Homework 6
MA/CS 375, Fall 2005
Due December 15

This homework will count as part of your grade so you must work independently. It is permissible to discuss it with your instructor, the TA, fellow students, and friends. However, the programs/scripts and report must be done only by the student doing the project. Please follow the guidelines in the syllabus when preparing your solutions.

1. (30 pts.) In this problem we examine the dynamics of repeated matrix multiplication and use spectral analysis to understand it.

   i Let:

$$W = \begin{pmatrix} -0.0325 & -0.5325 & 0.1400 & 0.1725 & -0.3125 \\ -0.5325 & -0.0325 & 0.1400 & 0.1725 & -0.3125 \\ 0.1400 & 0.1400 & 0.1033 & 0.4633 & 0.1833 \\ 0.1725 & 0.1725 & 0.4633 & -0.2092 & 0.4958 \\ -0.3125 & -0.3125 & 0.1833 & 0.4958 & 0.0708 \end{pmatrix}.$$

   Generate a random 5-vector x. Compute $W^n x$, $n = 1 : 100$. Plot $\frac{\|W^n x\|}{\|W^{n-1} x\|}$ versus n. What do you observe? For $n = 20 : 20 : 100$ print $\frac{W^n x}{\|W^n x\|}$. What do you observe?

   ii Use the Matlab command **eig** to compute the eigenvalues and eigenvectors of W. Use the results to explain your observations from part (i).

   iii Repeat (i) for the matrix:

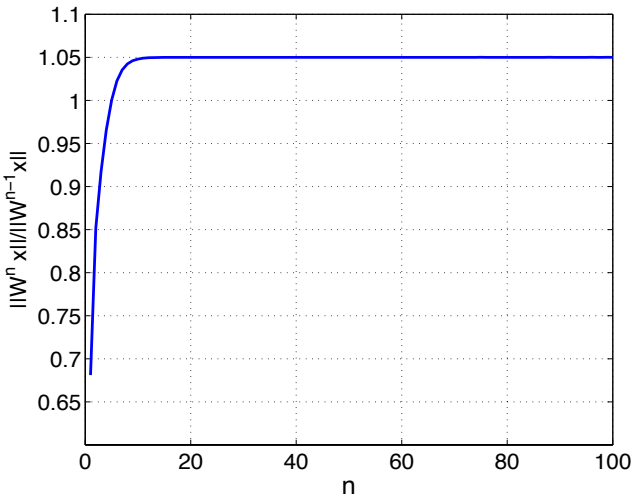
$$W = \begin{pmatrix} -\frac{5}{9} & -\frac{5}{9} & \frac{10}{9} & \frac{1}{9} \\ \frac{5}{9} & \frac{5}{9} & -\frac{4}{9} & \frac{5}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} & -\frac{1}{9} \\ \frac{7}{9} & -\frac{2}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}.$$

   iv Repeat (ii) for the results of part (iii).

   Solutions: For the first (symmetric) matrix the starting vector is generated using `x=rand(5,1)`. Then we can compute powers of W applied to x in the following way.

```
>>Wx=zeros(5,101); Wx(:,1)=x; nWx=zeros(101,1); nWx(1)=norm(x);
>>for n=2:101
>>Wx(:,n)=W*Wx(:,n-1); nWx(n)=norm(Wx(:,n));
>>end
>>plot(nWx(2:101)./nWx(1:100),'LineWidth',2); grid on;
```

For a plot, one obtains



The normalized vectors as described in the problem can be obtained by

```
>> A=Wx./repmat(nWx',5,1);
>> A(:,21:20:101)
```

which yields the vector for varying n Using Matlab to perform the eigenvalue decomposition,

