Homework 5

Due: March 1, 2021

1. The key to efficiency in the `chebfun` package, which you used in a previous homework exercise, is the ability to rapidly translate between the values of a function at the Chebyshev points, $\cos(\pi j/n)$, $j = 0, \ldots, n$, and the coefficients $a_0, \ldots, a_n$, in a Chebyshev expansion of the function's nth-degree polynomial interpolant: $p(x) = \sum_{j=0}^{n} a_j T_j(x)$, where $T_j(x) = \cos(j \arccos(x))$ is the jth degree Chebyshev polynomial. Knowing the coefficients $a_0, \ldots, a_n$, one can evaluate p at the Chebyshev points by evaluating the sums

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

These sums are much like the real part of the sums in the FFT,

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

but the argument of the cosine differs by a factor of 2 from the values that would make them equal. Explain how the FFT or a closely related procedure could be used to evaluate the sums in (1). To go in the other direction, and efficiently determine the coefficients $a_0, \ldots, a_n$ from the function values $f(\cos(k\pi/n))$, what method would you use?

Solution.

Define

$$\begin{aligned} a_{n+m} &= a_{n-m}, \quad m = 1, \ldots, n \\ A_k &= \sum_{j=0}^{2n-1} a_j e^{2\pi ijk/2n} = \sum_{j=0}^{2n-1} a_j e^{\pi ijk/n}, \quad k = 0, \ldots, n-1 \end{aligned}$$

Adding $a_0 = a_{2n}$ to both sides

$$\begin{aligned} A_k + a_0 &= \sum_{j=0}^{2n} a_j e^{\pi ijk/n} \\ &= \sum_{j=0}^{n} a_j e^{\pi ijk/n} + \sum_{j=n+1}^{2n} a_j e^{\pi ijk/n} \end{aligned}$$

We notice that

$$\cos(\pi(n-m)k/n) = \cos(\pi k)\cos(-\pi mk/n) = \cos(\pi(n+m)k/n)$$

$$\sin(\pi(n-m)k/n) = \cos(\pi k)\sin(-\pi mk/n) = -\sin(\pi(n+m)k/n)$$

Since j in $\sum_{j=0}^{n} a_j e^{\pi ijk/n}$ and $\sum_{j=n}^{2n} a_j e^{\pi ijk/n}$ differ by n, then

$$\begin{aligned}\sum_{j=0}^{n} a_j e^{\pi ijk/n} + \sum_{j=n}^{2n} a_j e^{\pi ijk/n} &= \sum_{j=0}^{n} a_j(\cos(\pi jk/n) - i\sin(\pi jk/n) + \cos(\pi jk/n) + i\sin(\pi jk/n)) \\ &= 2p(\cos(k\pi/n))\end{aligned}$$

Adding $a_n e^{\pi ik}$ to $A_k + a_0$

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

Rearranging

$$p(\cos(k\pi/n)) = \frac{1}{2}(A_k + a_n e^{\pi ik} + a_0)$$

If we want to go in the other direction, we use the expression

$$A_k = 2p(\cos(k\pi/n)) - a_n e^{\pi ik} - a_0$$

Then we can use the inverse DFT to compute the coefficients

$$a_j = \frac{1}{n}\sum_{k=0}^{n-1} e^{-2\pi ijk/n} A_k, \quad j = 0, \dots, n-1$$

2. On the course web page is a finite difference code (steady2d.m) to solve the boundary value problem:

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

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

where $a(x,y)=1+x^2+y^2$ and $f(x,y)=1$. It uses a direct solver for the linear system.

Replace this direct solver first by the Jacobi method, then by the Gauss Seidel method, and then by the SOR method. For each method, make a plot of the relative residual norm, $\|b-Au^k\|/\|b\|$ versus iteration number k. (Use a logarithmic scale for the residual; i.e., you may use `semilogy` in Matlab to do the plot.) Try several different values for the parameter ω in SOR, until you find one that seems to work well.

Then try solving the linear system using the conjugate gradient method. You may write your own CG code or use the one in Matlab (called **pcg**). First try CG without a preconditioner (i.e., with preconditioner equal to the identity) and then try CG with the Incomplete Cholesky decomposition as the preconditioner. You may use `ichol` in Matlab to generate the incomplete Cholesky decomposition. Again make a plot of relative residual norm versus iteration number for the CG method.

Experiment with a few different mesh sizes and comment on how the number of iterations required to reach a fixed level of accuracy seems to vary with h for each method.

