Corey Schulz
CS 3200 – Spring 2020
April 11, 2020

Homework 5 Report
Gradient Descent and Eigenvalues

Question 1

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

A

The problem does, indeed, fail with the existing code. I changed the *n* value to line up with the new matrix dimensions and ran the code. It produced the following output when it finished running:

```
Solution of the system is Inf Inf Inf NaN
 norm residual Inf in 17 iterations
>>
```

As such, the problem as stated fails and does not converge on a proper solution with the provided code.

B

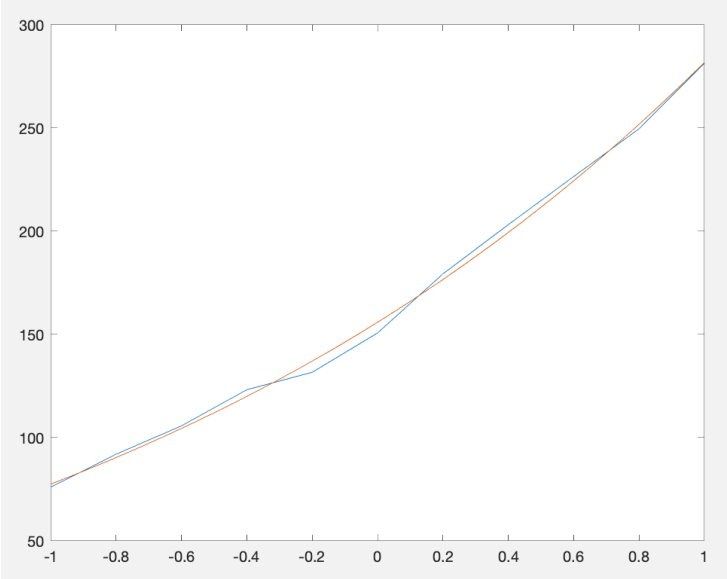
Scaling the year values in the *y* array by

$$s = \frac{(year - 1950)}{50}$$

…and running the code again, however, *can* solve the problem, giving a solution of:

```
Solution of the system is 155.904267 98.732292 23.726150 3.423350
 norm residual 10.207579 in 97 iterations
```

Additionally, the steepest descent can be graphed as the following, alongside the existing data:



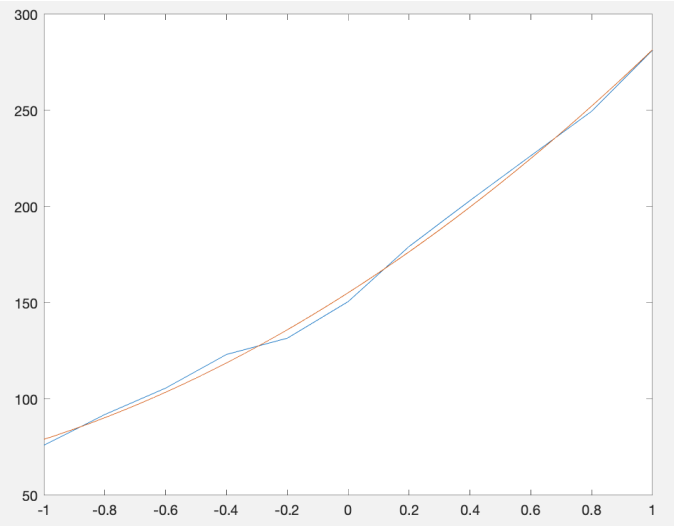
US Population (blue) vs US Population Steepest Descent (orange)

In all, the solution converged to the same point, but occasionally over or underestimated the solution, depending on the year. The descent solution was never off by a significant amount.

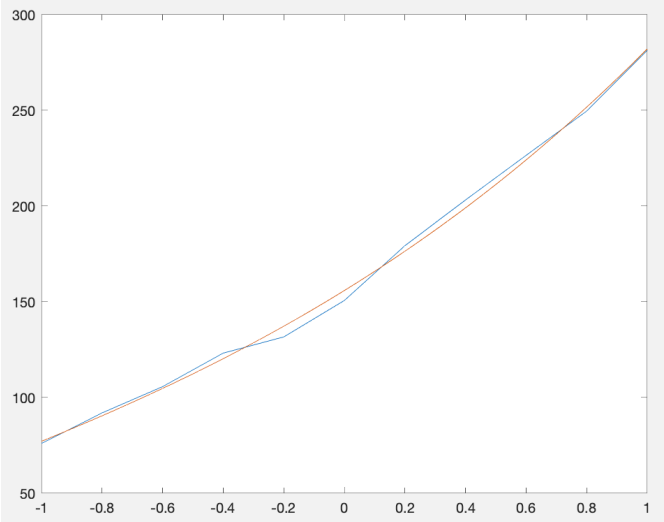
C

For this question, I refit the code in the given Steepest Descent function into its own Matlab function, in `steepest_descent.m`, and ran the code with a tolerance of 1e-5, and differing polynomial degrees of 2, 3, 4, and 5.

