1 Overview/Description of Code

In this section, I will briefly discuss all functions that I created to complete this assignment as well as touch on a few optimization measures implemented. Each code is well-documented and further explanation for each function can be found in the comments themselves.

- **powerIteration.m** - My powerIteration.m function is a blend of the power and inverse method. It takes in the usual matrix A, tolerance, initial guess, and maximum number of iterations, but it also receives a function handle used to generate the point x_{k+1} given x_k. For power method, this is $g(x) = Ax/\|Ax\|$, and for inverse iteration, this is $g(x) = (A\backslash x)/\|A\backslash x\|$. This flexibility allows for powerIteration.m to be used to solve questions 1 and 4.

- **anderson.m** - The heaviest code of this assignment, anderson.m applies Anderson acceleration to a given problem. It also takes in parameters A, tolerance, initial guess, maximum number of iterations, and function handle g. Furthermore, it receives the step m for $m-$level Anderson acceleration. As such, anderson.m is able to apply Anderson accelerations to both the power method and the inverse method (just how powerIteration.m can do both).
  The $m-$level acceleration was particularly tricky to code. In order for a clean, easy to read code, whenever $m > 0$, I chose to initialize $x_j = x_0$ for all $j < m$ and started iterating on step $m + 1$. Thus, for the first m iterations, there actually is enough history to look back at to appropriately solve the subproblem. This does not affect the subproblem at all since the additional variables are just duplicates of the desired variables. However, as a side note, this means that other outputs such as θ, $\|w\|$, and iteration count are slightly off. That is, for $m-$level Anderson acceleration, the first $m + 1$ values of $\theta, \|w\|$ will be 0, and the iteration count will be an overcount of $m + 1$. Please keep this in mind when viewing later plots.
  Lastly, although anderson.m is appropriately optimized in terms of not computing expensive operations twice (i.e. Ax is stored and never computed again), its subproblem is not optimized at all. The assignment allows for fminsearch to solve the subproblem and I used this explicitly. MATLABS's fminsearch actually struggles greatly to solve the subproblem, and even generates solutions such that $\theta_k > 1$, which is mathematically impossible. In this case, I manually set θ_k and note that this is simply a fault of the optimizer. Furthermore, because I pass in an explicit function handle for fminsearch to minimize (and not an anonymous one), the repeated function evaluations required of fminsearch take a large amount of time. This results in the Anderson acceleration of question 2 and 5 requiring more time than their unaccelerated counterparts, despite performing less iterations. One could remedy this by writing a specialized optimizer, but doing so would take away from the purpose of the assignment (and we were explicitly told that fminsearch was ok).

- **andersonQ6.m** - This is effectively the same code as anderson.m, except the function handle g required additional parameters for the tolerance proportional to the residual norm from the previous iteration. Rather than completely redesign anderson.m, I simply made this function (almost an exact replica) to handle the last problem with relative tolerance for pcg.

- **andersonSubProblem.m** This is a short function that acts as the input to fminsearch.

It takes in iterate history, m, and current iteration number and is then passed to fminsearch to be optimized over.

- **fpiPowerMethod.m** This is a specialized function that serves as an input to powerIteration.m and anderson.m. It computes $g(x) = Ax/\|x\|$.
- **fpiInverseMethod.m** This is the inverse method equivalent of fpiPowerMethod.m. It is used to compute $g(x) = (A\backslash x)/\|A\backslash x\|$. I realize that the inverse method is typically done by first LU factorizing A once and using those factors to compute $A\backslash x$ at every iteration, but after a few tests, I found that MATLAB's$A\backslash x$ outperformed my personal LU solver (even when LU was already computed) so I use $A\backslash x$ instead. This is also to be consistent with question 6 since it asks us to use pcg to solve the inverse problem.
- **fpiPCGMethod.m** Last but not least, this is the function handle input to andersonQ6.m that computes $g(x) \approx (A\backslash x)/\|A\backslash x\|$ using pcg with the incomplete Cholesky of A as a preconditioner.

2 The Assignment

1. Use the power method to find the largest eigenvalue of the $40^2 \times 40^2$ Poisson matrix with initial guess $x_0 = (1, 0, \ldots, 0)^T$. Discuss convergence results.

