Dan Ruley
CS 3200
Assignment #5
April 9, 2020

Problem 1:

a.)
The Steepest Descent function definitely fails for the x vector as is - the coefficients for the polynomial and the residual norm are either Inf or NaN.

b.)
Scaling the x vector to be within the range [-1, 1] results in reasonable values. The residual norm is 10.35 and the polynomial function approximates the x values fairly well.

c.)
The residual norms are all fairly close when comparing polynomials of degrees 2-5. The lowest results from the 5th degree polynomial; however, it seems the number of iterations increases significantly with the degree of the polynomial.

Degree	Normres	Iterations
2	10.36	20
3	10.35	79
4	9.049	351
5	7.85	1097

There seems to be a tradeoff - higher degree polynomials result in a lower residual norm and presumably a better fit for the given data, but require significantly more computations to arrive at the result.

d.)
All of the polynomials seem to be somewhat reasonable in terms of predicting the 2010 and 2019 populations. At least they are in the same ballpark as the actual numbers for those years: 308.745M for 2010 and 328.239M for 2019. I think the degree 4 polynomial is the best out of these, it predicts the 2010 and 2019 populations the best, and has the 2nd lowest normres. It also takes significantly fewer iterations than the 5th degree polynomial to arrive at the answer.

Degree	Normres	Iterations	2010 prediction	2019 prediction
2	10.36	20	313.0407	342.9856
3	10.35	79	315.465	348.2682
4	9.049	351	303.3283	320.0004
5	7.85	1097	315.2766	357.9125

e.)
I varied the value of alpha from 0.1 to 1.0 and analyzed the SteepestDescent function's behavior with a degree 4 polynomial and a varying alpha value. The larger alpha values in the range .89 to 1.0 resulted in the fewest iterations; however, these polynomials produced completely inaccurate results. The alpha value that still produced reasonable results but also minimized the runtime was 0.21. The degree 4 polynomial with alpha = 0.21 required 71 iterations and estimated the 2010 population at 329.86 and 2019 at 376.09, which is not great but fairly reasonable given that it cut down the number of iterations significantly.

Problem 2:

a.)
I used the suggested matrix A = [1, 2, 3; 4, 5, 6; 7, 8, 9] with an initial x of [1;3;5]. The results converged to an x vector of [0.2834; 0.6417; 1] in 8 iterations. Using the Rayleigh Quotient we can solve for $\boldsymbol{\lambda}$ = (x'*Ax)/(x'· x) = 16.1168. Using the Matlab eig(A), we can see the eigenvalues of A are [16.1168; -1.1168; 0]. Therefore, the Power Method correctly determined the largest eigenvalue for this matrix, 16.1168, which corresponds to the eigenvector [0.2834; 0.6417; 1] the function computed.

b.)
The method fails to converge after 100 iterations for B = [2, 3, 2; 1, 0, -2; -1, -3, -1] and x_0 = [2;3;2]. It seems as the new x values are iteratively calculated, they begin to oscillate between [1; 0; -1] and [0; -1; 0]; hence, norm(x - x_prev) is always 1 and will never decrease below tol. I believe this is because the eigenvalues of B are [3,1,-3]. Since |3| = |-3|, there is no single eigenvalue with the greatest magnitude and A^m * x will not converge to a particular value.

c.)
Using the vector [1; -1; 1] produces different results; the Power Method immediately converges in one iteration and returns the vector [1; -1; 1]. This makes sense because this is actually an

eigenvector for B, corresponding to the eigenvalue 1. B*[1; -1; 1] = [1; -1; 1]. This is actually the only thing the power method could possibly converge to, since [1; -1; 1] is an eigenvector for $\boldsymbol{\lambda} = 1$ it is orthogonal to the other eigenvectors of $\boldsymbol{\lambda} = -3, 3$.

d.)
I made a change so that the power method would use the inverse power method instead. I simply passed in a fourth parameter, inverse, and used an if/else statement to switch to the inverse power method if 1 is passed in for this parameter. The one character change that the function needs is simply computing y = A \ x_prev instead of y = A * x_prev. I then used the inverse version of the power method on matrix A and found the eigenvalue 0. This matches the Matlab eig function for A, which confirms $\boldsymbol{\lambda}$ with the smallest magnitude to be 0. Interestingly, the inverse power method worked for A even though it is a non-invertible matrix (its determinant is 0). I also used the inverse power method on matrix B and found its smallest eigenvalue to be 1, which is also confirmed with the Matlab eig function.

e.)
i.)
The largest eigenvalue for the matrix is 0.916.

ii.)
Since this value is less than 1, we expect the population to eventually die out.

iii.)
I simulated this over 1000 years, using the formula $P^{n+1} = A * P^n$. Indeed, after 1000 years, all of the P_i values were around 1e-39, or essentially zero. This matches our expectation that the population would go extinct based on the largest eigenvalue being < 1.

iv.)
Changing d_4 to 0.1 instead results in the largest eigenvalue of the matrix being 1.0292. Now, since this is >1, we expect the population to grow in an unbounded manner. I ran the simulation over 1000 years and observed that P = [3.3e14; 2.9e14; 2.2e14; 8.7e14]. Clearly, the population has grown exponentially, matching our expectations based on the largest eigenvalue of the population matrix.