

Problems

1) In class, we showed that

$$p_{k+1} = r_{k+1} - \frac{\langle p_k, r_{k+1} \rangle_A}{\|p_k\|_A^2} p_k. \quad (1)$$

(a) Using the fact that $r_{k+1} = r_k - \alpha_k A p_k$ and $r_{k+1}^T r_k = 0$, show that $\langle p_k, r_{k+1} \rangle_A = -\frac{\|r_{k+1}\|_2^2}{\alpha_k}$.

$$\begin{aligned} 0 &= r_{k+1}^T r_k = r_{k+1}^T (r_{k+1} + \alpha_k A p_k) \\ &= r_{k+1}^T r_{k+1} + \alpha_k r_{k+1}^T A p_k \\ \implies r_{k+1} A p_k &= -\frac{r_{k+1}^T r_{k+1}}{\alpha_k} \end{aligned}$$

which implies

$$\langle p_k, r_{k+1} \rangle_A = -\frac{\|r_{k+1}\|_2^2}{\alpha_k}.$$

(b) Rewrite $\|p_k\|_A^2$ in terms of r_k and α_k .

$$\begin{aligned} \|p_k\|_A^2 &= p_k^T A p_k \\ &= \left(r_k - \frac{\langle p_{k-1}, r_k \rangle}{\|p_{k-1}\|_A^2} p_{k-1} \right)^T \frac{1}{\alpha_k} (r_k - r_{k+1}) \\ &= \frac{1}{\alpha_k} (r_k^T r_k - r_k^T r_{k+1}) \quad \text{because } p_{k-1} \text{ is orthogonal to } r_k \text{ and } r_{k+1} \\ &= \frac{1}{\alpha_k} r_k^T r_k \\ &= \frac{1}{\alpha_k} \|r_k\|_2^2. \end{aligned}$$

(c) Plug these expressions into (1) to get a technique for evaluating the next basis vector for the residual space without any applications of the matrix A .

$$\begin{aligned} p_{k+1} &= r_{k+1} - \left(-\frac{\|r_{k+1}\|_2^2}{\alpha_k} \right) \left(\frac{\alpha_k}{\|r_k\|_2^2} \right) p_k \\ &= r_{k+1} + \left(\frac{\|r_{k+1}\|_2^2}{\|r_k\|_2^2} \right) p_k. \end{aligned}$$

2) Consider a sparse 500×500 matrix A constructed as follows.

- Put a 1 in each diagonal entry.
- In each off-diagonal entry put a random number from the uniform distribution on $[-1, 1]$ but make sure to maintain symmetry. Then replace each off-diagonal entry with $|A_{ij}| > \tau$ by 0, where τ for $\tau = 0.01, 0.05, 0.1$, and 0.2 .

Take the right hand side to be a random vector b and set the tolerance to 10^{-10} .