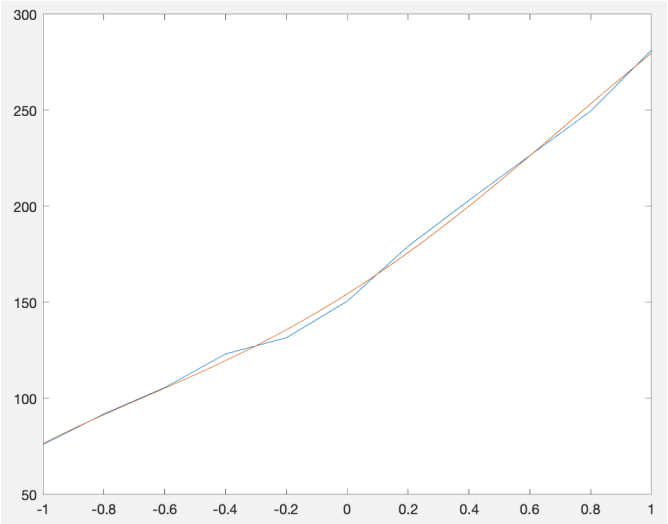
Below are the graphs of each of those functions.



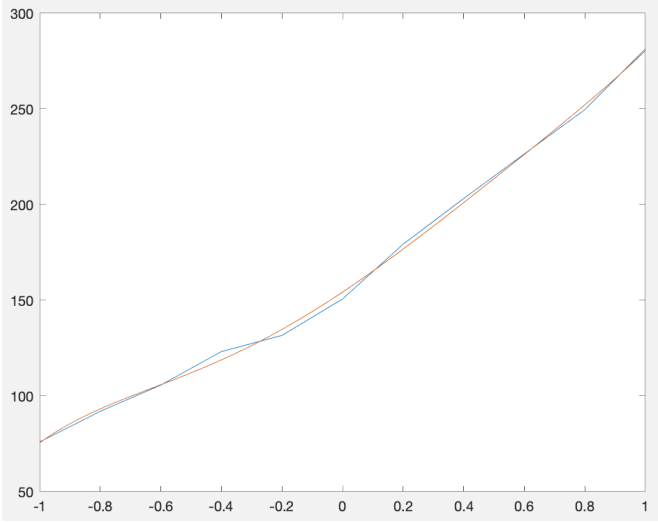
Polynomial degree: 2 (orange)



Polynomial degree: 3 (orange)



Polynomial degree: 4 (orange)



Polynomial degree: 5

As you can see, all the output polynomials are *very* similar, and all have the same structure. And so as you'd expect, they all produce rather similar results in the big picture. However, the residuals vary by up to 25% between all the graphs.

Polynomial Degree	Least Squares Residual (norm)
2	10.301042
3	10.363910
4	9.049163
5	7.852920

The residual norm trends downwards as the polynomial degree increases, with the largest jump being in-between degrees 4 and 5.

In general, one would assume that, given the norm of the least squares residual goes down as the polynomial degree goes up, it can be predicted that the polynomial degree 5 has, in general, the smallest least squares residual on average. Further, we can also predict that the degree of 5 will have a slightly higher accuracy than the other models. This hypothesis will be tested in the following question.

However, as in other questions presented in this class, this appears to be a tradeoff between computation time and accuracy – as the degree of polynomial goes up, the number of iterations required to converge can be on increasingly large orders of magnitude.

D

I modified the code further to extend the calculations to the years 2010 and 2019. From the data given, the actual population in 2010 and 2019 was 308.745M and 328.239M respectively.

The following table contains the results of the extrapolation.

Degree	2010 Estimation	2010 Difference from actual	2019 Estimation	2019 Difference from actual
2	312.894324	4.14932	342.752114	14.5131
3	315.550358	6.80536	348.432972	20.194
4	303.337447	-5.40755	320.023287	-8.21571
5	315.213200	6.4862	357.708524	29.4695

For “Difference from actual,” lower absolute value is better.

It seems that, contrary to my assumptions in the previous question, the degree of polynomial is not *necessarily* correlated with the accuracy of the model. For example, the model of degree 2 tends to be more accurate than the model of degree 3, but the model of degree 4 is more accurate than the degree 3 model.

From the limited data available, I have determined that the best overall model is the fourth model. The least squares residual is about in the middle of the others, sure, but the *prediction accuracy* is the highest of the other models, which I would prioritize over having a higher or lower least squares residual norm.

Of course, this might not be the right conclusion – there aren't enough data points to make an educated decision – but I believe the degree 4 model to be the best out of all the models presented in terms of accuracy. After all, that's what we really care about in the end regardless.

E

Changing the alpha value (from the default value of .61) can reduce computation time, but still produce acceptable results. I modified my code further to accept an alpha value as a parameter and return the number of iterations the steepest descent algorithm took to complete. Included in the table below is a summary of those findings. For the purposes of this problem, I used the model determined to be the best in Part D, which was the polynomial of degree 4.

Alpha	Iterations
.2	1019
.4	541
.6	372
.61	367
.62	361
.63	356
.64	351
.65	341
.66	336
.67	331
.68	327
.69	323
>= .7	fail

Thus, the best value of alpha to use for this problem is .69. Using this over other values can drastically reduce the workload required for convergence. It's definitely worth testing if the problem needs to be done repeatedly.

