Name: Hongda Li
AMATH 585 WINTER 2022 HW 6

Problem 1

Any function with chebyshev coefficients $a_0, a_1, \cdots, a_n$, evaluated at the chbyshev node is given as:

$$p(\cos(k\pi/n)) = \sum_{j=0}^{n} a_j \cos(jk\pi/n) \tag{1}$$

Our objective here is make use hf the FFT algorithm for DFT for the objective of: Interpolation the function at chebyshev node getting the values of $a_0, a_1, \cdots, a_n$, and evaluating the function value at the chebyshev nodes using the FFT algorithm. It's implied that k, n are integers in this context.

My claim here is that, if we tiled the vector in the following format:

$$\vec{f} = [a_0, a_1, \cdots, a_{n-1}, a_n, a_{n-1}, \cdots, a_1]$$

It's symmetric exclusing the first element, and then we put this into the DFT algorithm using FFT, then we obtain the following relationsship:

$$\frac{1}{2}(F_k + a_0 + (-1)^k) = \sum_{j=0}^{n} \cos\left(\frac{\pi jk}{n}\right) a_j \quad \forall 0 \le k \le n \tag{1.1}$$

Here, we assume that the vector $\vec{F} = [F_0, F_1, \cdots F_{2n-1}]$ are the output of the DFT after we feed $\vec{f}$ into the algorithm.

Before we prove (1.1) we wish to establish some basics about the vector $\vec{f}$. Observe that the vector is symmetric if we exclude the first argument, which means that $f_j = f_{2n-j}\ \forall\ 1 \le j \le 2n-1$. Next, the vector $\vec{f}$ has a total length of $2n$. And when we index the vector $\vec{f}, \vec{F}$, we let the **index starts with zero**.

First, consider the following algebra:

$$\begin{aligned}
\exp\left(-i\frac{\pi(2n-j)k}{n}\right) &= \exp\left(-i\frac{2\pi n - j\pi k}{n}\right) \\
&= \exp\left(-i\frac{2n\pi n}{n} + \frac{ij\pi k}{n}\right) \\
&= \exp\left(i\frac{jk\pi}{n}\right) \\
\sum_{j=0}^{2n-1} \exp\left(-i\frac{i\pi jk}{n}\right) &= \sum_{j=1}^{2n} \exp\left(-i\frac{2\pi(2n-j)k}{n}\right)
\end{aligned} \tag{1.2}$$

The second equality is just a trick where I swapp the index so it starts summing in the reverse order.

Now consider the DFT on vector $\vec{f}$, which by definition would be given as:

$$\begin{aligned}
F_k &= \sum_{j=0}^{2n-1} \exp\left(-i\frac{2\pi jk}{2n}\right) f_j = \sum_{j=0}^{2n-1} \exp\left(-i\frac{\pi jk}{n}\right) f_j \\
&= \frac{1}{2}\left(\sum_{j=0}^{2n-1} \exp\left(-i\frac{\pi jk}{n}\right) f_j + \sum_{j=1}^{2n} \exp\left(-i\frac{2\pi(2n-j)k}{n}\right) \underbrace{f_{2n-j}}_{=f_j}\right) \\
&= \frac{1}{2}\left(\sum_{j=0}^{2n-1} \exp\left(-i\frac{\pi jk}{n}\right) f_j + \sum_{j=1}^{2n} \exp\left(i\frac{jk\pi}{n}\right) f_j\right) \Longleftarrow \quad \text{by: (1.2)} \\
&= \frac{1}{2}\left(2f_0 + \sum_{j=1}^{2n-1} \exp\left(-i\frac{\pi jk}{n}\right) f_j + \sum_{j=1}^{2n-1} \exp\left(i\frac{jk\pi}{n}\right) f_j\right) \\
&= f_0 + \sum_{j=1}^{2n-1} \cos\left(\frac{\pi jk}{n}\right) f_j
\end{aligned} \tag{1.3}$$

Next, please observe the fact that the term for $j = 1$ equals to $j = 2n - 1$, due to the symmetry of cos and the symmetry of vector f_j $\forall 1 \le j \le 2n - 1$. And hence we obtained:

$$F_k = a_0 + \left(2\sum_{j=1}^{n-1} \cos\left(\frac{\pi jk}{n}\right) a_j\right) + (-1)^k a_n \tag{1.4}$$