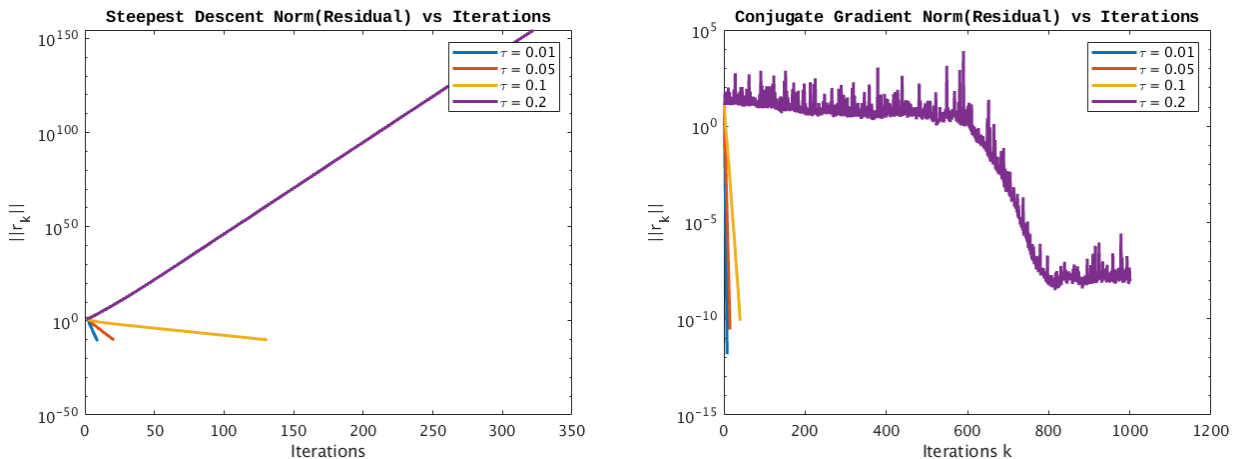


Figure 1: Convergence plot of the steepest descent method (left) and the conjugate gradient method (right).

- (a) Write the Steepest Descent (SD) and Conjugate Gradient (CG) solver.

My code is given at the end of the document.

- (b) Apply SD to solve each of the linear systems and plot the residual for each iteration $\|r_n\|$ versus the iteration n on a *semilogy* scale.

Using SD from my code, the convergence plot can be seen in left graphic of Figure 1.

- (c) Apply CG to solve each of the linear systems and plot the residual for each iteration $\|r_n\|$ versus the iteration n on a *semilogy* scale.

Using CG from my code, the convergence plot can be seen in right graphic of Figure 1.

- (d) What do you observe about the convergence of these methods? If the methods do not converge for any choices of τ explain what's happening.

In the case of my stochastic matrices, both methods converged for the first three values of τ but SD did not converge for $\tau = 0.2$. The reason SD did not converge for $\tau = 0.2$ is that the matrix was increasingly ill-conditioned which pushed each iteration in a random direction causing SD to spiral away from the solution.

For the iterations that did converge, the most prominent difference is that SD routinely more iterations to converge than CG for each τ . Furthermore, the very first iteration of SD did see much improvement in the residual while CG usually had it's greatest reduction in the residual for the very first iteration. We can explain the greatest initial improvement of CG by noting that CG resolves the solution in the direction of the eigenvectors which the greatest modulus eigenvalues. Thus, when we apply CG to each matrix, CG resolves the eigenvalues with magnitude one because they are the most prominent in each matrix. Therefore, the first iteration of CG makes the most ground in solving the system.

Another trend to note that is true for both SD and CG plots is the decrease in convergence rate as τ increases. As τ increases, the eigenvalues of each matrix become more spread out and the matrix might even obtain some negative eigenvalues which means the matrix is no longer SPD. Thus, as τ increases the conditioning of each system increases causing each method to converge at a slower rate.

(e) How do the residual compare with the error bounds provided in class?

For SD, the error bound from class states that

$$\|e_{k+1}\|_A \leq \sqrt{\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}} \|e_k\|_A.$$

For my random matrices, we have

$$\sqrt{\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}} = \begin{cases} 1.695606350621100e - 01, & \tau = 0.01 \\ 5.378379939896955e - 01, & \tau = 0.05 \\ 9.163567447907687e - 01, & \tau = 0.1 \\ 9.986824672367477e - 01, & \tau = 0.2 \end{cases}.$$

Thus, looking at our initial guess of 0 which has an approximate error of 10^1 , we would expect the first three iterations to converge in over 10 iterations because the error is roughly decreasing by 10^{-1} at each iteration. Our convergence plot SD shows this trend. Now for $\tau = 0.2$, our error coefficient is almost 1 which means the error might not decrease at all at each iteration. Furthermore, once you factor in rounding error, we might expect the solution to get worse and worse at each iteration. Looking at the convergence plot for SD shows this trend with the residuals getting larger and larger at each iteration.

