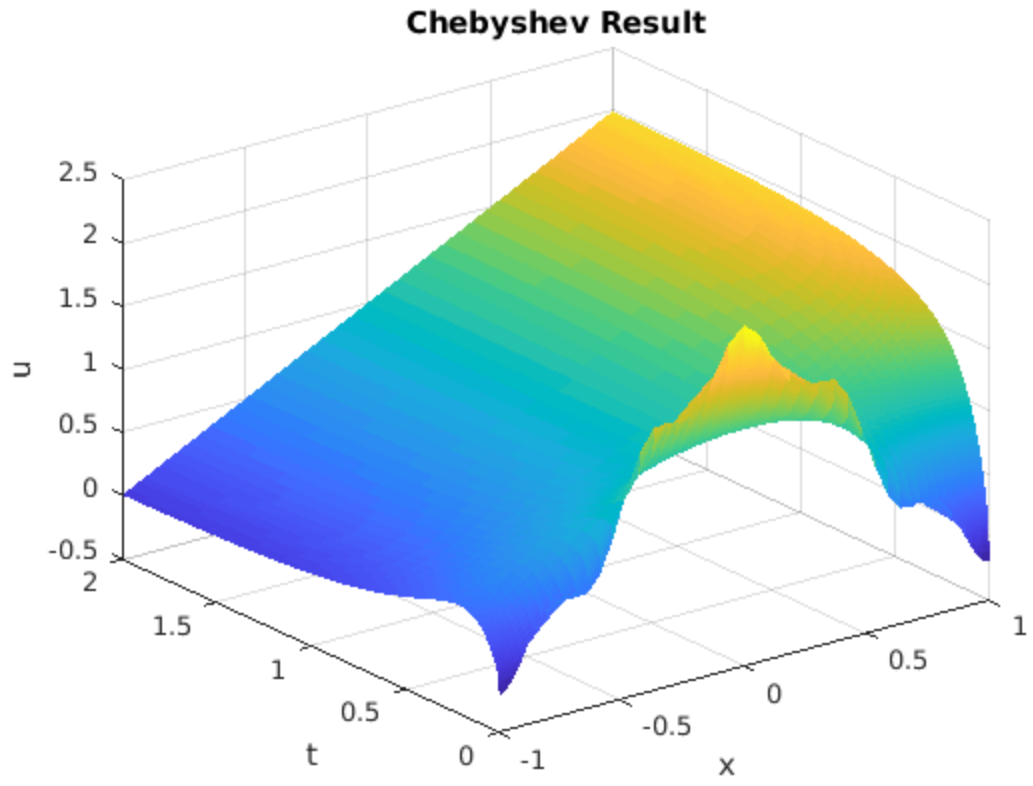
Table of Contents

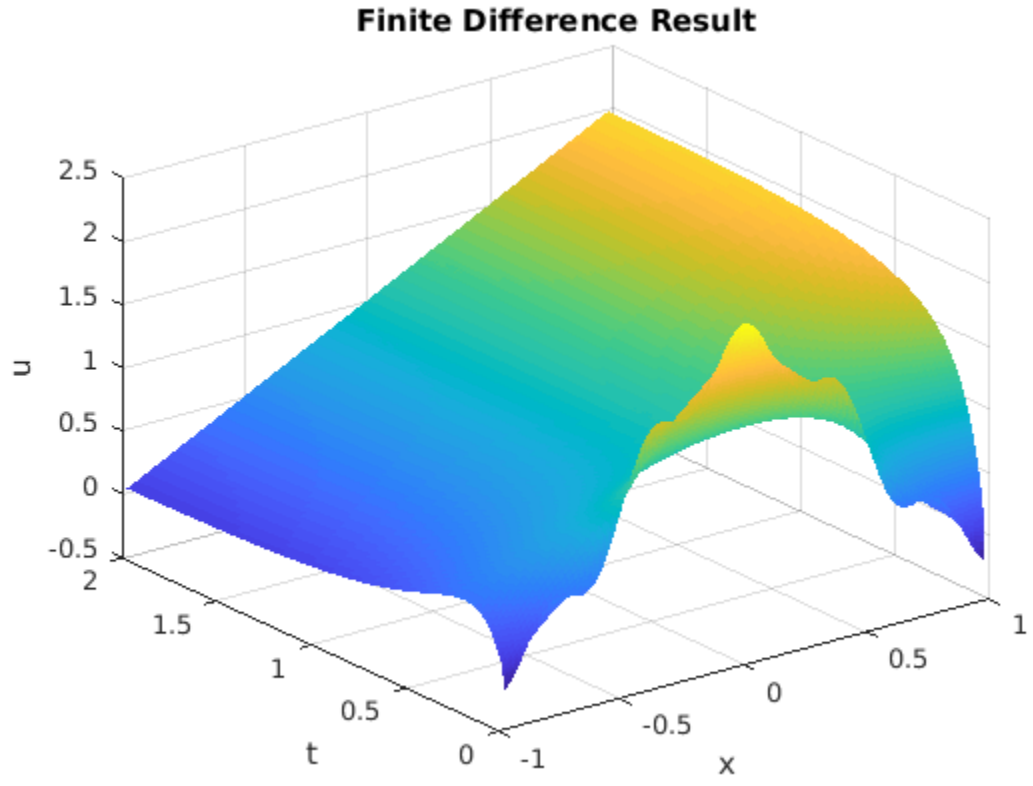
Homework 3 -- Problem 3

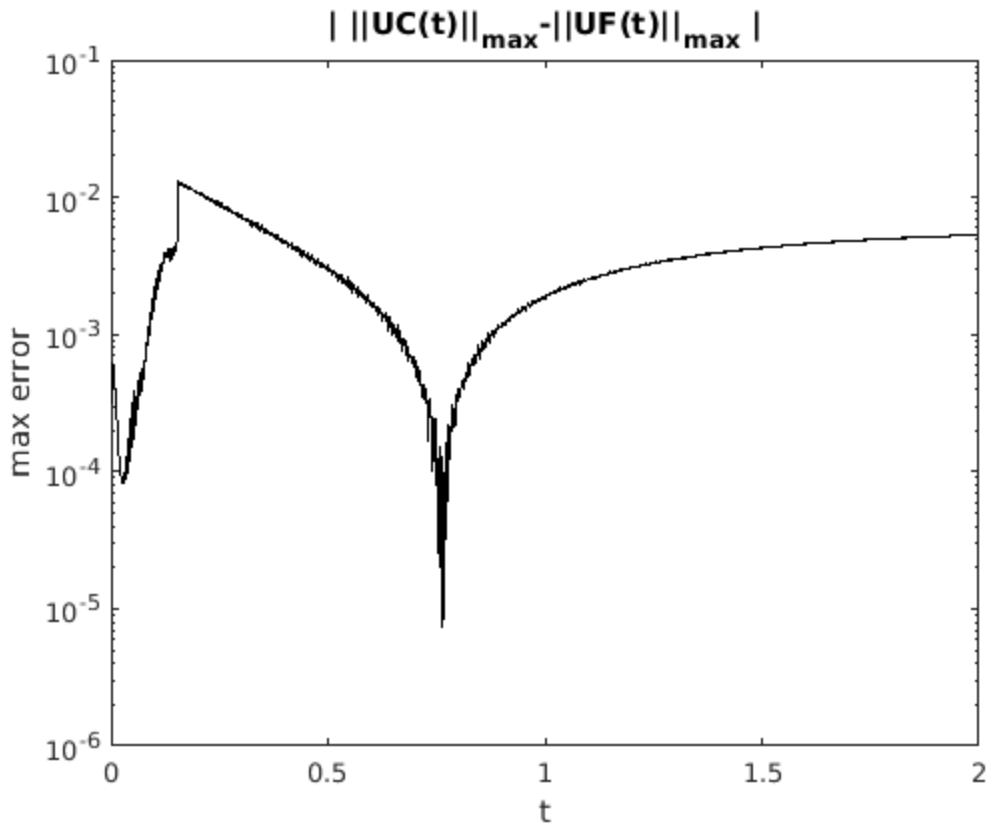
Solving diffusion equation with Neumann B.C.s of 1

```
clear all, close all, clc

Nc =  50; % Order of Cheb poly
Nf = 101; % Number of points for FD domain

disp(sprintf('Cheb Timing, num points: %d',Nc+1))
tic
[xc,tc,UC] = SM_Diff(Nc);
toc
disp(sprintf('Finite Difference timing, num points: %d',Nf))
tic
[xf,tf,UF] = FD_Diff(Nf);
toc

[XC,TC] = meshgrid(xc(2:end-1),tc);
[XF,TF] = meshgrid(xf(2:end-1),tf);

figure
h = surf(XC,TC,UC); set(h,'edgecolor','none')
xlabel('x'); ylabel('t'); zlabel('u');
title('Chebyshev Result')

figure
h = surf(XF,TF,UF); set(h,'edgecolor','none')
xlabel('x'); ylabel('t'); zlabel('u');
title('Finite Difference Result')

figure
maUC = max(abs(UC),[],2); maUF = max(abs(UF),[],2);
maer = abs(maUC-maUF);
semilogy(tc,maer,'k-')
xlabel('t'); ylabel('max error');
title('| ||UC(t)||_{max}-||UF(t)||_{max} |')
```

```
Cheb Timing, num points: 51
Elapsed time is 6.690054 seconds.
Finite Difference timing, num points: 101
Elapsed time is 0.846565 seconds.
```


