

Problem Set 7

1. (25 pts.) Consider the variable coefficient diffusion equation

$$u_t = (Du_x)_x, \quad 0 < x < 1, \quad t > 0$$

where $D > 0$ is a function of x , with initial conditions $u(x, 0) = u_0(x)$ and subject to boundary conditions $u_x(0, t) = 0$, $u(1, t) = 0$.

- (a) Show that this problem is well-posed by proving the energy estimate

$$\frac{d}{dt} \|u\|^2 \leq 0,$$

where $\|u\|^2 = (u, u) = \int_0^1 u^2 dx$.

$$\begin{aligned} u_t &= (Du_x)_x \\ u u_t &= u (Du_x)_x \\ \int_0^1 u u_t dx &= \int_0^1 u (Du_x)_x dx \\ \int_0^1 \frac{1}{2} \frac{d\|u\|^2}{dt} dx &= \cancel{u(1) (Du_x(1))} - \cancel{u(0) (Du_x(0))} - \int_0^1 u_x Du_x dx \\ \int_0^1 \frac{1}{2} \frac{d\|u\|^2}{dt} dx &= - \int_0^1 D (u_x)^2 dx \end{aligned}$$

Here, $D \geq 0$ and $(u_x)^2 \geq 0$ always and hence,

$$\frac{1}{2} \frac{d\|u\|^2}{dt} \leq 0$$

Thus, this PDE is well posed.

- (b) Propose a computational grid, a semi-discretization of the PDE in space, and a treatment of the boundary conditions for which you can prove a discrete energy estimate of the form $\frac{d}{dt} \|u\|_h^2 \leq 0$ for an appropriately defined discrete norm $\|\cdot\|_h$. Prove your discrete energy estimate.

$$\Delta x = 1/N$$

$$x_j = (j - \frac{1}{2})\Delta x, \quad j = 0, 1, 2, \dots, N$$

and here, $D_{-x}v_j = \frac{v_{j+\frac{1}{2}} - v_{j-\frac{1}{2}}}{\Delta x}$

$$\begin{aligned} \frac{dv_j}{dt} &= D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x}v_j \right) \\ v_j \frac{dv_j}{dt} &= v_j D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x}v_j \right) \\ \Delta x \sum_{j=0}^{N-1} \frac{1}{2} \frac{d\|v_j\|_h^2}{dt} &= \Delta x \sum_{j=0}^{N-1} v_j D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x}v_j \right) \\ \Delta x \sum_{j=0}^{N-1} \frac{1}{2} \frac{d\|v_j\|_h^2}{dt} &= \frac{\Delta x}{\Delta x^2} \sum_{j=0}^{N-1} v_j \Delta_{+x} \left(\nu_{j-\frac{1}{2}} \Delta_{-x}v_j \right) \end{aligned}$$

Using summation by parts, we can simplify and get to the form

$$\begin{aligned}\Delta x \sum_{j=0}^{N-1} \frac{1}{2} \frac{d||v_j||_h^2}{dt} &= \frac{\Delta x}{\Delta x^2} \left(\cancel{v_N} \left(\nu_{N-\frac{1}{2}} \Delta_{-x} v_N \right) - v_0 \nu_{-\frac{1}{2}} \Delta_{-x} v_0 - \sum_{j=0}^{N-1} \left(\nu_{j+\frac{1}{2}} \Delta_{-x} v_{j+1} \right) \Delta_{+x} v_j \right) \\ \sum_{j=0}^{N-1} \frac{1}{2} \frac{d||v_j||_h^2}{dt} &= \left(-\frac{1}{\Delta x} v_0 \nu_{-\frac{1}{2}} \cancel{D_{-x} v_0} - \frac{1}{\Delta x^2} \sum_{j=0}^{N-1} \left(\nu_{j+\frac{1}{2}} (\Delta_{+x} v_j)^2 \right) \right)\end{aligned}$$

Since $\nu_{j+\frac{1}{2}} \geq 0$ and $(\Delta_{+x} v_j)^2 \geq 0$,

$$\frac{d||v_j||_h^2}{dt} \leq 0$$

- (c) [Prove a fully discrete energy estimate for Crank-Nicolson integration of the scheme from part \(b\) above.](#)

$$\begin{aligned}\frac{v_j^{n+1} - v_j^n}{\Delta t} &= D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x} \right) \frac{v_j^{n+1} + v_j^n}{2} \\ (v_j^{n+1})^2 - (v_j^n)^2 &= \Delta t D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x} \right) \frac{v_j^{n+1} + v_j^n}{2} (v_j^{n+1} + v_j^n) \\ (v_j^{n+1})^2 - (v_j^n)^2 &= \frac{\Delta t}{4} D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x} \right) v_j^{n+\frac{1}{2}} v_j^{n+\frac{1}{2}} \\ \Delta x \sum_{j=0}^{N-1} (v_j^{n+1})^2 - (v_j^n)^2 &= -\Delta x \sum_{j=0}^{N-1} \left(D_{+x} v_j^{n+\frac{1}{2}}, D_{+x} \left(\nu_{j-\frac{1}{2}} v_j^{n+\frac{1}{2}} \right) \right) + \left[v_j^{n+\frac{1}{2}} D_{+x} v_j^{n+\frac{1}{2}} \right]_{j=0}^{j=N}\end{aligned}$$

