

AE 370: HW8

Linyi Hou

April 24, 2020

Part 1

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

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

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

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

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

$$u(x = b, t) = 0 \quad (4)$$

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

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

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

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

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:

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

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 :

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

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

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

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

$$\begin{bmatrix} \dot{u}_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \frac{\kappa}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} g(x_2, t) \\ g(x_3, t) \\ \vdots \\ g(x_{n-1}, t) \\ g(x_n, t) \end{bmatrix} \tag{10}$$

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,

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

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

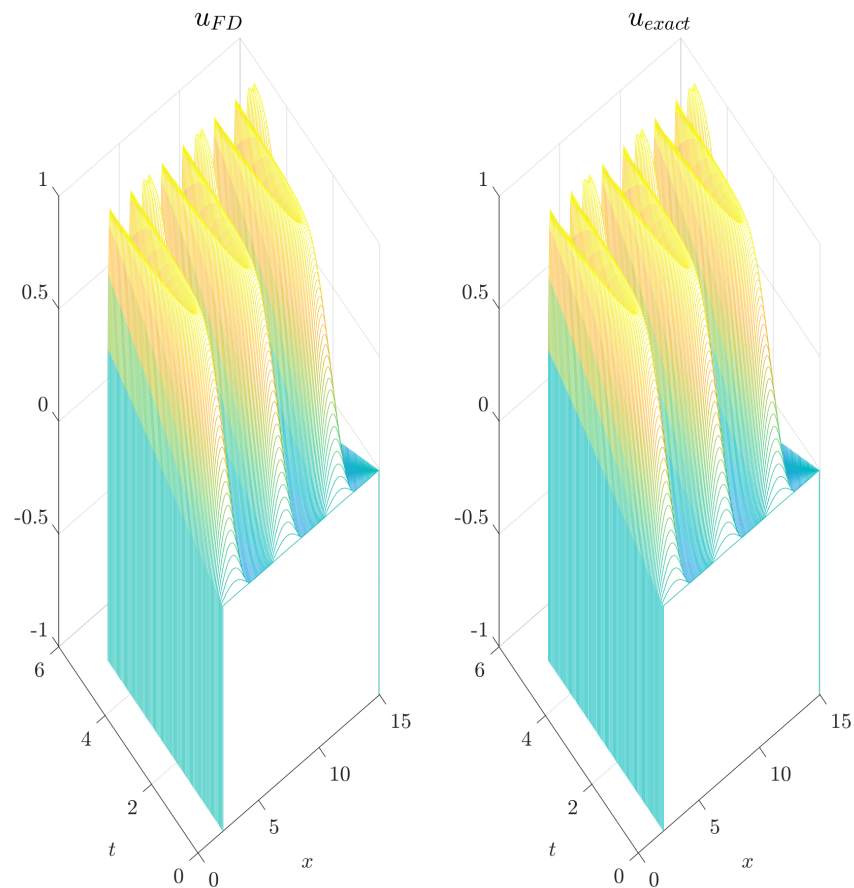


Figure 1: Part 3a Waterfall Plot

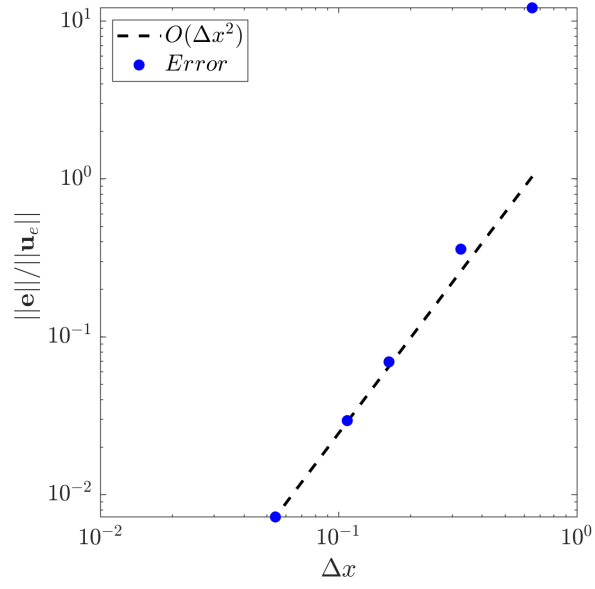


Figure 2: Part 3b Spatial Convergence Plot

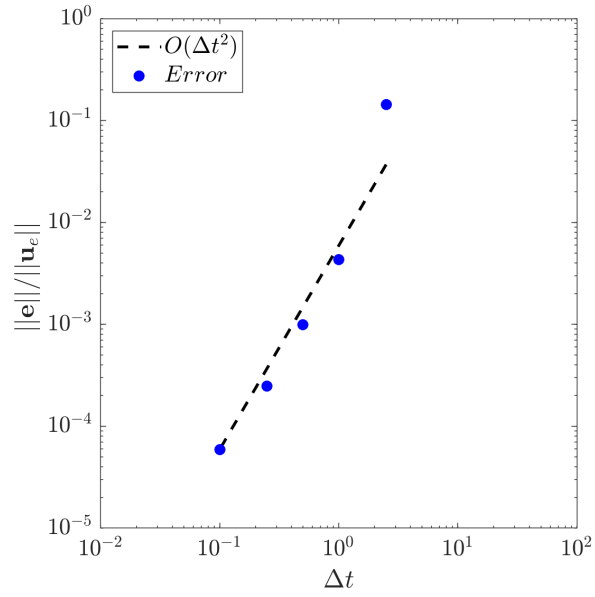


Figure 3: Part 4 Temporal Convergence Plot

Appendix B: Code

```
1 %% Problem 3
2 close all
3 clear;clc
4
5 %solve heat eqn over  $2 < x < 15$ 
6 %with BCs  $u(2,t) = u(15,t) = 0$ 
7 %and IC  $u(x,0) = 0$ 
8 %This code does a convergence test in space
9
10 %params for problem
11 a = 2;
12 b = 15;
13 kappa = 2;
