Carson Slater

6375 Homework Solutions

```
source(here::here("helpers.R"))
```

Question 1

I have read Trefethen and Bau Lectures 4-6.

Question 2

a.

Singular Value Decomposition, or SVD is a method to factor a matrix into three distinct matrices, each with different properties. A can define any matrix $\mathbf{A}$ to be decomposed into three matrices,

$$A = U\Sigma V^*.$$

When $\mathbf{A} \in \mathbb{C}^{m\times n}$ where $m \geq n$, $\mathbf{U}$ contains n left-singular vectors, which correspond to the n singular values, $\{\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0\}$ where $p = \min(m, n)$. These singular values comprise the diagonal matrix Σ. Additionally, $\mathbf{V}$ contains n right-singular vectors. All column vectors comprising $\mathbf{U}$ and $\mathbf{V}$ are such that $\mathbf{U}$ and $\mathbf{V}$ are unitary.

Reduced SVD is usually denoted as

$$A = \hat{U}\hat{\Sigma}V^*,$$

where there are n columns in $\hat{\mathbf{U}}$ that are orthonormal vectors, meaning the columns $u_{1:n} \in \mathbb{C}^m$. Unless $m = n$, $\mathbf{U}$ is not unitary. Full SVD alters $\hat{\mathbf{U}}$ by adjoining $m-n$ additional orthonormal columns to get $\mathbf{U} \in \mathbb{C}^{m\times m}$. Additionally, Full SVD appends $m - n$ zero vectors as rows, changing $\hat{\Sigma} \in \mathbb{R}^{n\times n}$ to $\Sigma \in \mathbb{R}^{m\times n}$.

b.

```
A <- matrix(1:24, nrow = 6, ncol = 4)

base::svd(A)
# $d
# [1] 6.988936e+01 3.934140e+00 2.439260e-15 7.988285e-16
#
# $u
#            [,1]        [,2]        [,3]        [,4]
```

```
# [1,] -0.3427168 -0.63745958  0.48536117  0.29647755
# [2,] -0.3680125 -0.39975602 -0.32347263  0.06224984
# [3,] -0.3933083 -0.16205245 -0.69147759 -0.20176071
# [4,] -0.4186040  0.07565111  0.35897775 -0.28204130
# [5,] -0.4438997  0.31335467  0.22356194 -0.56202233
# [6,] -0.4691954  0.55105823 -0.05295064  0.68709697
#
# $v
#            [,1]        [,2]       [,3]       [,4]
# [1,] -0.1283136  0.82676213  0.4710363  0.2795082
# [2,] -0.3374215  0.43144724 -0.4197153 -0.7237673
# [3,] -0.5465295  0.03613234 -0.5736783  0.6090100
# [4,] -0.7556374 -0.35918256  0.5223573 -0.1647509
```

c.

Here, `base::svd(A)$d` ($\hat{\Sigma}$) is a 1×4 array of four numbers, because it is only storing the diagonal values of matrix (Σ) where `base::svd(A)$u` ($\hat{\mathbf{U}}$) is a 6×4 array, and `base::svd(A)$d`($\mathbf{V}$) is a 4×4 array.

d.

```
with(base::svd(A),
     zapsmall(u %*% diag(d) %*% t(v))
)
#      [,1] [,2] [,3] [,4]
# [1,]    1    7   13   19
# [2,]    2    8   14   20
# [3,]    3    9   15   21
# [4,]    4   10   16   22
# [5,]    5   11   17   23
# [6,]    6   12   18   24
```

e.

Matrix $\mathbf{A}$ has a rank of 2, as rank would be the number of nonzero singular values in the Σ matrix along the diagonal.

f.

- The full version of the SVD has three matrices, $\mathbf{U} \in \mathbb{C}^{m \times m}$ unitary, $\mathbf{V} \in \mathbb{C}^{n \times n}$ unitary, and $\Sigma \in \mathbb{R}^{m \times n}$ diagonal with positive entries.
- Per Trefethen and Bau, the reduced SVD has $\mathbf{U} \in \mathbb{C}^{m \times n}$ unitary, $\mathbf{V} \in \mathbb{C}^{n \times n}$ unitary, and $\Sigma \in \mathbb{R}^{n \times n}$ diagonal with positive entries.
- From earlier, we say that `base::svd(A)` where $\mathbf{A} \in \mathbb{C}^{6 \times 4}$ produced a $\mathbf{U}$ matrix that had 6 rows and 4 columns. This is not a square matrix, and thus we can conclude that `base::svd()` computes the SVD using reduced SVD.

