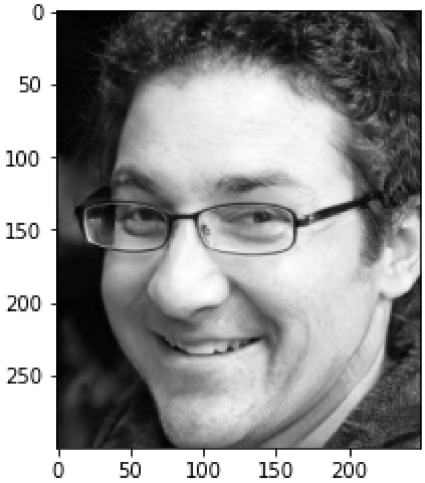
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline


from PIL import Image
from io import BytesIO
import requests


url = "https://bgr.com/wp-content/uploads/2019/02/download-1.jpeg?quality=98&strip=all&w=300"
response = requests.get(url)
img = Image.open(BytesIO(response.content))
img = np.array(img)
# convert to grayscale
img = img[:, :, 0] * 0.2989 + img[:, :, 1] * 0.5870 + img[:, :, 2] * 0.1140
# Crop the image
img = img[:, 50:]
# Normalize pixel values
img = img / np.max(np.max(img))
print("Size of image:", img.shape)
```

```
Size of image: (300, 250)
```

```
plt.imshow(1 - img, cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x7f79cd7c0210>
```




```
np.linalg.matrix_rank(img)
```

```
250
```

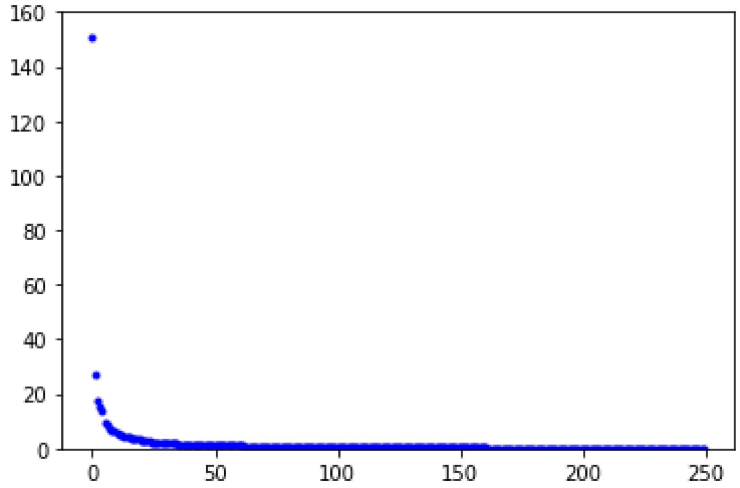
```
# Perform singular value decomposition in NumPy
U, diag, Vt = np.linalg.svd(img)
```

```
print(U.shape)
print(diag.shape)
print(Vt.shape)
print(diag[:10])
print(diag[-10:])
```

```
(300, 300)
(250,)
(250, 250)
[150.97801226  26.71522236  17.48466668  15.24428641  14.08614048
   9.59730112   8.44389932   7.48559835   6.71709645   6.26468411]
[0.03409647 0.03251174 0.02917226 0.02908885 0.02668024 0.02556615
 0.02246924 0.02120552 0.01778257 0.01560985]
```

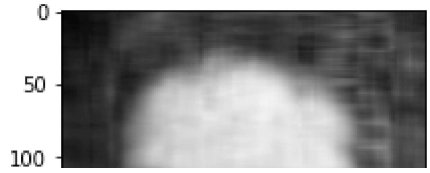
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(diag, 'b.')
plt.ylim([0, 160])
```

```
(0.0, 160.0)
```



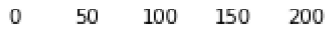

```
k = 10
Ur = U[:, :k]
Vtr = Vt[:k, :]
Sr = np.diag(diag[:k])
img_r = Ur.dot(Sr).dot(Vtr)
plt.imshow(1 - img_r, cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x7f79c51de310>
```




```
print(Ur.shape)
print(Sr.shape)
print(Vtr.shape)
```

```
(300, 10)
(10, 10)
(10, 250)
```