   Solution. This question/analysis is computed by the MATLAB script scriptQ1.m. The power method applied to a particular matrix A should converge linearly with rate $r = \lambda_{n-1}/\lambda_n$. Here, λ_{n-1} and λ_n are the second largest and largest eigenvalues in absolute value, respectively.Using MATLAB's eigs, I determined numerically that the $40^2 \times 40^2$ Poisson matrix has $\lambda_{n-1}/\lambda_n \approx 0.9978$. The eigenvectors computed by the power method should converge linearly with this rate, and the eigenvalues should converge linearly with rate $r^2 \approx 0.9956$. Figure 2 shows that the eigenvalues do indeed converge linearly. Furthermore, by analyzing e_{n+1}/e_n for the computed iterates, I obtained Figure 1 which shows the observed rate matches the theoretical one, at least asymptotically.

Iteration	e_n/e_{n-1}
2815	0.9956
2816	0.9956
2817	0.9956
2818	0.9956
2819	0.9956
2820	0.9956

Figure 1: Observed Error Rates for Power Method

2. Apply Anderson acceleration with $m = 1, 2, 3, 4, 5$ to the power method, using the same A, tol, and x_0 from part 1. Describe the convergence behavior you observed. Plot θ_k and $\|w_k\|$ for each computation.

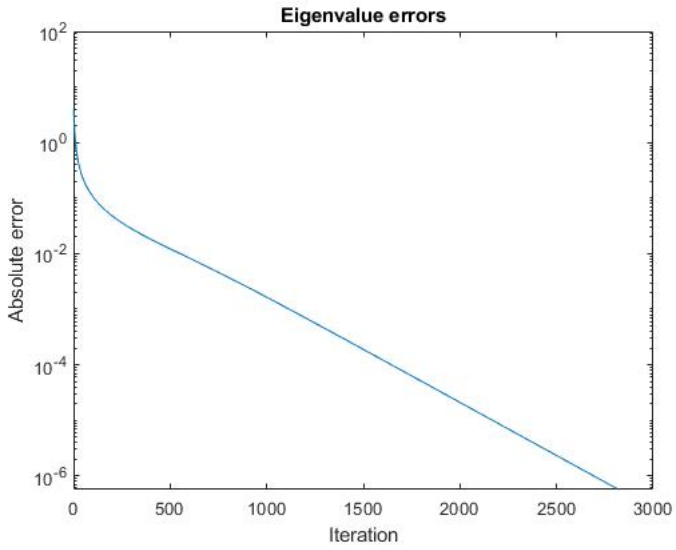

Figure 2: Plot of eigenvalue errors using power method computed via MATLAB script

Solution. This question/analysis is computed by the MATLAB script scriptQ2.m. Please see the "code notes" section as well as the script comments for details of implementation. Plots of θ_k and $\|w_k\|_2$ are provided below in Figure 3. It is easy to see here that $\|w_k\|$ tends to 0 as $k \to \infty$. Also, it helps to notice that although the script itself might take longer than the power method to run, the number of iterations to reach the tolerance for any value of $m \in \{1, 2, 3, 4, 5\}$ is less than the required number of iterations by power method. Thus, the results here show that Anderson iterations do indeed "accelerate" the performance of power method iterations[1].

3. Use the convergence theory above for Anderson acceleration to explain the observed convergence behavior.

Solution. Another look at Figure 3 showcases the flexibility of Anderson Acceleration. Note that for this problem $\kappa \approx 0.9978 \approx 1$, thus the bound provided

$$\|w_{k+1}\| \leq \theta_k \kappa \|w_k\| + C\|w_k\|(\|w_k\| + \cdots + \|w_{k-m}\|)$$

tells us that the improvement in $\|w_{k+1}\|$ is dominated by θ_k for low orders of m, and could be different for higher orders. Indeed, in Figure 3, note that when $m = 1$ and $\theta = 1$, almost no improvements in $\|w_k\|$ are made. Furthermore, note that the behavior of $\|w_k\|$ fluctuates more when $m = 5$, likely due to the additional constants that do not depend on θ_k. Without the knowledge of the constant C, however, it is difficult to assess whether

