AE 370: HW8

Linyi Hou

April 24, 2020

Part 1

We begin with the 1D heat equation provided in the problem statement:

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + g(x,t), \quad 0 \leq t \leq T, \ a \leq x \leq b \tag{1}$$

where $a = 2, b = 15, \kappa = 2, T = 5$. The initial and boundary conditions are:

$$\text{I.C.s}: \ u(x, t = 0) = 0 \tag{2}$$

$$\text{B.C.s}: \ u(x = a, t) = 0 \tag{3}$$

$$u(x = b, t) = 0 \tag{4}$$

<u>**Discretizing the spatial domain:**</u> First, we discretize the space variable x into $n + 1$ pieces using the following equation:

$$x_j = a + \frac{(b-a)(j-1)}{n}, \ j = 1, \ldots, n+1 \tag{5}$$

Then, per requirement from the problem statement, we locally construct 2^{nd} order polynomials. We choose to use centered Lagrange polynomials, and arrive at the formulation

$$u(x,t) \approx \sum_{i=j-1}^{j+1} c_i(t) L_i^{(j)}(x) \tag{6}$$

where $L_i^{(j)}(x)$ is the Lagrange basis polynomial and $c_i(t)$ are the unknown coefficients that need to be solved for. Using the property of Lagrange polynomials:

$$L_i^{(j)}(x_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \tag{7}$$

Observe from Equations (6) and (7) that $u(x_j, t) \approx c_j(t)$. Therefore $c_j(t)$ is an approximation to $u(x_j, t)$, and thus solving for coefficients $b_i(t)$ is equivalent to solving for approximations to the exact solutions of u at x_i:

$$u(x,t) \approx \sum_{i=j-1}^{j+1} u_i(t) L_i^{(j)}(x) \tag{8}$$

Conversion to initial value problem: Substitute Equation (8) into Equation (1) to get:

$$
\begin{aligned}
\sum_{i=j-1}^{j+1} \dot{u}_i(t) L_j^{(i)}(x_j) &= \kappa \sum_{i=j-1}^{j+1} u_i(t) \left.\frac{d^2 L_j^{(i)}}{dx^2}\right|_{x=x_j} + g(x_j, t) \quad (j = 2, \ldots, n) \\
\Rightarrow \quad \dot{u}_j(t) &= \kappa \sum_{i=j-1}^{j+1} u_i(t) \left.\frac{d^2 L_j^{(i)}}{dx^2}\right|_{x=x_j} + g(x_j, t) \quad (j = 2, \ldots, n) \\
\Rightarrow \quad \dot{u}_j(t) &= \frac{\kappa}{\Delta x^2}[u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)] + g(x_j, t)
\end{aligned} \tag{9}
$$

Matrix form: The IVP shown in Equation (9) can be expressed as $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u} + \mathbf{g}$,

$$
\dot{\begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}} = \frac{\kappa}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} g(x_2, t) \\ g(x_3, t) \\ \vdots \\ g(x_{n-1}, t) \\ g(x_n, t) \end{bmatrix} \tag{10}
$$

with initial conditions provided by Equation (2)

$$
\mathbf{u}(t = 0) = \mathbf{0} \tag{11}
$$

Note also that the boundary conditions from Equations (3) and (4) have already been included in Equation (10), where the first and last elements of the $\mathbf{g}$ term enforce the boundary conditions at a and b, respectively.

Part 2

Taking the trapezoid method,

$$
u_{k+1} = u_k + \frac{1}{2}\Delta t f(u_k, tk) + \frac{1}{2}\Delta t f(u_{k+1}, t_{k+1}) \tag{12}
$$

we can simplify the particular IVP shown in Equation (10) using the fact that $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u} + \mathbf{g}$:

$$
u_{k+1} = u_k + \frac{1}{2}\Delta t f(u_k, t_k) + \frac{1}{2}\Delta t A u_{k+1} + \frac{1}{2}\Delta t g(t_{k+1}) \tag{13}
$$

and finally solve for $\mathbf{u}_{k+1}$:

$$
u_{k+1} = \left(\mathbb{I} - \frac{1}{2}\Delta t A\right)^{-1} \left(u_k + \frac{1}{2}\Delta t f(u_k) + \frac{1}{2}\Delta t g(t_{k+1})\right) \tag{14}
$$

Part 3

Figures and code have been attached in Appendix A and Appendix B, respectively.

