AMATH 585 Homework 5

Cade Ballew

March 2, 2022

1 Problem 1

Say that we know coefficients $a_0, \dots, a_n$ and wish to evaluate

$$p(\cos(k\pi/n)) = \sum_{j=0}^{n} a_j \cos(jk\pi/n), \quad k = 0, \dots, n.$$

using the FFT

$$F_k = \sum_{j=0}^{n-1} e^{2\pi ijk/n} f_j, \quad k = 0, \dots, n-1.$$

To go about doing this, we extend our vector of coefficients a reversing the order and appending to the end. Namely, we let $a_{2n-j} = a_j$ for $j = 1, \dots, n-1$. Applying the FFT to this extended vector,

$$\begin{aligned} A_k = \sum_{j=0}^{2n-1} e^{2\pi ijk/(2n)} a_j &= \sum_{j=0}^{n} a_j e^{\pi ijk/n} + \sum_{j=n+1}^{2n-1} a_j e^{\pi ijk/n} \\ &= a_0 + \sum_{j=1}^{n-1} a_j e^{\pi ijk/n} + a_n e^{\pi ik} + \sum_{j=n+1}^{2n-1} a_j e^{\pi ijk/n} \\ &= a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j e^{\pi ijk/n} + \sum_{j=n+1}^{2n-1} a_j e^{\pi ijk/n}. \end{aligned}$$

Now, we perform a change of variables $j \to 2n - j$ on the second summation to get

$$\begin{aligned}
A_k &= a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j e^{\pi ijk/n} + \sum_{j=2n-(n+1)}^{2n-(2n-1)} a_{2n-j} e^{\pi i(2n-j)k/n} \\
&= a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j e^{\pi ijk/n} + \sum_{j=1}^{n+1} a_{2n-j} e^{\pi i(2n-j)k/n} \\
&= a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j e^{\pi ijk/n} + \sum_{j=1}^{n+1} a_j e^{2\pi ik - \pi ijk/n} \\
&= a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j e^{\pi ijk/n} + \sum_{j=1}^{n+1} a_j e^{-\pi ijk/n} \\
&= a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j \left(e^{\pi ijk/n} + \overline{e^{\pi ijk/n}} \right) = a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j 2\Re e^{\pi ijk/n} \\
&= a_0 + a_n(-1)^k + 2\sum_{j=1}^{n-1} a_j \cos(\pi ijk/n).
\end{aligned}$$

Thus,

$$a_0 + a_n(-1)^k + A_k = 2\left(a_0 + a_n(-1)^k + \sum_{j=1}^{n-1} a_j \cos(\pi ijk/n) \right) = 2\sum_{j=0}^{n} a_j \cos(\pi ijk/n),$$

meaning that we can compute this sum by computing A_k via the FFT and letting

$$p(\cos(k\pi/n)) = \sum_{j=0}^{n} a_j \cos(\pi ijk/n) = \frac{a_0 + a_n(-1)^k + A_k}{2}.$$

Note that our complexity is still $\mathrm{O}(n \log n)$, because adding the correction terms and dividing by 2 is certainly less work and $\mathrm{O}(2n \log 2n) = \mathrm{O}(2n \log 2 + 2n \log n) = \mathrm{O}(n + n \log n) = \mathrm{O}(n \log n)$.

To go in the reverse direction is analogous. Namely, let $P_k = p(\cos(k\pi/n))$ and let P denote the extended vector where $P_{2n-k} = P_k$ for $k = 1, \ldots, n-1$. Then, we can apply the IFFT to P to get p_j and can compute

$$a_j = \frac{P_0 + P_n(-1)^j + p_j}{2}.$$

One can see that this holds by the same argument as the forward direction where the FFT is replaced with the IFFT.

2 Problem 2

Using the provided MATLAB code to create the matrix equation required to solve the BVP

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(a(x,y)\frac{\partial u}{\partial y}\right)=f(x,y) \quad \text{in } (0,1)\times(0,1)$$

$$u(x,0)=u(x,1)=u(0,y)=u(1,y)=0,$$

where $a(x,y)=1+x^2+y^2$ and $f(x,y)=1$ with finite differences, we use the following MATLAB function to apply simple iterations to $Ax=b$ for a given matrix M.