[1] As a side comment, questions like this are very difficult to verify. I'm by no means an expert at Anderson acceleration so it is difficult for me to tell whether or not the behavior above is accurate. I am moderately confident in my Anderson acceleration code, but would not know if it needed to be debugged or not. The only thing I do know for sure is that the acceleration reduced the number of iterations in all cases, and still converged to the correct eigenvalue. This is in contrast to problems such as power/inverse iteration where I can verify that my results match the theory exactly

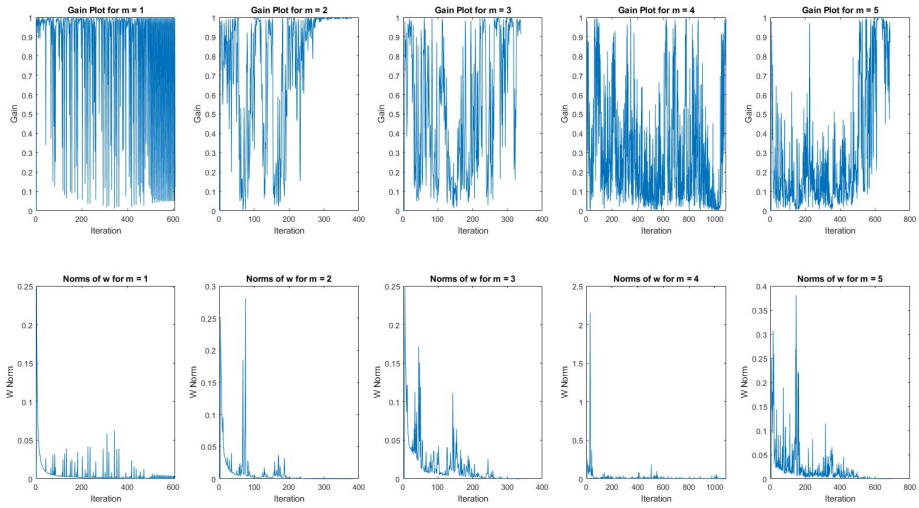

Figure 3: Subplots of gain and $\|w_k\|$ for each m

or not the results here truly match the theory.

As a whole, it is clear to see that Anderson acceleration does the linear convergence rate by a significant margin for good choices of m. Here, $m = 2$ appears to achieve the fastest convergence. The choice of optimal m should be problem dependent.

4. Use the inverse power method to find the smallest eigenvalue of the $128^2 \times 128^2$ Poisson matrix, using $x_0 = (1, 0, \ldots, 0)^T$ and $tol = 10^{-8}$. Compare the convergence results with those in problem 1, and explain why one computation converges faster than the other.

   Solution. This question/analysis is computed by the MATLAB script scriptQ4.m. The inverse method applied to a particular matrix A should converge linearly with rate $r = \lambda_1/\lambda_2$. Here, λ_1 and λ_2 are the smallest and second smallest eigenvalues in absolute value, respectively. Using MATLAB's eigs, I determined numerically that the $128^2 \times 128^2$ Poisson matrix has $\lambda_1/\lambda_2 \approx 0.4$. The eigenvectors computed by the power method should converge linearly with this rate, and the eigenvalues should converge linearly with rate $r^2 \approx 0.16$. Figure 5 shows that the eigenvalues do indeed converge linearly. Furthermore, by analyzing e_{n+1}/e_n for the computed iterates, I obtained Figure 4 which shows the observed rate matches the theoretical one, at least asymptotically.

   Note the drastic improvements in convergence despite increasing both the tolerance and size of the matrix. This is due to the fact that the two smallest eigenvalues of A are much more different in modulus, at least relatively, than the two largest eigenvalues of A are. Thus, although both are linearly convergent, the inverse method dominates here. Using a shift sufficiently close to the smallest eigenvalue would also improve the convergence rate.

