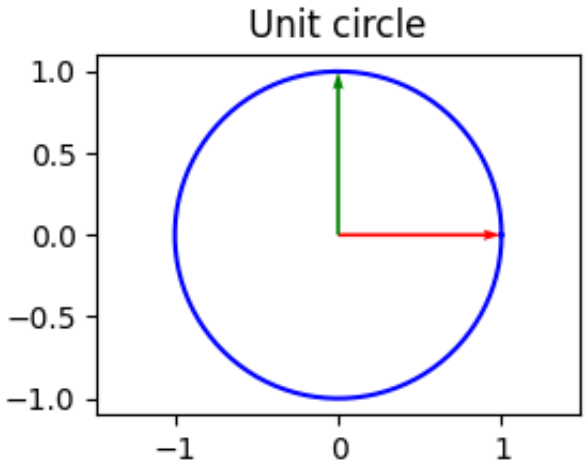
```
# Project 14: Singular Value Decomposition and image compression
import numpy as np
import matplotlib.pyplot as plt


print("\nSubtask 1\n")
# Task 1: Plotting the unit circle and basis vectors
t = np.linspace(0, 2 * np.pi, 100)
X = np.array([np.cos(t), np.sin(t)])
plt.subplot(2, 2, 1)
plt.plot(X[0, :], X[1, :], 'b')
plt.quiver(0, 0, 1, 0, color='r', angles='xy', scale_units='xy', scale=1)
plt.quiver(0, 0, 0, 1, color='g', angles='xy', scale_units='xy', scale=1)
plt.axis('equal')
plt.title('Unit circle')
plt.show()
```

```
Subtask 1
```



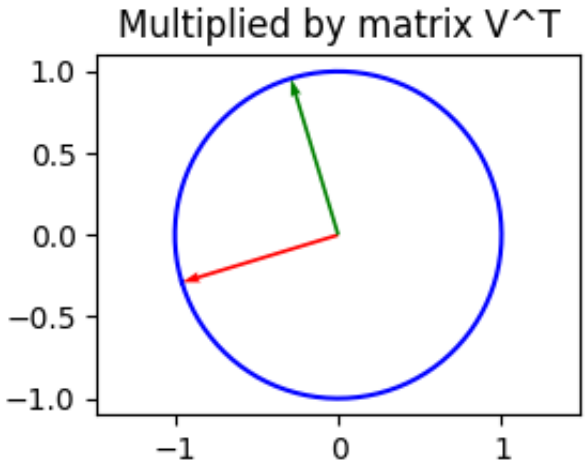
```
print("\nSubtask 2\n")
A = np.array([[2, 1], [-1, 1]])
U, S, V = np.linalg.svd(A)
print("U:\n", U)
print("S:\n", S)
print("V:\n", V)
# Verify orthogonality
print("U' * U:\n", np.dot(U.T, U))
print("V' * V:\n", np.dot(V.T, V))
```

```
Subtask 2

U:
 [[-0.95709203  0.28978415]
 [ 0.28978415  0.95709203]]
S:
 [2.30277564 1.30277564]
V:
 [[-0.95709203 -0.28978415]
 [-0.28978415  0.95709203]]
U' * U:
 [[1.00000000e+00 1.77671996e-17]
 [1.77671996e-17 1.00000000e+00]]
V' * V:
 [[ 1.00000000e+00 -2.42191841e-17]
 [-2.42191841e-17  1.00000000e+00]]
```

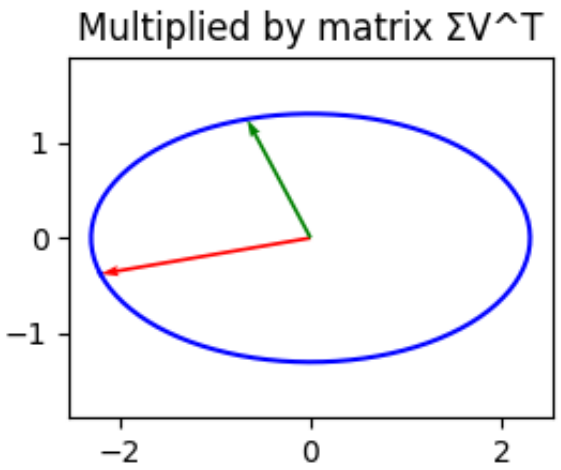
```
print("\nSubtask 3\n")
VX = np.dot(V.T, X)
plt.subplot(2, 2, 2)
plt.plot(VX[0, :], VX[1, :], 'b')
plt.quiver(0, 0, V[0, 0], V[0, 1], color='r', angles='xy', scale_units='xy', sc
plt.quiver(0, 0, V[1, 0], V[1, 1], color='g', angles='xy', scale_units='xy', sc
plt.axis('equal')
plt.title('Multiplied by matrix V^T')
plt.show()
```

```
Subtask 3
```