Solution. We implement the Jacobi, Gauss-Seidel and SOR as follows

```
function [res_vec,iter]=jacobi(u,A,b,tol,maxiter)
M=sparse(diag(diag(A)));
iter=-1;
% rel_res=1;
% while rel_res>tol
for i=0:maxiter
    res=b-A*u;
    u=u+M\res;
    rel_res=norm(res)/norm(b);
    res_vec(i+1)=rel_res;
    iter=iter+1;
end
```

```
function [res_vec,iter]=gauss_seidel(u,A,b,tol,maxiter)
M=sparse(tril(A));
iter=-1;
% rel_res=1;
% while rel_res>tol
for i=0:maxiter
```

```
    res=b-A*u;
    u=u+M\res;
    rel_res=norm(res)/norm(b);
    res_vec(i+1)=rel_res;
    iter=iter+1;
end
```

```
function [res_vec,iter]=sor(u,w,A,b,tol,maxiter)
M=sparse(diag(diag(A))+w*tril(A,-1))/w;
iter=-1;
% rel_res=1;
% while rel_res>tol
for i=0:maxiter
    res=b-A*u;
    u=u+M\res;
    rel_res=norm(res)/norm(b);
    res_vec(i+1)=rel_res;
    iter=iter+1;
end
```

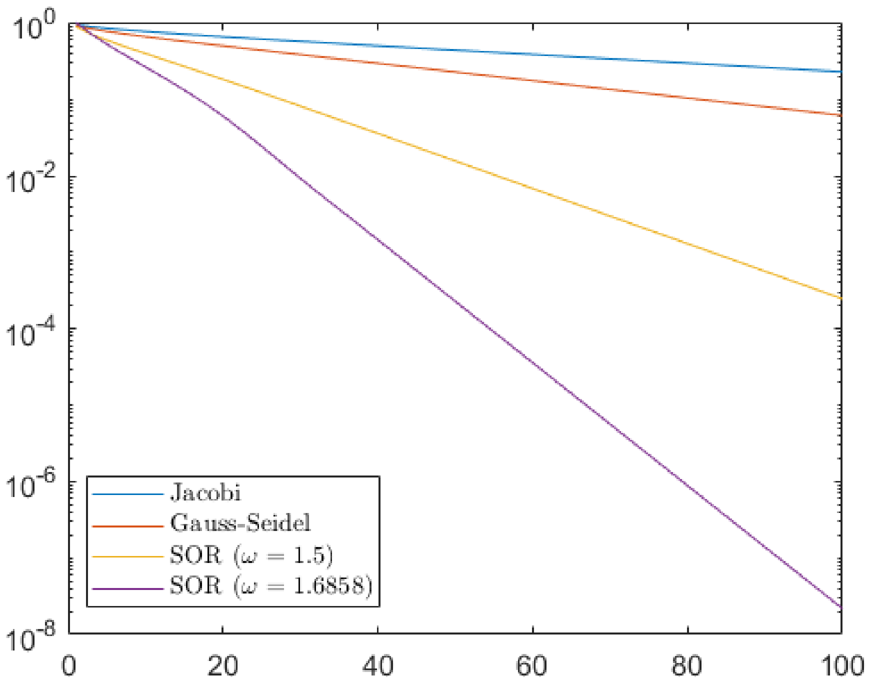
We use `pcg` for the conjugate gradient method. We use `ichol` to generate a preconditioner M

```
L=ichol(A);
M=L'*L;
```

We use `steady2D` to compute the Poisson matrix A and the source term b. Then we run the iterative solvers with the following inputs

```
u=sparse(zeros(N,1));
tol=1e-6;
maxiter=100;
```

It is evident from Figure 1 that SOR converges faster than both Jacobi and Gauss-Seidel methods and CG with a preconditioner converges faster than CG without one. The optimal ω for SOR is given by $\omega_{\text{opt}} \approx 2 - 2\pi h$ and $\omega_{\text{opt}} = 1.6858$ for $h = 0.05$. We observe from Figure 2 that Jacobi and Gauss-Seidel requires larger number of iterations as the mesh size decreases. For SOR methods, as ω gets closer to 2, the number of iterations becomes more stable.