- (d) [Write a code to implement the scheme in part \(d\). Demonstrate second-order convergence using a manufactured solution.](#)

Method of manufactured solutions is used to verify second-order of convergence.

$$\begin{aligned}u_{ex} &= \cos \left(\frac{\pi x}{2} \right) \sin t, \quad u_{ex}(x=1, t) = 0 \\ u_t &= \cos \left(\frac{\pi x}{2} \right) \cos t \\ u_x &= -\frac{\pi}{2} \sin \left(\frac{\pi x}{2} \right) \sin t, \quad u_x(x=0, t) = 0 \\ u_{xx} &= -\frac{\pi^2}{4} \cos \left(\frac{\pi x}{2} \right) \sin t\end{aligned}$$

$$u_t - (\nu u_x)_x = u_t - (\nu_x u_x + \nu u_{xx}) = f(x, t)$$

This solution satisfies the boundary conditions specified in the problem. The discrete set of equations for the Crank-Nicholson scheme is,

$$\begin{aligned}D_{+t} v_j^n &= D_{+x} \left(\nu_{j-\frac{1}{2}} D_{-x} \right) \left(\frac{v_j^{n+1} + v_j^n}{2} \right) + \left(\frac{f_j^{n+1} + f_j^n}{2} \right) \\ D_{+t} v_j^n &= \frac{1}{2} D_{+x} \left(\nu_{j-\frac{1}{2}} v_j^{n+1} \right) + \frac{1}{2} D_{+x} \left(\nu_{j-\frac{1}{2}} v_j^n \right) + \left(\frac{f_j^{n+1} + f_j^n}{2} \right)\end{aligned}$$

Setting $\hat{F} = \left(\frac{f_j^{n+1} + f_j^n}{2} \right)$ and $\Gamma = \frac{\Delta t}{\Delta x^2}$, this simplifies to,

$$-\frac{\Gamma}{2}\nu_{j-\frac{1}{2}}v_{j-1}^{n+1} + \left(1 + \frac{\Gamma}{2}(\nu_{j+\frac{1}{2}} + \nu_{j-\frac{1}{2}})\right)v_j^{n+1} - \frac{\Gamma}{2}\nu_{j+\frac{1}{2}}v_{j+1}^{n+1} =$$

$$\frac{\Gamma}{2}\nu_{j-\frac{1}{2}}v_{j-1}^n + \left(1 - \frac{\Gamma}{2}(\nu_{j+\frac{1}{2}} + \nu_{j-\frac{1}{2}})\right)v_j^n + \frac{\Gamma}{2}\nu_{j+\frac{1}{2}}v_{j+1}^n + \Delta t \hat{F}$$

$$j = 1, 2, \dots, N-1$$

Boundary conditions are given by,

$$\frac{v_1^{n+1} - v_0^{n+1}}{\Delta x} = 0, \quad \frac{v_N^{n+1} + v_{N-1}^{n+1}}{2} = 0$$

The code is shown in Listing 1. The order of convergence is shown in Fig 1.

```

1 function [err,norm_err,xd] = HeatEqnEnergyCN2(N,nStep,tf,fOption,cOption)
2
3 xlim1 = 0;
4 xlim2 = 1;
5 tlim1 = 0;
6 tlim2 = tf;
7
8 dx = (xlim2-xlim1)/N;
9 dt = (tlim2-tlim1)/nStep;
10
11
12 x = (xlim1+dx/2:dx:xlim2-dx/2);
13 x = [xlim1-dx/2 x xlim2+dx/2];
14
15 NTot = length(x);
16
17 assert(length(x)==NTot,'Length mismatch');
18
19 t = (tlim1:dt:tlim2);
20
21 u = zeros(NTot,1);
22 nu = zeros(NTot,1);
23 for j=1:NTot
24     u(j) = getEx(x(j),tlim1,fOption);
25     nu(j) = getNu(x(j),cOption);
26 end
27
28 A = zeros(NTot);
29
30 Gamma = dt/dx^2;
31
32 for j=1:NTot
33     if j==1
34         A(j,j) = -1;
35         A(j,j+1) = 1;
36     elseif j==NTot
37         A(j,j) = 1;
38         A(j,j-1) = 1;
39     else
40         nmh = 0.5*(nu(j-1)+nu(j));
41         npf = 0.5*(nu(j)+nu(j+1));
42         A(j,j-1) = -(Gamma/2)*nmh;
43         A(j,j) = 1 + (Gamma/2)*(npf+nmh);