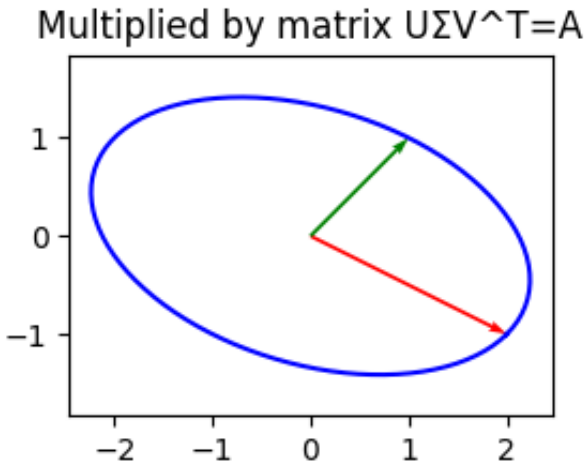
```
print("\nSubtask 4\n")
S_matrix = np.diag(S)
SVX = np.dot(S_matrix, VX)
plt.subplot(2, 2, 3)
plt.plot(SVX[0, :], SVX[1, :], 'b')
plt.quiver(0, 0, S[0] * V[0, 0], S[1] * V[0, 1], color='r', angles='xy', scale_
plt.quiver(0, 0, S[0] * V[1, 0], S[1] * V[1, 1], color='g', angles='xy', scale_
plt.axis('equal')
plt.title('Multiplied by matrix ΣV^T')
plt.show()
```

```
Subtask 4
```



```
print("\nSubtask 5\n")
AX = np.dot(U, SVX)
plt.subplot(2, 2, 4)
plt.plot(AX[0, :], AX[1, :], 'b')
plt.quiver(0, 0, U[0, 0] * S[0] * V[0, 0] + U[0, 1] * S[1] * V[0, 1],
U[1, 0] * S[0] * V[0, 0] + U[1, 1] * S[1] * V[0, 1], color='r', angles='xy',
scale_units='xy', scale=1)
plt.quiver(0, 0, U[0, 0] * S[0] * V[1, 0] + U[0, 1] * S[1] * V[1, 1],
U[1, 0] * S[0] * V[1, 0] + U[1, 1] * S[1] * V[1, 1], color='g', angles='xy',
scale_units='xy', scale=1)
plt.axis('equal')
plt.title('Multiplied by matrix UΣV^T=A')
plt.show()
```

```
Subtask 5
```




```
print("\nSubtask 6\n")
# Modification example for U and V (this is just a random example, modificatior
U1 = U
V1 = V.T
print("U1 * S * V1.T:\n", np.dot(U1, np.dot(S_matrix, V1.T)))
```

```
Subtask 6

U1 * S * V1.T:
 [[ 2.  1.]
 [-1.  1.]]
```

```
print("\nSubtask 7\n")
Av1 = np.dot(A, V.T[:, 0])
Av2 = np.dot(A, V.T[:, 1])
print("Av1:\n", Av1)
print("σ1 * u1:\n", S[0] * U[:, 0])
print("Av2:\n", Av2)
print("σ2 * u2:\n", S[1] * U[:, 1])
# Numerical check
print("A * V - U * S:\n", np.dot(A, V.T) - np.dot(U, S_matrix))
```

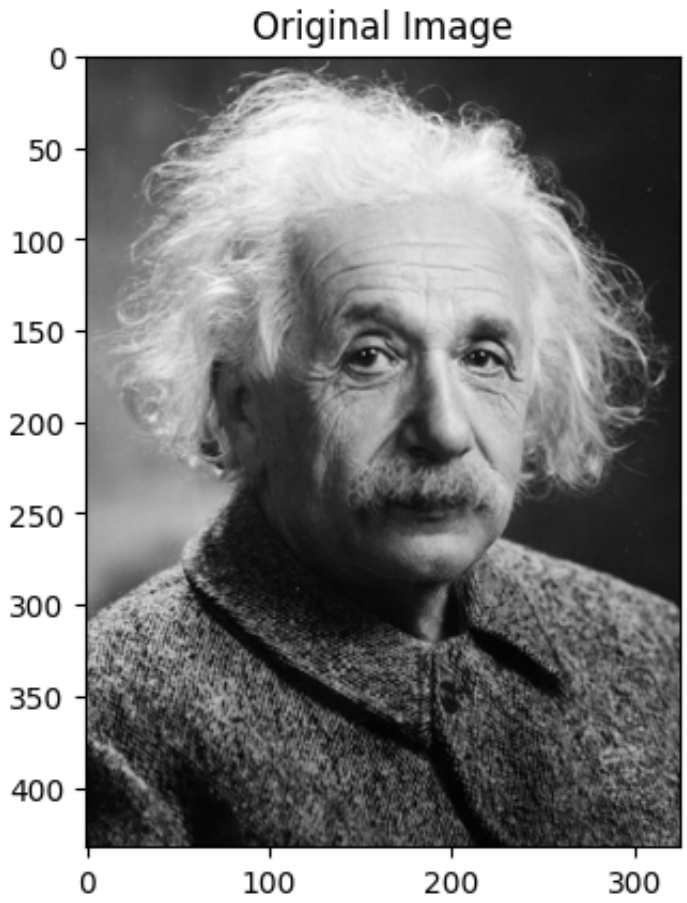
```
Subtask 7

Av1:
 [-2.2039682   0.66730788]
σ1 * u1:
 [-2.2039682   0.66730788]
Av2:
 [0.37752373 1.24687618]
σ2 * u2:
 [0.37752373 1.24687618]
A * V - U * S:
 [[-8.88178420e-16  2.77555756e-16]
 [ 3.33066907e-16  2.22044605e-16]]
```