Here, take note of the extra term, when $j = n$, $f_j = n$, which is right in the middle of the symmetric part of $\vec{f}$, and it only repeats once, so I take it out from the sum and it produces the term $(-1)^k a_n$. All other terms repeats 2 times and $f_0 = a_0$. Rearranging the above equation we have:

$$\begin{aligned}
\frac{1}{2}\left(F_k - a_0 - (-1)^k a_n\right) &= \sum_{j=1}^{n-1} \cos\left(\frac{\pi jk}{n}\right) a_j \\
\frac{1}{2}\left(F_k + a_0 + (-1)^k a_n\right) &= \sum_{j=0}^{n} \cos\left(\frac{\pi jk}{n}\right) a_j \\
\frac{1}{2}\left(F_k + a_0 + (-1)^k a_n\right) &= p\left(\cos\left(\frac{k\pi}{n}\right)\right)
\end{aligned} \tag{1.5}$$

From the frist line to the second line, I added $a_0, (-1)^k a_n$ to both side of the equation. At this point, we have proven that (1.1) is true, and we can make use of the algorithm fast evaluate the chebyshev series at the chebyshev nodes. Simply make the vector $\vec{f}$ as said above, and then evalute it to get F_k, and then use that above expression, for $k = 0, \cdots, n$. There will be $2n$ output vectors, but we can ignore the part where it gets symmetric.

Next, to reverse the process for looking for the chebyshee coefficients, we simply consider: "What is F_k"? And then make use of the IDFT algorithm which uses IFFT.

$$\begin{aligned}
p\left(\cos\left(\frac{k\pi}{n}\right)\right) &= \frac{1}{2}\left(F_k + a_0 + (-1)^k a_n\right) \\
2p\left(\cos\left(\frac{k\pi}{n}\right)\right) &= F_k + a_0 + (-1)^k a_n \\
2p\left(\cos\left(\frac{k\pi}{n}\right)\right) - a_0 - (-1)^k a_n &= F_k
\end{aligned} \tag{1.6}$$

Do this for $k = 0, \cdots, 2n - 1$ and then invoke the IFDT using FFT, and then we get back the veoctr $\vec{f}$, and the first $n + 1$ elements are the chbyshev coefficients.