```

```

44     A(j,j+1) = -(Gamma/2)*nph;
45     end
46 end
47
48 b = zeros(NTot,1);
49 for i=2:length(t)
50     uold = u;
51     for j=1:NTot
52         if j==1
53             b(j) = dx*getUx(xlim1,t(i),fOption);
54         elseif j==NTot
55             b(j) = 2*getEx(xlim2,t(i),fOption);
56         else
57             nmh = 0.5*(nu(j-1)+nu(j));
58             nph = 0.5*(nu(j)+nu(j+1));
59
60             fnp = getF(x(j),t(i),fOption,cOption);
61             fn = getF(x(j),t(i-1),fOption,cOption);
62             fhat = 0.5*(fnp+fn);
63
64             b(j) = (Gamma/2)*nmh*uold(j-1)+...
65                 (1-Gamma*0.5*(nph+nmh))*uold(j)+...
66                 (Gamma/2)*nph*uold(j+1) + dt*fhat;
67         end
68     end
69     u = A\b;
70 end
71
72 uex = zeros(NTot,1);
73 for j=1:NTot
74     uex(j) = getEx(x(j),tlim2,fOption);
75 end
76
77 err = u(2:NTot-1)-uex(2:NTot-1);
78 norm_err = max(abs(err));
79
80
81 xd = x(2:NTot-1);
82 end
83
84 %% nu functions
85
86 function nu = getNu(x,cOption)
87 if cOption==1
88     m = 1;
89     nu0 = 0.5;
90     nu = m*x + nu0;
91 elseif cOption ==2
92     nu = (x+0.5)^2;
93 else
94     nu = 1;
95 end
96 end
97
98
99 function nux = getNux(x,cOption)
100 if cOption==1
101     m = 1;
102     nux = m*(x^0);
103 elseif cOption ==2
104     nux = 2*(x+0.5);

```

```

105 else
106     nux = 0;
107 end
108 end
109
110 %% u functions
111
112 function uex = getEx(x,t,fOption)
113 if fOption==1
114     uex = cos(pi*x/2)*sin(t);
115 else
116     uex = 0;
117 end
118 end
119
120 function ut = getUt(x,t,fOption)
121 if fOption==1
122     ut = cos(pi*x/2)*cos(t);
123 else
124     ut = 0;
125 end
126 end
127
128 function ux = getUx(x,t,fOption)
129 if fOption==1
130     ux = -(pi/2)*sin(pi*x/2)*sin(t);
131 else
132     ux = 0;
133 end
134 end
135
136 function uxx = getUxx(x,t,fOption)
137 if fOption==1
138     uxx = -(pi^2/4)*cos(pi*x/2)*sin(t);
139 else
140     uxx = 0;
141 end
142 end
143
144 function f = getF(x,t,fOption,cOption)
145 if fOption==1
146     ut = getUt(x,t,fOption);
147     ux = getUx(x,t,fOption);
148     uxx = getUxx(x,t,fOption);
149
150     nu = getNu(x,cOption);
151     nux = getNux(x,cOption);
152
153     f = ut - (nux*ux + nu*uxx);
154 else
155     end
156 end

```

Listing 1: Crank-Nicholson scheme - variable co-efficient Heat Equation

2. (25 pts.) Consider a heat conduction problem in an annular section

$$u_t = \nu \left[\frac{1}{r} (ru_r)_r + \frac{1}{r^2} u_{\theta\theta} \right], \quad 1 < r < 2, \quad 0 < \theta < \frac{\pi}{2}, \quad t > 0$$

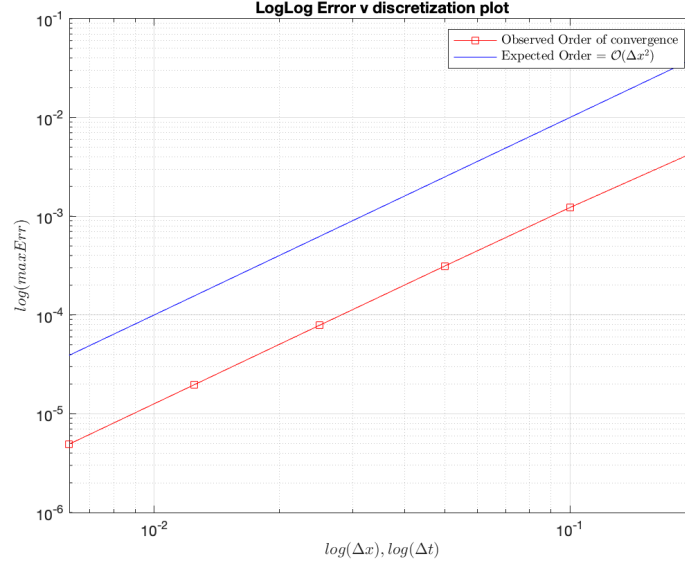


Figure 1: Order of Convergence plot

with initial condition $u(r, \theta, 0) = u_0(r, \theta)$, and boundary conditions

$$\begin{aligned} u_r(1, \theta, t) &= \alpha_1(\theta, t), & u_r(2, \theta, t) &= \alpha_2(\theta, t) \\ u(r, 0, t) &= \beta_1(r, t) & u(r, \frac{\pi}{2}, t) &= \beta_2(r, t). \end{aligned}$$