Part 4

Figures and code have been attached in Appendix A and Appendix B, respectively.

Appendix A: Figures

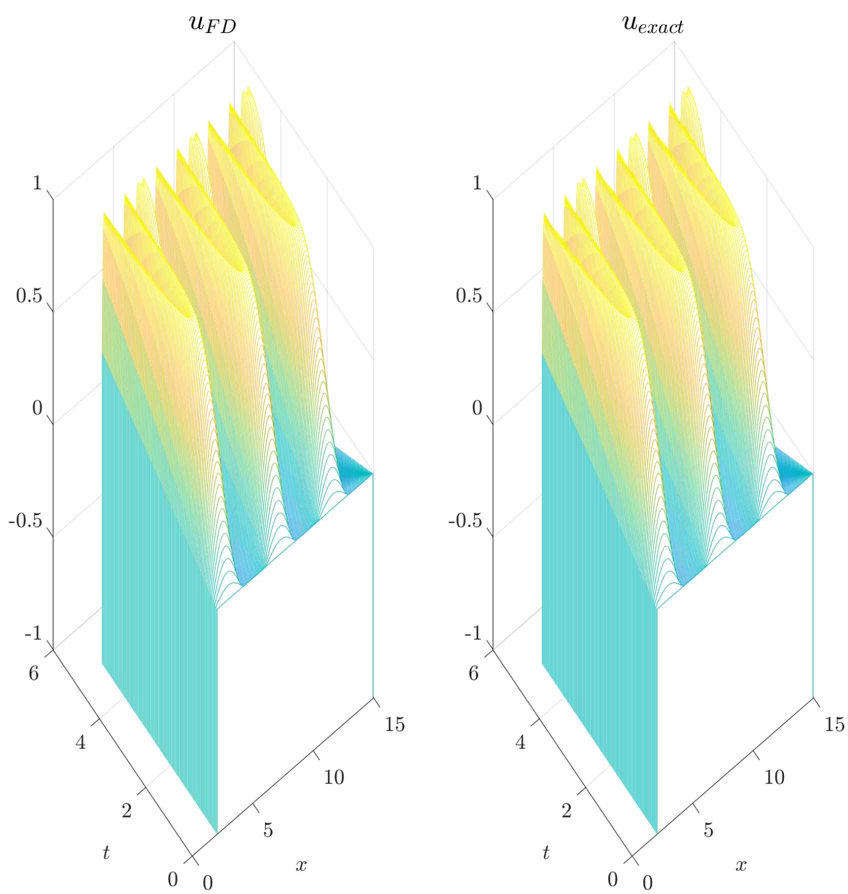

Figure 1: Part 3a Waterfall Plot

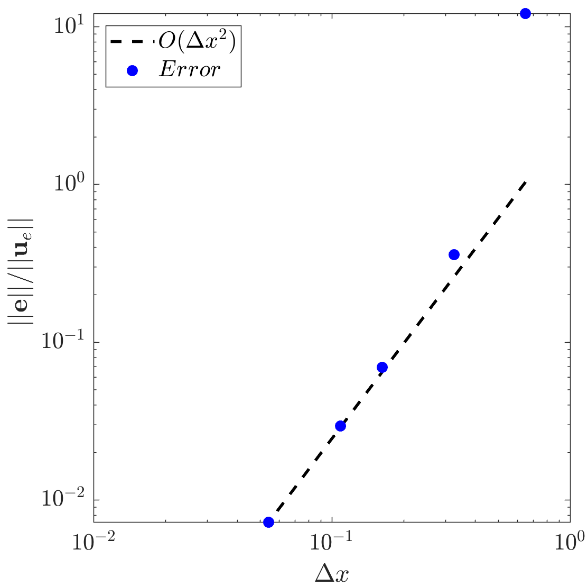

Figure 2: Part 3b Spatial Convergence Plot

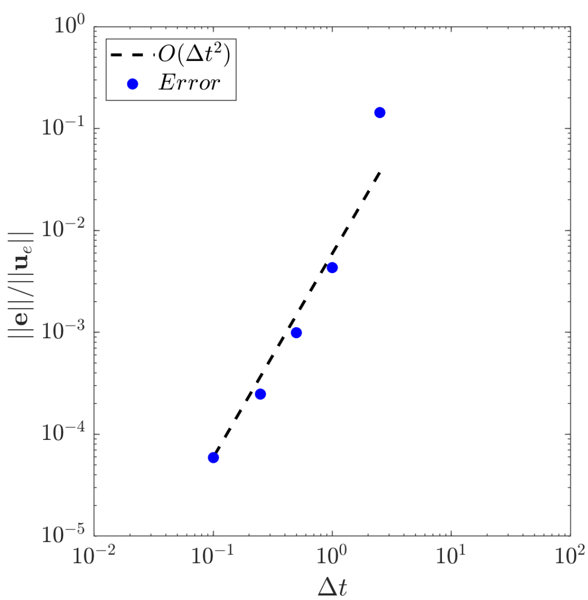

Figure 3: Part 4 Temporal Convergence Plot

Appendix B: Code

```
%% Problem 3
close all
clear;clc

%solve heat eqn over 2 < x < 15
%with BCs u(2,t) = u(15,t) = 0
%and IC u(x,0) = 0
%This code does a convergence test in space

%params for problem
a = 2;
b = 15;
kappa = 2;
ln = b-a;
T = 5; %Final time to run to
dt = 1e-3; %Make dt small in spatial convergence test so that time error
           %doesn't pollute convergence


%exact sol
uex = @(x,t) sin( t.* sin(6.*pi.*(x-a)./ln) );

%initial condition
eta = @(x) uex(x,0);

%g(x) =
fcn = @(x,t) - kappa*((36*t.^2.*pi^2.*cos((6.*pi.*(a - x))./ln).^2 ...
    .*sin(t.*sin((6.*pi.*(a - x))./ln)))./ln.^2 + ...
    (36.*t.*pi^2.*sin((6.*pi.*(a - x))./ln).*cos(t.*sin((6.*pi.*(a - x))
        ...
    ./ln)))/ln.^2) - sin((6.*pi.*(a - x))./ln)...
    .*cos(t.*sin((6.*pi.*(a - x))./ln));

%# of n points to use
nvect = [20; 40; 80; 120; 240];

%initialize error vect
err = zeros( size( nvect ) );
```

```matlab

for j = 1 : length( nvect )


    %---Build n, xj points, A matrix and g vector
        %# of grid points
        n = nvect( j );

        %Build interp points
        xj = linspace(a,b,n+1)';

        %grid spacing (uniform)
        dx = ( b - a ) / n;

        %Build A matrix
        %Use truncated version from lecture notes
        A = ( kappa / dx^2 ) ...
          * ( diag(ones(n-2,1),1) + diag(ones(n-2,1),-1) - 2*eye(n-1) );

        %Also build identity mat (same size as A)
        I = eye(size(A));

        %Build g for this set of xj
        g = @(t) fcn(xj(2:end-1),t);

        %Build RHS for IVP, f(u,t)
        f = @(u,t) A*u + g(t);
    %---

    %---Initialize for time stepping
        uk = eta(xj(2:end-1));
        tk = 0;
        tvect = dt : dt : T;

        %# snapshots to save (don't mess with this; it sets things up so
        %the solution is only saved a relatively small number of times
        %to keep your data storage from growing to large)
        nsnps = 100;
        ind = max( 1, round(length(tvect)/nsnps) );
        tsv = tvect( 1 : ind : end );
```