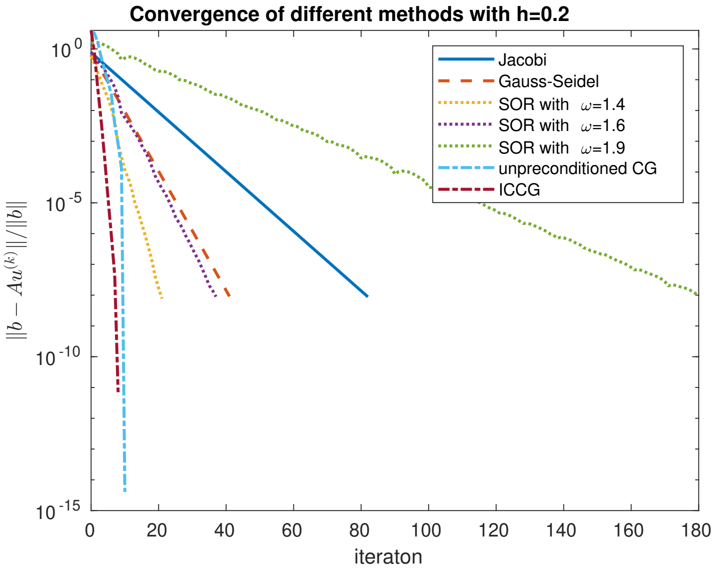
```
function [x,iter,residvec] = simpleiter(A,b,M,x0,tol,maxiter)
x = x0; r = b-A*x0;
z = M\r;
errnorm = norm(r);
iter = 0;
nb = norm(b); %save norm(b)
while errnorm/nb>tol && iter<maxiter
    x = x + z;
    r = b - A*x;
    z = M\r;
    iter = iter + 1;
    errnorm = norm(r);
    residvec(iter) = errnorm/nb; %relative residual
end
end
```

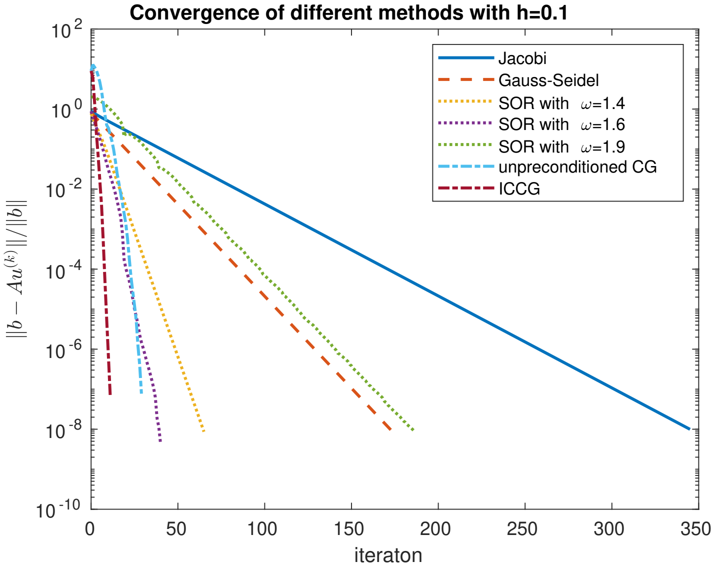
If we write $A=D-L-U$ where D is diagonal and L and U are strictly lower and upper triangular, respectively, then we get Jacobi by taking $M=D$, Gauss-Seidel by $M=D-L$, and SOR by $M=\omega^{-1}D-L$ for some $\omega\in(0,2)$. We apply these methods and use MATLAB's `pcg` to compute both an unpreconditioned conjugate gradient (set the preconditioner to the identity) and incomplete Cholesky conjugate graident using MATLAB's `ichol`. The following code segment executes this after our system is built. Note that we set our maximum iterations at 1000 and tolerance for the relative error at 10^{-8}.