20	40	60	80	100
0.499998808689738	0.49999999999829	0.5	0.5	0.5
0.49999891372587	0.499999999998327	0.5	0.5	0.5
0.00126896745479932	1.51279458226553e-06	1.80327124483189e-09	2.14952494430216e-12	2.53866898685795e-15
-0.49872954784572	-0.499998487203602	-0.499999998196729	-0.49999999999785	-0.499999999999998
0.501267897844052	0.500001512792916	0.500000001803271	0.50000000000215	0.500000000000003

```
>>[S,L]=eig(W);
```

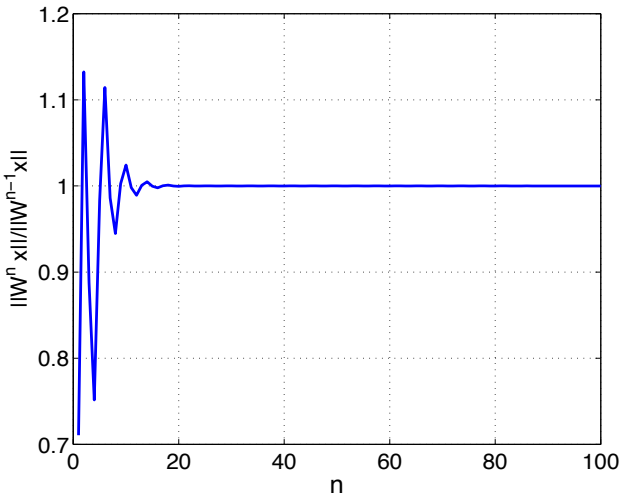
The matrices of eigenvectors and eigenvalues are

$$S = \begin{pmatrix} 0.5000 & 0.3162 & 0.3873 & -0.7071 & 0.0000 \\ 0.5000 & 0.3162 & 0.3873 & 0.7071 & 0.0000 \\ 0.0000 & -0.6325 & 0.5164 & 0.0000 & 0.5774 \\ -0.5000 & 0.6325 & 0.1291 & -0.0000 & 0.5774 \\ 0.5000 & 0 & -0.6455 & 0 & 0.5774 \end{pmatrix}$$

$$L = \begin{pmatrix} -1.0500 & 0 & 0 & 0 & 0 \\ 0 & -0.5000 & 0 & 0 & 0 \\ 0 & 0 & 0.2000 & 0 & 0 \\ 0 & 0 & 0 & 0.5000 & 0 \\ 0 & 0 & 0 & 0 & 0.7499 \end{pmatrix}$$

By repeatedly multiplying a vector by the matrix, the ratio of successive norms converges to the modulus of the largest eigenvalue, whereas the normalized resulting vector converges (to within a sign) of the corresponding eigenvector. This is called the power method of determining the largest eigenvalue/vector pair.

The second matrix is not symmetric and as such may not necessarily have real eigenvalues, however, since its entries are real, complex eigenvalues must occur in conjugate pairs. Using the same method as before, one obtains the plot



The normalized vector is

20	40	60	80	100
0.686780234414052	0.6865846197281	0.686584560882313	0.686584560864617	0.686584560864612
-0.686353439417874	-0.686584491397969	-0.686584560843721	-0.686584560864605	-0.686584560864611
0.169480426839158	0.169120304580162	0.169120196290588	0.169120196258022	0.169120196258012
-0.168903080895085	-0.169120130981875	-0.169120196238382	-0.169120196258006	-0.169120196258012

The matrices of eigenvectors and eigenvalues are

$$S = \begin{pmatrix} -0.5000 + 0.0000i & -0.5000 - 0.0000i & 0.3162 & 0.7071 \\ 0.5000 & 0.5000 & 0.6325 & 0.0000 \\ 0.0000 - 0.5000i & 0.0000 + 0.5000i & 0.6325 & -0.0000 \\ -0.0000 + 0.5000i & -0.0000 - 0.5000i & 0.3162 & -0.7071 \end{pmatrix}$$

$$L = \begin{pmatrix} 0.0000 + 1.0000i & 0 & 0 & 0 \\ 0 & 0.0000 - 1.0000i & 0 & 0 \\ 0 & 0 & 0.6667 & 0 \\ 0 & 0 & 0 & -0.6667 \end{pmatrix}$$