```

        u = zeros( n−1, length(tsv));
        cnt = 1;
    %−−−

    %−−−Do time stepping

    cf = inv(eye(size(A))−0.5*dt*A); % coefficient to solve for ukp1
                                     % pre−compute to improve speed

    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        %Update solution at next time using trap method
        ukp1 = cf * (uk + 0.5*dt*f(uk,tk) + 0.5*dt*g((tkp1)));

        %Update solution variable & time
        uk = ukp1;
        tk = tkp1;

        %Again, leave this. It sets things up to only save for a
            relatively
        %small # of times.
        if min(abs( tkp1−tsv ) ) < 1e−8

            u(:,cnt) = uk;
            cnt = cnt + 1;
        end

    end
    %−−−

    err(j) = norm( uk − uex(xj(2:end−1),tk) ) / norm( uex(xj(2:end−1),tk)
         );

end


%−−Waterfall plot of solns
```

```
    [X,T] = meshgrid( xj(2:end-1), tsv );

    figure(1), subplot(1,2,1)
    waterfall( X,T, u.' ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-1 1])

    subplot(1,2,2)
    waterfall( X,T, uex(X,T) ), hold on
    set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
    title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
    xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
    ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
    zlim([-1 1])

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [25 25])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 25 25])
    set(gcf, 'PaperPosition', [0 0 25 25])

    svnm = 'p3_waterfall';
    print( '-dpng', svnm, '-r200' )
%--

%--Error plots
    figure(2)
    c = err(end)/(dx^2);
    loglog( ln./nvect, c*(ln./nvect).^2, 'k--', 'linewidth', 2 ), hold on

    %plot err
    loglog( ln./nvect, err , 'b.', 'markersize', 26 )
    xlim([1e-2 1])
    h = legend('$O(\Delta x^2)$', '$Error$');
```

```
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest'
        )

    %make pretty
    xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex',
        'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])

    svnm = 'p3_error';
    print( '-dpng', svnm, '-r200' )

%--

%% Problem 4
close all
clear;clc

%solve heat eqn over 2 < x < 15
%with BCs u(2,t) = u(15,t) = 0
%and IC u(x,0) = 0
%This code does a convergence test in time

%params for problem
a = 2;
b = 15;
kappa = 2;
ln = b-a;
T = 5; %Final time to run to