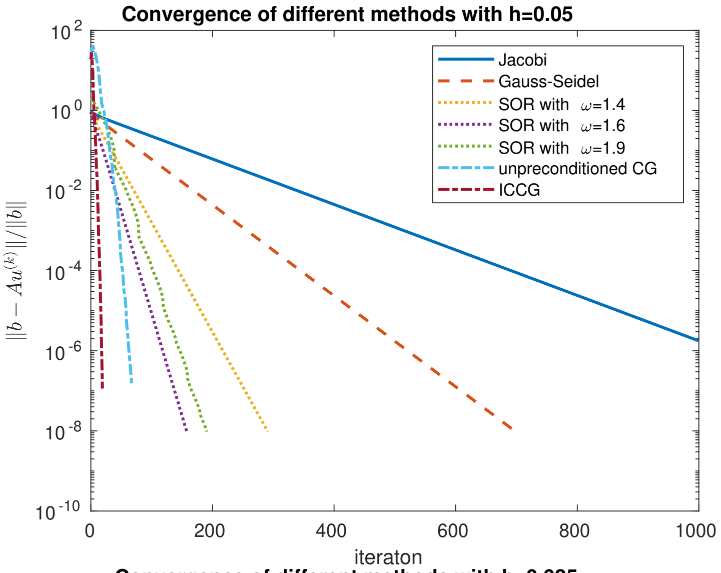
```
x0 = zeros(length(b),1); %initial guess
tol = 1e-8; maxiter = 1000;

D = diag(diag(A));
[x_J,iter,residvec_J] = simpleiter(A,b,D,x0,tol,maxiter);
figure(1)
semilogy(0:iter-1,residvec_J,'LineWidth',1.6)
hold on
fprintf('Jacobi takes %i iterations.\n',iter)
```

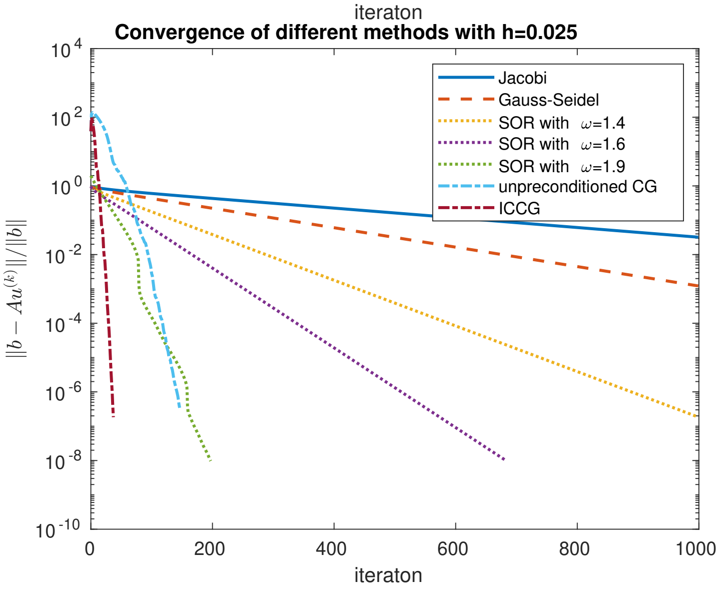
```
L = tril(A);
[x_G,iter,residvec_G] = simpleiter(A,b,L,x0,tol,maxiter);
semilogy(0:iter-1,residvec_G,’--’,’LineWidth’,1.6)
fprintf(’Gauss-Seidel takes %i iterations.\n’,iter)

omega1=1.4;
M1 = D/omega1+tril(A,-1);
[x_S1,iter,residvec_S1] = simpleiter(A,b,M1,x0,tol,maxiter);
semilogy(0:length(residvec_S1)-1,residvec_S1,’:’,’LineWidth’,1.6)
fprintf(’SOR with omega=%d takes %i iterations.\n’,omega1, iter)

omega2=1.6;
M2 = D/omega2+tril(A,-1);
[x_S2,iter,residvec_S2] = simpleiter(A,b,M2,x0,tol,maxiter);
semilogy(0:length(residvec_S2)-1,residvec_S2,’:’,’LineWidth’,1.6)
fprintf(’SOR with omega=%d takes %i iterations.\n’,omega2, iter)

omega3=1.9;
M3 = D/omega3+tril(A,-1);
[x_S3,iter,residvec_S3] = simpleiter(A,b,M3,x0,tol,maxiter);
semilogy(0:length(residvec_S3)-1,residvec_S3,’:’,’LineWidth’,1.6)
fprintf(’SOR with omega=%d takes %i iterations.\n’,omega3, iter)

% nonpreconditioned cg
[x_cg,˜,˜,iter,residvec_cg] = pcg(A,b,tol,maxiter,eye(size(A)));
semilogy(0:length(residvec_cg)-1,residvec_cg,’-.’,’LineWidth’,1.6)
fprintf(’CG takes %i iterations.\n’,iter)

L = ichol(A); %incomplete Cholesky
[x_icg,˜,˜,iter,residvec_icg] = pcg(A,b,tol,maxiter,L,L’);
semilogy(0:length(residvec_icg)-1,residvec_icg,’-.’,’LineWidth’,1.6)
fprintf(’ICCG takes %i iterations.\n’,iter)
xlabel(’iteraton’); ylabel(’$\|b-Au^{(k)}\|/\|b\|$’,’Interpreter’,’latex’)
legend(’Jacobi’,’Gauss-Seidel’,strcat(’SOR with \omega=’,num2str(omega1)) ...
    ,strcat(’SOR with \omega=’,num2str(omega2)),strcat(’SOR with \omega=’,num2str(omega3)) .
    ,’unpreconditioned CG’,’ICCG’)
title(strcat(’Convergence of different methods with h=’,num2str(h)))
hold off
```