While again the ratio of successive norms does converge to the modulus of the largest eigenvalue, the function oscillates initially. The normalized vector produced does not appear in the matrix of eigenvectors. The fact that there are two eigenvalues with modulus 1 suggests that the $W^n x$ vector may be a combination of the two corresponding eigenvectors. We can find the expansion coefficients of the $W^n x$ vector in terms of the eigenvectors by multiplying it by S^{-1}.

```
>>c=S\A(:,101);
```

This produces the vector

$$c = \begin{pmatrix} -0.6866 + 0.1691i \\ -0.6866 - 0.1691i \\ 0.0000 + 0.0000i \\ 0.0000 - 0.0000i \end{pmatrix}$$

Notice how the last two entries are effectively zero. This means that the solution vector has no component in the span of the eigenvectors corresponding to the eigenvalues $\pm\frac{2}{3}$. Clearly, the solution vector is a linear combination of the two eigenvectors corresponding to the eigenvalues $\pm i$.

2. (30 pts.) The method of successive overrelaxation or SOR is a way to accelerate the convergence of the Gauss-Seidel method for solving $Ax = b$. Let ω be a parameter and $x^{(k)}$ be the kth iterate. Then $x^{(k+1)}$ is computed via the formula:

$$\bar{x}_i^{(k+1)} = \frac{(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)})}{a_{ii}},$$

$$x_i^{(k+1)} = (1 - \omega) x_i^{(k)} + \omega \bar{x}_i^{(k+1)}.$$

Note that if $\omega = 1$ then SOR is identical to Gauss-Seidel. Modify the book's program itermeth.m to accept a new parameter ω as an input and replace the Gauss-Seidel option by SOR with parameter

ω. Use your method to solve $Ax = b$ where A is the 100×100 tridiagonal matrix whose diagonal entries are 2.01 and whose offdiagonal entries are -1. Considering $\omega = .2 : .2 : 1.8$ record the number of iterations required to converge to a tolerance of 10^{-4}. What is the best choice? How much of a difference does it make? Change the diagonal element to 2.001 and repeat the experiment. How do your results change?

Solution: The modifications to `itermeth.m` which change the Gauss-Seidel method to the SOR method are in the function declaration

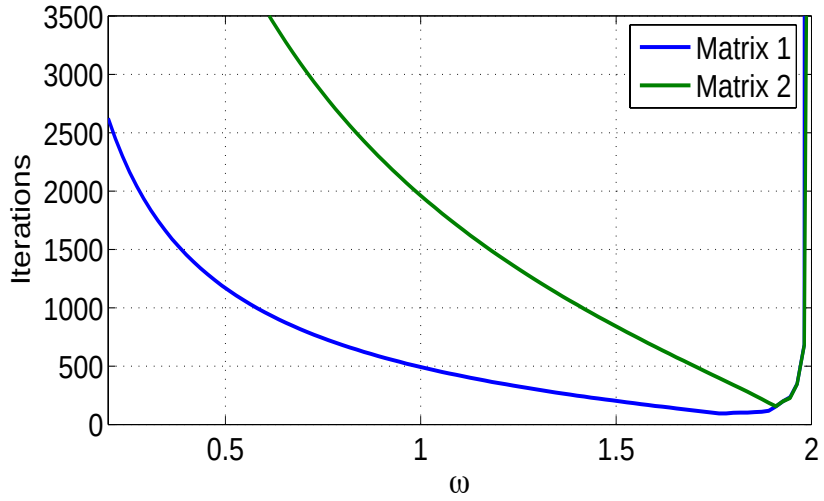
```
function [x, iter]= itermeth(A,b,x0,nmax,tol,P,w)
```