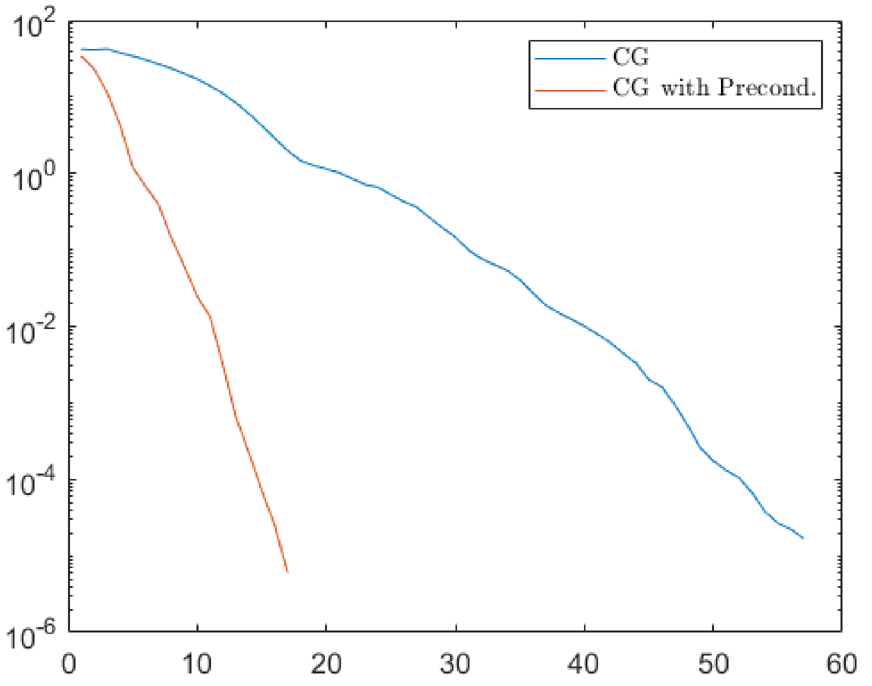
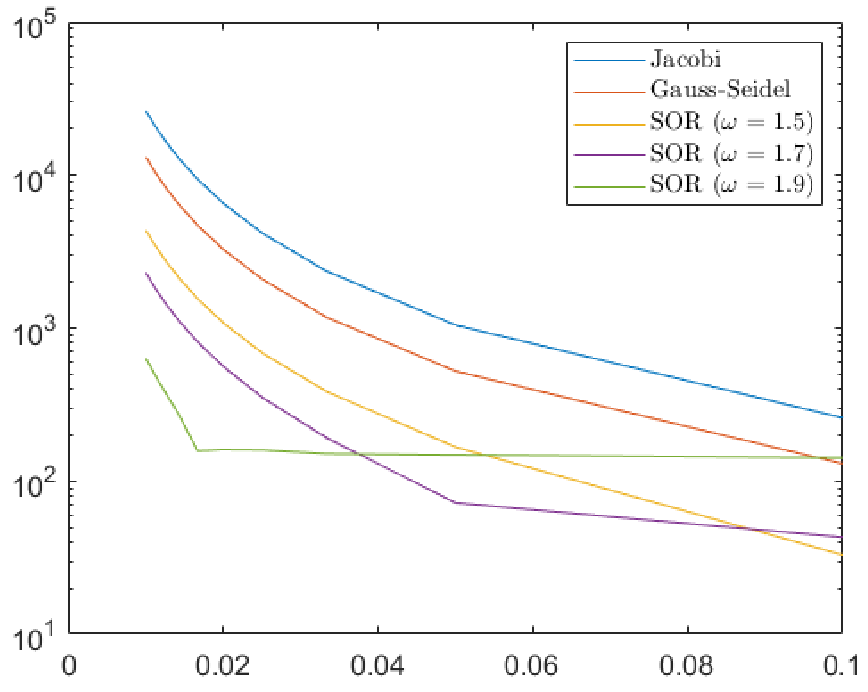


Figure 1: Relative residue norm against number of iterations. A mesh size $h = 0.05$ is used.