CUR Decomposition

```
probs = []
for i in range(img.shape[1]):
    prob = np.sum(img[:, i] ** 2)
    probs.append(prob)
probs = np.array(probs) / np.sum(probs)
print(np.sum(probs))
print(probs)
```

```
1.0
[0.00023228 0.00022633 0.00022818 0.00023194 0.00024099 0.00025282
 0.00026271 0.00027457 0.00028241 0.00029053 0.00029362 0.00029152
 0.00026663 0.0002477  0.00023525 0.00021223 0.00018336 0.00016248
 0.00016084 0.00034331 0.00039509 0.00039065 0.00037192 0.00033332
 0.00030679 0.00027655 0.00024402 0.00024186 0.00023941 0.00025167
 0.00023785 0.00030435 0.00041119 0.00049716 0.00055156 0.00059498
 0.00062708 0.000678   0.00073968 0.00083653 0.00103043 0.0011789
 0.00126962 0.00136562 0.00145543 0.0015155  0.00155242 0.00158404
 0.00164529 0.00168584 0.00172874 0.00177713 0.00182412 0.00187359
 0.00193633 0.00197305 0.00199845 0.00203135 0.00207311 0.0021032
 0.00217867 0.00225907 0.00237154 0.00249885 0.00263634 0.00274183
 0.00284584 0.00297155 0.00303985 0.00316395 0.00329109 0.00341552
 0.00353103 0.0036222  0.00371592 0.00382601 0.00392058 0.00413981
 0.0043619  0.00454773 0.00477645 0.00497843 0.00519517 0.00535614
 0.005548   0.00570843 0.00577943 0.00588565 0.00597914 0.00605778
 0.00615846 0.00623782 0.00628136 0.00628122 0.00621924 0.00624042
 0.00621281 0.00634943 0.00647939 0.00648017 0.00651161 0.00658255
 0.00667489 0.006689   0.0067782  0.00684794 0.00684389 0.00680737
 0.00678398 0.00681415 0.00682887 0.00682075 0.00681034 0.00678258
 0.00675286 0.00670926 0.00659406 0.00649963 0.0064098  0.00633747
 0.00629757 0.00631274 0.0062785  0.00626429 0.00623033 0.00612401
 0.00608747 0.00609884 0.00613246 0.00613001 0.00615142 0.00614228
 0.00613588 0.0061226  0.00609593 0.00611711 0.00609742 0.00609636
 0.00605579 0.00606175 0.00604206 0.00597326 0.00592933 0.00583342
 0.00583559 0.00579833 0.00577815 0.00583095 0.00584282 0.00592802
 0.00598799 0.00601674 0.00609032 0.00617444 0.00626175 0.00629974
```

```
 0.00632881 0.00637816 0.00645147 0.00651009 0.00654702 0.00649141
 0.0062393  0.00630082 0.00651287 0.00670675 0.0067953  0.00683943
 0.00683857 0.00687365 0.00688517 0.0068129  0.00683502 0.00695751
 0.00703209 0.00696249 0.00692711 0.00691764 0.00688569 0.00686568
 0.00686796 0.00679282 0.00669207 0.00661916 0.00656152 0.00646794
 0.00635671 0.00625307 0.00616493 0.00609875 0.00604839 0.00601532
 0.00589495 0.00581532 0.00572679 0.00558283 0.00544308 0.00529861
 0.00518723 0.00511383 0.00500343 0.00482854 0.00467723 0.00452478
 0.0043619  0.00419372 0.00397722 0.00375655 0.00355304 0.00339876
 0.0032258  0.00308977 0.00299627 0.00292165 0.00288033 0.00294058
 0.0029065  0.00289657 0.00295547 0.00294885 0.00291002 0.00287179
 0.00286182 0.00276502 0.00262522 0.00247387 0.00232666 0.00213854
 0.00191852 0.00180844 0.00177916 0.00175884 0.00180936 0.0018577
 0.00185069 0.00193167 0.00194121 0.00209995 0.00202385 0.00197479
 0.00196359 0.00192321 0.00197022 0.00201238 0.00199324 0.00199721
 0.00198737 0.00193365 0.00184399 0.00184426]
```

```
c = 50
samples_c = np.random.choice(np.arange(img.shape[1]), size=c, p=probs)
print(samples_c)
```

```
[120 166  48 236  83 122 249 187 187 209   7  84 179 129 208 245 106 138
 104 103 225  52 212  42 173 169 150 108  96 124  85 234 108 115  79  87
 195 151 136 137 101 168 199 203  95  58 188 154 172 152]
```

```
C = np.empty([300, 0])
for sample in samples_c:
    C = np.hstack([C, img[:, [sample]] / np.sqrt(c * probs[sample])])
print(C)
```

```
[[0.7180772  0.36724284 0.09740221 ... 0.84119844 0.25113511 0.57243544]
 [0.50041636 0.55500443 0.12495234 ... 0.32524419 0.15526764 0.32184613]
 [0.57062958 0.17410227 0.12495234 ... 0.53154722 0.09348353 0.20729102]
 ...
 [0.64102025 0.86033773 0.42418356 ... 0.89624238 0.99241936 0.95931672]
 [0.53230355 0.90100654 0.42433235 ... 0.90169332 0.95862732 0.97363611]
 [0.47961054 0.92384775 0.39678222 ... 0.92541149 0.93794588 0.98846317]]
```

```
C.shape
```

```
(300, 50)
```