and on line 18

```
U = eye(n); beta = 1; alpha = w;
```

Alternately, one could write a simple SOR script which exploits the tridiagonal structure for the given problem

```
function [x,iter]=sor3band(A,b,w,x0,tol,maxit)

% Solve the symmetric positive definite tridiagonal system Ax=b
% using the Successive Over-Relaxation method

n=length(A);
L=spdiags(diag(A,-1),-1,n,n); U=L'; D=spdiags(diag(A),0,n,n);
x=x0;  x0=x+rand(n,1); iter=0;
Mw=D+w*L; Nw=(1-w)*D-w*L'; wb=w*b;

while (norm(x-x0)>tol & iter<maxit) % Do iteration
    x0=x;    x=Mw\(Nw*x0+wb);
    iter=iter+1;
end
```



The relaxation parameter which minimizes the number of iterations needed from the first matrix was found to be $\omega = 1.6061$. The number of iterations needed to solve the second system is minimized when ω is at its upper limit of 1.8. For all values of ω, the convergence of the SOR method first system is superior due to the greater diagonal dominance. In most practical computations, the optimal relaxation parameter is determined from information about the eigenvalues of the matrix.

3. (40 pts.) In this problem we compare three different schemes for solving the first order wave equation with periodic boundary conditions:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x},\ a \leq x \leq b, u(a,t) = u(b,t),\ u(x,0) = f(x) \tag{1}$$

We introduce space and time discretizations

$$x_m = a + m * h,\ h = \frac{b-a}{M},\ m = 0, ..., M$$

$$t_n = n * k,\ h = \frac{T}{N},\ n = 0, ..., N$$

(a) The forward-time, forward-space scheme (**FF** scheme):

$$\frac{u_m^{n+1} - u_m^n}{k} = c\frac{u_{m+1}^n - u_m^n}{h} \tag{2}$$

$$u_m^{n+1} = u_m^n + \alpha(u_{m+1}^n - u_m^n),\ \alpha = \frac{ck}{h},\ m = 1, ..., M,\ n = 0, ..., N-1 \tag{3}$$

(b) The forward-time, centered-space scheme (**CF** scheme):

$$\frac{u_m^{n+1} - u_m^n}{k} = c\frac{u_{m+1}^n - u_{m-1}^n}{2h} \tag{4}$$

$$u_m^{n+1} = u_m^n + \frac{\alpha}{2}(u_{m+1}^n - u_{m-1}^n),\ m = 1, ..., M,\ n = 0, ..., N-1 \tag{5}$$

(c) The leap-frog (or centered-time, centered-space scheme (**LF** scheme):

$$\frac{u_m^{n+1} - u_m^{n-1}}{2k} = c\frac{u_{m+1}^n - u_{m-1}^n}{2h} \tag{6}$$

$$u_m^{n+1} = u_m^{n-1} + \alpha(u_{m+1}^n - u_{m-1}^n),\ m = 1, ..., M,\ n = 0, ..., N-1 \tag{7}$$

Since the leap-frog scheme is two-levels deep in time, it is called a multi-step scheme. To get it started, we need to specify data for the first two time steps. Here we will do it using the scheme (5). Because of the periodic boundary conditions, the solution can be assumed to extend periodically in x outside the computational domain. Thus, we can identify the values $U_{M+k}^n = u_k^n$, for $k = -1, 0, 1$ needed in the above formulas.

If we write

$$\mathbf{U}^n = (u_1^n, u_2^n, ..., u_M^n)^\top$$

then the above 3 schemes can be written in matrix form:

(a) The FF scheme (3)

$$\mathbf{U}^{n+1} = A_F\mathbf{U}^n \tag{8}$$

(b) The CF scheme (5)

$$\mathbf{U}^{n+1} = A_C\mathbf{U}^n \tag{9}$$

(c) The LF scheme (7)

$$\mathbf{U}^{n+1} = A_L\mathbf{U}^n + \mathbf{U}^{n-1} \tag{10}$$

You must implement these schemes to solve equation (1) with

$$a = 1,\ b = 1,\ T = 2$$

with initial condition

$$u(x, 0) = f(x) = \mathrm{sech}(5\sin(\pi x))$$

(a) Use $M = 200,\ 400$ and $N = 300$, and $c = .9$. Compute the numerical solutions at time $t = T$ and plot each against the exact solution

$$u_{exact}(x,t) = \mathrm{sech}(5\sin(\pi(x+ct)))$$

How do the computations with the two different values of M differ?