Question 3

a.

```r
matrix_rank <- function(A) {
  if (is.matrix(A) == FALSE) {
    stop("`matrix_rank()` only accepts an array or matrix object.")
  } else {
    n_nonzero(base::svd(A)$d)
  }
}
```

```r
# Testing matrix_rank
B <- matrix(1:4, nrow = 2)
matrix_rank(B)
# [1] 2

(C <- matrix(1:9, nrow = 3))
#      [,1] [,2] [,3]
# [1,]    1    4    7
# [2,]    2    5    8
# [3,]    3    6    9
matrix_rank(C)
# [1] 2

set.seed(1)
rnorm(10*5) |>
  matrix(nrow = 10, ncol = 5) |>
  matrix_rank()
# [1] 5
```

b.

The column space, or range($\mathbf{A}$) is the set of vectors that $\mathbf{A}$ can map non-zero vectors to. Using the $\mathbf{A}x = b$ example, the column space would be any column vector $b \in \mathbb{C}^m$ for which x is transformed by $\mathbf{A}$ where $\mathbf{A}x \neq 0$. Therefore, the column space is a subset of $\mathbb{C}^m$.

c.

```r
col_space_basis <- function(A) {
  if (is.matrix(A) == FALSE) {
    stop("`col_space_basis()` only accepts an array or matrix object.")
  } else {
    base::svd(A)$u[,seq(matrix_rank(A))]
  }
}
```

```r
# Testing Column Space of A
col_space_basis(A) |> round(2)
#       [,1]  [,2]
# [1,] -0.34 -0.64
# [2,] -0.37 -0.40
# [3,] -0.39 -0.16
# [4,] -0.42  0.08
# [5,] -0.44  0.31
# [6,] -0.47  0.55

col_space_basis(B) |> round(2)
#       [,1]  [,2]
# [1,] -0.58 -0.82
# [2,] -0.82  0.58

col_space_basis(C) |> round(2)
#       [,1]  [,2]
# [1,] -0.48  0.78
# [2,] -0.57  0.08
# [3,] -0.67 -0.63
```

d.

The null space, or null($\mathbf{A}$) is the set of vectors that $\mathbf{A}$ maps to the zero vector, 0_m. Using the $\mathbf{A}x = b$ example, the null space would be any row vector $x \in \mathbb{C}^n$ for which $\mathbf{A}x = 0$. Therefore, the null space is a subset of $\mathbb{C}^n$.

e.

```
null_space_basis <- function(A) {
  if (is.matrix(A) == FALSE) {
    stop("`null_space_basis()` only accepts an array or matrix object.")
  } else {
    base::svd(A)$v[,-seq(matrix_rank(A))]
  }
}
```

```
# Testing null_space_basis

null_space_basis(A) |> round(2)
#       [,1]  [,2]
# [1,]  0.47  0.28
# [2,] -0.42 -0.72
# [3,] -0.57  0.61
# [4,]  0.52 -0.16

A %*% null_space_basis(A)
#                [,1]         [,2]
# [1,] -1.776357e-15 1.776357e-15
# [2,]  0.000000e+00 2.664535e-15
# [3,]  0.000000e+00 3.996803e-15
# [4,]  1.776357e-15 5.329071e-15
# [5,]  3.552714e-15 5.329071e-15
# [6,]  5.329071e-15 4.884981e-15

(null_space_basis(B) |> round(2)) # = matrix(ncol = 0, nrow = m)
#
# [1,]
# [2,]
```

f.

```r
# Got help from Jacob Moore for this result (3f)

# Finding ||B||
t <- seq(0, 2*pi, length = 1000)

vectors <- matrix(c(cos(t), sin(t)), byrow = TRUE, nrow = 2)

(B %*% vectors) |> apply(MARGIN = 2, norm) |> max()
# [1] 5.464986

# Checking to see if answers are the same as the B
max(base::svd(B)$d)
# [1] 5.464986
```