- (a) Write a second-order accurate code to solve this problem using centered differencing and Crank-Nicolson time integration.

$$\begin{aligned} & -\frac{r_1}{2r_{j,k}} r_{j-\frac{1}{2},k} v_{j-1,k}^{n+1} + \left(1 + \frac{r_1}{2r_{j,k}} (r_{j+\frac{1}{2},k} + r_{j-\frac{1}{2},k}) + \frac{r_2}{r_{j,k}^2} \right) v_{j,k}^{n+1} - \frac{r_1}{2r_{j,k}} r_{j+\frac{1}{2},k} v_{j+1,k}^{n+1} \\ & - \frac{r_2}{2r_{j,k}^2} v_{j,k-1}^{n+1} - \frac{r_2}{2r_{j,k}^2} v_{j,k+1}^{n+1} = \frac{r_1}{2r_{j,k}} r_{j-\frac{1}{2},k} v_{j-1,k}^n + \\ & \left(1 - \frac{r_1}{2r_{j,k}} (r_{j+\frac{1}{2},k} + r_{j-\frac{1}{2},k}) - \frac{r_2}{r_{j,k}^2} \right) v_{j,k}^n + \\ & \frac{r_1}{2r_{j,k}} r_{j+\frac{1}{2},k} v_{j+1,k}^n + \frac{r_2}{2r_{j,k}^2} v_{j,k-1}^n + \frac{r_2}{2r_{j,k}^2} v_{j,k+1}^n + \Delta t \hat{F} \\ & j = 0, 1, 2, \dots, N_r - 1, N_r \\ & k = 0, 1, 2, \dots, N_\theta - 1, N_\theta \end{aligned}$$

Boundary conditions are given by,

$$\begin{aligned} \frac{v_{1,k}^{n+1} - v_{-1,k}^{n+1}}{2\Delta r} &= \alpha_1(r = 1, \theta, (n+1)\Delta t) & \frac{v_{j,-1}^{n+1} + v_{j,1}^{n+1}}{2} &= \beta_1(r, \theta = 0, (n+1)\Delta t) \\ \frac{v_{N_r+1,k}^{n+1} - v_{N_r-1,k}^{n+1}}{2\Delta r} &= \alpha_2(r = 2, \theta, (n+1)\Delta t) & \frac{v_{j,N_\theta-1}^{n+1} + v_{j,N_\theta+1}^{n+1}}{2} &= \beta_2\left(r, \theta = \frac{\pi}{2}, (n+1)\Delta t\right) \end{aligned}$$

The code written for this is shown in Listing 2, Listing 3, and Listing 4.

```

1 function mesh = genMesh(rlim1,rlim2,slim1,slim2,Nr,Ns)
2
3 dr = (rlim2-rlim1)/Nr;
4 ds = (slim2-slim1)/Ns;
5
6 ng = 1;
7 NrTot = Nr+1+2*ng;
8 NsTot = Ns+1+2*ng;
9 jar = ng+1;
10 jbr = NrTot-ng;
11 jas = ng+1;
12 jbs = NsTot-ng;
13
14 r = (rlim1:dr:rlim2);
15 r = [rlim1-dr r rlim2+dr];
16 s = (slim1:ds:slim2);
17 s = [slim1-ds s slim2+ds];
18
19 GridFn = cell(NsTot,NrTot);
20 DOF = zeros(NsTot,NrTot);
21 IDX = cell(NsTot*NrTot,1);
22 dof = 1;
23 for k=1:NsTot
24     for j=1:NrTot
25         GridFn{k,j} = [r(j) s(k)];
26         DOF(k,j) = dof;
27         IDX{dof} = [k,j];
28         dof = dof + 1;
29     end
30 end
31
32 mesh.grid = GridFn;
33 mesh.DOF = DOF;
34 mesh.IDX = IDX;
35 mesh.NrTot = NrTot;
36 mesh.NsTot = NsTot;
37 mesh.ng = ng;
38 mesh.jar = jar;
39 mesh.jbr = jbr;
40 mesh.jas = jas;
41 mesh.jbs = jbs;
42 mesh.dr = dr;
43 mesh.ds = ds;
44 end

```

Listing 2: Mesh Generation code

```

1 function [max_err,ud,uex,err] = HeatEqn2DMapping(Nr,Ns,nStep,tf,iOption)
2
3 nu = 1;
4
5 rlim1 = 1;
6 rlim2 = 2;
7 slim1 = 0;
8 slim2 = pi/2;
9 tlim1 = 0;
10
11 mesh = genMesh(rlim1,rlim2,slim1,slim2,Nr,Ns);
12
13 NrTot = mesh.NrTot;

