Corey Schulz
CS 3200 – Spring 2020
April 20, 2020

Homework 6 Report
Least Squares and Image Processing

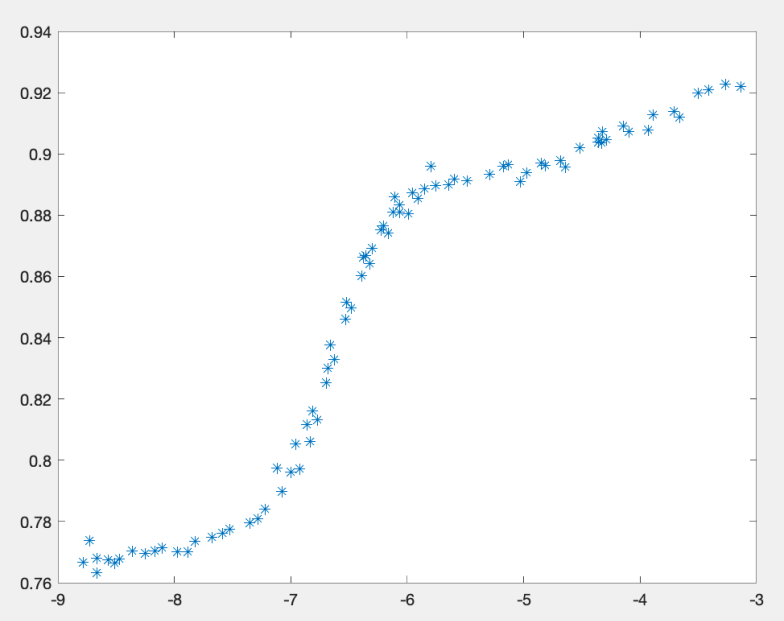
Question 1

> All parts of this question can be viewed in Matlab by running `question_1.m`.

A

I modified the given NIST data, now in `input_data_nist.txt`, and got rid of the excess words at the top and the variable spacing between the `y` and `x` values. Then, I used the Matlab reader code to go line-by-line and store the values.

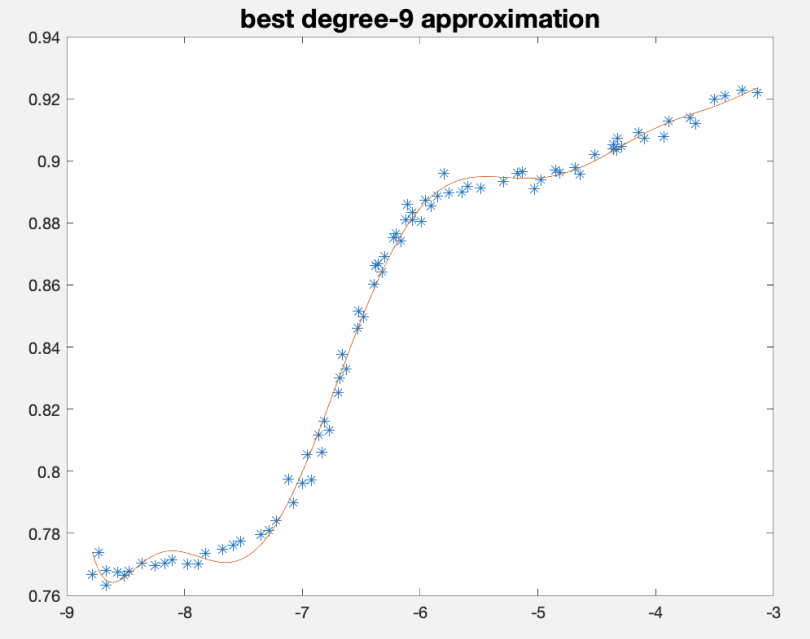
Here's what they look like, plotted:

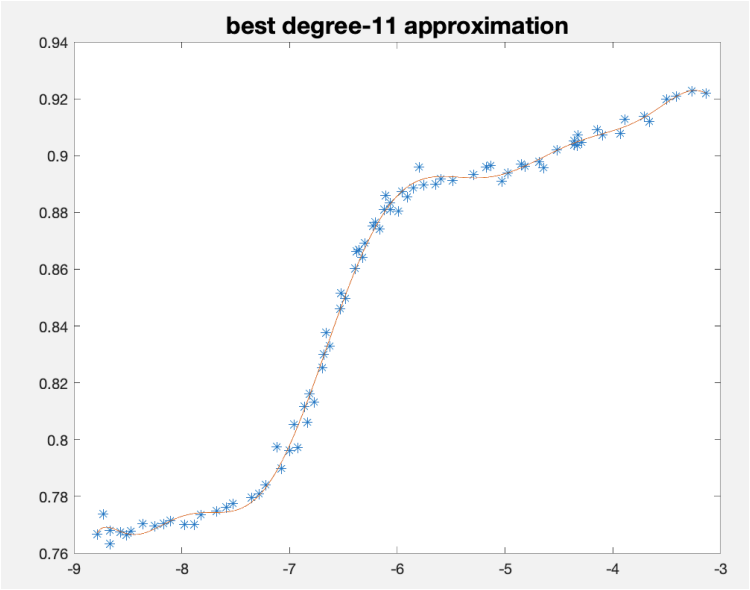


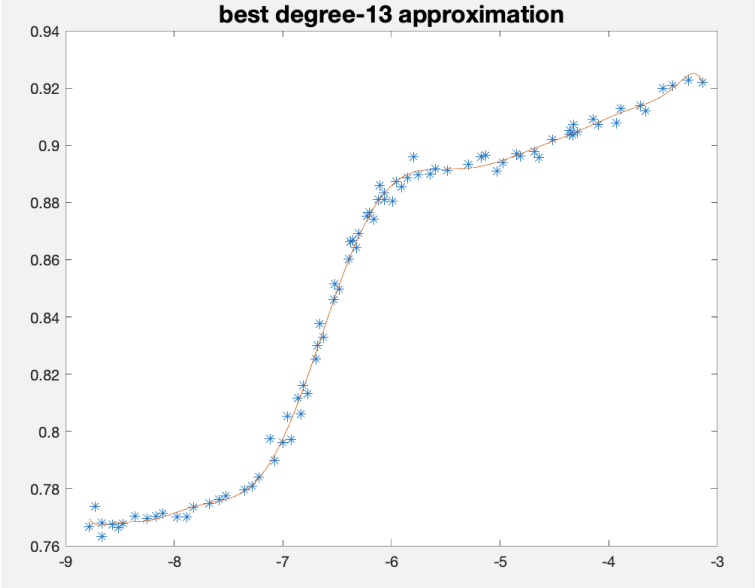
NIST Data

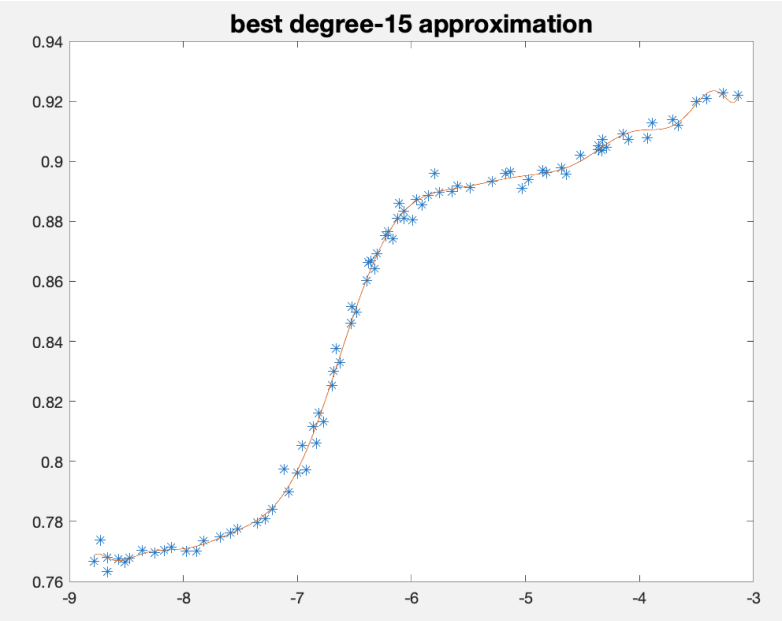
B

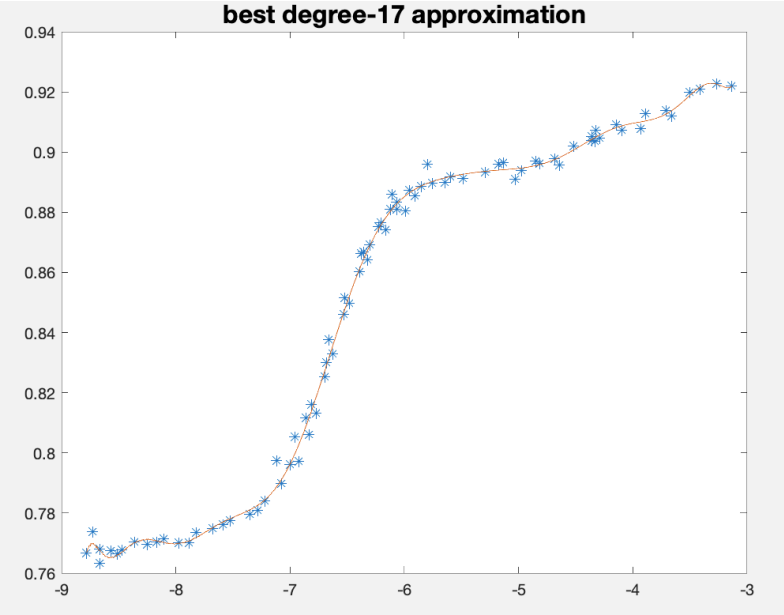
Using the supplied code as somewhat of a starting point, after I learned how the given method worked, I translated its functionality using provided methods in the Matlab API. In doing so, I plotted 1,001 points in an approximation to the given data, based on the dimensionality of the produced function. Here's what those graphs looked like! (*And, sorry, there are a lot of them for this question.)*

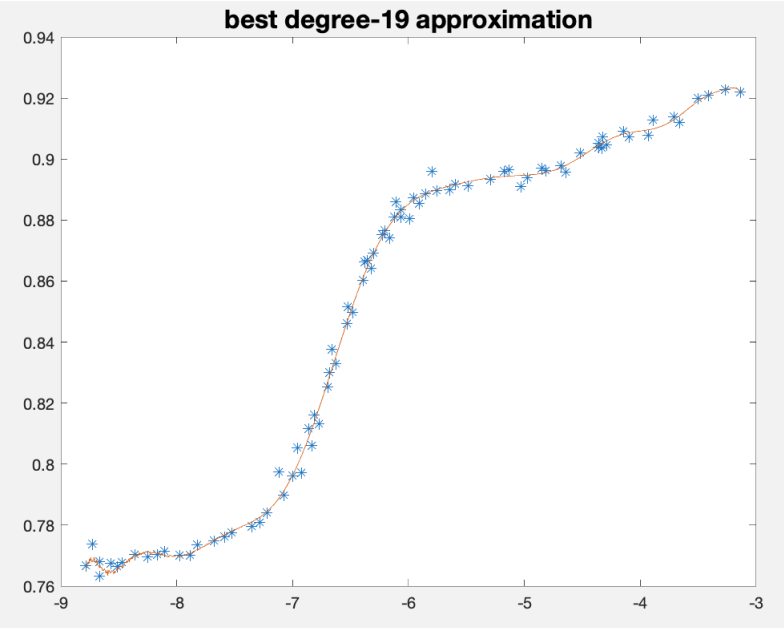


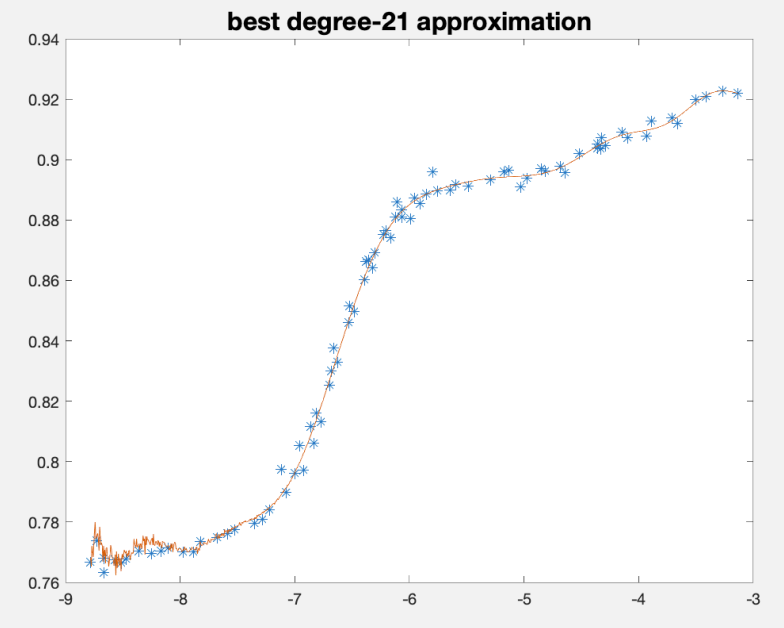

best degree-11 approximation
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
-9
-8
-7
-6
-5
-4
-3

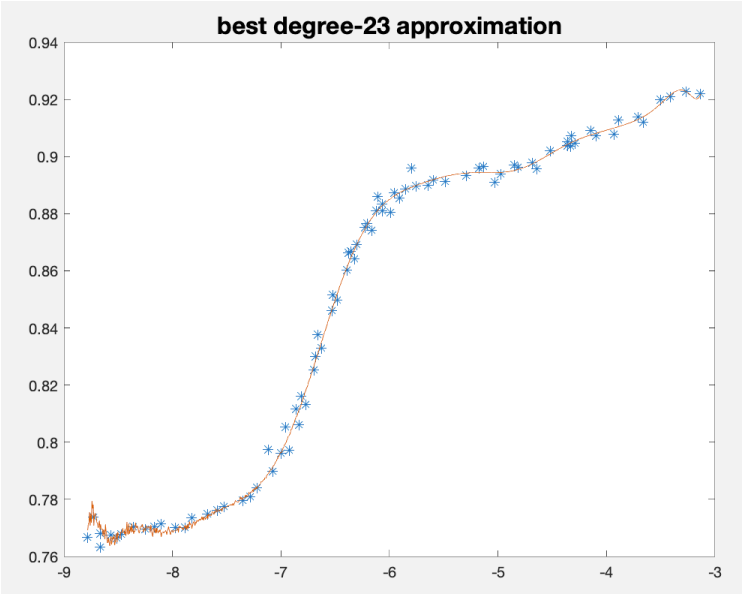

best degree-13 approximation
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
-9
-8
-7
-6
-5
-4
-3