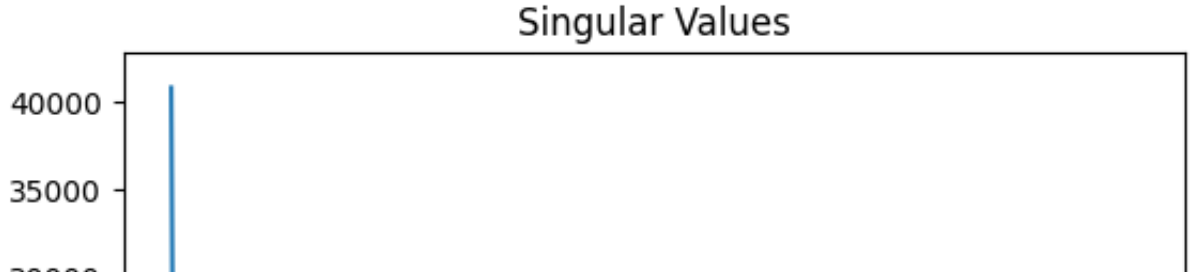
```
print("\nSubtask 8-11\n")
import cv2
# Load the image
ImJPG = cv2.imread("/content/einstein.jpg",cv2.IMREAD_GRAYSCALE)
plt.figure()
plt.imshow(ImJPG, cmap='gray')
plt.title('Original Image')
plt.show()
# Singular Value Decomposition
UIm, SIm, VIm = np.linalg.svd(ImJPG.astype(np.float64), full_matrices=False)
print(UIm)
# Plot Singular Values
plt.figure()
plt.plot(np.arange(len(SIm)), SIm)
plt.title('Singular Values')
plt.show()
# Image compression using truncated SVD
for k in [50, 100, 150]:
    ImJPG_comp = np.dot(UIm[:, :k], np.dot(np.diag(SIm[:k]), VIm[:k, :]))
    plt.figure()
    plt.imshow(ImJPG_comp, cmap='gray')
    plt.title(f'Compressed Image with {k} Singular Values')
    plt.show()
```

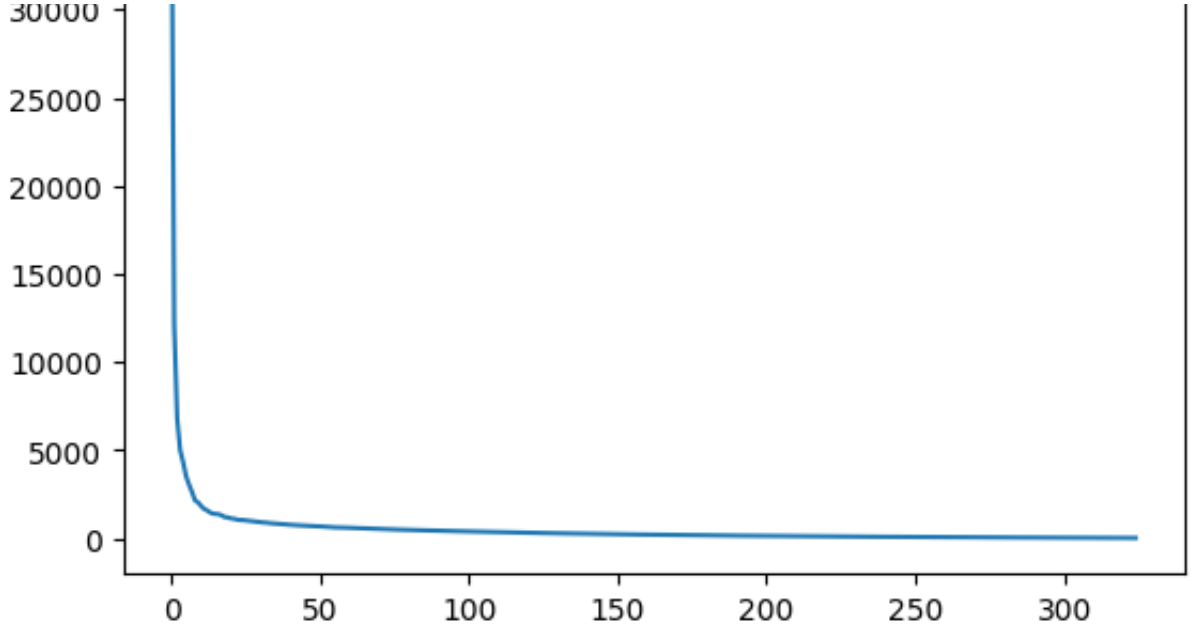
```
pct = 1 - (np.size(UIm[:, :k]) + np.size(VIm[:k, :]) * np.size(np.diag(SIm|
print(f'Compression percentage for {k} singular values: {pct:.3f}')
```