```

```

14 NsTot = mesh.NsTot;
15 ng    = mesh.ng;
16 jar    = mesh.jar;
17 jbr    = mesh.jbr;
18 jas    = mesh.jas;
19 jbs    = mesh.jbs;
20 dr     = mesh.dr;
21 ds     = mesh.ds;
22 dt     = (tf-tlim1)/nStep;
23
24 t      = (tlim1:dt:tf);
25
26 r1     = nu*dt/dr^2;
27 r2     = nu*dt/ds^2;
28
29 X      = zeros(NsTot,NrTot);
30 Y      = zeros(NsTot,NrTot);
31
32 for k=1:NsTot
33     for j=1:NrTot
34         loc = mesh.grid{k,j};
35         xloc = loc(1)*cos(loc(2));
36         yloc = loc(1)*sin(loc(2));
37         X(k,j) = xloc;
38         Y(k,j) = yloc;
39     end
40 end
41
42 u = zeros(NsTot,NrTot);
43 for k=1:NsTot
44     for j=1:NrTot
45         loc = mesh.grid{k,j};
46         rad = loc(1);
47         theta = loc(2);
48         u(k,j) = getEx(rad,theta,tlim1,iOption);
49     end
50 end
51
52 A = zeros(NsTot*NrTot);
53 b = zeros(NsTot*NrTot,1);
54 vnp = zeros(NsTot*NrTot,1);
55
56 for k=1:NsTot
57     for j=1:NrTot
58         row = mesh.DOF(k,j);
59         if (k==ng&&j==ng) || (k==1&&j==NrTot) || ...
60             (k==NsTot&&j==1) || (k==NsTot&&j==NrTot) % corner check
61             col = row;
62             A(row,col) = 1;
63         else
64             if k==ng % bottom boundary (Dirchlet)
65                 col = mesh.DOF(jas+1,j);
66                 A(row,row) = 1;
67                 A(row,col) = 1;
68             elseif j==ng % left boundary (Neumann)
69                 col = mesh.DOF(k,jar+1);
70                 A(row,row) = -1;
71                 A(row,col) = 1;
72             elseif j==NrTot % right boundary (Neumann)
73                 col = mesh.DOF(k,jbr-1);
74                 A(row,row) = 1;

```



```

75         A(row,col) = -1;
76         elseif k==NsTot % top boundary (Dirichlet)
77             col = mesh.DOF(jbs-1,j);
78             A(row,row) = 1;
79             A(row,col) = 1;
80         else
81             rjmk = mesh.grid{k,j-1}(1);
82             rjk = mesh.grid{k,j}(1);
83             rjpk = mesh.grid{k,j+1}(1);
84
85             rjmh = 0.5*(rjk+rjmk);
86             rjph = 0.5*(rjk+rjpk);
87
88             colm = mesh.DOF(k,j-1);
89             colp = mesh.DOF(k,j+1);
90             rowm = mesh.DOF(k-1,j);
91             rowp = mesh.DOF(k+1,j);
92
93             A(row,colm) = -(r1/(2*rjk))*rjmh;
94             A(row,row) = (1+ (r1/(2*rjk))*(rjmh+rjph) + (r2/rjk^2));
95             A(row,colp) = -(r1/(2*rjk))*rjph;
96             A(row,rowm) = -(r2/(2*rjk^2));
97             A(row,rowp) = -(r2/(2*rjk^2));
98         end
99     end
100 end
101
102 for i=2:length(t)
103     uold = u;
104     for k=1:NsTot
105         for j=1:NrTot
106             row = mesh.DOF(k,j);
107             rjk = mesh.grid{k,j}(1);
108             tjk = mesh.grid{k,j}(2);
109             if (k==ng&&j==ng) || (k==ng&&j==NrTot) || ...
110                 (k==NsTot&&j==ng) || (k==NsTot&&j==NrTot) % corners
111                 b(row) = getEx(rjk,tjk,t(i),iOption);
112             else
113                 if k==ng % bottom Boundary condition
114                     b(row) = 2*getEx(rjk,slim1,t(i),iOption);
115                 % b(row) = 2*getBC1(rjk,t(i),iOption);
116                 elseif j==ng % left Boundary condition
117                     b(row) = 2*dr*getUr(rlim1,tjk,t(i),iOption);
118                 elseif j==NrTot % right Boundary condition
119                     b(row) = 2*dr*getUr(rlim2,tjk,t(i),iOption);
120                 elseif k==NsTot % top Boundary condition
121                     b(row) = 2*getEx(rjk,slim2,t(i),iOption);
122                 % b(row) = 2*getBC2(rjk,t(i),iOption);
123                 else
124                     rjmk = mesh.grid{k,j-1}(1);
125                     rjpk = mesh.grid{k,j+1}(1);
126
127                     rjmh = 0.5*(rjk+rjmk);
128                     rjph = 0.5*(rjk+rjpk);
129
130                     fnp = getF(nu,rjk,tjk,t(i),iOption);
131                     fn = getF(nu,rjk,tjk,t(i-1),iOption);
132                     Fhat = 0.5*(fnp+fn);
133
134                     b(row) = (0.5*r1/rjk)*rjmh*uold(k,j-1) + ...