14 ln = b-a;
15 T = 5; %Final time to run to
16 dt = 1e-3; %Make dt small in spatial convergence test so that time error
17           %doesn't pollute convergence
18
19
20 %exact sol
21 uex = @(x,t) sin( t.* sin(6.*pi.*(x-a)./ln) );
22
23 %initial condition
24 eta = @(x) uex(x,0);
25
26 %g(x) =
27 fcn = @(x,t) - kappa*((36*t.^2.*pi^2.*cos((6.*pi.*(a - x))./ln).^2 ...
28               .*sin(t.*sin((6.*pi.*(a - x))./ln)))./ln.^2 + ...
29               (36.*t.*pi^2.*sin((6.*pi.*(a - x))./ln).*cos(t.*sin((6.*pi.*(a - x))
30               ...
31               ./ln)))/ln.^2) - sin((6.*pi.*(a - x))./ln)...
32               .*cos(t.*sin((6.*pi.*(a - x))./ln));
33
34 %# of n points to use
35 nvect = [20; 40; 80; 120; 240];
36
37 %initialize error vect
38 err = zeros( size( nvect ) );
```

```

38
39 for j = 1 : length( nvect )
40
41
42     %——Build n, xj points, A matrix and g vector
43     %# of grid points
44     n = nvect( j );
45
46     %Build interp points
47     xj = linspace(a,b,n+1)';
48
49     %grid spacing (uniform)
50     dx = ( b - a ) / n;
51
52     %Build A matrix
53     %Use truncated version from lecture notes
54     A = ( kappa / dx^2 ) ...
55         * ( diag(ones(n-2,1),1) + diag(ones(n-2,1),-1) - 2*eye(n-1) );
56
57     %Also build identity mat (same size as A)
58     I = eye(size(A));
59
60     %Build g for this set of xj
61     g = @(t) fcn(xj(2:end-1),t);
62
63     %Build RHS for IVP, f(u,t)
64     f = @(u,t) A*u + g(t);
65     %——
66
67     %——Initialize for time stepping
68     uk = eta(xj(2:end-1));
69     tk = 0;
70     tvect = dt : dt : T;
71
72     %# snapshots to save (don't mess with this; it sets things up so
73     %the solution is only saved a relatively small number of times
74     %to keep your data storage from growing to large)
75     nsmps = 100;
76     ind = max( 1, round(length(tvect)/nsmps) );
77     tsv = tvect( 1 : ind : end );