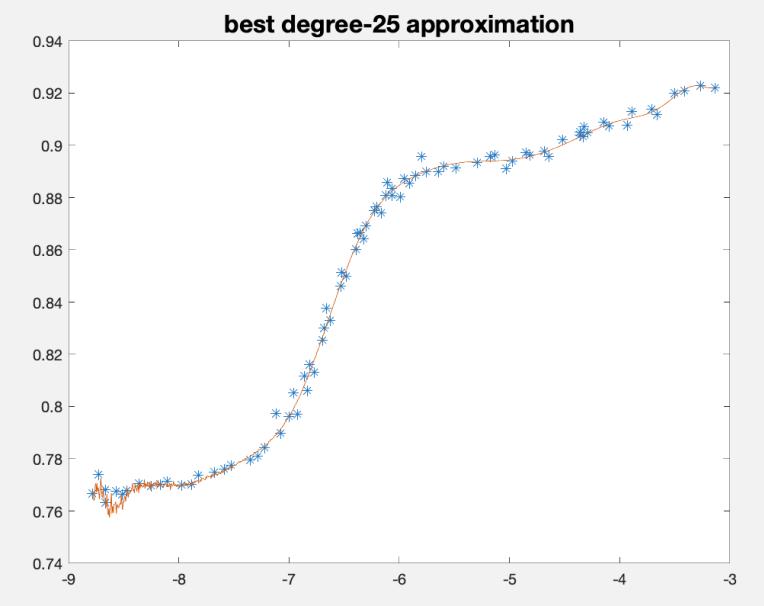

best degree-15 approximation
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
-9
-8
-7
-6
-5
-4
-3


best degree-17 approximation
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
-9
-8
-7
-6
-5
-4
-3


best degree-19 approximation
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
-9
-8
-7
-6
-5
-4
-3


best degree-21 approximation
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
-9
-8
-7
-6
-5
-4
-3





…As you can see, each of these graphs have a *very* similar shape, with minor variations dependent on the degree of polynomial. Just from eyeballing the images, which one's closer to the actual thing is *too hard to tell.* In fact, I'm surprised this method was able to get such a good approximation to the actual at all, given the resulting Vandermonde matrix is ill-conditioned, and even singular in some cases. Read on for the errors of each approximation.

C

Using Matlab's implementation of the following error equation:

$$E = \sum_{i=1}^{82} (y_i - p(x_i))^2$$

…We can gather the average least squares error of the approximations from Part B. That data is included below.

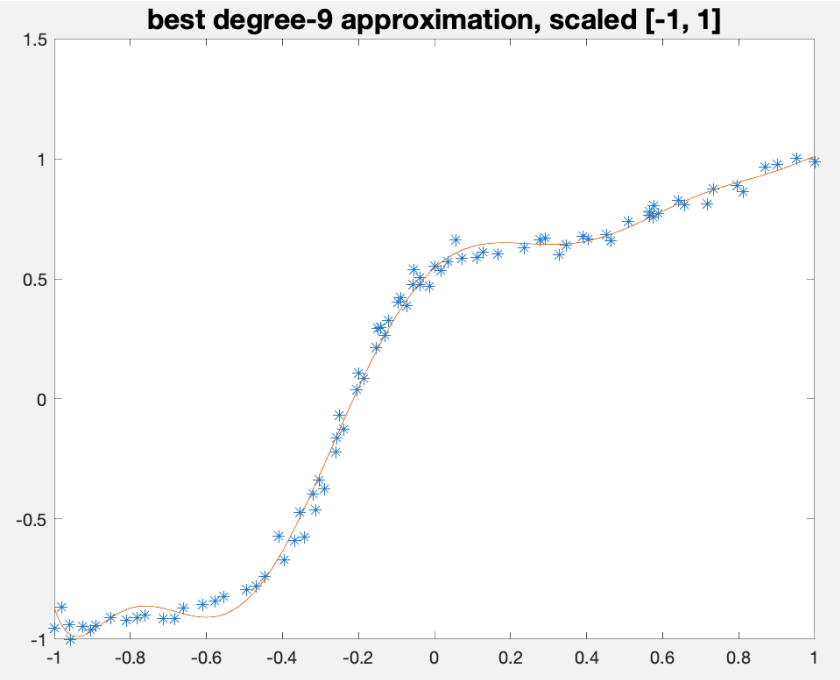
Least Squares Error (NIST Data)

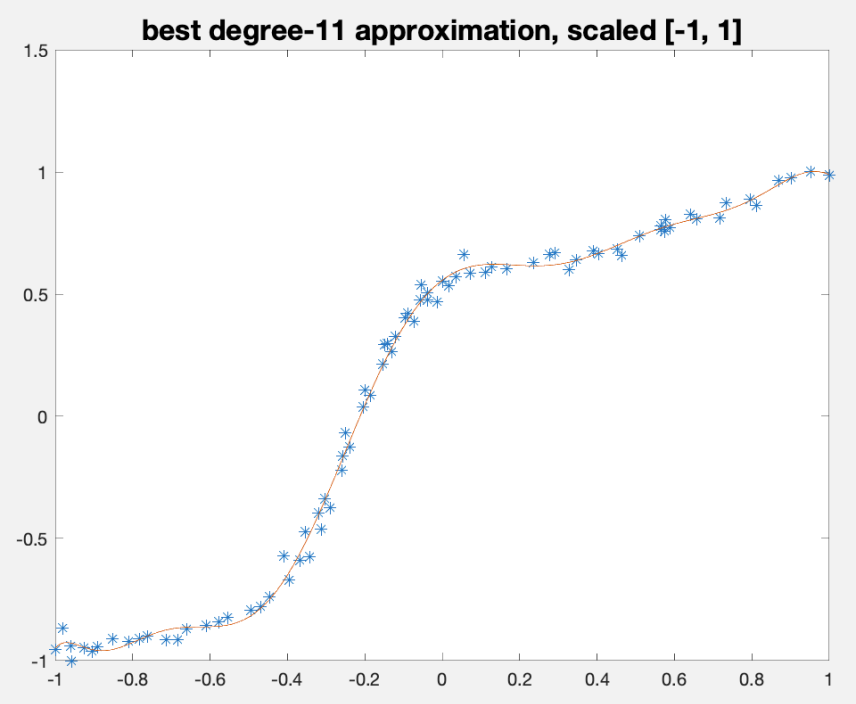
Polynomial degree	Average error for approximation
9	0.0040
11	0.0034
13	0.0034
15	0.0032
17	0.0033
19	0.0034
21	0.0037
23	0.0044
25	0.0047

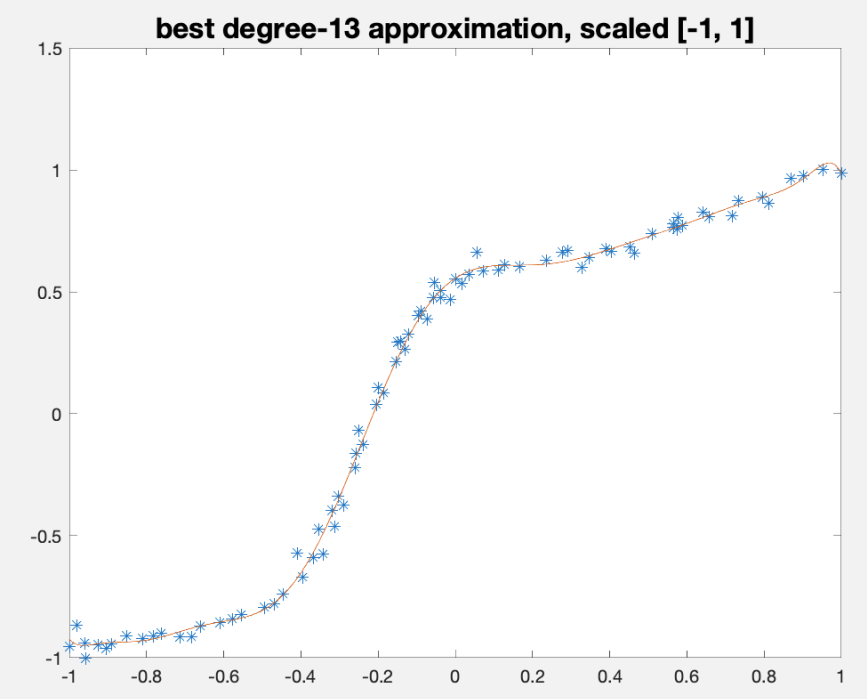
So, the lowest error hovers around .32, with a degree 15 approximation. This is as I would expect, given that a higher polynomial degree might tend to overfit to the existing data, where a lower polynomial degree might be *too* general to get the job done. Looking at the graphs in the previous question, this does appear to be the case. So, using *these* values of X and Y with no additional scaling, I'd say that an approximation using polynomial degree 15 is the best bet, though as most other questions in this class the *real* answer comes down to a tradeoff between computation time and accuracy.