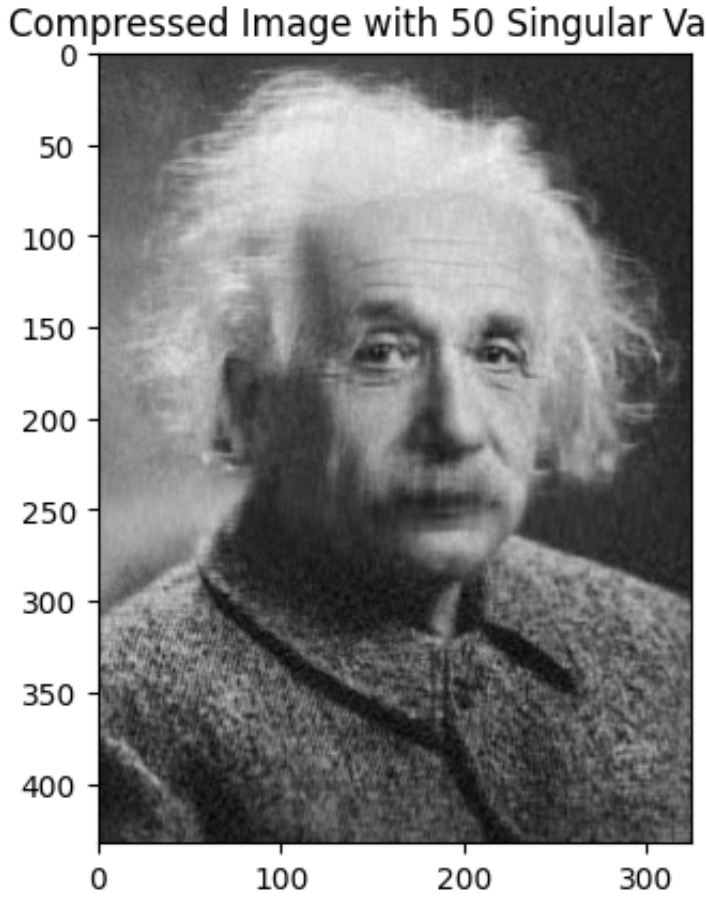
Subtask 8-11



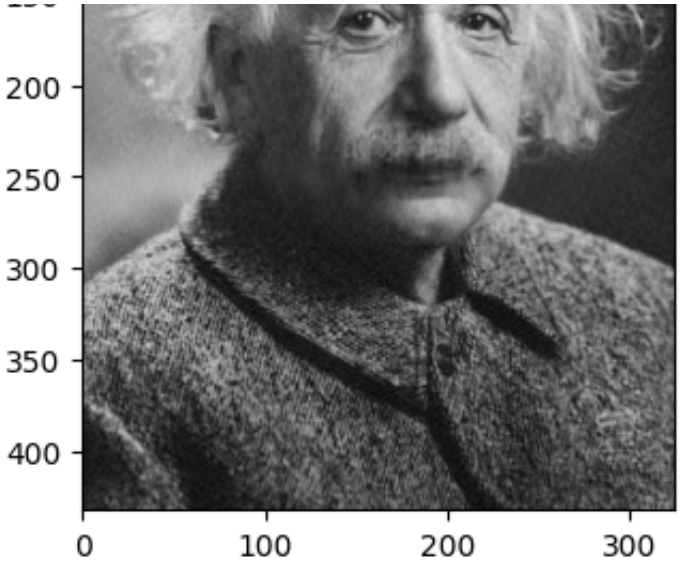

```
[[-0.01973233  0.00558642 -0.02123928 ... -0.00261544 -0.05072357
  -0.03953517]
 [-0.01978487  0.00571134 -0.02162803 ...  0.0073873  -0.03997558
  -0.03325746]
 [-0.01991589  0.00594026 -0.02212336 ... -0.00760312 -0.0560467
  -0.09563301]
 ...
 [-0.01950539 -0.00595175 -0.04791957 ...  0.00209835  0.06150357
   0.04754879]
 [-0.02000929 -0.00796643 -0.0507054  ... -0.012451   -0.0317809
  -0.05879769]
 [-0.02011501 -0.00876505 -0.05297971 ... -0.00155527 -0.01497408
   0.07202208]]
```