```

```
%Make dx small in spatial convergence test so that spatial error
%doesn't pollute convergence
n = 3000;
dx = (b-a)/n;

%exact sol
uex = @(x,t) sin( t.* sin(6.*pi.*(x-a)./ln) );

%initial condition
eta = @(x) uex(x,0);

%g(x) =
fcn = @(x,t) - kappa*((36*t.^2.*pi^2.*cos((6.*pi.*(a - x))./ln).^2 ...
    .*sin(t.*sin((6.*pi.*(a - x))./ln)))./ln.^2 + ...
    (36.*t.*pi^2.*sin((6.*pi.*(a - x))./ln).*cos(t.*sin((6.*pi.*(a - x))
        ...
    ./ln)))/ln.^2) - sin((6.*pi.*(a - x))./ln)...
    .*cos(t.*sin((6.*pi.*(a - x))./ln));

%# of n points to use
dtvect = [2.5; 1; 5e-1; 2.5e-1; 1e-1];

%initialize error vect
err = zeros( size( dtvect ) );

for j = 1 : length( dtvect )


    %---Build xj points, A matrix and g vector
        %time step size
        dt = dtvect( j );

        %Build interp points
        xj = linspace(a,b,n+1)';

        %Build A matrix
        %Use truncated version from lecture notes
        A = ( kappa / dx^2 ) ...
          * ( diag(ones(n-2,1),1) + diag(ones(n-2,1),-1) - 2*eye(n-1) );

```

```
        %Also build identity mat (same size as A)
        I = eye(size(A));

        %Build g for this set of xj
        g = @(t) fcn(xj(2:end-1),t);

        %Build RHS for IVP, f(u,t)
        f = @(u,t) A*u + g(t);
    %---

    %---Initialize for time stepping
        uk = eta(xj(2:end-1));
        tk = 0;
        tvect = dt : dt : T;
    %---

    %---Do time stepping

    cf = inv(eye(size(A))-0.5*dt*A); % coefficient to solve for ukp1
                                     % pre-compute to improve speed

    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        %Update solution at next time using trap method
        ukp1 = cf * (uk + 0.5*dt*f(uk,tk) + 0.5*dt*g((tkp1)));

        uk = ukp1;
        tk = tkp1;

    end
    %---

    err(j) = norm( uk - uex(xj(2:end-1),tk) ) / norm( uex(xj(2:end-1),tk) );
end


%--Error plots
```

```
    figure(3)
    c = err(end)*(1./dt^2);
    loglog( dtvect, c*(dtvect).^2, 'k--', 'linewidth', 2 ), hold on

    %plot err
    loglog( dtvect, err , 'b.', 'markersize', 26 )
    xlim([1e-2 100])


    %make pretty
    h = legend('$O(\Delta t^2)$', '$Error$');
    set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )

    xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
    ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', 'fontsize', 16)

    set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

    set(gcf, 'PaperPositionMode', 'manual')
    set(gcf, 'Color', [1 1 1])
    set(gca, 'Color', [1 1 1])
    set(gcf, 'PaperUnits', 'centimeters')
    set(gcf, 'PaperSize', [15 15])
    set(gcf, 'Units', 'centimeters' )
    set(gcf, 'Position', [0 0 15 15])
    set(gcf, 'PaperPosition', [0 0 15 15])

    svnm = 'p4_error';
    print( '-dpng', svnm, '-r200' )

%--
```