Problem 2

Code

```
close all; clear all; clc;

%% Stationary Iterative Methods Convergence.
[A, b] = MakeTestProblem(8);
[~, errsJB] = StationaryIterative(A, b);
[~, errsGS] = StationaryIterative(A, b, [], "gs");
[~, errsSOR] = StationaryIterative(A, b, [], "sor");
ErrsJBLogged = log10(errsJB);
ErrsGSLogged = log10(errsGS);
ErrsSORLogged = log10(errsSOR);
figure;
plot(ErrsJBLogged,"-+");
hold on;
plot(ErrsGSLogged, "-o");
plot(ErrsSORLogged, "-.");
legend(["JB", "GS","SOR"]);
xlabel("Iteration");
ylabel("Log10 of Relative Residual");
title("Stationary Iterative Method Convergence");
saveas(gcf, "stationary_methods.png");
% Stationary Iterative Method: Convergence estimate wrt h, the grid size.

%% CG with and without and with Preconditioning.
[A, b] = MakeTestProblem(20);
[~, ErrsCG] = PerformCG(A, b);
[~, ErrsPCG] = PerformCG(A, b, 1);
figure;
plot(log10(ErrsCG));
hold on;
plot(log10(ErrsPCG));
xlabel("Iteration Count");
ylabel("Log10 Relative Residual");
legend(["cg", "pcg+ichol"]);
saveas(gcf, "pcg_vs_cgs_itr.png");

%% Convergence rate wrt to h the grid size for all method.
GridDivisions = 5:30;
ItrJB = [];
ItrGS = [];
ItrSOR = [];
ItrCGS = [];
ItrPCG = [];
for n = GridDivisions
    ItrJB(end + 1) = GetIterationCountForMethod(@StationaryIterative, n);
    ItrGS(end + 1) = GetIterationCountForMethod( ...
        @(A, b) StationaryIterative(A, b, [], "gs"), n ...
        );
    ItrSOR(end + 1) = GetIterationCountForMethod( ...
        @(A, b) StationaryIterative(A, b, [], "sor"), n ...
        );
    ItrCGS(end + 1) = GetIterationCountForMethod(@PerformCG, n);
    ItrPCG(end + 1) = GetIterationCountForMethod( ...
        @(A, b) PerformCG(A, b, 1), n ...
    );
end

%% Plotting it out.
close all;
figure;
plot(GridDivisions, ItrJB);hold on
plot(GridDivisions, ItrGS);
plot(GridDivisions, ItrSOR);
plot(GridDivisions, ItrCGS);
```

```
plot(GridDivisions, ItrPCG);
legend(["JB", "GS", "SOR", "CGS", "PCG"], "location", "northwest");
xlabel("Number of Grids Partition on one Dimension");
ylabel("Iterations of Methods");
title("Iteration Count vs Grid Division")
saveas(gcf, "h_vs_methods_itr.png");

figure;
loglog(GridDivisions, ItrJB, "-x");hold on
loglog(GridDivisions, ItrGS, "-o");
loglog(GridDivisions, ItrSOR, "-.");
legend(["JB", "GS", "SOR"], "location", "northwest");
xlabel("Log of Iteration count");
ylabel("Log of Numer of Grid Partition on one Dimension");
title("The Stationary Methods");
saveas(gcf, "h_vs_stationary_methods.png");


figure;
loglog(GridDivisions, ItrPCG, '-x'); hold on ;
loglog(GridDivisions, ItrSOR, '-o');
legend(["pcg", "sor"]);
xlabel("Log of Iteration count");
ylabel("Log of Numer of Grid Partition on one Dimension");
title("The PGC and SOR")
saveas(gcf, "h_vs_pcg_sor.png");



%% Numerically Compute it.
% Iteration vs number of grid division on one dimension
ConvergenceRateSOR = LogLogSlopeEstimate(GridDivisions, ItrSOR);
disp(ConvergenceRateSOR);
ConvergenceRateGS = LogLogSlopeEstimate(GridDivisions, ItrGS);
disp(ConvergenceRateGS);
ConvergenceRatePCG = LogLogSlopeEstimate(GridDivisions, ItrPCG);
disp(ConvergenceRatePCG);




function [soln, RelativeErrs] = PerformCG(A, b, precon)
    if nargin < 3
        precon = 0;
    end
    tol = 1e-8;
    if precon == 0
        [soln, FLAG, ~, ~, RelativeErrs] = cgs(A, b, tol, length(b)*10);
        disp("The Flag for cgs is: ");
        disp(num2str(FLAG));
    else
        L = ichol(A);
        M = L*L';
        [soln, FLAG, ~, ~, RelativeErrs] = pcg(A, b, tol, length(b)*10, M);
        disp("The Flag for pcg is: ");
        disp(num2str(FLAG));
    end
end

function TotalItr = GetIterationCountForMethod(fxn, n)
    [A, b] = MakeTestProblem(n);
    [~, Errs] = fxn(A, b);
    TotalItr = length(Errs);
end

function slope = LogLogSlopeEstimate(x, y)
    slope = mean( ...
            (log(y(1:end - 1)) - log(y(2: end))) ...
```