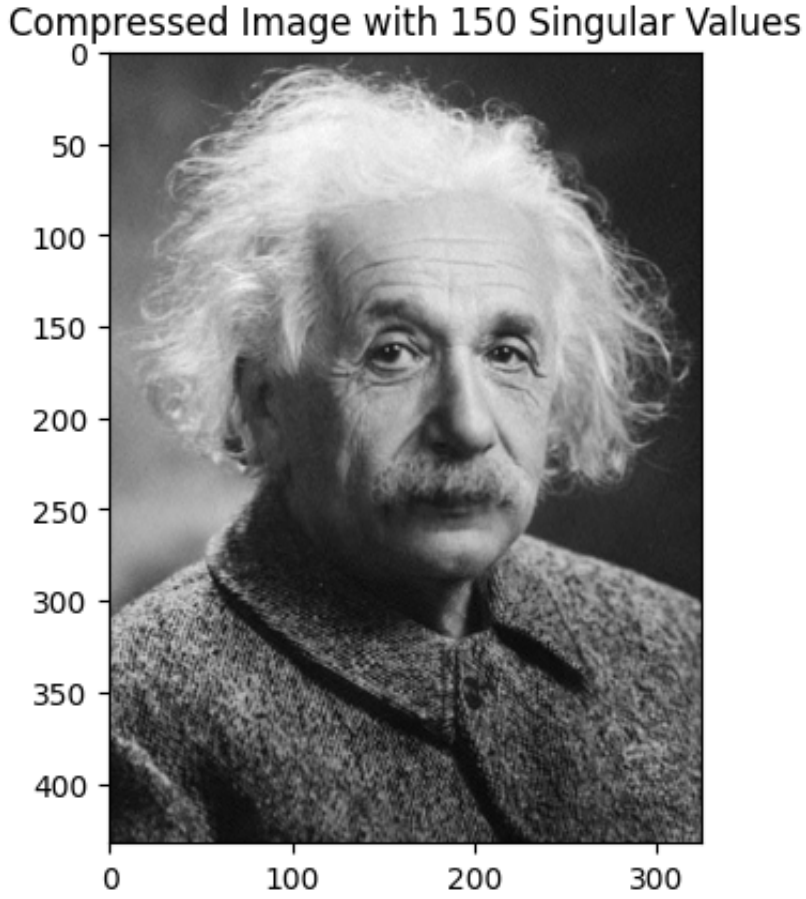





```
Compression percentage for 50 singular values: -287.837
```




```
Compression percentage for 100 singular values: -2308.777
```