g.

```r
B <- matrix(1:4, nrow = 2)

# Computing the Frobenious Norm
mat_norm(B, "f")
# [1] 5.477226

# Computing Frobemious Norm Based on Theorem  5.3
(base::svd(B)$d)^2 |> sum() |> sqrt()
# [1] 5.477226
```

h.

```r
# Illustrating Theorem 5.4 on Matrix A
e_vals <- eigen(crossprod(A))$values
e_vals[is.nonzero(e_vals)] |> zapsmall() |> sqrt()
# [1] 69.889363  3.934082
```

i.

```r
D <- matrix(c(9, 4, 2, 4, 5, 1, 2, 1, 1), nrow = 3)
# Computing Singular Values Using Theorem 5.5
```

```r
eigen(D)$values |> abs()
# [1] 11.9262377  2.5470415  0.5267208

# Checking Results
base::svd(D)$d
# [1] 11.9262377  2.5470415  0.5267208
```

j.

```r
# Computing Determinant of D
det(D)
# [1] 16

# Product of Singular Values
prod(base::svd(D)$d)
# [1] 16
```

k.

```r
low_rank <- function(A, nu) {
  if (nu > matrix_rank(A)) {
    stop("'nu' must be less than or equal to the rank of matrix A.")
  } else {
    SVD <- base::svd(A)
    SVD$d[(nu + 1):matrix_rank(A)] <- 0
    with(SVD,
        zapsmall(u %*% diag(d) %*% t(v)))
  }
}
```

```r
# Testing low_rank
A |> low_rank(1) |> round(2)
#      [,1]  [,2]  [,3]  [,4]
# [1,] 3.07  8.08 13.09 18.10
# [2,] 3.30  8.68 14.06 19.44
# [3,] 3.53  9.28 15.02 20.77
# [4,] 3.75  9.87 15.99 22.11
# [5,] 3.98 10.47 16.96 23.44
# [6,] 4.21 11.06 17.92 24.78
```

I.

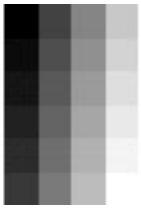

Figure 1: Plot of matrix A.

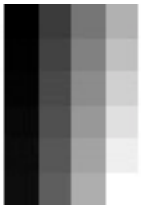

Figure 2: Plot of rank 1 approximation of matrix A.

Going off of the plots, the rank 1 approximation is actually a fairly good approximation of the original matrix.