```

```

136                                     (1- (0.5*r1/rjk)*(rjmh+rjph) - (r2/rjk^2)
    )*uold(k,j) + ...
137                                     (0.5*r1/rjk)*rjph*uold(k,j+1) + ...
138                                     (r2/(2*rjk^2))*uold(k-1,j) + ...
139                                     (r2/(2*rjk^2))*uold(k+1,j) + dt*Fhat;
140                                     end
141                                     end
142                                     end
143                                     end
144                                     vnp = A\b;
145                                     u = reconstructSol(mesh.IDX,vnp);
146 end
147
148 t = '$t_f=';
149 t = strcat(t,num2str(tf));
150 t = strcat(t,'s$');
151 %
152 % figure
153 % surf(X(jas:jbs,jar:jbr),Y(jas:jbs,jar:jbr),u(jas:jbs,jar:jbr));
154 % colorbar
155 % xlabel('$x$', 'Interpreter','latex');
156 % ylabel('$y$', 'Interpreter','latex');
157 % zlabel('$u(r,\theta,t)$', 'Interpreter','latex');
158 % title('$Numerical Solution, \ u(r,\theta,t)$',t,'Interpreter','latex');
159 %
160 % figure
161 % contourf(X(jas:jbs,jar:jbr),Y(jas:jbs,jar:jbr),u(jas:jbs,jar:jbr));
162 % colorbar
163 % xlabel('$x$', 'Interpreter','latex');
164 % ylabel('$y$', 'Interpreter','latex');
165 % zlabel('$u(r,\theta,t)$', 'Interpreter','latex');
166 % title('Numerical Solution, $\ u(r,\theta,t)$',t,'Interpreter','latex');
167
168 ud = u(jas:jbs,jar:jbr);
169
170 if iOption==1
171     uex = zeros(NsTot,NrTot);
172     for k=1:NsTot
173         for j=1:NrTot
174             loc = mesh.grid{k,j};
175             rad = loc(1);
176             theta = loc(2);
177             uex(k,j)= getEx(rad,theta,t,iOption);
178         end
179     end
180     err = u-uex;
181     max_err = max(max(abs(err(jas:jbs,jar:jbr)))));
182
183 %     figure
184 %     surf(err)
185
186 else
187     max_err = 0;
188     uex = ud;
189 end
190
191 end
192
193 %% functions
194
195 function uex = getEx(r,theta,t,iOption)

```

```

196 if iOption==1
197     uex = cos(pi*r)*sin(2*theta)*sin(t);
198 else
199     uex = 0;
200 end
201 end
202
203 function ubc1 = getBC1(r,t,iOption)
204 if iOption==1
205     ubc1 = cos(pi*r)*sin(t)*0;
206 else
207     ubc1 = 0;
208 end
209 end
210
211 function ubc2 = getBC2(r,t,iOption)
212 if iOption ==1
213     ubc2 = cos(pi*r)*sin(t)*0;
214 else
215     ubc2 = (r-1)^2*(r-2)^2*(t^0);
216 end
217 end
218
219 function ur = getUr(r,theta,t,iOption)
220 if iOption==1
221     ur = -pi*sin(pi*r)*sin(2*theta)*sin(t);
222 else
223     ur = 0;
224 end
225 end
226
227 function ut = getUt(r,theta,t,iOption)
228 if iOption==1
229     ut = cos(pi*r)*sin(2*theta)*cos(t);
230 else
231     ut = 0;
232 end
233 end
234
235 function Vrr = getVrr(r,theta,t,iOption)
236 if iOption==1
237     Vrr = -pi*sin(2*theta)*sin(t)*((sin(pi*r)/r)+pi*cos(pi*r));
238 else
239     Vrr = 0;
240 end
241 end
242
243 function Vtt = getVtt(r,theta,t,iOption)
244 if iOption==1
245     Vtt = -(4/r^2)*cos(pi*r)*sin(2*theta)*sin(t);
246 else
247     Vtt = 0;
248 end
249 end
250
251 function f = getF(nu,r,theta,t,iOption)
252 if iOption==1
253     ut = getUt(r,theta,t,iOption);
254     Vrr = getVrr(r,theta,t,iOption);
255     Vtt = getVtt(r,theta,t,iOption);
256

```

```

257     f = ut - nu*(Vrr+Vtt);
258 else
259     f = 0;
260 end
261 end

```

Listing 3: Heat Equation 2D under domain mapping

```

1 function u = reconstructSol(IDX,v)
2
3 for i=1:length(v)
4     idx = IDX{i};
5     k   = idx(1);
6     j   = idx(2);
7     u(k,j) = v(i);
8 end
9
10 end