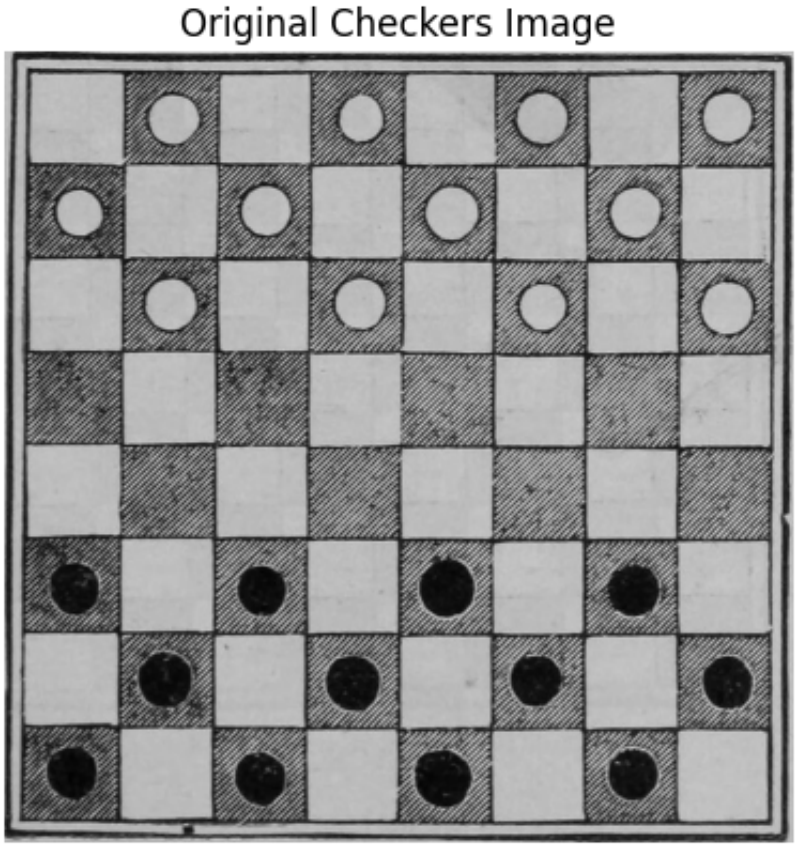


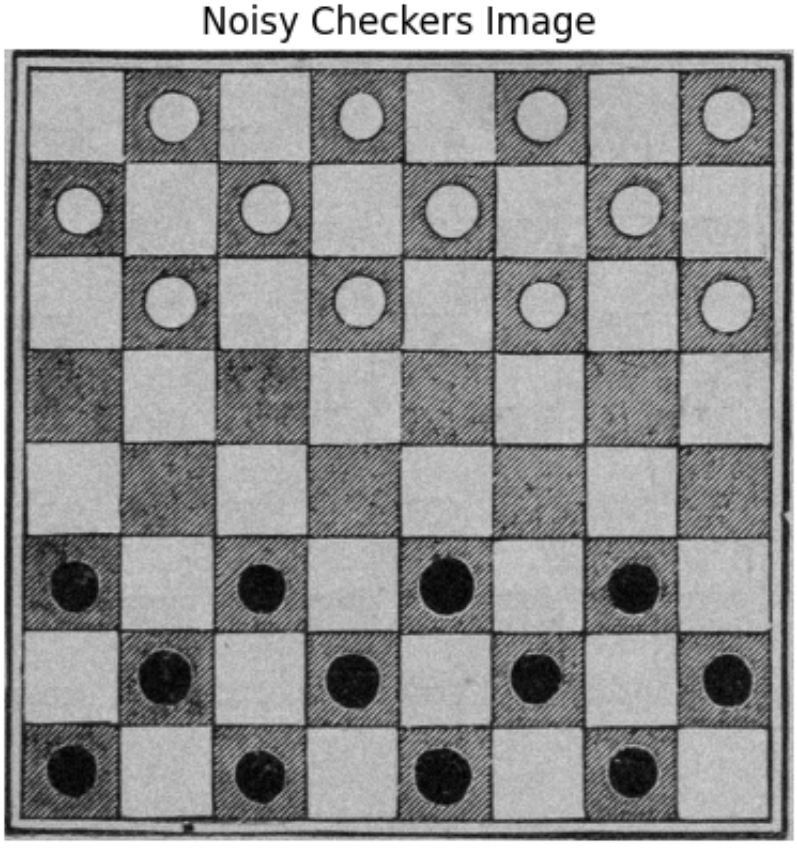

```
Compression percentage for 150 singular values: -7793.919
```

```
print("\nSubtask 12\n")
#12
import numpy as np
import matplotlib.pyplot as plt
import cv2
# Load the image
ImJPG = cv2.imread("/content/checkers.pgm",cv2.IMREAD_GRAYSCALE)
# Add noise to the image
```

```
# Add noise to the image
m, n = ImJPG.shape
ImJPG_Noisy = ImJPG.astype(np.float64) + 50 * (np.random.rand(m, n) - 0.5)
ImJPG_Noisy = np.clip(ImJPG_Noisy, 0, 255) # Ensure values are within valid rang
# Display the original and noisy images
plt.figure()
plt.imshow(ImJPG, cmap='gray')
plt.title('Original Checkers Image')
plt.axis('off')
plt.show()
plt.figure()
plt.imshow(ImJPG_Noisy, cmap='gray')
plt.title('Noisy Checkers Image')
plt.axis('off')
plt.show()
```