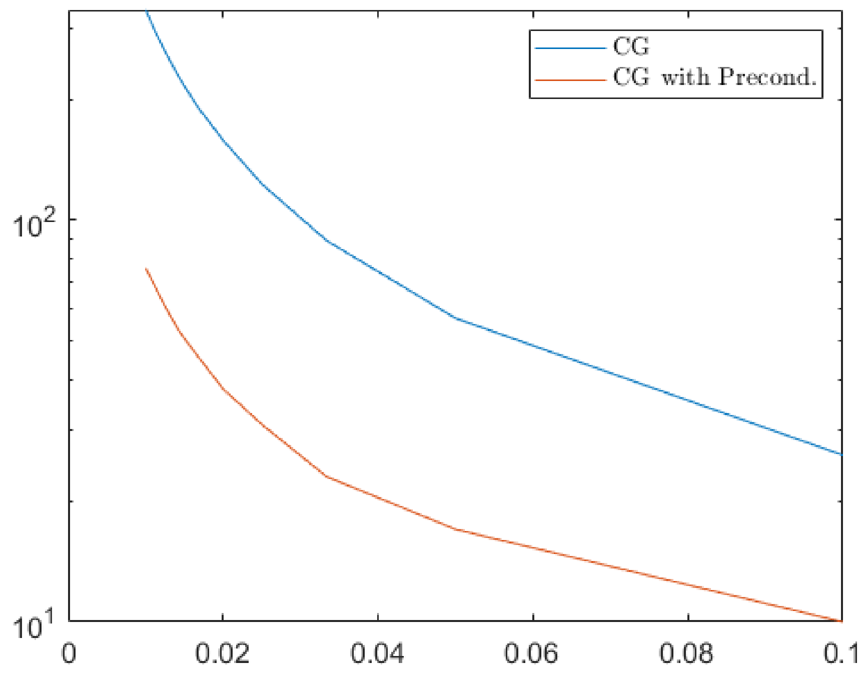


Figure 2: Number of iterations required to reach 1e-6 tolerance against varying mesh sizes.

3. Suppose a symmetric positive definite matrix A has one thousand eigenvalues uniformly distributed between 1 and 10, and one eigenvalue of 10^4. Suppose another symmetric positive definite matrix B has an eigenvalue of 1 and has one thousand eigenvalues uniformly distributed between 10^3 and 10^4. Since each matrix has condition number $\kappa = 10^4$, we have seen that the error at step k of the CG algorithm satisfies

$$\frac{\|e^{(k)}\|_{A,B}}{\|e^{(0)}\|_{A,B}} \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k = 2\left(\frac{99}{101}\right)^k.$$

Give another bound on $\|e^{(k)}\|_A/\|e^{(0)}\|_A$ based on a polynomial that is a product of a degree $k-1$ Tchebyshev polynomial on the interval $[1,10]$ and a linear polynomial that is 1 at the origin and 0 at 10^4. Give another bound on $\|e^{(k)}\|_B/\|e^{(0)}\|_B$ based on a polynomial that is a product of a degree $k-1$ Tchebyshev polynomial on the interval $[10^3,10^4]$ and a linear polynomial that is 1 at the origin and 0 at 1. For which matrix would you expect CG to converge more rapidly? [If you are not sure, you may try it in Matlab.]

Solution.

For matrix A, let $\lambda_1 = 1, \lambda_m = 10, \lambda_m + 1 = 10^4$. Define

$$\tilde{P}_{k,A}(x) = \frac{T_{k-1}(\frac{\lambda_m+\lambda_1-2x}{\lambda_m-\lambda_1})}{T_{k-1}(\frac{\lambda_m+\lambda_1}{\lambda_m-\lambda_1})}\left(\frac{\lambda_{m+1}-x}{\lambda_{m+1}}\right)$$

where T_{k-1} is the Chebyshev polynomial of degree $k-1$. Then we have shifted the Chebyshev polynomial from the interval $[-1,1]$ to $[\lambda_1,\lambda_m]$. We know T_{k-1} takes extreme values ± 1 at the endpoints. Since $\frac{\lambda_n-x}{\lambda_n}$ is strictly decreasing, then $\tilde{P}_{k,A}(x)$ also takes extreme values at the endpoints. Notice that $\tilde{P}_{k,A}(\lambda_n) = 0$, then it must be true that

$$\max_{1\le j\le m+1} |\tilde{P}_{k,A}(\lambda_j)| = \tilde{P}_{k,A}(\lambda_1)$$

Then a bound on the A-norm of the error is

$$\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A} \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{k-1}\left|\frac{\lambda_{m+1}-\lambda_1}{\lambda_{m+1}}\right| \approx 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{k-1}$$

For matrix B, let $\lambda_{m+1} = 1, \lambda_1 = 10^3, \lambda_m = 10^4$. Define

$$\tilde{P}_{k,B}(x) = \frac{T_{k-1}(\frac{\lambda_m+\lambda_1-2x}{\lambda_m-\lambda_1})}{T_{k-1}(\frac{\lambda_m+\lambda_1}{\lambda_m-\lambda_1})}\left(\frac{\lambda_{m+1}-x}{\lambda_{m+1}}\right)$$

Then we can argue similarly as before that

$$\max_{1\le j\le m+1} |\tilde{P}_{k,B}(\lambda_j)| = \tilde{P}_{k,B}(\lambda_m)$$

Then a bound on the B-norm of the error is

$$\frac{\|e^{(k)}\|_B}{\|e^{(0)}\|_B} \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{k-1}\left|\frac{\lambda_{m+1}-\lambda_m}{\lambda_{m+1}}\right| \approx 2\cdot 10^4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{k-1}$$

Thus we would expect CG to converge more rapidly for A.