We do this for three different values of $\omega = 1.4, 1.6, 1.9$ and $h = 0.2, 0.1, 0.05, 0.025$ and observe the following error plots.


Convergence of different methods with h=0.2
10^{0}
10^{-5}
10^{-10}
10^{-15}
$\|b - Au^{(k)}\|/\|b\|$
0
20
40
60
80
100
120
140
160
180
iteraton
Jacobi
Gauss-Seidel
SOR with ω=1.4
SOR with ω=1.6
SOR with ω=1.9
unpreconditioned CG
ICCG


Convergence of different methods with h=0.1
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
$\|b - Au^{(k)}\|/\|b\|$
0
50
100
150
200
250
300
350
iteraton
Jacobi
Gauss-Seidel
SOR with ω=1.4
SOR with ω=1.6
SOR with ω=1.9
unpreconditioned CG
ICCG





We observe that our optimal parameter ω for SOR seems to vary has h changes, apparently approaching 2 as $h \to 0$. The following table presents the number of iterations needed to converge under our tolerance for different methods and h values.

h	Jacobi	Gauss-Seidel	SOR ($\omega = 1.4$)	SOR ($\omega = 1.6$)	SOR ($\omega = 1.9$)	CG	ICCG
0.2	83	42	22	38	181	10	8
0.1	346	174	66	41	187	29	11
0.05	1000	699	292	159	192	67	19
0.025	1000	1000	1000	684	198	146	37

Noting that we cut h in half each time, we now give the factor of increase in the number of iterations compared to the previous h (iter(h)/iter($2h$)).

h	Jacobi	Gauss-Seidel	SOR ($\omega = 1.4$)	SOR ($\omega = 1.6$)	SOR ($\omega = 1.9$)	CG	ICCG
0.1	4.1687	4.1429	3	1.0789	1.0331	2.9	1.375
0.05	N/A	4.0172	4.4242	3.8780	1.0267	2.3103	1.7273
0.025	N/A	N/A	N/A	4.3019	1.0312	2.1791	1.9474

Both Jacobi and Gauss-Seidel grow like O(h^{-2}) while both CG and ICCG appear to be converging to growing like O(h^{-1}). The different SOR methods are trickier, but appear to be near constant for sufficiently large h and converge to growing like O(h^{-2}) as h becomes sufficiently small.

3 Problem 3

Consider a symmetric positive definite matrix A that has one thousand eigenvalues uniformly distributed between 1 and 10, and one eigenvalue of 10^4 and another symmetric positive definite matrix B that has an eigenvalue of 1 and has one thousand eigenvalues uniformly distributed between 10^3 and 10^4. First considering the case of matrix A, let the eigenvalues be ordered such that $\lambda_1 \leq \ldots \leq \lambda_{n-1} \ll \lambda_n$ and consider the polynomial

$$p_k(z) = \left(T_{k-1}\left(\frac{2z - \lambda_{n-1} - \lambda_1}{\lambda_{n-1} - \lambda_1}\right) \Big/ T_{k-1}\left(\frac{-\lambda_{n-1} - \lambda_1}{\lambda_{n-1} - \lambda_1}\right)\right) \frac{\lambda_n - z}{\lambda_n}.$$