```

```

78
79     u = zeros( n-1, length(tsv));
80     cnt = 1;
81     %——
82
83     %——Do time stepping
84
85     cf = inv(eye(size(A))-0.5*dt*A); % coefficient to solve for ukp1
86                                     % pre-compute to improve speed
87
88     for jj = 1 : length( tvect )
89
90         tkp1 = tk + dt;
91
92         %Update solution at next time using trap method
93         ukp1 = cf * (uk + 0.5*dt*f(uk,tk) + 0.5*dt*g((tkp1)));
94
95         %Update solution variable & time
96         uk = ukp1;
97         tk = tkp1;
98
99         %Again, leave this. It sets things up to only save for a
100            relatively
101            %small # of times.
102         if min(abs( tkp1-tsv ) ) < 1e-8
103
104             u(:,cnt) = uk;
105             cnt = cnt + 1;
106         end
107     end
108     %——
109
110     err(jj) = norm( uk - uex(xj(2:end-1),tk) ) / norm( uex(xj(2:end-1),tk)
111         );
112 end
113
114
115 %——Waterfall plot of solns

```

```

116 [X,T] = meshgrid( xj(2:end-1), tsv );
117
118 figure(1), subplot(1,2,1)
119 waterfall( X,T, u.' ), hold on
120 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
121 title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
122 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
123 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
124 zlim([-1 1])
125
126 subplot(1,2,2)
127 waterfall( X,T, uex(X,T) ), hold on
128 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
129 title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
130 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
131 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
132 zlim([-1 1])
133
134 set(gcf, 'PaperPositionMode', 'manual')
135 set(gcf, 'Color', [1 1 1])
136 set(gca, 'Color', [1 1 1])
137 set(gcf, 'PaperUnits', 'centimeters')
138 set(gcf, 'PaperSize', [25 25])
139 set(gcf, 'Units', 'centimeters' )
140 set(gcf, 'Position', [0 0 25 25])
141 set(gcf, 'PaperPosition', [0 0 25 25])
142
143 svnm = 'p3_waterfall';
144 print( '-dpng', svnm, '-r200' )
145 %—
146
147 %—Error plots
148 figure(2)
149 c = err(end)/(dx^2);
150 loglog( ln./nvect, c*(ln./nvect).^2, 'k—', 'linewidth', 2 ), hold on
151
152 %plot err
153 loglog( ln./nvect, err , 'b.', 'markersize', 26 )
154 xlim([1e-2 1])
155 h = legend('$0(\Delta x^2)$', '$Error$');

```



```

156     set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest'
157         )
158     %make pretty
159     xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
160     ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex',
161         'fontsize', 16)
162
163     set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
164
165     set(gcf, 'PaperPositionMode', 'manual')
166     set(gcf, 'Color', [1 1 1])
167     set(gca, 'Color', [1 1 1])
168     set(gcf, 'PaperUnits', 'centimeters')
169     set(gcf, 'PaperSize', [15 15])
170     set(gcf, 'Units', 'centimeters' )
171     set(gcf, 'Position', [0 0 15 15])
172     set(gcf, 'PaperPosition', [0 0 15 15])
173
174     svnm = 'p3_error';
175     print( '-dpng', svnm, '-r200' )
176
177
178 %% Problem 4
179 close all
180 clear;clc
181
182 %solve heat eqn over  $2 < x < 15$ 
183 %with BCs  $u(2,t) = u(15,t) = 0$ 
184 %and IC  $u(x,0) = 0$ 
185 %This code does a convergence test in time
186
187 %params for problem
188 a = 2;
189 b = 15;
190 kappa = 2;
191 ln = b-a;
192 T = 5; %Final time to run to
193