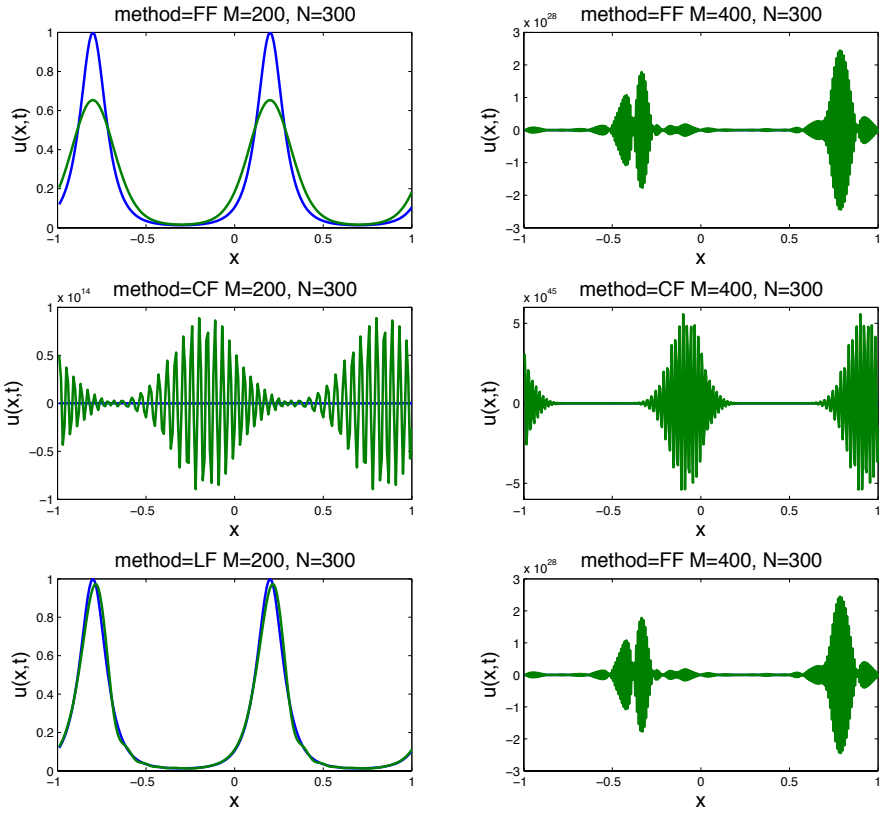
(b) Now use $M = 200$, $N = 300$ and $M = 400$, $N = 600$. Compare the performance of the schemes (1) and (3)