```

Listing 4: Reconstructing solution code

- (b) Verify the accuracy of your code using manufactured solutions.

$$\begin{aligned}
 u(r, \theta, t) &= \cos(\pi r) \sin 2\theta \sin t, \quad r \in (1, 2), \quad \theta \in \left(0, \frac{\pi}{2}\right) \\
 u_r(r, \theta, t) &= -\pi \sin(\pi r) \sin 2\theta \sin t \\
 u_t(r, \theta, t) &= \cos(\pi r) \sin 2\theta \cos t \\
 \frac{1}{r} (ru_r)_r &= -\pi \sin 2\theta \sin t \left(\frac{\sin \pi r}{r} + \pi \cos \pi r \right) \\
 \frac{1}{r^2} (u_{\theta\theta}) &= -\frac{4}{r^2} \cos(\pi r) \sin 2\theta \sin t \\
 f(r, \theta, t) &= u_t - \nu \left(\frac{1}{r} (ru_r)_r + \frac{1}{r^2} u_{\theta\theta} \right)
 \end{aligned}$$

Boundary conditions are,

$$\begin{aligned}
 u_r(r=1) &= 0, & u(\theta=0) &= 0 \\
 u_r(r=2) &= 0, & u\left(\theta=\frac{\pi}{2}\right) &= 0
 \end{aligned}$$

The order of convergence is shown in Fig 2.

- (c) Now set

$$\begin{aligned}
 u_0(r, \theta) &= 0 \\
 \alpha_1(\theta, t) &= 0 \\
 \alpha_2(\theta, t) &= 0 \\
 \beta_1(r, t) &= 0 \\
 \beta_2(r, t) &= (r-1)^2(r-2)^2.
 \end{aligned}$$

Using $\nu = 1$ and 40 grid lines in both the radial and angular coordinate directions, compute numerical solutions to this problem at $t = 0, .1, .5, 1.5$. Create surface plots of the solution for each time.

The solutions are plotted in Fig 3 and Fig 4.

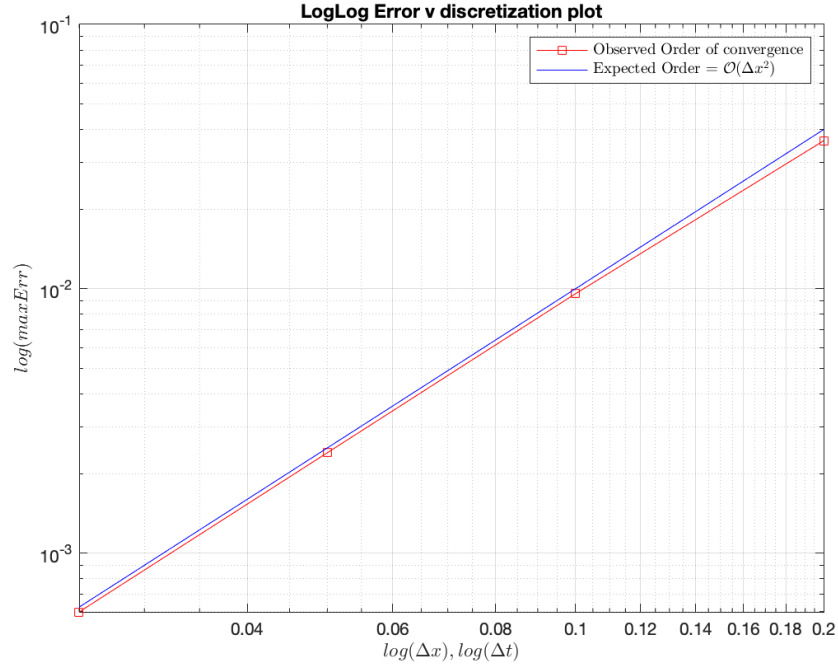


Figure 2: Order of Convergence

- (d) Create a single line plot with four curves showing the solutions from part (c) along the inner radius ($r = 1$), as a function of θ for all four times $t = 0, .1, .5, 1.5$. The 1-D slice of the function is shown in Fig 5 and Fig 6.