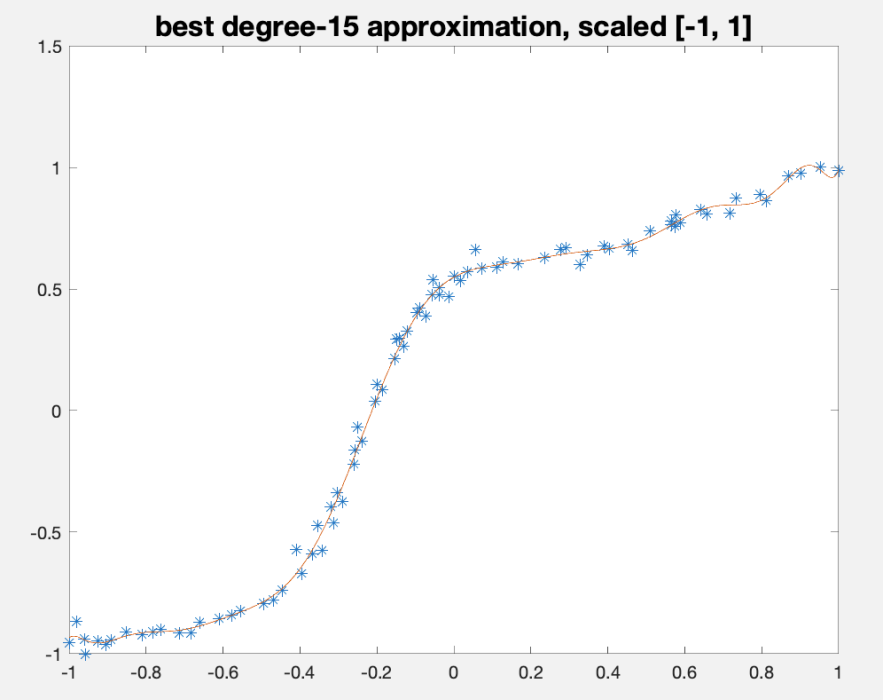
D

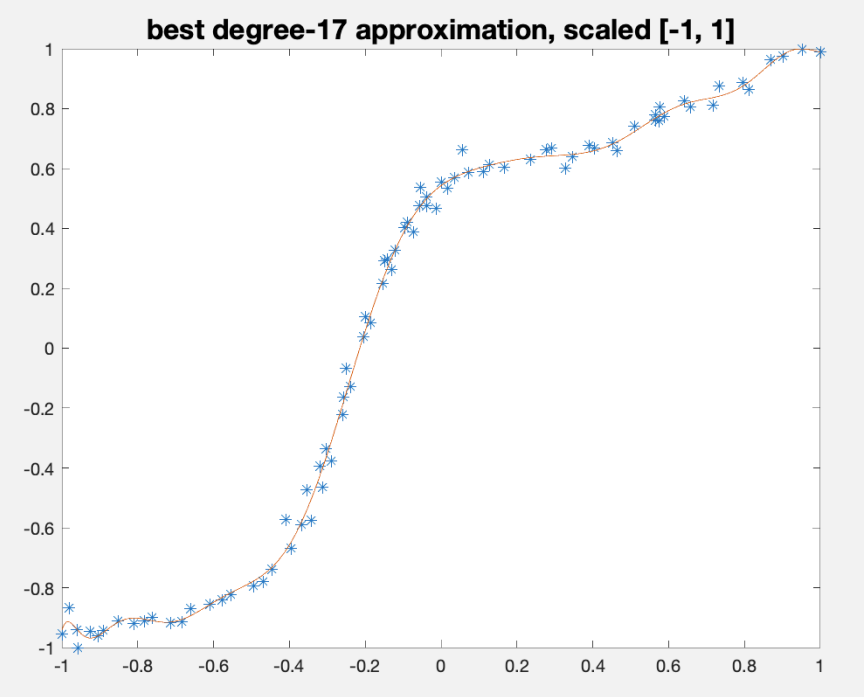
Doing all the same work as in Parts B and C, but this time scaling the X and Y values to both be in the range [-1, 1], we get the following graphs this time:

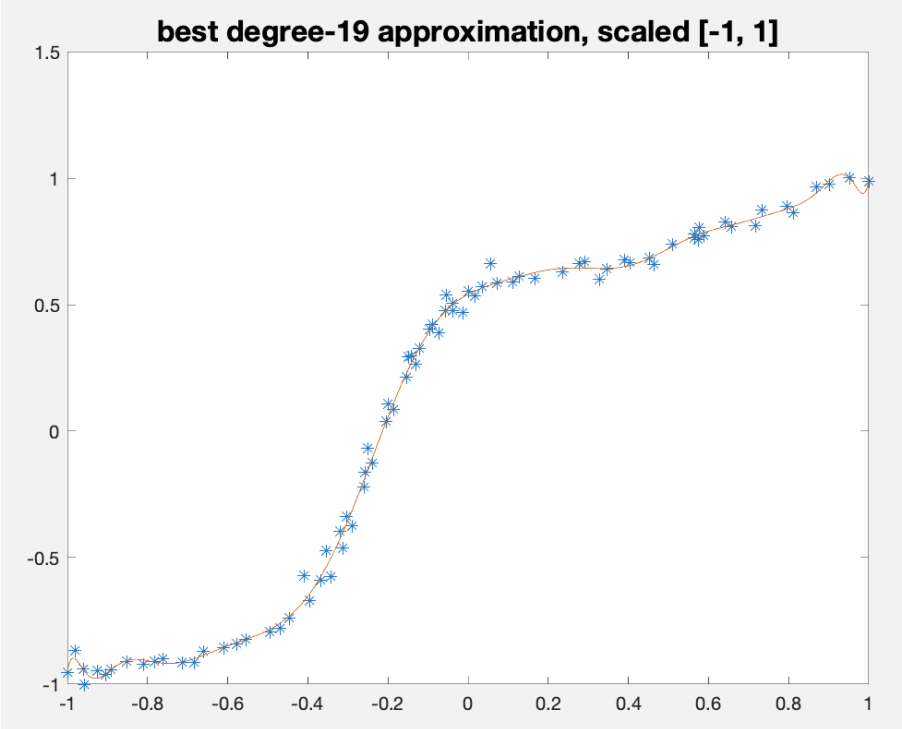

best degree-9 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

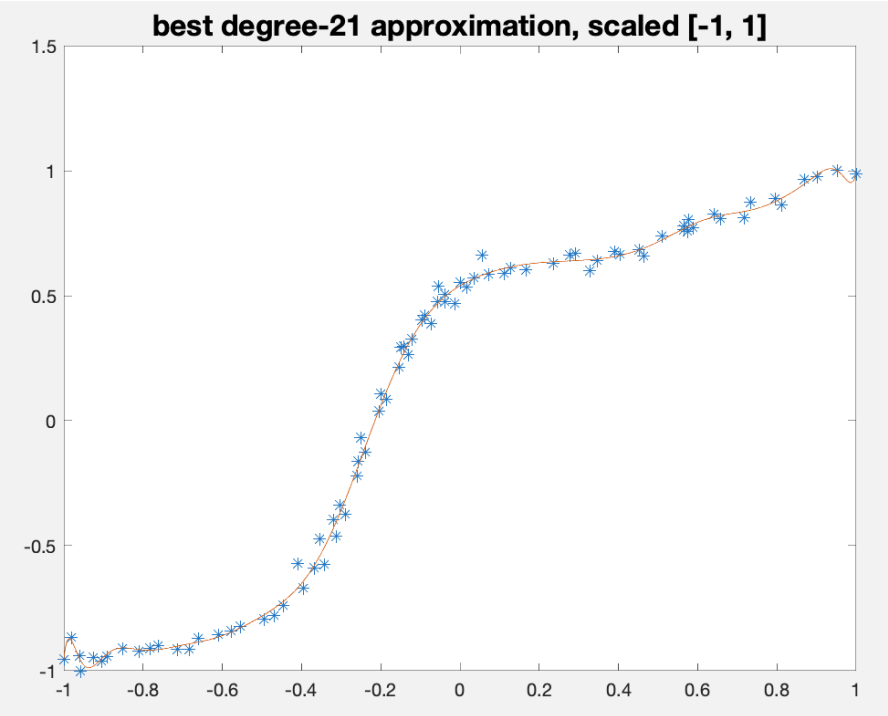

best degree-11 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

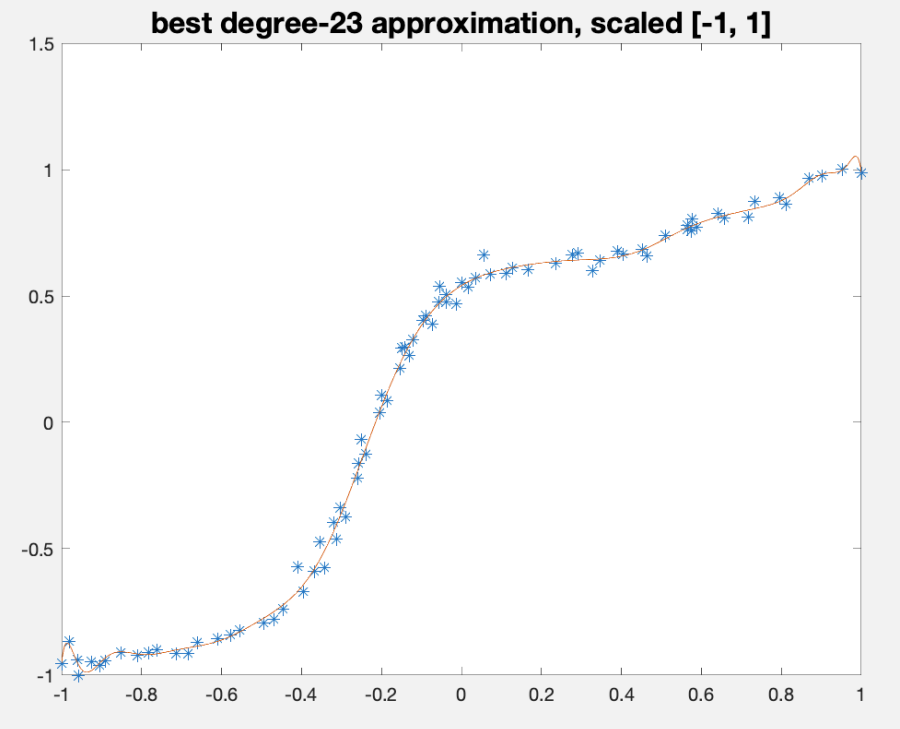

best degree-13 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

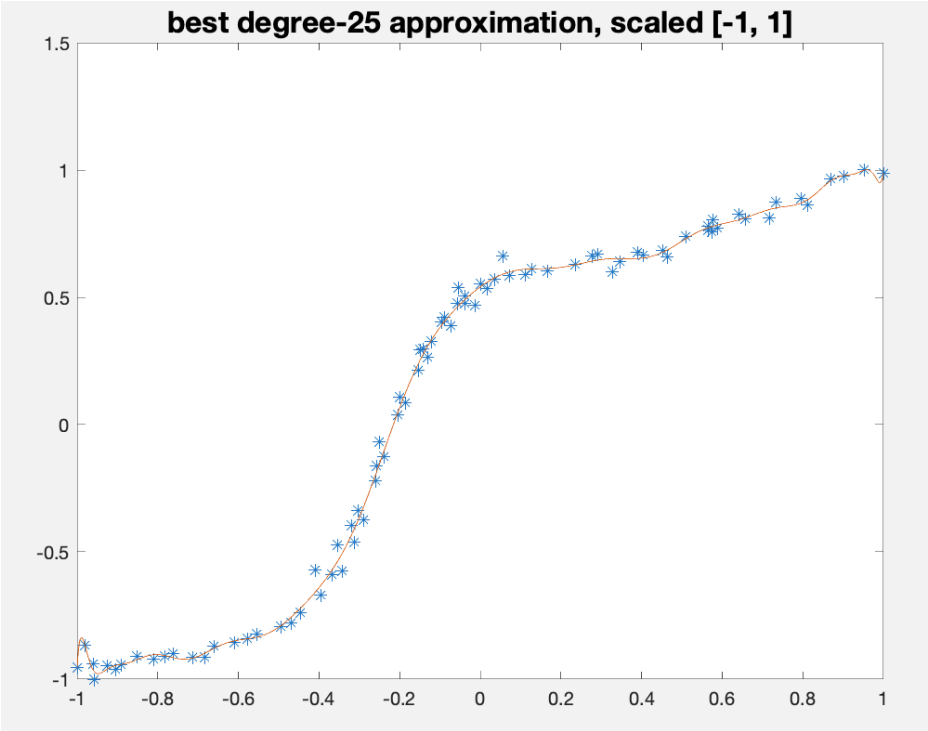

best degree-15 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


best degree-17 approximation, scaled [-1, 1]
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


best degree-19 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


best degree-21 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


best degree-23 approximation, scaled [-1, 1]
1.5
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1



Here, as in the first parts of this question, it's rather inconclusive by looking at the graphs which one is the best. The higher rank polynomial graphs *seem to be* a lot smoother and less willing to overfit to the given data. *However,* we can see the gathered errors in the table below:

Least Squares Error (NIST Data, scaled [-1, 1])

Polynomial degree	Average error for approximation
9	0.0502
11	0.0429
13	0.0422
15	0.0407
17	0.0411
19	0.0423
21	0.0441
23	0.0475
25	0.0541

Again, like in the first parts of this problem, the degree `m` of which estimated polynomial is more accurate remains the same – the polynomial with the lowest error overall is of degree 15.

E

So, then, on the basis of these experiments the value of `m` that proved to be the best in terms of accuracy and desired results was 15.

Here, the polynomial appears to be the closest fit to the data *and* has the lowest error. It's also in the middle ground of computation time between all the given polynomial degrees, too, so it would seem that it's the best in terms of accuracy *and* computation!

