Numerical Analysis Final

MTH – 452 MSU Spring '19

Patrick Thornton

Question 1

Description

Solve the boundary value problem using Finite Difference Method

$$\begin{cases} \nabla^2 u + 2u = g, & inside\ \Omega = [0,1]\ x\ [0,1] \\ u = 0 & on\ the\ boudary\ of\ \Omega, \end{cases}$$

Where $g(x,y)=(xy+1)(xy - x - y) + x^2 + y^2$. The exact solution is know as $u = 0.5xy(x-1)(y-1)$. Use Gauss-Seidel procedure to solve the obtained linear equation starting with $u_0(x_i,y_i) = x_i,y_i$.

Procedure

With the given information in the description. I created an algorithm to use the boundary conditions, and the given functions to approximate the given values within the boundary. To tell my algorithm when it was a good time to stop, I set an error tolerance of 10^{-4}. If the algorithm did not get lower than that tolerance, it was set to stop after 100 iterations. Lastly, I set the grid to be a 10x10 area, making predictions at the corner of every section (i.e. $g(x_i,y_i)$ for y,x in [0,0.1,0.2,…,0.8,0.9,1]).

Result

The result of my function was a 10x10 matrix whose first and last column and row were 0's and whose interior points were the approximations for the *ith, jth* step.

0	0	0	0	0	0	0	0	0	0
0	-0.1400	-0.1900	-0.2135	-0.2204	-0.2134	-0.1932	-0.1593	-0.1059	0
0	-0.1903	-0.1972	-0.2044	-0.2061	-0.2014	-0.1913	-0.1794	-0.1749	0
0	-0.2141	-0.2050	-0.2023	-0.2001	-0.1968	-0.1933	-0.1938	-0.2056	0
0	-0.2216	-0.2076	-0.2011	-0.1973	-0.1946	-0.1938	-0.1986	-0.2152	0
0	-0.2149	-0.2035	-0.1987	-0.1957	-0.1928	-0.1909	-0.1939	-0.2078	0
0	-0.1949	-0.1939	-0.1959	-0.1957	-0.1919	-0.1854	-0.1799	-0.1833	0
0	-0.1608	-0.1818	-0.1963	-0.2008	-0.1953	-0.1805	-0.1582	-0.1366	0
0	-0.1068	-0.1764	-0.2073	-0.2167	-0.2089	-0.1840	-0.1369	-0.0455	0
0	0	0	0	0	0	0	0	0	0

Figure 1: Matrix approximation output

My algorithm also outputs the approximations for specific x,y values in a table form, for the sake of space, I used a 3x3 grid this time. This lead to the output of the following values:

Table 1: Tabular results for 4x4 grid

x	y	W
0	[0, 0.25, 0.5, 0.75,1]	0
[0, 0.25, 0.5, 0.75,1]	0	0
0.25	0.25	-1.643456e-01
0.25	0.5	-1.686911e-01
0.25	0.75	-1.168476e-01
0.5	0.25	-1.688148e-01
0.5	0.5	-1.539425e-01
0.5	0.75	-1.391940e-01
0.75	0.25	-1.170331e-01
0.75	0.5	-1.393176e-01
0.75	0.75	-4.635677e-02
1	[0, 0.25, 0.5, 0.75,1]	0
[0, 0.25, 0.5, 0.75,1]	1	0