Homework

- Describe the algorithm that samples rows.
- Apply the algorithm to create 50 samples of row index `samples_r` and the row-matrix `R`.
- Execute the remaining code. The program should generate an image with reasonable quality.

```
# Your Code Here
```

```
probs = []
for i in range(img.shape[0]):
    prob = np.sum(img[i, :] ** 2)
    probs.append(prob)
probs = np.array(probs) / np.sum(probs)
print(np.sum(probs))
print(probs)
```

```
1.0
[0.00062855 0.00054854 0.00047686 0.00038523 0.00034289 0.00032666
 0.00030981 0.0003018  0.00028153 0.00032684 0.00036922 0.00032754
 0.00041489 0.00041231 0.00043763 0.00047071 0.00042218 0.00037594
 0.00038036 0.00034603 0.0003045  0.00034407 0.00036354 0.00036919
 0.00035398 0.00038169 0.00042679 0.00049681 0.00053874 0.00048897
 0.00054207 0.00062634 0.00070761 0.00083662 0.00088511 0.00098549
 0.00111457 0.00119358 0.0013341  0.00148251 0.00159986 0.00170052
 0.00183331 0.00201802 0.00216035 0.00230898 0.00253877 0.00274944
 0.0029127  0.00306945 0.00320606 0.00328804 0.00340496 0.00351706
 0.00358768 0.00374376 0.00389536 0.00399645 0.00407891 0.00416349
 0.00420628 0.00424428 0.0043209  0.00441275 0.00455369 0.00463208
 0.00467443 0.00468433 0.00471927 0.00472432 0.00476015 0.0047996
 0.00481834 0.00484992 0.00487531 0.00492627 0.0049291  0.00494488
 0.00495303 0.00494654 0.00491748 0.00491419 0.00492604 0.00492489
 0.00491543 0.00490145 0.00493355 0.00498627 0.00502769 0.00502081
 0.00500244 0.00497285 0.00497644 0.00502356 0.00511899 0.00511371
 0.00507363 0.00512607 0.00512576 0.00512721 0.00514529 0.00514466
 0.00510365 0.00509286 0.00506441 0.00504112 0.00498744 0.00490223
 0.00479425 0.0046308  0.00445488 0.00430709 0.00408455 0.00382569
 0.0036387  0.00338796 0.003209   0.00308953 0.00295125 0.00283569
 0.00291605 0.00301109 0.00312235 0.00319626 0.00323707 0.0033012
 0.00334545 0.00333766 0.00333442 0.00331652 0.00323712 0.00303316
 0.00276062 0.00235157 0.00237984 0.00210435 0.00207618 0.00206462
 0.00194912 0.00186199 0.00173228 0.00185283 0.00206199 0.00234128
 0.00270546 0.00314853 0.00358293 0.00380686 0.0039671  0.00407482
 0.00402905 0.00403238 0.00403176 0.00399847 0.00407635 0.00421059
 0.00434637 0.00451817 0.0046553  0.00485654 0.00507141 0.00524474
 0.00546396 0.00552213 0.00520407 0.00515629 0.00525073 0.00515775
 0.00570435 0.00584435 0.00584761 0.00591521 0.00600327 0.00605288
 0.00616763 0.00624181 0.00630085 0.00631176 0.00631335 0.00625321
 0.00622736 0.00614965 0.00609528 0.00607617 0.00601178 0.00595794
 0.0059362  0.00587153 0.00576839 0.00569304 0.00556489 0.00541592
 0.00528504 0.00511232 0.00491024 0.0047344  0.00457489 0.00441224
 0.00429566 0.00417375 0.00406121 0.00392114 0.00376762 0.00352464
 0.00338609 0.00331677 0.00325882 0.00324681 0.0032333  0.00319862
 0.00314276 0.00311321 0.00311263 0.0031123  0.00312344 0.00312413
 0.00313586 0.00312535 0.00311084 0.0031202  0.00313187 0.0031371
 0.00309663 0.00300148 0.00279022 0.00255355 0.00251353 0.00259386
 0.00271084 0.00281817 0.00294992 0.00307438 0.00314753 0.00317159
 0.00315699 0.00307198 0.00308792 0.00335854 0.00355745 0.00370693
 0.00378067 0.00378109 0.00373928 0.00365333 0.00362967 0.00357103
 0.00346695 0.00335    0.00327659 0.00324054 0.00322993 0.00322679
 0.00320139 0.00315786 0.0031357  0.00309329 0.00313398 0.00315927
 0.00310392 0.00306552 0.00308391 0.00307447 0.00307098 0.00312517
```

```
       0.00314097 0.00315438 0.00317417 0.00318139 0.00318528 0.00315004
       0.00311159 0.00306711 0.00305129 0.00298408 0.00289927 0.00285415
       0.00280064 0.00270654 0.00259928 0.00255931 0.00246723 0.00242169
       0.00237048 0.00226718 0.00215531 0.00203705 0.00196973 0.00188756
       0.00180547 0.0017071  0.00158975 0.00151507 0.00143593 0.00137012
       0.00131106 0.00124336 0.00116771 0.00109826 0.00100865 0.00094456]
```

```
r = 50
samples_r = np.random.choice(np.arange(img.shape[0]), size=r, p=probs)
print(samples_r)
```

```
     [253  70 112  82 156 176 142  93  60 242 169 222 109  41 231  79 191  83
      143 263 258 149  77 179 105  38 168 115 159  99 174  54 203  85 160 181
      124 268 179 158  69 179  87 275  68 241 129  78 188 260]
```

```
R = np.empty([0, 250])
for sample in samples_r:
    R = np.vstack([R, img[[sample],:] / np.sqrt(r * probs[sample])])
print(R)
```

```
     [[0.0965653  0.11645133 0.13633736 ... 1.06990995 1.05002392 1.07985297]
      [0.06351553 0.06351553 0.07161402 ... 0.1866654  0.20052897 0.30647107]
      [0.69032428 0.62850595 0.58056268 ... 0.37542342 0.39960358 0.32291342]
      ...
      [0.08608427 0.07814503 0.08608427 ... 0.36482791 0.4119004  0.31188297]
      [0.09684023 0.09676077 0.089404   ... 0.36729774 0.37392833 0.30175414]
      [0.20062558 0.22074861 0.21068709 ... 1.04689483 1.21865405 1.3767393 ]]
```

```
R.shape
```

```
     (50, 250)
```

```
W = img[samples_r, :][:, samples_c]
W.shape
```

```
     (50, 50)
```