Chebyshev Result
u
2.5
2
1.5
1
0.5
0
-0.5
2
1.5
1
0.5
0
t
-1
-0.5
0
0.5
1
x


Finite Difference Result
u
2.5
2
1.5
1
0.5
0
-0.5
2
1.5
1
0.5
0
t
-1
-0.5
0
0.5
1
x



Spectral Code

```
function [x,T,U] = SM_Diff(N)
  %% u_t = u_xx -1<x<1, with Neumann boundary conditions
  [D,x] = cheb(N);
  x = -x;   % Reverse order of x so x(1) = -1.
  D = -D;   % D is affected by reorder.
  D2 = D^2; % Thank Chebyshev for this wonderful property.
  u0 = cos(pi*x)+0.2*cos(5*pi*x)+1; % initial distribution.
  g0 = 1; gN = 1; gBC = [g0;gN];  % Neumann boundary conditions.
  B  = D2(2:end-1,[1 end]);  % u0 and uN still want to party,
  D2 = D2(2:end-1,2:end-1);  % but they werent invited to D2.
  A  = D([1 end],[1,end]);   % Dem no flux conditions
  C  = D([1 end],2:end-1);   % allow us to rewrite B*[u0;uN]
  % Solving on interior of field:
  %    RHS = @(t,u) D2*u + B*inv(A)*(gBC - C*u)
  % These are constant matrices! Compute them ONLY once.
  BiA   = B/A;
  Q     = D2 - BiA*C;
  b     = BiA*gBC;
  RHS   = @(t,u) Q*u+b; % I like this RHS better than the first one.
  t     = 0:0.001:2;
  [T,U] = ode45(RHS, t, u0(2:end-1));
end
```

Second Order Finite Difference Code

```
function [x,T,U] = FD_Diff(N)
  x  = linspace(-1,1,N); % get dat domain
  dx = x(2)-x(1);        % get dat spatial stepsize
  D2 = (1/dx^2)*toeplitz([ -2 1 zeros(1,N-2) ]');% 2nd order central
  u0 = cos(pi*x)+0.2*cos(5*pi*x)+1; % init distribution
  g0 = 1; gN = 1; gBC=[g0;gN]; % Seinfeld's least fav. BC
  B  = D2(2:end-1,[1 end]);    % To B D2,
  D2 = D2(2:end-1,2:end-1);    % or not D2 D2.
  A = [-1 0; 0 1]/dx;          % Fix D for ODE and optimize comp.
  C = [1 zeros(1,N-3); zeros(1,N-3) -1]/dx;
  BiA= B/A;
  b  = BiA*gBC;
  Q  = sparse(D2 - BiA*C); % This is still a tridiagonal matrix
  RHS = @(t,u) Q*u+b;
  t = 0:0.001:2;
  [T,U] = ode45(RHS, t, u0(2:end-1));
end
```

Doing this problem with the expressive linear algebra syntax shown in the spectral solver provides the following right hand side,

```
RHS = @(t,u) D2*u + (B/A)*([g0;gN]-C*u),
```

where

```
A = D([1 end],[1 end]);
C = D([1 end],2:end-1);
```

and `D` is the differental operator, which we have already defined within the interior with the second order central scheme, but now needs to be defined along the boundaries (along the first and last rows):

```
D([1 end],:) = [ -1 1 zeros(1,N-2); ...
                 zeros(1,N-2) -1 1 ] / dx;
```

Also, since `B` is only nonzero in its upper left and lower right corner, implies that `B*inv(A)*C` is going to be nonzero only in its upper left and lower right corners as well. Therefore, unlike the spectral collocation method, the final `Q` operator will benefit from a sparse representation (as it's tridiagonal).

Notice above the use of Euler on boundaries (essentially just Taylor series expansions). The one step truncation error of these methods is second order, which is consistent with the second order central scheme used on the interior. That is,

```
u_0 = u_1 - dx*g0
    = u(-1+dx) - dx*g0
    = u_0 + dx u_0' + .5 dx^2 u_0'' + O(dx^3) - dx * g0
    = u_0 + dx g0   + .5 dx^2 u_0'' + O(dx^3) - dx * g0
    = u_0 + O(dx^2),

u_N = u_{N-1} + dx*gN = u(1-dx) + dx*gN = u_N + O(dx^2),
```

since `g0 = u_x(-1)`, and `gN = u_x(1)`. This leads to the explicit right hand side for the interior field:

```
RHS = @(t,u) D2*u + B*[u(1)-dx*g0;u(end)+dx*gN].
```

However, the representation used by the code is about 9% more efficient when `Q` is not sparse, and 33% more efficient when it is sparse (running on mathpost.asu.edu):

```
              | avg time of 10 runs (N=100) (s)
--------------|---------------------------------
 b explicit   |        1.2
 Q non sparse |        1.1
 Q sparse     |         .9
```

Published with MATLAB® R2017a