Question 2

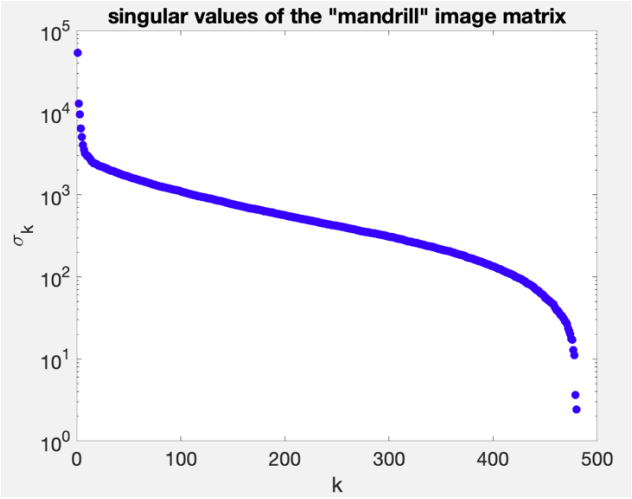
> Code for this question is contained in the files `question_2_durer.m` and `question_2_mandrill.m` respectively for each image.

A

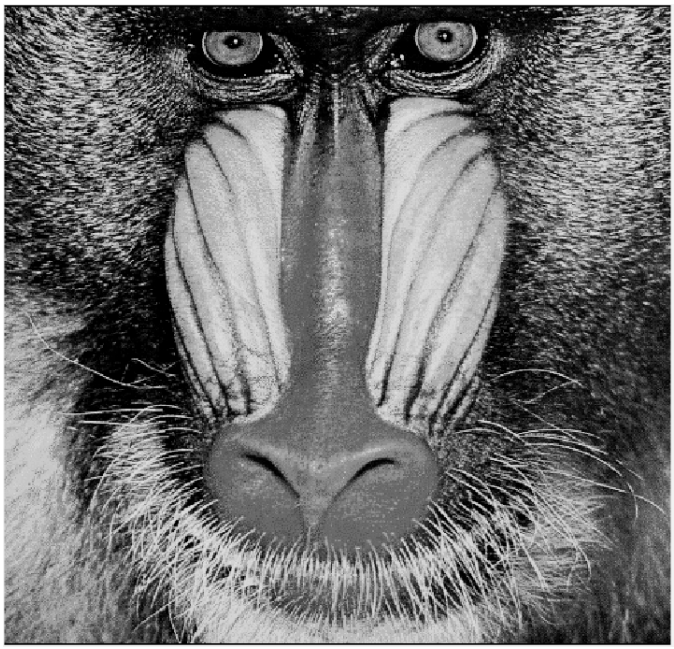
For the purposes of this question, I modified the given Gatlin image code, changed the colormap to grey, and made a `for` loop to go over the requisite image ranks {2, 4, 8, 16, 32, 64, 128}.

Below are the results for each of the images.

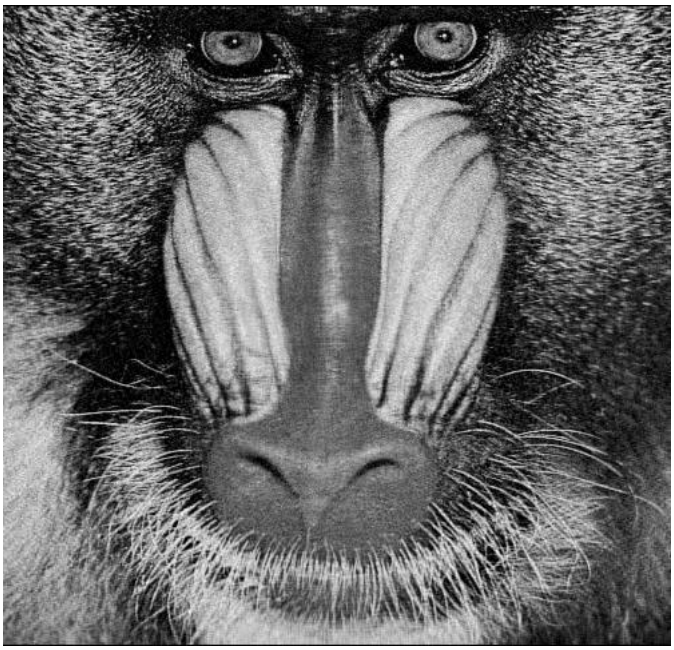
Mandrill



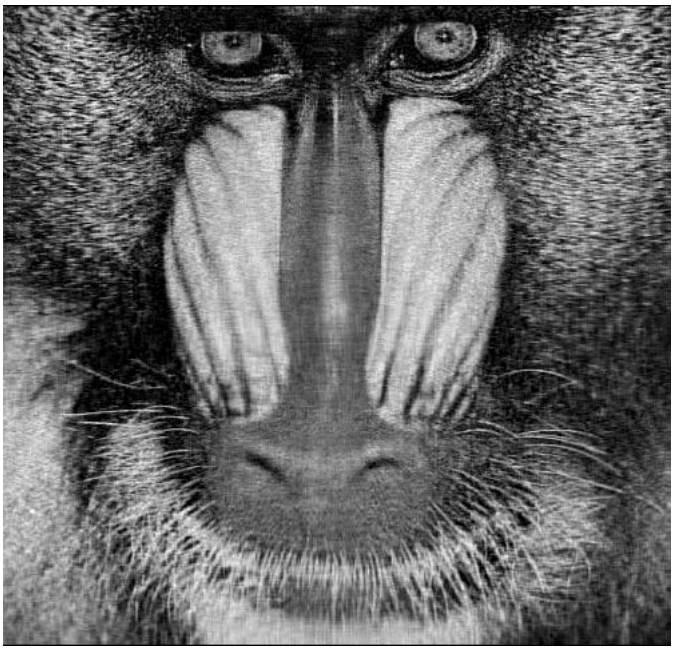
As you can see, the SVD of the image drop rapidly, slow down in the middle and then continue dropping exponentially as K surpasses a certain threshold, sort of like an odd logarithm.



