Problems

1) In class, we showed that

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

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

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

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.

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

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}.

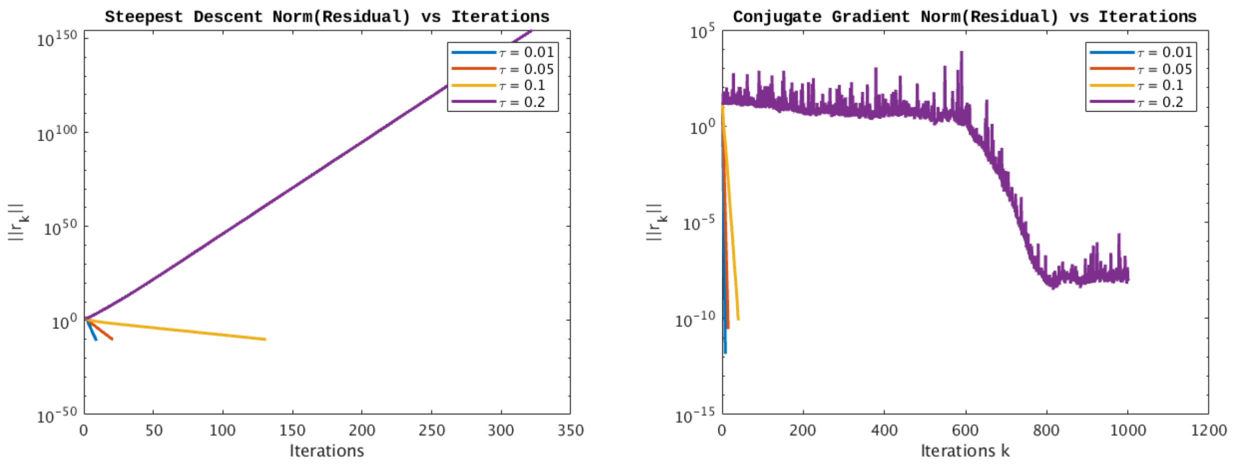

Figure 1: Convergence plot of the steepest descent method (left) and the conjugate gradient method (right).