For CG, our error bound from class states

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|e_0\|_A.$$

For my random matrices, we have

$$\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} = \begin{cases} 1.437837641951423e - 02, & \tau = 0.01 \\ 1.477941280879589e - 01, & \tau = 0.05 \\ 5.441933365915878e - 01, & \tau = 0.1 \\ 9.299248381593058e - 01, & \tau = 0.2 \end{cases}.$$

Using these error coefficients for $\tau = 0.01$, we would expect CG to converge in under 10 iterations because the error which starts at about 10^1 decreases by 10^{-2} at each iteration. This is exactly what our plot for the convergence of CG shows. Next, for $\tau = 0.05$ and $\tau = 0.1$, we would expect the method to converge in under 100 iterations because the error bound decreases by 10^{-1} at each iteration. Once again, the convergence plot for CG shows this trend as well. Finally, when $\tau = 0.2$, our error bound is almost given by a power of 1 which means the error might be stagnant at each iteration. The convergence plot for CG shows that for iterations up to about 600, we do have some stagnation in our convergence and then our method finally starts to hone in on the solution but rounding error keeps it from getting all the way there.

Therefore, our theoretical error bounds apply to both of our actual test problems!

- 3) Suppose CG is applied to a symmetric positive definite matrix A with the result $\|e_0\|_A = 1$, and $\|e_{10}\|_A = 2 \cdot 2^{-10}$, where $\|e_k\|_A = \|x_k - x^*\|_A$ and x^* is the true solution. Based solely on this data,

- (a) What bound can you give on $\kappa(A)$?

To compute a bound for $\kappa(A)$, note that the error bound for CG applied to SPD matrices is given by

$$\|e_k\|_A \leq 2 \left(\frac{1 - \sqrt{\frac{1}{\kappa(A)}}}{1 + \sqrt{\frac{1}{\kappa(A)}}} \right)^n \|e_0\|_A = 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^n \|e_0\|_A.$$

Then, putting our given errors together, we have

$$\|e_{10}\|_A = 2 \cdot 2^{-10} \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^{10} 1$$

which implies

$$\begin{aligned} 2^{-10} &\leq \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^{10} \\ \Rightarrow \quad \frac{1}{2} &\leq \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \\ \Rightarrow \quad \frac{1}{2} \sqrt{\kappa(A)} + \frac{1}{2} &\leq \sqrt{\kappa(A)} - 1 \\ \Rightarrow \quad \frac{3}{2} &\leq \frac{1}{2} \sqrt{\kappa(A)} \\ \Rightarrow \quad 9 &\leq \kappa(A) \end{aligned}$$

- (b) What bound can you give on $\|e_{20}\|_A$?

If we take the lower bound on $\kappa(A) = 9$, then we have the error bound

$$\|e_{20}\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^{20} \|e_0\|_A = 2 \cdot 2^{-20}.$$

In reality, we are on no ground to claim that $\kappa(A) = 9$ and so the only meaningful error bound we can make is

$$\|e_{20}\|_A \leq \|e_{10}\|_A.$$

- 4) Consider the task of solving the following system of nonlinear equations.

$$f_1(x, y) = 3x^2 + 4y^2 - 1 = 0 \text{ and } f_2(x, y) = y^3 - 8x^3 - 1 = 0$$

for the solution α near $(x, y) = (-0.5, 0.25)$.

- (a) Apply the fixed point iteration with

$$g(x) = x - \begin{pmatrix} 0.016 & -0.17 \\ 0.52 & -0.26 \end{pmatrix} \begin{pmatrix} 3x^2 + 4y^2 - 1 \\ y^3 - 8x^3 - 1 \end{pmatrix}.$$

You can use $(-0.5, 0.25)$ as the initial condition. How many steps are needed to get an approximation to 7 digits of accuracy?

Using my code (given at the end of the document), the fixed point iteration converges in 4 iterations to an answer of

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -0.497251208023980 \\ 0.254078591468912 \end{pmatrix}$$

which is surprisingly fast!