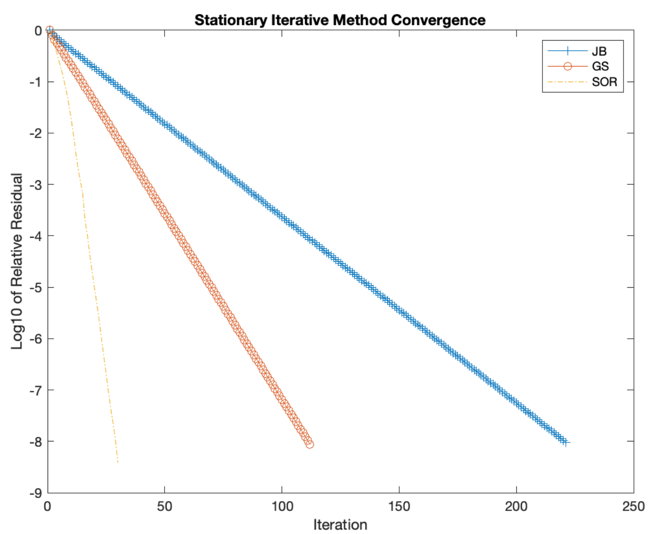
```
            /(log(x(1: end - 1)) - log(x(2: end))) ...
        );
end

function [A, b] = MakeTestProblem(n)
%
%  Solves the steady-state heat equation in a square with conductivity
%  c(x,y) = 1 + x^2 + y^2:
%
%     -d/dx( (1+x^2+y^2) du/dx ) - d/dy( (1+x^2+y^2) du/dy ) = f(x),
%                                                        0 < x,y < 1
%     u(x,0) = u(x,1) = u(0,y) = u(1,y) = 0
%
%  Uses a centered finite difference method.

%  Set up grid.
% n = input(' Enter number of subintervals in each direction: ');

    h = 1/n;
    N = (n-1)^2;

    %  Form block tridiagonal finite difference matrix A and right-hand side
    %  vector b.

    A=sparse(zeros(N,N));
    b = ones(N,1);
    % Use right-hand side vector of all 1's.

    %  Loop over grid points in y direction.

    for j=1:n-1
      yj = j*h;
      yjph = yj+h/2;  yjmh = yj-h/2;

    %    Loop over grid points in x direction.

      for i=1:n-1
        xi = i*h;
        xiph = xi+h/2;  ximh = xi-h/2;
        aiphj = 1 + xiph^2 + yj^2;
        aimhj = 1 + ximh^2 + yj^2;
        aijph = 1 + xi^2 + yjph^2;
        aijmh = 1 + xi^2 + yjmh^2;
        k = (j-1)*(n-1) + i;
        A(k,k) = aiphj+aimhj+aijph+aijmh;
        if i > 1
            A(k,k-1) = -aimhj;
        end
        if i < n-1
            A(k,k+1) = -aiphj;
        end
        if j > 1
            A(k,k-(n-1)) = -aijmh;
        end
        if j < n-1
            A(k,k+(n-1)) = -aijph;
        end
      end
    end
    A = (1/h^2)*A;   % Remember to multiply A by (1/h^2).
    % Solve linear system.

end

function [soln, RelativeResErr] = StationaryIterative(A, b, epsilon, arg3, w, x0, maxitr)
%%% Function performs stational iterations
%%% INTPUT:
%%%     A:
```

```
%%%         The sparse matrix for performing the vector operation.
%%%     b:
%%%         The b vector for the RHS of the system.
%%%     arg3:
%%%         The type of method that we are using for the system. By default
%%%         it uses the Jacobi iteration if this is not set.
%%%
%%% OUTPUT:
%%%     soln: The solution in the end.
%%%     RelativeResErr: List of ||Ax - b||/||b|| during the iteration.


    if ~exist("epsilon", "var") || isempty(epsilon)
        epsilon = 1e-8;
    end
    if ~exist("arg3", "var") || isempty(arg3)
        arg3 = "jb";
    end
    if ~exist("w", "var") || isempty(w)
        w = 1.5;
    end
    if ~exist("x0", "var") || isempty(x0)
        x0 = zeros(size(b));
    end
    if ~exist("maxitr", "var") || isempty(maxitr)
        maxitr = max(2*size(b, 1), 1000);
    end

    L = tril(A, -1); U = triu(A, 1); d = diag(A);
    D = diag(d);

    if ~any(["jb", "gs", "sor"] == arg3)
        error("arg3 must be one of the following: jb, gs, sor. ")
    else
        maxEig = abs(eigs((L + U)./d, 1, 'largestabs'));
        if arg3 == "jb"
            disp("Stationary Iterative Method: jb")
            if maxEig > 1 || isnan(maxEig)
                disp("Jacobi might not converge, max eig of the matrix is:");
                disp(num2str(maxEig));
                disp("Here is the plot of the matrix")
                figure;
                imagesc((L + U)./d); colorbar;
            end
        else
            if arg3 == "gs"
                disp("Staionary Iterative method: GS");
                w = 1;
            else  % sor
                disp("Stationary Iterative method: SOR");
                w = 2/(1 + sqrt(1 - maxEig^2));
                disp("sor determined relaxtion factor is: ");
                disp(num2str(w));
            end
        end
    end

    RelativeResErr = zeros(1, maxitr);
    RelativeResErr(1) = norm(b - A*x0)/norm(b);


    for Itr = 1: maxitr
        if arg3 == "jb"
            x0 = (b - (L + U)*x0)./d;
        else  % gs or sor
            x0 = (D + w*L)\(w*b - (w*U + (w - 1)*D)*x0);
        end
```