Iteration	e_n/e_{n-1}
11	0.1600
12	0.1600
13	0.1600
14	0.1600
15	0.1601
16	0.1605

Figure 4: Observed Error Rates for Inverse Method

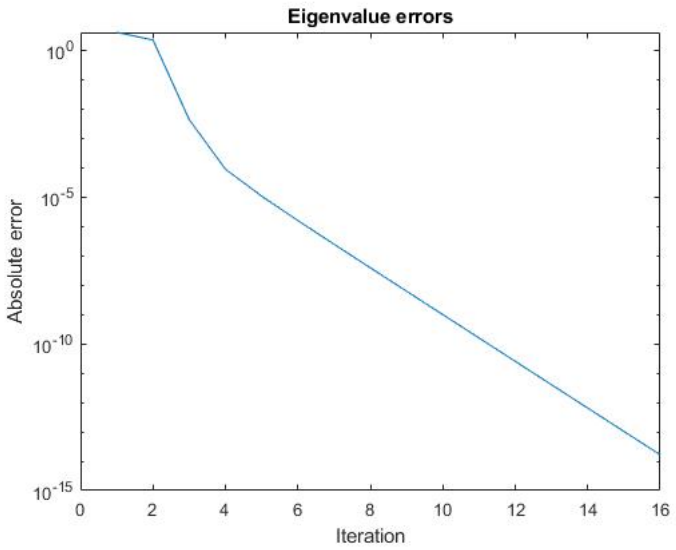

Figure 5: Plot of eigenvalue errors using inverse method computed via MATLAB script

5. Apply Anderson acceleration with $m = 1$ and $m = 2$ to the inverse power method, using the same A, tol, and x_0 from problem 4. Plot θ_k and $\|w_k\|$ for each computation.

   Solution. This question/analysis is computed by the MATLAB script scriptQ5.m. Similar to question 2, in Figure 6 it is seen that after initialization, $\|w_k\|$ effectively monotonically decreasing[2]. Since $m = 1$ and $m = 2$ here, the behavior of $\|w_k\|$ is more controlled. Furthermore, although the difference is less significant, the number of iterations required of inverse iteration (shown in Figure 5) are slightly more than that of in Figure 6. This again leads us to believe that the Anderson acceleration is indeed working properly. Lastly, it is slightly easier to see here that the gain θ_k improves whenever m increases. This is simply because as m increases, we are optimizing over a wider feasible set. Thus, there is more possibility for improvement.

6. Finally, repeat problem 5, but use MATLAB's pcg to approximately compute the inverse iteration with the prconditioner constructed by ichol, for Anderson acceleration. First use fixed tolerance of 10^{-8}, then try tolerance $tol = c\|g(x_{i-2}) - x_{i-2}\|$ with $c = 0.1, 0.01,$

[2]If the fact that $\|w_1\| = 0$ is troubling, please read the first section for clarification

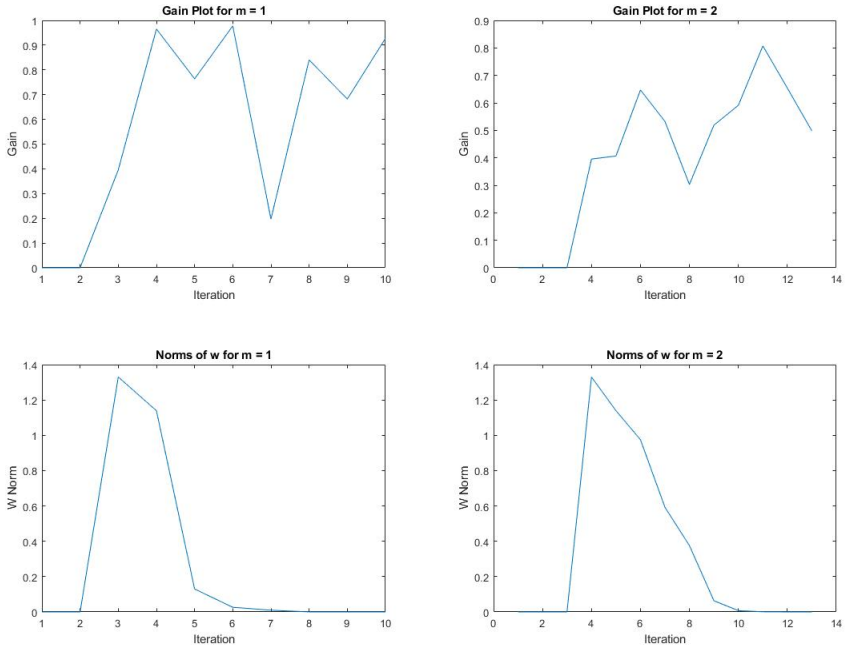

Figure 6: Subplots of gain and $\|w_k\|$ for each m

or 0.001 for pcg. Explore if the overall convergence of Anderson accleration with fixed-tolerance and variable-tolerance pcg solves is about the same.