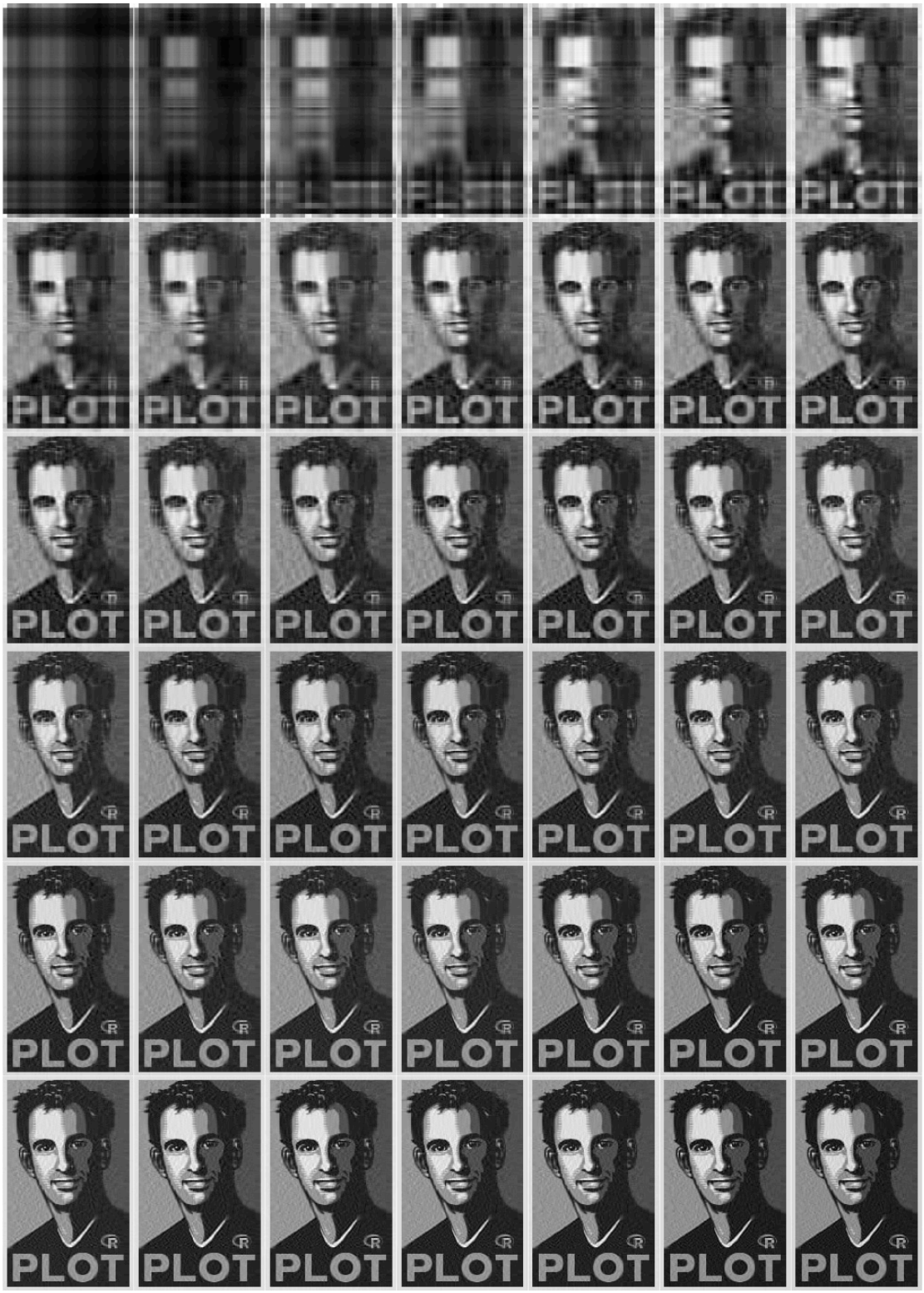

Figure 3: Low rank approximations for `Hadley.jpg`, for $\nu_{\in\mathbb{Z}} \in [1, 42]$.

m.

`Hadley.jpeg` is transformed into a matrix with 720×438 dimensions. This means that 315,360 numbers represent and communicate this image to others. Low-rank approximations up to $\nu = 42$ can severely reduce the amount of numbers needed to represent this image. To find the optimal ν rank approximation, we can see how much each rank reduces the size of the image.

Rank	Count of Numbers for Low Rank Approximation	Proportion of Size Reduced from Original
1	1158	0.9963
2	2316	0.9927
3	3474	0.9890
4	4632	0.9853
5	5790	0.9816
6	6948	0.9780
7	8106	0.9743
8	9264	0.9706
9	10422	0.9670
10	11580	0.9633
11	12738	0.9596
12	13896	0.9559
13	15054	0.9523
14	16212	0.9486
15	17370	0.9449
16	18528	0.9412
17	19686	0.9376
18	20844	0.9339
19	22002	0.9302
20	23160	0.9266
21	24318	0.9229
22	25476	0.9192
23	26634	0.9155
24	27792	0.9119
25	28950	0.9082
26	30108	0.9045
27	31266	0.9009
28	32424	0.8972
29	33582	0.8935
30	34740	0.8898
31	35898	0.8862
32	37056	0.8825
33	38214	0.8788
34	39372	0.8752
35	40530	0.8715
36	41688	0.8678
37	42846	0.8641
38	44004	0.8605
39	45162	0.8568
40	46320	0.8531
41	47478	0.8494
42	48636	0.8458

We can see that for an increase in ν, the proportion of size reduced shrinks approximately 0.367%. So then subjectively, I would argue that an optimal rank with decent image quality is $\nu = 24$. This is only because it is the best balance between size and image quality. Clear

distinctions like the 'PLOT' letters are visible in lower-rank approximations with $\nu < 24$, but image clarity seems to be the more acceptable at $\nu = 24$ given we are trying to optimize.