Solutions: The Matlab code for these three schemes is

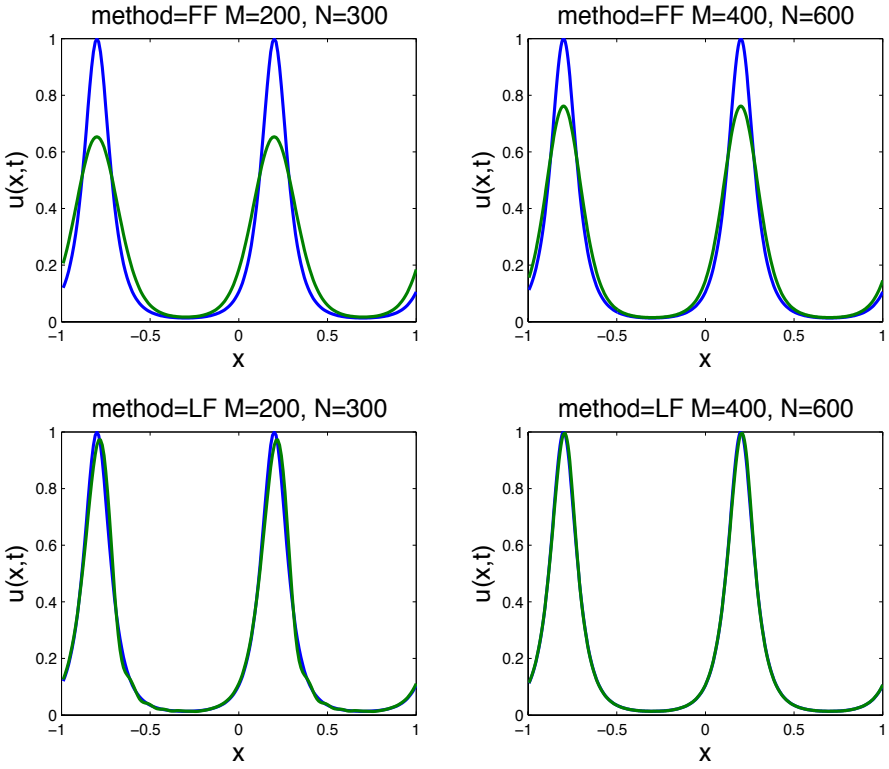
```
function advect(M,N,T,ab,c,f,method)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Solve the 1D advection (first order wave) equation
%
% u_t=c*u_x,  u(x,0)=f(x)
%
% using difference discretization in time and space
%
% Parameters:
%
% M - number of grid points
% N - number of time steps
% ab - 1x2 vector containg the left and right endpoints
% c - wave velocity
% exact - the exact solution in (x,t)
% method - which method to use 'FF' 'CF' or 'LF'
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m=(1:M)'; h=(ab(2)-ab(1))/M; x=ab(1)+m*h; k=T/N;
e=ones(M,1);    alpha=c*k/h;  u0=feval(f,x);
exact=feval(f,x+c*T);
if method=='FF' % Use forward time and forward space
    A=spdiags([e*(1-alpha) e*alpha],0:1,M,M); A(M,1)=alpha;
    for n=1:N u1=A*u0; u0=u1; end
```

```
    plot(x,exact,x,u1,'LineWidth',2)
else
    A=spdiags([-e*alpha/2 e e*alpha/2],-1:1,M,M);
    A(M,1)=alpha/2; A(1,M)=-alpha/2;
    if method=='CF' % Use forward time and centered space
        for n=1:N  u1=A*u0; u0=u1; end
        plot(x,exact,x,u1,'LineWidth',2)
    else % Use Leap-frog
        u1=A*u0;    % Use CF scheme to obtain u(x,t=k)
        A=spdiags([-e 0*e e]*alpha,-1:1,M,M);
        A(M,1)=alpha; A(1,M)=-alpha;
        for n=2:N u2=A*u1+u0; u0=u1; u1=u2; end
        plot(x,exact,x,u2,'LineWidth',2)
    end
end

xlabel('x','FontSize',16);
ylabel('u(x,t)','FontSize',16);
title(['method=',method,' M=',num2str(M),', N=',num2str(N)],'FontSize',16);
```

The results for part (a) are



The CF scheme is unstable in both cases, while the FF and LF schemes are stable in the case where the $\alpha < 1$ and unstable when $\alpha > 1$. The main difference is that the attenuation of the computed wave using the FF scheme is substantial whereas the LF scheme produces a result which is much closer to the exact solution. The LF scheme has the advantage of being second order in spatial and temporal discretization whereas the FF scheme is first order in both. In both cases, there appears to be a slight phase lag between the computed and exact solutions.
The results for part (b) are



Refinement of the grid and time step size improves both the error in the FF and LF schemes, but the improvement is slight in the FF case due to the lower order of convergence and the LF scheme exhibits far less dissipation.