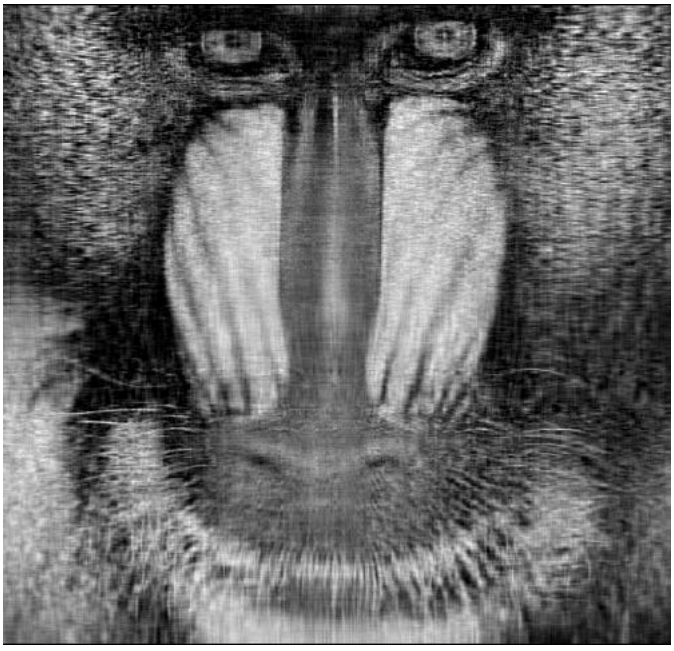
Mandrill True Image



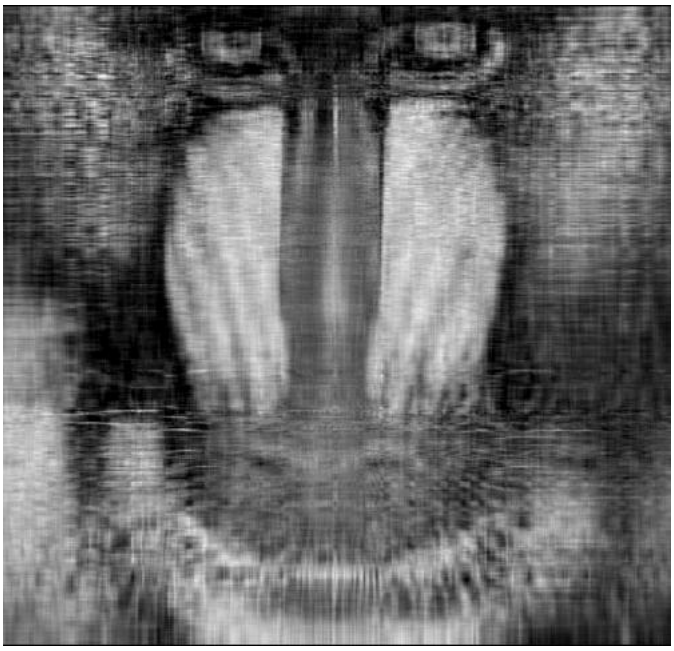
Mandrill Rank 128 Image



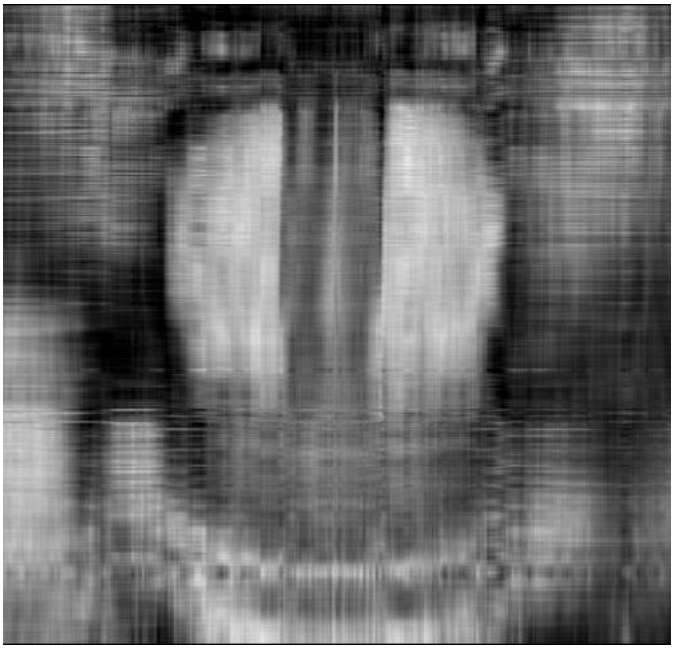
Mandrill Rank 64 Image



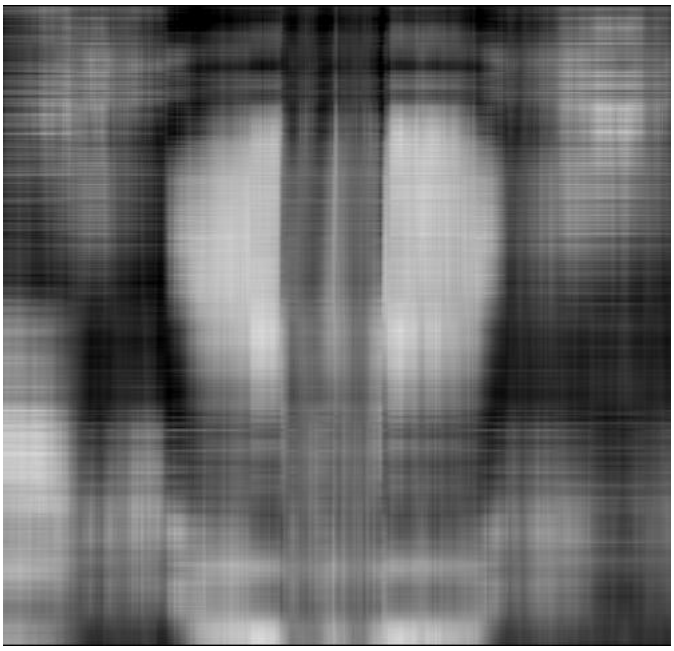
Mandrill Rank 32 Image



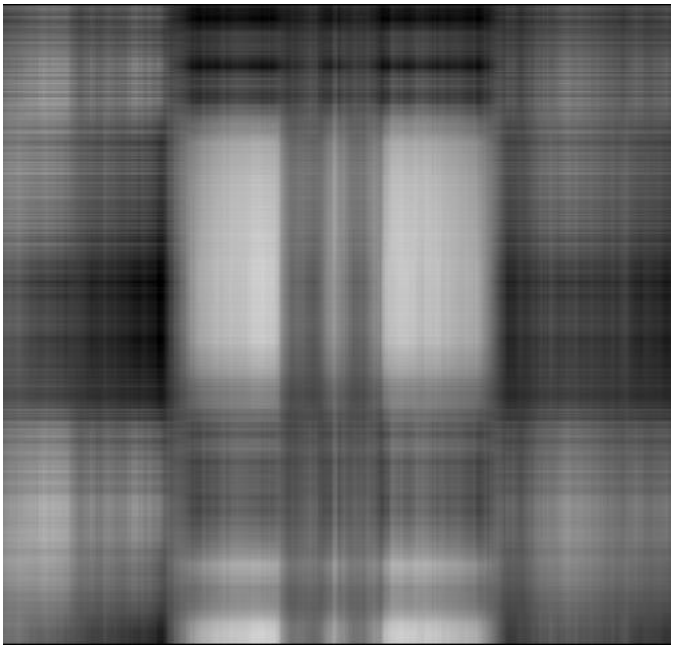
Mandrill Rank 16 Image



Mandrill Rank 8 Image



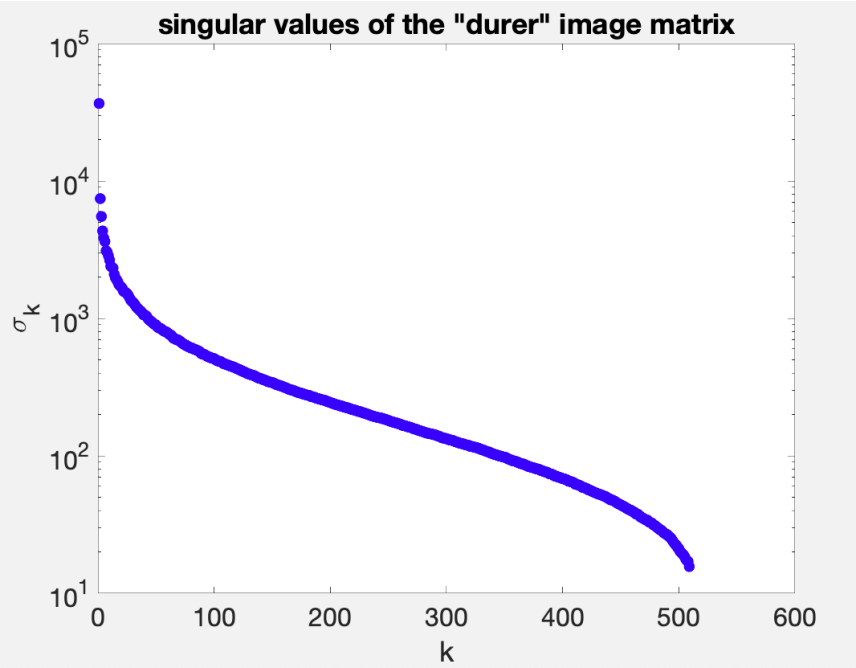
Mandrill Rank 4 Image



Mandrill Rank 2 Image

There's a steady drop in image quality from Rank 32 and down, *however,* the Rank 32 and below images save potentially drastically in terms of file size on disk. See part B for more details!