(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

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

   (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

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

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

```
%%
% Homework 6, problem 2 code
% Linear system solving using SD and CG
%
% Author: Caleb Jacobs
% Date last modified: 7-10-2021

close all
clear
format longE

%% Settings
tau    = [0.01; 0.05; 0.1; 0.2];  % Cutoff values
N      = 500;                     % Size of matrices
seed   = 210;                     % Random number seed
tol    = 1e-10;                   % Error Toloerance
maxIts = 1000;                    % Maximum allowed iterations
x0     = zeros(N, 1);             % Initial solution guess


%% Linear system setup
% Construct test matrices
A = zeros(N, N, length(tau));
for i = 1 : length(tau)
    A(:,:,i) = genMat(N, tau(i), seed);
end

% Construct b
b = rand(N, 1);

% True solutions for error calculations
xTrue = zeros(N, length(tau));
for i = 1 : length(tau)
    xTrue(:, i) = A(:, :, i) \ b;
end

%% Driver
% Run steepest descent
figure()
x1 = zeros(N, length(tau));
for i = 1 : length(tau)
    [x1(:, i), r] = sd(A(:,:,i), b, x0, tol, maxIts);

%     norm(A(:,:,i) * x1(:,i) - b);
    semilogy(r, 'LineWidth', 2)
    hold on
end
title('Steepest Descent Norm(Residual) vs Iterations')
xlabel('Iterations')
ylabel('||r_k||')
legend('\tau = 0.01', '\tau = 0.05', '\tau = 0.1', '\tau = 0.2')

% Run conjugate gradient
figure()
x2 = zeros(N, length(tau));
for i = 1 : length(tau)
    [x2(:, i), r] = cg(A(:,:,i), b, x0, tol, maxIts);

%     norm(A(:,:,i) * x2(:,i) - b);
    semilogy(r, 'LineWidth', 2)
    hold on
end
title('Conjugate Gradient Norm(Residual) vs Iterations')
xlabel('Iterations k')
ylabel('||r_k||')
legend('\tau = 0.01', '\tau = 0.05', '\tau = 0.1', '\tau = 0.2')

%% Compute error bounds
% Get largest and smallest eigenvalues of each matrix
lambda = zeros(length(tau), 2);
for i = 1 : length(tau)
    lambda(i, 1) = eigs(A(:,:,i), 1, 'smallestabs');   % Smallest modulus
   eigenvalue
    lambda(i, 2) = eigs(A(:,:,i), 1, 'largestabs');    % Largest modulus
   eigenvalue
end

% Get condition number of each matrix
conds = zeros(length(tau), 1);
for i = 1 : length(tau)
    conds(i) = cond(A(:,:,i));
end
conds

% SD error coefficients
sdErrCoef = zeros(length(tau), 1);
for i = 1 : length(tau)
    sdErrCoef(i) = sqrt((lambda(i, 2) - lambda(i, 1)) ./ (lambda(i, 2) +
   lambda(i, 1)));
end
sdErrCoef

% CG error bounds
c = (sqrt(conds) - 1) ./ (sqrt(conds) + 1)

%% Generate random matrix with specified tau
```

```matlab
function A = genMat(n, tau, s)
    rng(s);                                 % Seed random # generator
    A = 2 * rand(n) - 1;                    % Generate random matrix

    % Fix diagonal entries
    for i = 1 : n
        A(i,i) = 1;
    end

    % Strip matrix and make symmetric
    for i = 1 : n
        for j = i + 1 : n
            if abs(A(i, j)) > tau
                A(i, j) = 0;
            end

            A(j, i) = A(i, j);
        end
    end
end

%% Steepest Descent solver
function [x, r] = sd(A, b, x0, tol, maxIts)
    rk = 1 - A * x0;                % Initialize residual

    r = zeros(maxIts + 1, 1);       % Initial residual norm storage
    r(1) = norm(rk);                % Store first residual

    for i = 1 : maxIts
        % Check convergence criteria
        if r(i) < tol
            break;
        end

        % Compute next iterate
        x0 = x0 + (rk' * rk) / (rk' * A * rk) * rk;

        rk = b - A * x0;            % Compute new residual
        r(i + 1) = norm(rk);        % Store new residual
    end

    x = x0;                         % Return solution
end

%% Conjugate Gradient solver
function [x, r] = cg(A, b, x0, tol, maxIts)
    rk  = b - A * x0;                       % Initial residual
    p   = rk;                               % Initial search direction
    rri = rk' * rk;                         % ||r_k||^2

    r    = zeros(maxIts + 1, 1);            % Full residual storage
    r(1) = sqrt(rri);                         % Store initial residual

    for i = 1:maxIts
        Ap  = A * p;                          % A*p
        a   = rri / (p' * Ap);                % search length
        x0  = x0 + a * p;                     % Get next iterate
        rk  = rk - a * Ap;                    % Compute new residual
        rrf = rk' * rk;                       % Compute new ||r_k||^2

        r(i + 1) = sqrt(rrf);                 % Store new ||r_k||

        % Check convergence criteria
        if sqrt(rrf) < tol
            break;
        end

        p = rk + (rrf / rri) * p;             % Compute next search direction
        rri = rrf;                            % Save ||r_k||^2
    end

    x = x0;                                   % Return solution
end
```

Problem 4

```
%%
% Homework 6, problem 4 code
% Newton like iteration
%
% Author: Caleb Jacobs
% Date last modified: 07-10-2021

clear
close all
format long

%% Setup
f1 = @(x) 3*x(1).^2 + 4*x(2).^2 - 1;
f2 = @(x) x(2).^3 - 8*x(1).^3 - 1;
g  = @(x) x - [0.016, -0.17; 0.52, -0.26] * [f1(x); f2(x)];

%% Begin fixed point iterations
x0 = [-0.5; 0.25];
for i = 1:100
    x0 = g(x0);
    if abs(f1(x0) - f2(x0)) < 1e-7
        break
    end
end

%% Display information
i
x0
f1(x0)
f2(x0)
```