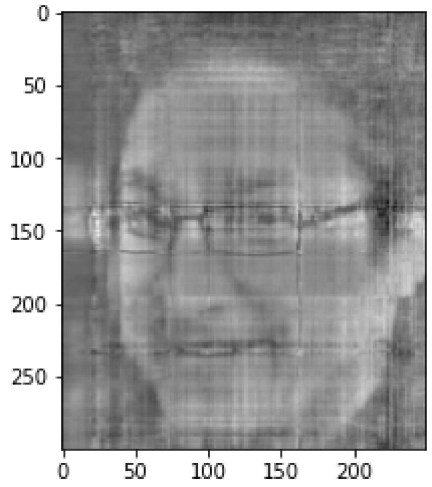
```
X, diag, Y = np.linalg.svd(W)
```

```
limit = 30
sigma = np.diag(np.concatenate([1 / diag[:limit], np.zeros(c - limit)]))
print(sigma.shape)
U = Y.T.dot(sigma).dot(X.T)
# U = Y.T.dot(np.diag(1 / diag)).dot(X.T)
# U = np.linalg.pinv(W)
print(U.shape)
```

```
     (50, 50)
     (50, 50)
```

```
img_cur = C.dot(U).dot(R)
plt.imshow(1 - img_cur, cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x7f79c51d0790>
```



✓ 0s completed at 9:22 PM