Note that the second term in this product is zero at $z = \lambda_n$, so

$$\max_{i=1,\ldots,n} |p_k(\lambda_i)| = \max_{i=1,\ldots,n-1} |p_k(\lambda_i)|.$$

Then, letting $\kappa' = \lambda_{n-1}/\lambda_1 \leq 10$, we can apply (4.59) in the text to the first function in this product to get that

$$\max_{i=1,\ldots,n-1} \left| T_{k-1}\left(\frac{2z - \lambda_{n-1} - \lambda_1}{\lambda_{n-1} - \lambda_1}\right) \Big/ T_{k-1}\left(\frac{-\lambda_{n-1} - \lambda_1}{\lambda_{n-1} - \lambda_1}\right)\right| \leq 2\left(\frac{\sqrt{\kappa'} - 1}{\sqrt{\kappa'} + 1}\right)^{k-1}.$$

Coupling this with the fact that the second term in our product is linear, we can apply (4.51) in the text to get that

$$\begin{aligned}\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A} &\leq \max_{i=1,\ldots,n} |p_k(\lambda_i)| \leq 2\left(\frac{\sqrt{\kappa'} - 1}{\sqrt{\kappa'} + 1}\right)^{k-1} \frac{\lambda_n - \lambda_1}{\lambda_n} \\ &\leq 2\left(\frac{\sqrt{10} - 1}{\sqrt{10} + 1}\right)^{k-1} \frac{10^4 - 1}{10^4} = 2 * 0.9999\left(\frac{\sqrt{10} - 1}{\sqrt{10} + 1}\right)^{k-1}.\end{aligned}$$

For matrix B, order the eigenvalues such that $\lambda_1 \ll \lambda_2 \leq \ldots \leq \lambda_n$ and consider the polynomial

$$p_k(z) = \left(T_{k-1}\left(\frac{2z - \lambda_n - \lambda_2}{\lambda_n - \lambda_2}\right) \Big/ T_{k-1}\left(\frac{-\lambda_n - \lambda_2}{\lambda_n - \lambda_2}\right)\right) \frac{\lambda_1 - z}{\lambda_1}.$$

Note that the second term in this product is zero at $z = \lambda_1$, so

$$\max_{i=1,\ldots,n} |p_k(\lambda_i)| = \max_{i=2,\ldots,n} |p_k(\lambda_i)|.$$

Then, letting $\kappa' = \lambda_n/\lambda_2 \leq 10$, we can apply (4.59) in the text to the first function in this product to get that

$$\max_{i=2,\ldots,n} \left| T_{k-1}\left(\frac{2z - \lambda_n - \lambda_2}{\lambda_n - \lambda_2}\right) \Big/ T_{k-1}\left(\frac{-\lambda_n - \lambda_2}{\lambda_n - \lambda_2}\right)\right| \leq 2\left(\frac{\sqrt{\kappa'} - 1}{\sqrt{\kappa'} + 1}\right)^{k-1}.$$

Coupling this with the fact that the second term in our product is linear, we can apply (4.51) in the text to get that

$$\begin{aligned}\frac{\|e^{(k)}\|_B}{\|e^{(0)}\|_B} \leq \max_{i=1,\ldots,n} |p_k(\lambda_i)| &\leq 2\left(\frac{\sqrt{\kappa'} - 1}{\sqrt{\kappa'} + 1}\right)^{k-1} \left|\frac{\lambda_1 - \lambda_n}{\lambda_1}\right| \\ &\leq 2\left(\frac{\sqrt{10} - 1}{\sqrt{10} + 1}\right)^{k-1} \frac{10^4 - 1}{1} = 2 * 9999 \left(\frac{\sqrt{10} - 1}{\sqrt{10} + 1}\right)^{k-1}.\end{aligned}$$

In attempting to analyze the convergence of CG for these matrices, it is important to note that we have not established the tightness of these bounds, so they may not give an accurate depcition of the convergence rates in the slightest. If we do assume that these bounds are sufficiently tight to give us information about the convergence rates, we expect CG to reduce relative error at the same rate of $\frac{\sqrt{10}-1}{\sqrt{10}+1}$ for both A and B in their respective norms; however, the bound on the convergence of B is 10000 times that of A, so we expect CG to take longer to reach a set relative error when applied to B instead of A.