- (b) Why is this a good choice for $g(x)$.

To understand why this is a good choice for $g(x)$, let's look at the Jacobian of f_1 and f_2 at $(-0.5, 0.25)$:

$$J = \begin{pmatrix} -3 & 2 \\ -6 & 3/16 \end{pmatrix}.$$

Then, inverting J yields

$$J^{-1} = \begin{pmatrix} 1/61 & -32/183 \\ 32/61 & -16/61 \end{pmatrix} \approx \begin{pmatrix} 0.016393 & -0.174863 \\ 0.52459 & -0.262295 \end{pmatrix}.$$

Thus, J^{-1} is the almost exactly the same as the 2×2 matrix in $g(x)$. Furthermore, the vector function in $g(x)$ is just the vector function formed from f_1 and f_2 . All of this together implies that $g(x)$ is sort of Newton's Method but with a fixed inverse Jacobian. Then, because our initial solution guess is close to the true solution, $g(x)$ should almost have quadratic convergence to the solution because it is like a local Newton's method.

Code Used

Problem 2

```
1 %%
2 % Homework 6, problem 2 code
3 % Linear system solving using SD and CG
4 %
5 % Author: Caleb Jacobs
6 % Date last modified: 7-10-2021
7
8 close all
9 clear
10 format longE
11
12 %% Settings
13 tau      = [0.01; 0.05; 0.1; 0.2]; % Cutoff values
14 N        = 500;                    % Size of matrices
15 seed     = 210;                    % Random number seed
16 tol      = 1e-10;                  % Error Tolerance
17 maxIts   = 1000;                   % Maximum allowed iterations
18 x0       = zeros(N, 1);            % Initial solution guess
19
20
21 %% Linear system setup
22 % Construct test matrices
23 A = zeros(N, N, length(tau));
24 for i = 1 : length(tau)
25     A(:,:,i) = genMat(N, tau(i), seed);
26 end
27
28 % Construct b
29 b = rand(N, 1);
30
31 % True solutions for error calculations
32 xTrue = zeros(N, length(tau));
33 for i = 1 : length(tau)
34     xTrue(:, i) = A(:, :, i) \ b;
35 end
36
37 %% Driver
38 % Run steepest descent
39 figure()
40 x1 = zeros(N, length(tau));
41 for i = 1 : length(tau)
42     [x1(:, i), r] = sd(A(:,:,i), b, x0, tol, maxIts);
43
44     norm(A(:,:,i) * x1(:,i) - b);
45     semilogy(r, 'LineWidth', 2)
```

```
46     hold on
47 end
48 title('Steepest Descent Norm(Residual) vs Iterations')
49 xlabel('Iterations')
50 ylabel('||r_k||')
51 legend('\tau = 0.01', '\tau = 0.05', '\tau = 0.1', '\tau = 0.2')
52
53 % Run conjugate gradient
54 figure()
55 x2 = zeros(N, length(tau));
56 for i = 1 : length(tau)
57     [x2(:, i), r] = cg(A(:,:,i), b, x0, tol, maxIts);
58
59     norm(A(:,:,i) * x2(:,i) - b);
60     semilogy(r, 'LineWidth', 2)
61     hold on
62 end
63 title('Conjugate Gradient Norm(Residual) vs Iterations')
64 xlabel('Iterations k')
65 ylabel('||r_k||')
66 legend('\tau = 0.01', '\tau = 0.05', '\tau = 0.1', '\tau = 0.2')
67
68 %% Compute error bounds
69 % Get largest and smallest eigenvalues of each matrix
70 lambda = zeros(length(tau), 2);
71 for i = 1 : length(tau)
72     lambda(i, 1) = eigs(A(:,:,i), 1, 'smallestabs'); % Smallest modulus
       eigenvalue
73     lambda(i, 2) = eigs(A(:,:,i), 1, 'largestabs'); % Largest modulus
       eigenvalue
74 end
75
76 % Get condition number of each matrix
77 conds = zeros(length(tau), 1);