Subtask 12



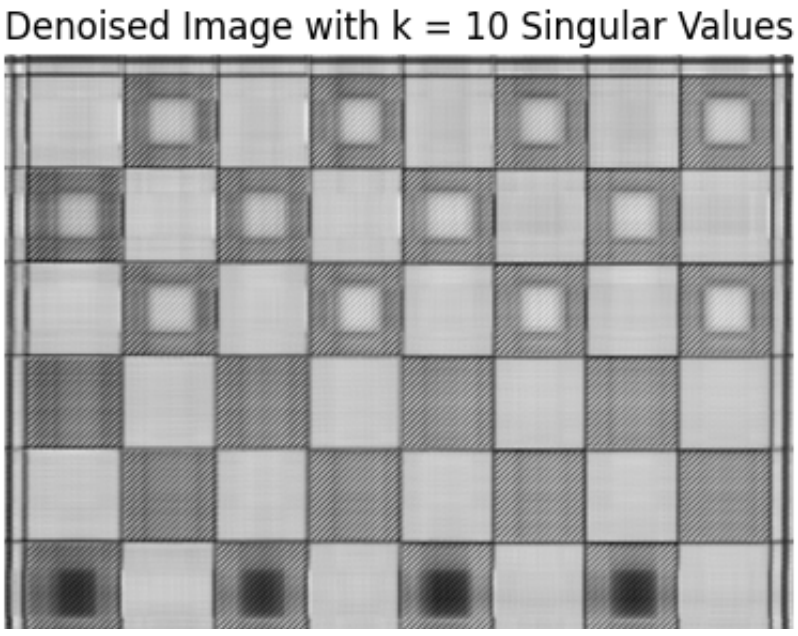


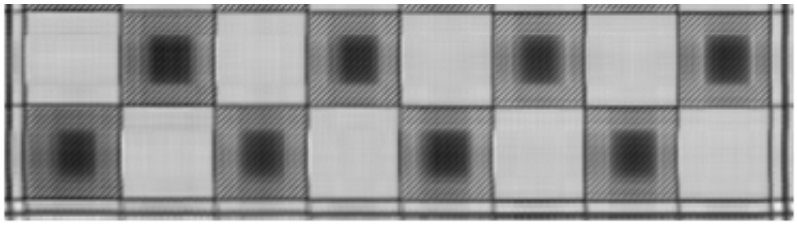
```
print("\nSubtask 13\n")
# Compute SVD of the noisy image
UIm, SIm, VIm = np.linalg.svd(ImJPG_Noisy, full_matrices=False)
# Variables: UIm, SIm, VIm
```

```
Subtask 13
```

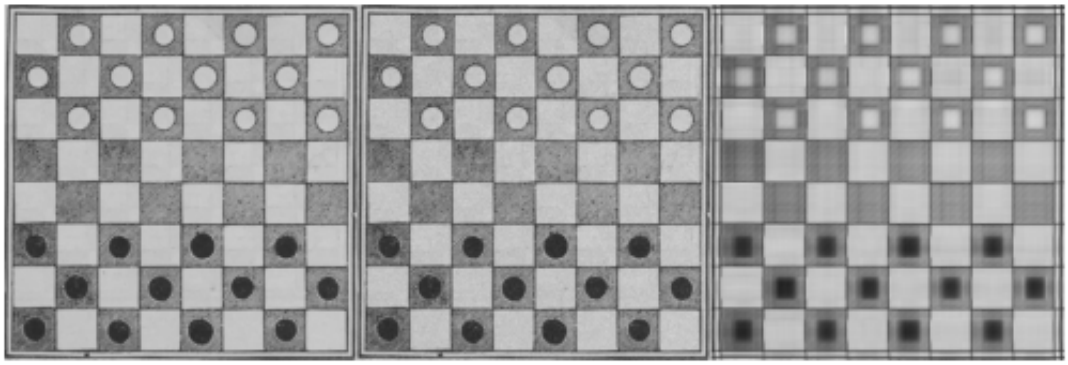
```
print("\nSubtask 14\n")
# Function to approximate the image with k singular values
def approximate_image(U, S, V, k):
    return np.dot(U[:, :k], np.dot(np.diag(S[:k]), V[:k, :]))
# Approximations with k = 10, k = 30, k = 50 singular values
ks = [10, 30, 50]
for k in ks:
    ImJPG_approx = approximate_image(UIm, SIm, VIm, k)
    plt.figure()
    plt.imshow(ImJPG_approx, cmap='gray')
    plt.title(f'Denoised Image with k = {k} Singular Values')
    plt.axis('off')
    plt.show()
    # Compare the images to the initial noisy image
    plt.figure()
    plt.imshow(np.hstack((ImJPG, ImJPG_Noisy, ImJPG_approx)), cmap='gray')
    plt.title(f'Original, Noisy, and Denoised (k = {k}) Images')
    plt.axis('off')
    plt.show()
```

```
Subtask 14
```

