Plot the results
fig, ax = plt.subplots(4, 10, figsize=(10, 6.5),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47), cmap='binary_r')
    ax[1, i].imshow(projected_very_small[i].reshape(62, 47), cmap='binary_r')
    ax[2, i].imshow(projected_small[i].reshape(62, 47), cmap='binary_r')
    ax[3, i].imshow(projected[i].reshape(62, 47), cmap='binary_r')
```

```
ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('2-dim\nreconstruction');
ax[2, 0].set_ylabel('10-dim\nreconstruction');
ax[3, 0].set_ylabel('150-dim\nreconstruction');
```


Really cool demo of SVD image compression: https://timbaumann.info/svd-image-compression-demo/

### *Now you*

Code up your own image compression using SVD and show the left and right singular vectors, the singular values, and the reconstructed images.

Share with the class!