Question 2

> All parts of this question can be viewed in Matlab by running `question_2.m`.

A

I used the provided PowerMethod code in Matlab to evaluate the following matrix A:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

With an initial guess of [1 ; 1 ; 1].
This produced the eigenvector of the largest eigenvalue.
Using the Rayleigh Quotient,

$$\lambda_1 = \frac{x^T A x}{x^T x}$$

…We can derive the Eigenvalue from that, which ended up being 16.1168 in this case.
Matlab's builtin `eigs` method got the same result for the same matrix A.

B

I repeated the same process as in part a, this time with the following matrix:

$$\begin{matrix} 2 & 3 & 2 \\ 1 & 0 & -2 \\ -1 & -3 & -1 \end{matrix}$$

…and an initial guess of [2 ; 3 ; 2].

This time, the result returned from the PowerMethod function was `NaN` -- this is not the correct answer. So, why did this happen? The computation used all the max of all 100 iterations to arrive at its incorrect answer. The norm used to determine convergence started off large at 3.1176, and went down, eventually seeming to converge on 1. However, once the number actually reached 1, the x_prev and x values used to determine the norm got near zero – and eventually, zero. The error here is,

most likely, an edge case with this implementation of the algorithm that relies on the norm being able to be computed as-is. But if `NaN` is somehow passed as an argument to the norm function, it can fail and timeout the computation. Ultimately, the numbers in this initial guess cause a `NaN` error in the code, preventing actual convergence.
I'm not entirely sure if it makes a difference here, but in this case the initial guess is a transposed version of the top row of the initial matrix. However, upon further inspection, this doesn't appear to be directly related to the problem and isn't repeatable with different matrices.

C

I tried the same process as in step 2, this time with a modified initial guess of [1 ; -1 ; 1].
Using this initial guess causes the PowerMethod function to immediately converge, and the largest eigenvalue reported here after using the Rayleigh Quotient is a value of 1. This *is* an eigenvalue of the given matrix, albeit not the correct one. The eigenvalues provided from Matlab are {-3, 3, 1}, with the largest eigenvalue being 3.

This error is another edge case that's arisen with the given PowerMethod code.
This is because, in the first iteration of the method, the following updates are done to x:

```
y = A*x_prev;
x = y / norm(y,inf);
x = x*sign(x(1));
```

Following these updates to x, x still is equivalent to x_old. `y` is equivalent to x_old as well. The solution immediately converges because of this. `x-x_prev` is a zero vector, and the norm of that, too, is zero. This is, of course, less than the tolerance, so the solution immediately converges, returning the input guess [1 ; -1 ; 1]. This immediate convergence is an edge case with this Power Method implementation. An obvious fix is to change the input guess and the correct answer can then be obtained.

D

For this question, I made a new function, `inverse_power_method.m`, to calculate the *smallest* eigenvalue.

The change in the code ended up being only one character. Since we know that

$$y = A^{-1} * x$$

And that, for the Power Method

$$y = A * x_{prev}$$

We can determine, for the inverse power method the following:

$$y = A \setminus x_{prev}$$

I implemented this in my code, and found the smallest eigenvalue to be 0 in this case. I interpreted the question's *smallest* meaning least magnitude, which 0 is in this case. To prove it, I inserted the inverse of A into the original PowerMethod function, and it also output zero when the eigenvector was turned into the proper eigenvalue.

E

Part I

Using the provided data, I made the following matrix:

0.3000	0.3000	0.3000	0.1000
0.9000	0	0	0
0	0.8000	0	0
0	0	0.5000	0.1000

I used the PowerMethod function provided, and along with the Rayleigh Quotient, calculated that the following is the largest eigenvalue of the matrix: 0.9016.

Part II

Based on the answer from Part I, I expect the population to slowly die over time. Since the largest eigenvalue is less than 1, the deaths outpace the births and the population will eventually, given enough time, drop to zero.

Part III

I used the following equation from the slides to project population over a period of 100 years from the given initial values.

$$\begin{pmatrix} P_1^n \\ P_2^n \\ P_3^n \\ P_4^n \end{pmatrix} = \begin{pmatrix} b_1 & b_2 & b_3 & b_4 \\ (1-d_1) & 0 & 0 & 0 \\ 0 & (1-d_2) & 0 & 0 \\ 0 & 0 & (1-d_3) & (1-d_4) \end{pmatrix}^n \begin{pmatrix} P_1^0 \\ P_2^0 \\ P_3^0 \\ P_4 \end{pmatrix}$$

The projection of this population was:

Population Group	Projected population after 100 years
P1	0.0125
P2	0.0124
P3	0.0109
P4	0.0067

And since population should be cast to an integer, this model predicts that after 100 years, the population would total zero.

Part IV

I did the calculations again with an updated value of `d4 = .01`.

This time, the largest eigenvalue is 1.0797. Given that this is greater than 1, I expect the population to grow unbounded.

Doing the same population projection results in the following data. This time, it's much different.

Population Group	Projected population after 100 years
P1	168110
P2	140130
P3	103830
P4	578840

So, the largest eigenvalue *does* help project population growth very well!