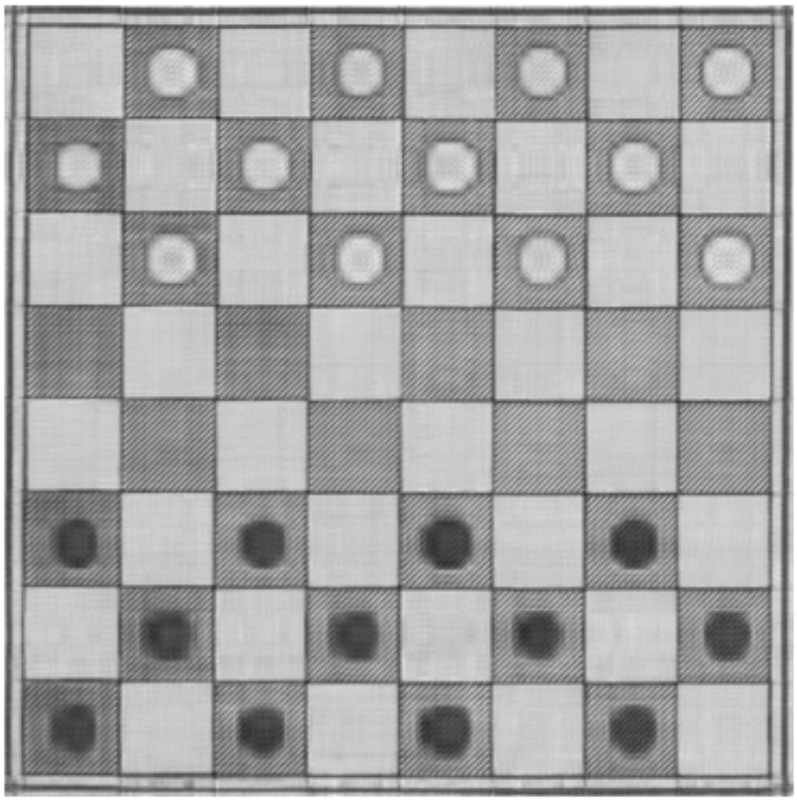




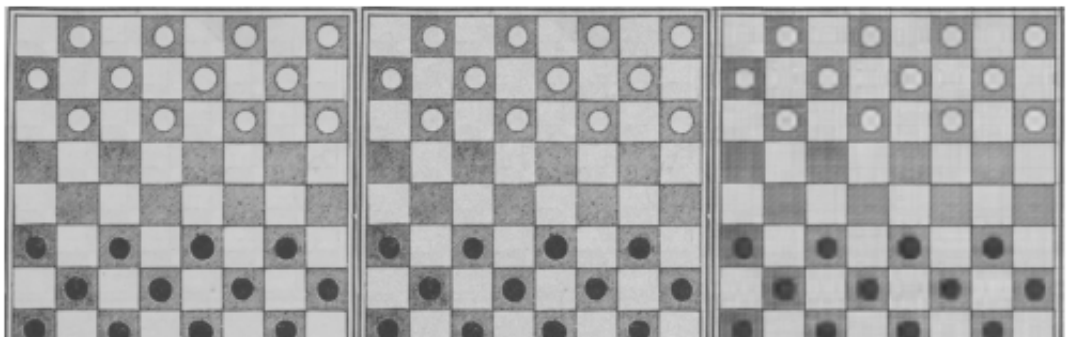
Original, Noisy, and Denoised (k = 10) Images



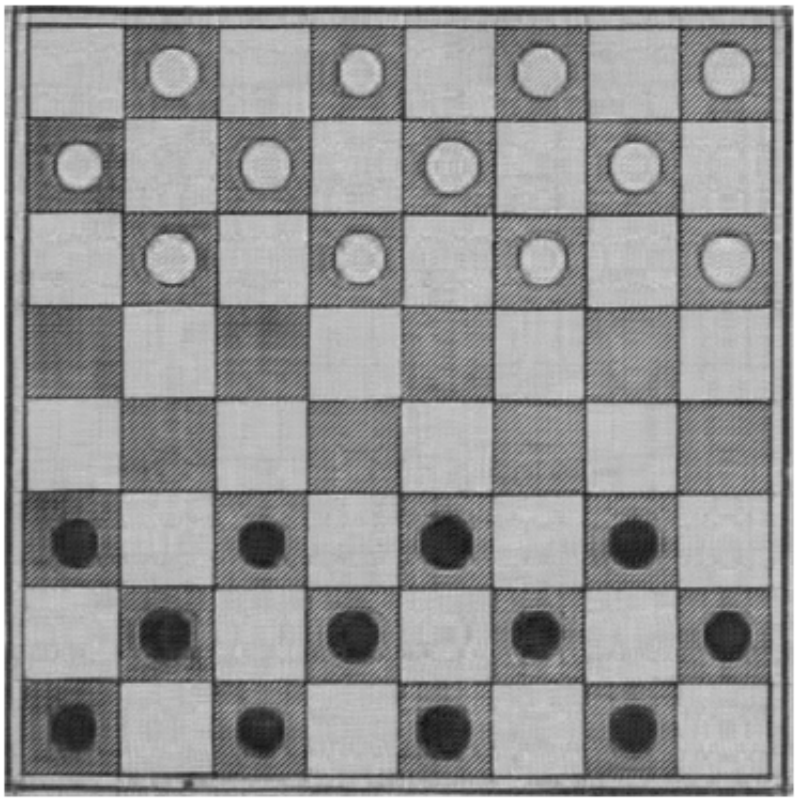
Denoised Image with k = 30 Singular Values



Original, Noisy, and Denoised (k = 30) Images



Denoised Image with k = 50 Singular Values



Original, Noisy, and Denoised (k = 50) Images