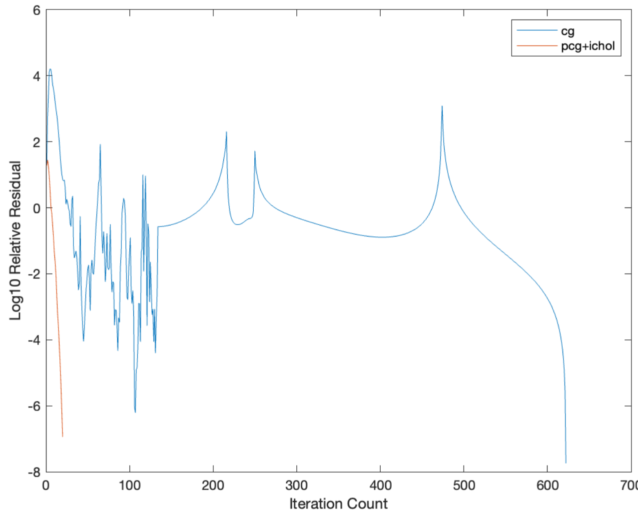
```
        RelativeResErr(Itr + 1) = norm(b - A*x0)/norm(b);

        if RelativeResErr(Itr + 1) < epsilon
            break;
        end

        if isnan(RelativeResErr(Itr + 1)) || isinf(RelativeResErr(Itr + 1))
            disp("Error during stationary iteration hit nan or inf.");
            break;
        end

    end

    RelativeResErr = RelativeResErr(1:Itr + 1);
    soln = x0;

end
```

Error for all method are measured using the relative error of the residual, which is given as $\|Ax_k - b\|/\|b\|$, and it's put under $\log 10$. All methods tolerance are set to be 10^{-8}.

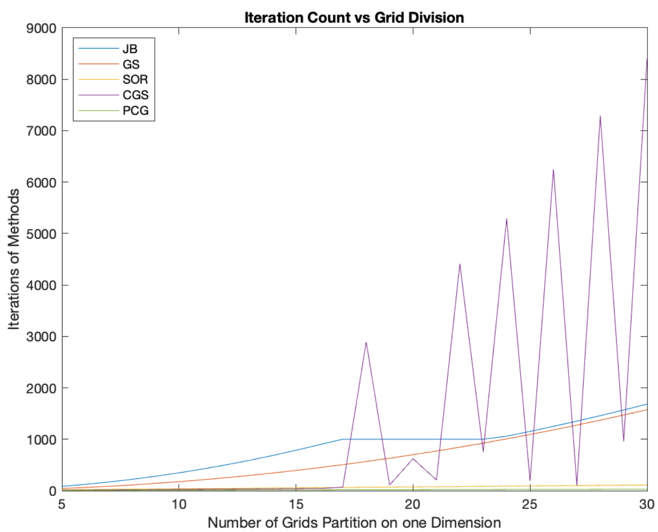
Results are produced. This is a plot of the convergence of the errors for Staionary Iterative Methods: "JB, GS, SOR":



Take note that, I computed the relexation factor using the maximal absolute eigen value of the Jacobi Split matrix: $-D^{-1}(L+U)$. And then used the formula: $\omega_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho(G_j)^2}}$, which gives the fast descend of the error.