78 for i = 1 : length(tau)
79     conds(i) = cond(A(:,:,i));
80 end
81 conds
82
83 % SD error coefficients
84 sdErrCoef = zeros(length(tau), 1);
85 for i = 1 : length(tau)
86     sdErrCoef(i) = sqrt((lambda(i, 2) - lambda(i, 1)) ./ (lambda(i, 2) +
       lambda(i, 1)));
87 end
88 sdErrCoef
89
90 % CG error bounds
91 c = (sqrt(conds) - 1) ./ (sqrt(conds) + 1)
92
93 %% Generate random matrix with specified tau
```

```
94 function A = genMat(n, tau, s)
95     rng(s); % Seed random # generator
96     A = 2 * rand(n) - 1; % Generate random matrix
97
98     % Fix diagonal entries
99     for i = 1 : n
100         A(i,i) = 1;
101     end
102
103     % Strip matrix and make symmetric
104     for i = 1 : n
105         for j = i + 1 : n
106             if abs(A(i, j)) > tau
107                 A(i, j) = 0;
108             end
109
110             A(j, i) = A(i, j);
111         end
112     end
113 end
114
115 %% Steepest Descent solver
116 function [x, r] = sd(A, b, x0, tol, maxIts)
117     rk = 1 - A * x0; % Initialize residual
118
119     r = zeros(maxIts + 1, 1); % Initial residual norm storage
120     r(1) = norm(rk); % Store first residual
121
122     for i = 1 : maxIts
123         % Check convergence criteria
124         if r(i) < tol
125             break;
126         end
127
128         % Compute next iterate
129         x0 = x0 + (rk' * rk) / (rk' * A * rk) * rk;
130
131         rk = b - A * x0; % Compute new residual
132         r(i + 1) = norm(rk); % Store new residual
133     end
134
135     x = x0; % Return solution
136 end
137
138 %% Conjugate Gradient solver
139 function [x, r] = cg(A, b, x0, tol, maxIts)
140     rk = b - A * x0; % Initial residual
141     p = rk; % Initial search direction
142     rri = rk' * rk; % ||r_k||^2
143
144     r = zeros(maxIts + 1, 1); % Full residual storage
```



```

145     r(1) = sqrt(rri); % Store initial residual
146
147     for i = 1:maxIts
148         Ap = A * p; % A*p
149         a = rri / (p' * Ap); % search length
150         x0 = x0 + a * p; % Get next iterate
151         rk = rk - a * Ap; % Compute new residual
152         rrf = rk' * rk; % Compute new ||r_k||^2
153
154         r(i + 1) = sqrt(rrf); % Store new ||r_k||
155
156         % Check convergence criteria
157         if sqrt(rrf) < tol
158             break;
159         end
160
161         p = rk + (rrf / rri) * p; % Compute next search direction
162         rri = rrf; % Save ||r_k||^2
163     end
164
165     x = x0; % Return solution
166 end

```

Problem 4

```
1 %%
2 % Homework 6, problem 4 code
3 % Newton like iteration
4 %
5 % Author: Caleb Jacobs
6 % Date last modified: 07-10-2021
7
8 clear
9 close all
10 format long
11
12 %% Setup
13 f1 = @(x) 3*x(1).^2 + 4*x(2).^2 - 1;
14 f2 = @(x) x(2).^3 - 8*x(1).^3 - 1;
15 g = @(x) x - [0.016, -0.17; 0.52, -0.26] * [f1(x); f2(x)];
16
17 %% Begin fixed point iterations
18 x0 = [-0.5; 0.25];
19 for i = 1:100
20     x0 = g(x0);
21     if abs(f1(x0) - f2(x0)) < 1e-7
22         break
23     end
24 end
25
26 %% Display information
27 i
28 x0
29 f1(x0)
30 f2(x0)
```