Code

```
function [w] = poisson(a,b,c,d,m,n,tol,iter)
    % This function approximates the solution to a elliptical partial
    % differential equation using the Finite difference Method
    % Author: Patrick Thornton

    % Input:
    % a,b - range of possible x values
    % c,d - range of possible y values
    % m - number of steps
    % n - number of time steps
    % tol - error tolerance for algorithm to stop
    % iter - max iterations if tolerance is not met

    % output(w) - approximations for each x,y point


    h = (b-a)./n; % get steps size
    k = (d-c)./m; % get time step size

    % initialize vectors
    xi = [];
    yi = [];
    g = @(x,y) (x*y+1)*(x*y-x-y)+x^2+y^2;          % g(x,y) function
    n = n+1;
    m = m+1;

    % populate step vectors
    for i = 1:n-1
        xi_new = a+i*h;
        xi = [xi xi_new];
    end
    for j = 1:m-1
        yi_new = c+j*k;
```

```
        yi = [yi yi_new];
    end

    % initialize variables
    w = zeros(n-1,m-1);
    lam = h^2/k^2;
    mu = 2*(1+lam);

    % Gauss-Seidel iterations
    for l = 1:iter
        z = (g(a,yi(m-1))+lam*g(xi(1),d)+lam*w(2,m-3)+w(3,m-2))/mu;
        norm = abs(z-w(2,m-2));
        w(2,m-2) = z;
        for i = 3:(n-3)
            z = (lam*g(xi(i),d)+w(i-1,m-2)+w(i+1,m-2)+lam*w(i,m-3))/mu;
            if abs(w(i,m-2)-z) > norm
                norm = abs(w(i,m-2)-z);
            end
            w(i,m-2)= z;
        end
        z = (g(b,yi(m-1))+lam*g(xi(n-1),d)+w(n-2,m-2)+lam*w(n-2,m-3))/mu;

        if abs(w(n-2,m-2)-z) > norm
            norm = abs(w(n-2,m-2)-z);
        end
        w(n-2,m-2) = z;

        for j = (m-3):-1:3
            z = (g(a,yi(j))+lam*w(2,j+1)+lam*w(2,j-1)+w(3,j))./mu;
            if abs(w(2,j)-z) > norm
                norm = abs(w(2,j)-z);
            end
            w(2,j) = z;
            for i=3:(n-3)
                z = (w(i-1,j)+lam*w(i,j+1)+w(i+1,j)+lam*w(i,j-1))/mu;
                if abs(w(i,j)-z) > norm
                    norm = abs(w(i,j)-z);
                end
                w(i,j) = z;



            end
            z = (g(b,yi(j))+w(n-3,j)+lam*w(n-2,j+1)+lam*w(n-2,j-1))/mu;
            if abs(w(n-2,j)-z) > norm
                norm = abs(w(n-2,j)-z);
            end
            w(n-2,j) = z;
        end
        z = (g(a,yi(1))+lam*g(xi(1),c)+lam*w(2,3)+w(3,2))/mu;
%         z = (g(b,yi(m-1))+lam*g(xi(n-1),d)+w(n-2,m-2)+lam*w(n-2,m-3))/mu;
        if abs(w(2,2)-z) > norm
            norm = abs(w(2,2)-z);
        end
        w(2,2) = z;
        for i =3:(n-3)
            z = (lam*g(xi(i),c)+w(i-1,2)+lam*w(i,3)+w(i+1,2))/mu;
```

```
            if abs(w(i,2)-z) > norm
                norm = abs(w(i,2)-z);
            end
            w(i,2) = z;
        end
        z = (g(b,yi(1))+lam*g(xi(n-1),c)+w(n-3,2)+lam*w(n-2,3))/mu;
        if abs(w(n-2,3)-z) > norm
            norm = abs(w(n-2,2)-z);
        end
        w(n-2,2) = z;

        % error
        norm

        % return approximations if tolerance met
        if norm <= tol
            for i =1:(n-1)
                for j=1:(m-1)
                    x = xi(i);
                    y = yi(j);
                    wo = w(i,j);
                    dispp = sprintf('x: %d, y: %d, wi: %d : ',x,y,wo);
                    disp(dispp)
                end
            end
            return
        end
    end
    % return if iterations are note met
    outt = sprintf('Maximum bumber of iterations %d exceeded',iter);
    disp(outt)
end
```

Question 2

Description

Code and run the Crank-Nicolson method with different choices of h and k for the following parabolic equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial^2 x}, \qquad 0 < x < 2, \quad t > 0$$

With

$$u(0,t) = u(2,t) = 0$$

and

$$u(x,0) = sin(\pi x(x - \frac{1}{2})),$$

Illustrate how the scheme converges with decreasing (h,k). Try k=h, for example.

Procedure

Using the above functions, I created a function that would calculate approximations to the function *u.* In this function, the endpoint l=2 was inputted along with two variables *m, n* which are used to calculate the step size *h* and the time step *k*. Next, using the steps described in Algorithm 12.3 of Numerical Analysis by Richard Bruden, I tested the approximations the Crank-Nicholson algorithm produced using different h and k values.