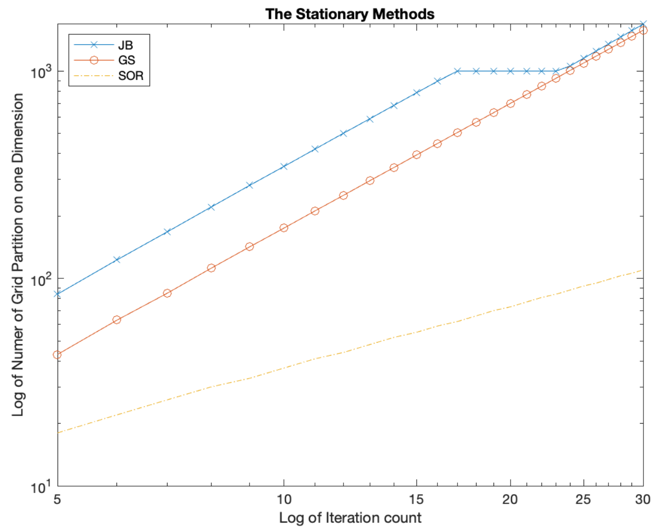


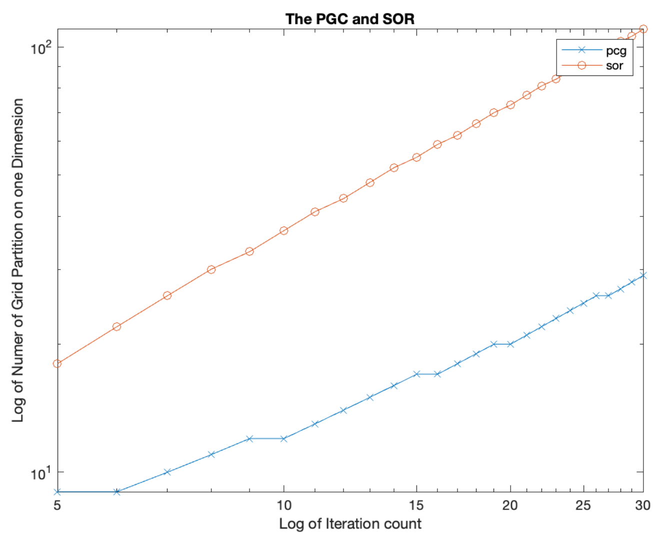
This is the error of the Conjugate Gradient with and without preconditioneer. For the Preconditioner, we use Incomplete Cholesky Factorization. Observe that the relative residual is not monotonically decreasing for CG, because CG aims for minization on the energy norm induces by matrix A, and we are the relative residual error is not measured under that norm, therefore, it looks wacky like that.



This is a plot of all methods iterations count versus the number of division along one dimension of the grid, which is basically $1/h$.

Please observe that, there are wilde oscilations (I don't know why) on the Conjugate Gradient methods, but the over all tenency grows quadratically. This is expected becaue the total number of elements in the matrix is about m^2 where m is the number of divisions along one dimension.





For further investiation, I plotted out the loglog of iteration count against the number of division along one axis. I computed the slope under the log log plot and obtained:

1. SOR: 1.0259
2. GS: 2.0179
3. PCG: 0.5556

I can explain the first 2, but the third one is one information for me. The SOR is expected to have this relation because how the spectrum of the iteration matrix, denoted by $\rho(G_{\text{sor}}) = 1 - \mathcal{O}(h)$, and for GS, the spectrum of the iteration matrix $\rho(G_{GS}) = 1 - \mathcal{O}(h^2)$. And this is the reason why the above plot is observed. However, this doesn't mean that PC is necessary faster than all methods, because the Implete Cholesky is already a at least $O(m^2)$ cost. But it's obvious that CGS doesn't work well consistently, and it's on the same ballmark compare to GS and JB.

Here is a short justification for how SOR method has this relation between log of iteration cout and the log of number of discretizations along one dimension. Let n denotes the number of iterations, h denotes the width of the grid, and $h = 1/m$, and ϵ denotes the tolerance of the method.

$$\begin{aligned}
(1 - \mathcal{O}(h))^n &< \epsilon \\
1 - n\mathcal{O}(h) &< \epsilon \\
1 + \epsilon &< n\mathcal{O}(h) \\
\frac{1-\epsilon}{\mathcal{O}(h)} &< n \\
\implies n \propto m &\propto \frac{1}{n}
\end{aligned} \tag{2.1}$$

This implies a linear relations between the numbter of grid discritizations and the number of steps required for the SOR method to converge, it's expected to be a linear function, which matches with our experiement. A similar relations can also be derived for the GS method, and we will get back a quadratic relations.

Problem 3