```

```

194 %Make dx small in spatial convergence test so that spatial error
195 %doesn't pollute convergence
196 n = 3000;
197 dx = (b-a)/n;
198
199 %exact sol
200 uex = @(x,t) sin( t.* sin(6.*pi.*(x-a)./ln) );
201
202 %initial condition
203 eta = @(x) uex(x,0);
204
205 %g(x) =
206 fcn = @(x,t) - kappa*((36*t.^2.*pi^2.*cos((6.*pi.*(a - x))./ln).^2 ...
207     .*sin(t.*sin((6.*pi.*(a - x))./ln)))./ln.^2 + ...
208     (36.*t.*pi^2.*sin((6.*pi.*(a - x))./ln).*cos(t.*sin((6.*pi.*(a - x))
209     ...
210     ./ln)))./ln.^2) - sin((6.*pi.*(a - x))./ln)...
211     .*cos(t.*sin((6.*pi.*(a - x))./ln));
212
213 %# of n points to use
214 dtvect = [2.5; 1; 5e-1; 2.5e-1; 1e-1];
215
216 %initialize error vect
217 err = zeros( size( dtvect ) );
218
219 for j = 1 : length( dtvect )
220
221     %——Build xj points, A matrix and g vector
222     %time step size
223     dt = dtvect( j );
224
225     %Build interp points
226     xj = linspace(a,b,n+1)';
227
228     %Build A matrix
229     %Use truncated version from lecture notes
230     A = ( kappa / dx^2 ) ...
231         * ( diag(ones(n-2,1),1) + diag(ones(n-2,1),-1) - 2*eye(n-1) );
232

```

```

233     %Also build identity mat (same size as A)
234     I = eye(size(A));
235
236     %Build g for this set of xj
237     g = @(t) fcn(xj(2:end-1),t);
238
239     %Build RHS for IVP, f(u,t)
240     f = @(u,t) A*u + g(t);
241     %——
242
243     %——Initialize for time stepping
244     uk = eta(xj(2:end-1));
245     tk = 0;
246     tvect = dt : dt : T;
247     %——
248
249     %——Do time stepping
250
251     cf = inv(eye(size(A))-0.5*dt*A); % coefficient to solve for ukp1
252                                     % pre-compute to improve speed
253
254     for jj = 1 : length( tvect )
255
256         tkp1 = tk + dt;
257
258         %Update solution at next time using trap method
259         ukp1 = cf * (uk + 0.5*dt*f(uk,tk) + 0.5*dt*g((tkp1)));
260
261         uk = ukp1;
262         tk = tkp1;
263
264     end
265     %——
266
267     err(jj) = norm( uk - uex(xj(2:end-1),tk) ) / norm( uex(xj(2:end-1),tk)
268                                     );
269
270
271     %——Error plots

```

```

272 figure(3)
273 c = err(end)*(1./dt^2);
274 loglog( dtvect, c*(dtvect).^2, 'k—', 'linewidth', 2 ), hold on
275
276 %plot err
277 loglog( dtvect, err , 'b.', 'markersize', 26 )
278 xlim([1e-2 100])
279
280
281 %make pretty
282 h = legend('$0(\Delta t^2)$', '$Error$');
283 set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest'
284      )
285
286 xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
287 ylabel( '$||\textbf{e}||/||\textbf{u}_e||$', 'interpreter', 'latex',
288        'fontsize', 16)
289
290 set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
291
292 set(gcf, 'PaperPositionMode', 'manual')
293 set(gcf, 'Color', [1 1 1])
294 set(gca, 'Color', [1 1 1])
295 set(gcf, 'PaperUnits', 'centimeters')
296 set(gcf, 'PaperSize', [15 15])
297 set(gcf, 'Units', 'centimeters' )
298 set(gcf, 'Position', [0 0 15 15])
299 set(gcf, 'PaperPosition', [0 0 15 15])
300
301 svnm = 'p4_error';
302 print( '-dpng', svnm, '-r200' )
303 %—

```