Durer



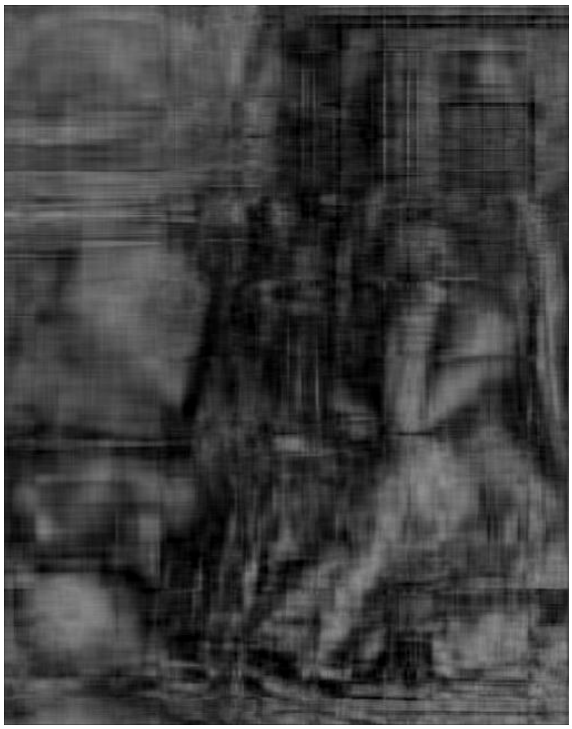
Here, the drops in SVD of the image follow a similar trend as the Mandrill image, but with a *slightly* different shape.

Durer True Image

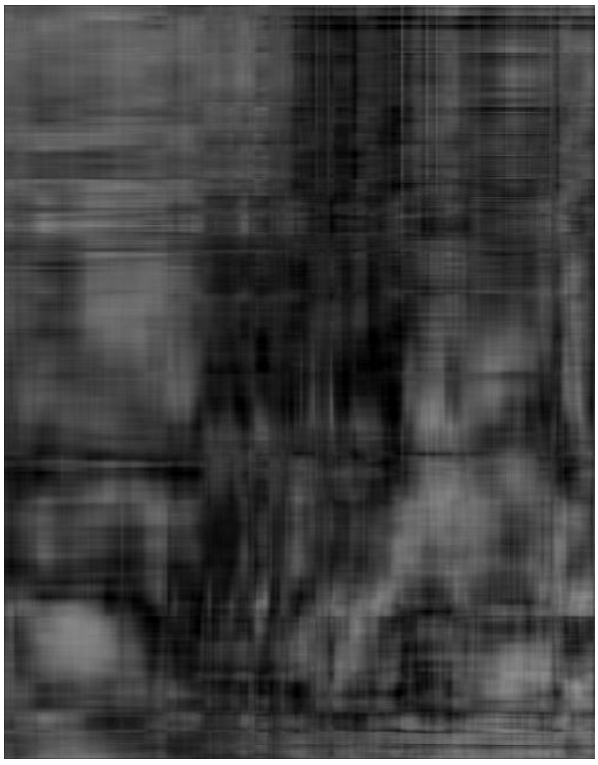
Durer Rank 128 Image

Durer Rank 64 Image

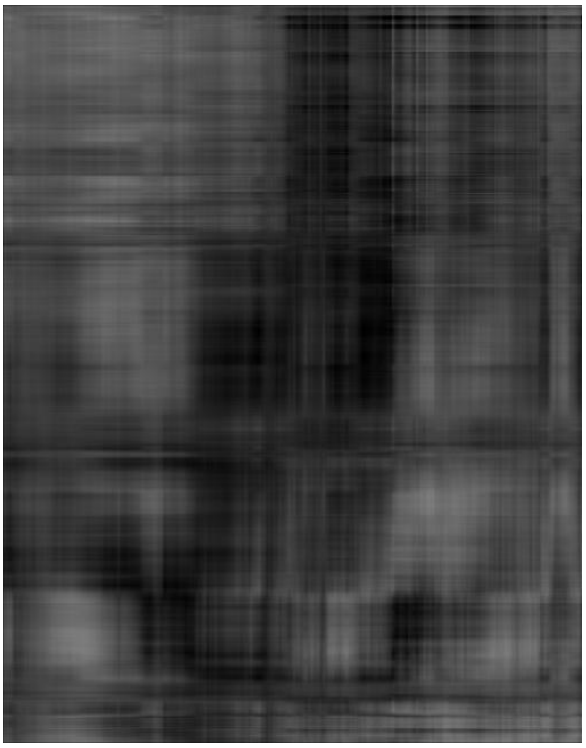
Durer Rank 32 Image



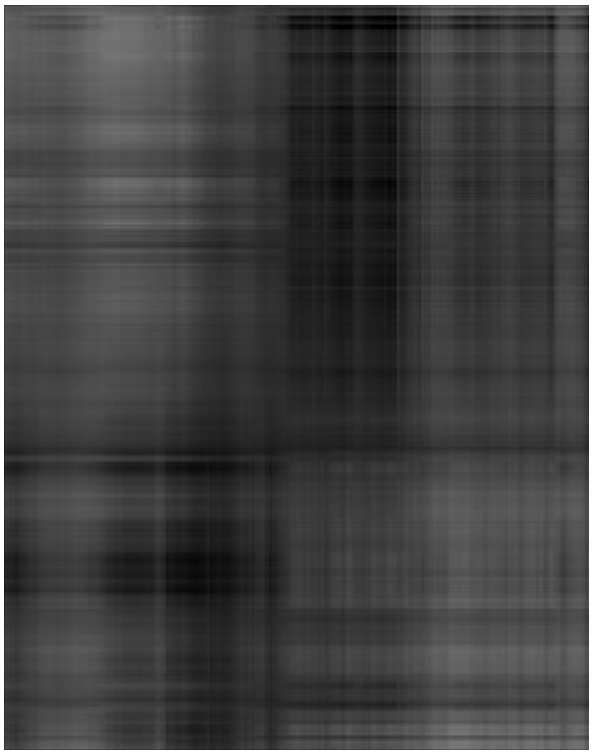
Durer Rank 16 Image



Durer Rank 8 Image



Durer Rank 4 Image



Durer Rank 2 Image

There's a steady drop in image quality from Rank 32 and down, *however,* the Rank 32 and below images save potentially drastically in terms of file size on disk. See part B for more details!

B

For this part of the problem, I saved each image to the disk as a `.jpg` image. Of course, the size of these images depends on the filetype used – in this case, I chose JPG, a common enough standard, but the same trends *did* apply when I tried the images in a PNG format.

So, as the rank changes, the following tables describe the filesize of the respective images:

Mandrill .jpg

Image Rank	Image Filesize	Size Difference v. Previous
2	24 KB	~
4	28 KB	4 KB
8	35 KB	7 KB
16	44 KB	9 KB
32	55 KB	11 KB
64	66 KB	11 KB
128	76 KB	10 KB

Durer .jpg

Image Rank	Image Filesize	Size Difference vs Previous
2	23 KB	~
4	27 KB	4 KB
8	32 KB	5 KB
16	38 KB	6 KB
32	45 KB	7 KB
64	53 KB	8 KB
128	60 KB	7 KB

…As the rank doubles, roughly a constant is added to the filesize for each file each time. Speaking in terms of Big O, this is *roughly in the ballpark* O(log(n)) complexity – as the input size doubles, a roughly constant amount of work is added. Of course, it isn't exactly the same principle here, but it's comparable.

There are diminishing returns in how much you can compress an image using this method, however – after a certain point, only the skeleton remains and the image is unrecognizable as its former self. *However,* as is the case with many other problems presented in this class, it's a tradeoff between space and efficiency that has to be decided given the circumstances surrounding the problem.