Solution. This question/analysis is computed by the MATLAB script scriptQ6.m. Figure 7 shows that even though we use a preconditioner and pcg to solve $y_i = A^{-1}x_{i-1}$, the performance does not take a significant hit. Indeed, compare this figure with that of Figure 6 and one will see that the results are nearly identical. However, by using pcg to solve the fixed point subproblem, the cost per iteration will decrease, and thus improve timing.

Additionally, note that Figures 8,9, and 10 are practically identical. There are slight changes, but overall, the trend is the same. That is, using a preconditioner to solve the system, does not diminish convergence empirically, but can drastically reduce the time needed for each iteration. This appears to be the case whether a fixed tolerance for pcg is used or not.

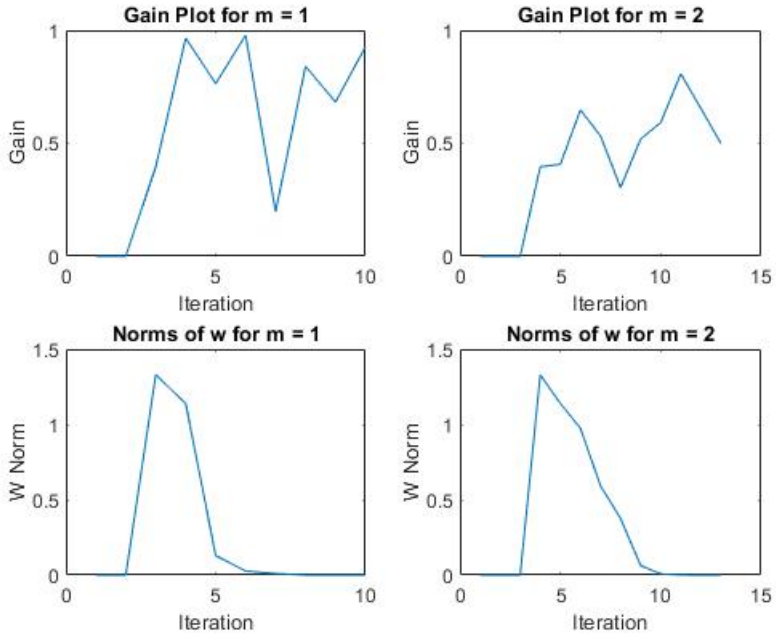

Figure 7: Subplots of gain and $\|w_k\|$ for each m with fixed tolerance PCG

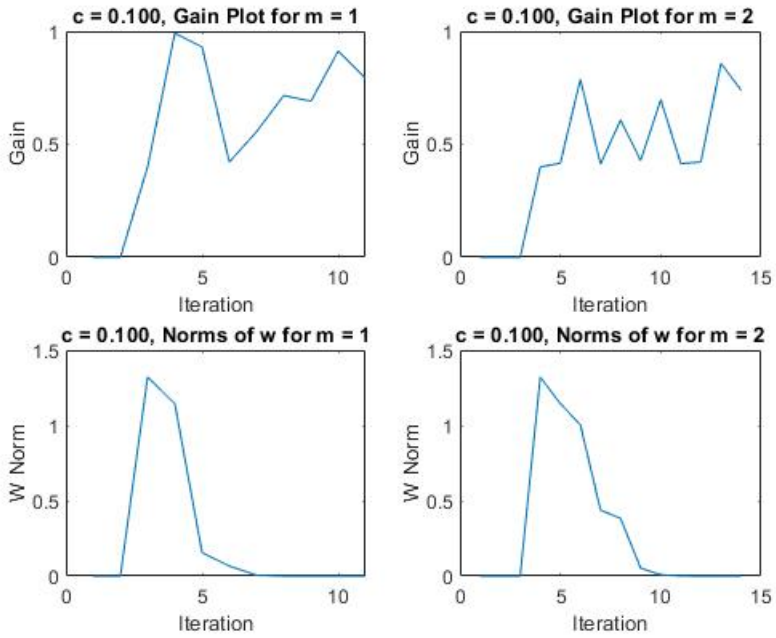

Figure 8: Subplots of gain and $\|w_k\|$ for each m with variable tolerance for $c = 0.1$

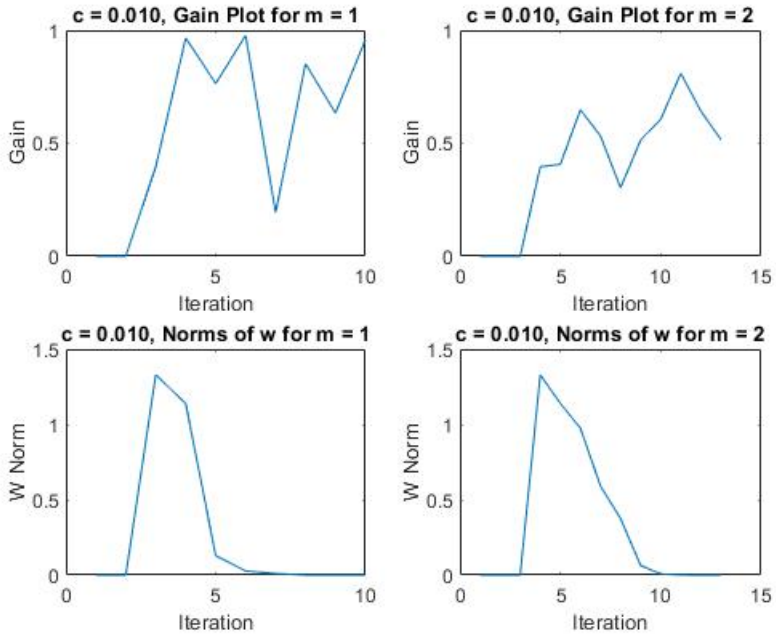

Figure 9: Subplots of gain and $\|w_k\|$ for each m with variable tolerance for $c = 0.01$

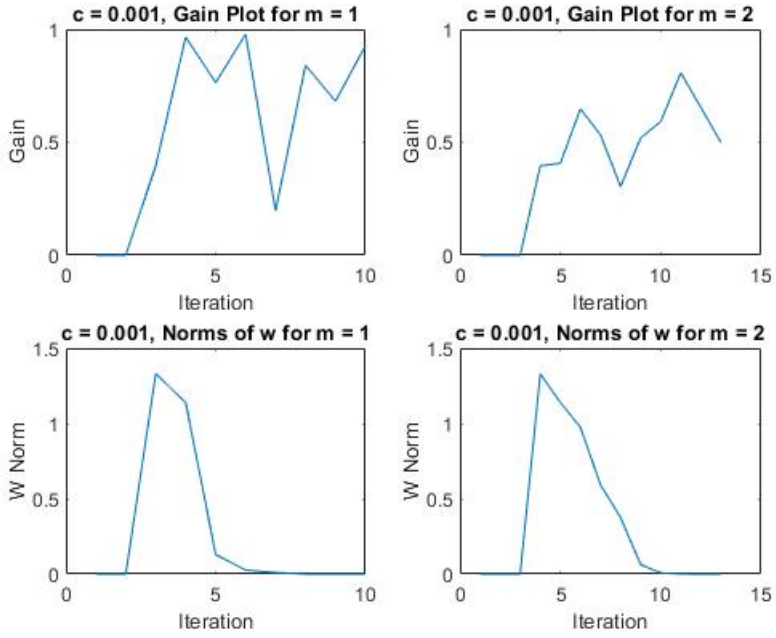

Figure 10: Subplots of gain and $\|w_k\|$ for each m with variable tolerance for $c = 0.001$