Result

For consistent visualization, I set k to 0.01, and then altered the h values. The list of h values is found below, with each of the values text being respective to their line color on the graph.

h = [0.1, 0.01, 0.002, 0.001, 0.0005]

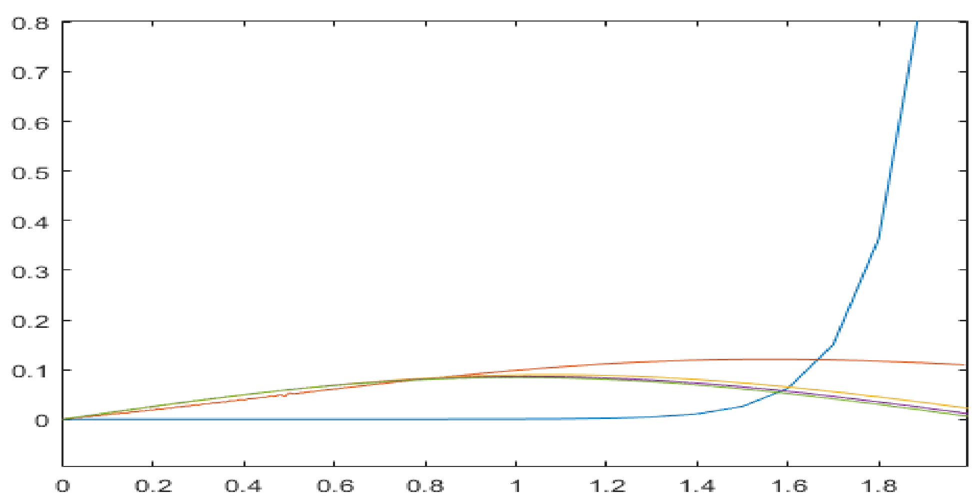

Figure 2: Results from Crank-Nicolson Algorithm

What this graph tells us I that the smaller the h, the more accurate the representation of u. I repeated this procedure with different values of k as well, and the pattern repeated itself compared to Figure 1, it just took different sizes of h respective to the new size of k. When h = 0.1 and k = 0.01 with t=0.5, the result was:

Table 2: Tabular results for Crank Nicolson

x_i	$W_{i,20}$
0	0
0.1	-0.0265
0.2	-0.0510
0.3	-0.0837
0.4	-0.0954
0.5	-0.0762
0.6	-0.0114
0.7	0.1019
0.8	0.2543
0.9	0.4243
1.0	0.5770
1.1	0.6679
1.2	0.6540
1.3	0.5131
1.4	0.2656
1.5	-0.0151
1.6	-0.2162
1.7	-0.2210
1.8	0.0766
1.9	0.8763
2.0	0

Code

```
function [w] = CrankNicolson(l,t,alpha,m,n)
    % This function approximates the solution to a parabolic partial
    % differential equation using the Crank-Nicolson Method
    % Author: Patrick Thornton

    % Input:
    % l - max value of x
    % t - max time
    % alpha - constant for f(x), set to 1
    % m - number of steps
    % n - number of time steps

    % output(w) - approximations for each step

    h = l/m; % get step size
    k = t/n; % get time step size
    lam = k./h.^2;
```

```
    w = [0]; % initialize approximation vector
    xi = [0]; % initialize step vector

    % populate vectors
    for i = 1:(m-1)
        x = h*i;
        xi = [xi x]
        y = sin(pi*x.*(x-0.5)); % solution vector
        w = [w y];
    end

    % Solving a tridiagonol linear system
    w = [w 0];
    li = [1 + lam];
    u = [-lam./(2*(1+lam))];
    for i = 2:(m-1)
        new_l = 1+lam+lam.*u(i-1)./2;
        li = [li new_l];
        new_u = -lam./(2.*li(i));
        u = [u new_u];
    end
    new_l = 1+lam+lam.*u(m-1)./2;
    li = [li new_l];

    for j = 1:n
        tj = j*k;
        z = [((1-lam).*w(1)+lam./2.*w(2))./li(1)];
        for i=2:(m-1)
            zi = ((1-lam).*w(i)+lam./2.*(w(i+1)+w(i-1)+z(i-1)))./li(i);
            z = [z zi];
        end
        w(m-1) = z(m-1);
        for i=(m-2):-1:1
            w(i+1) = z(i)-u(i).*w(i+2);
        end
%         disp(tj);

        %display results
        for i=1:(m-1)
            x = i*h;
            wi = w(i);
            out = sprintf('x: %d, wi: %d : ',x,wi);
            disp(out)
        end
    end
    %plot graph
    wp = w(1:m);
    plot(xi,wp)
end
```