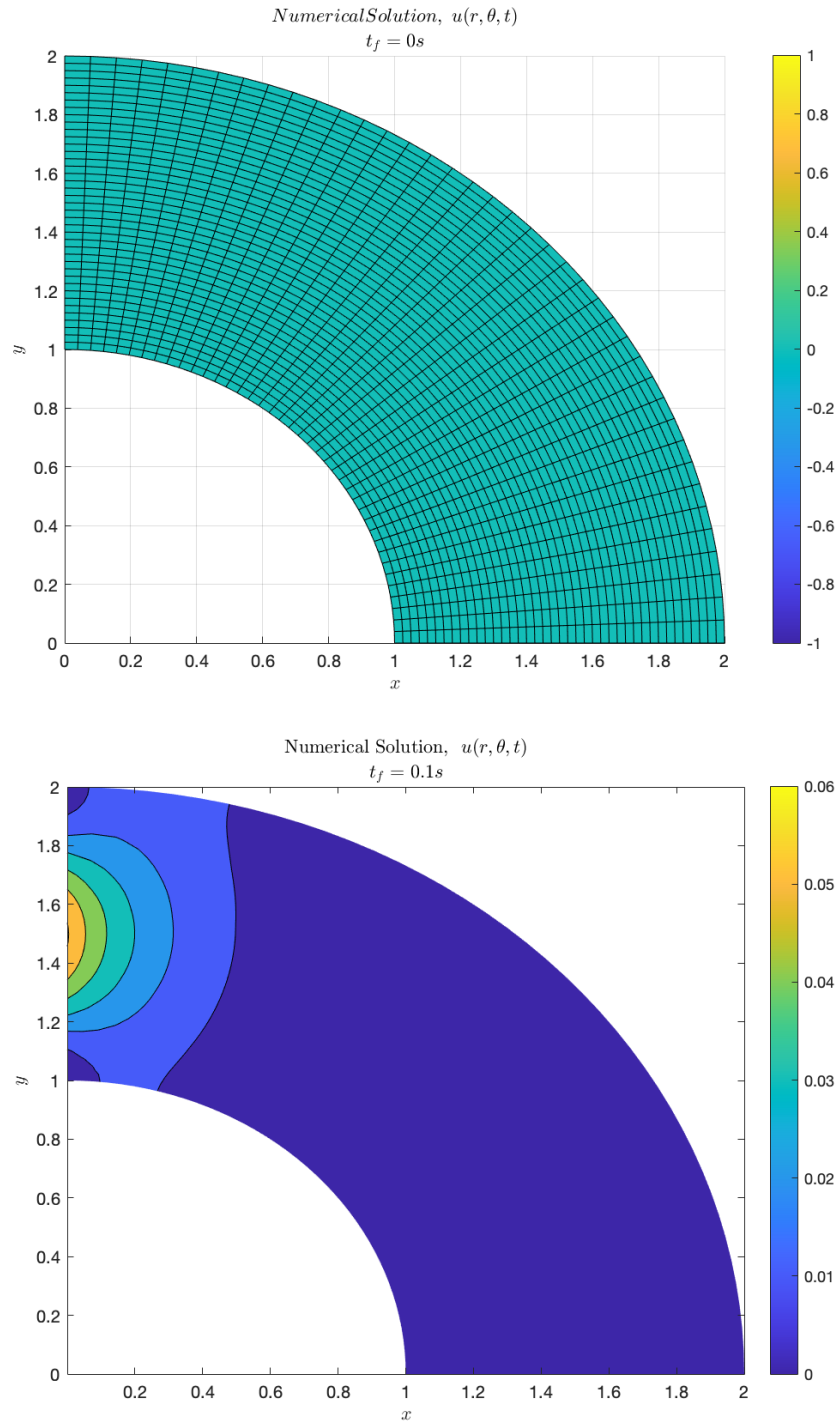


Figure 3: Solution at different final times

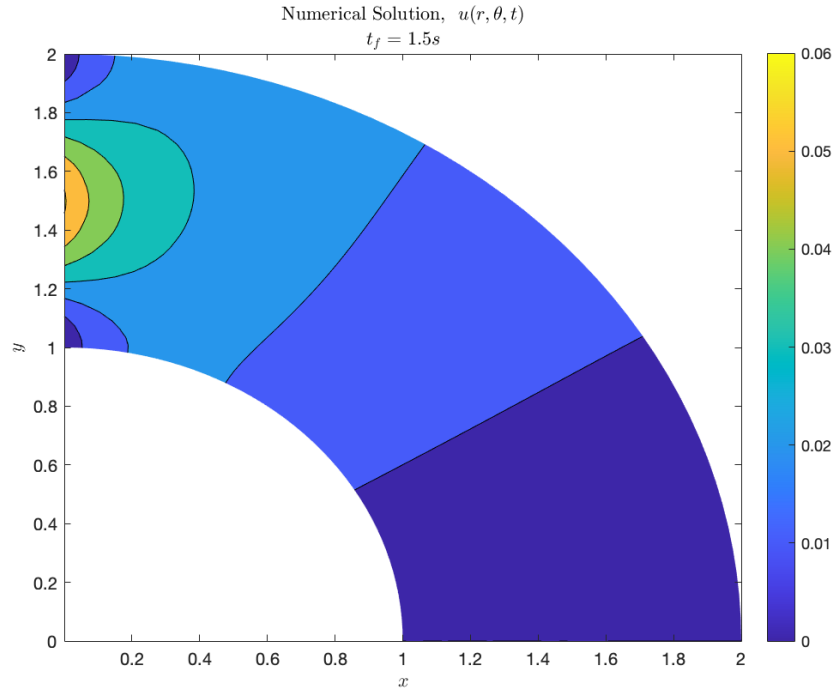
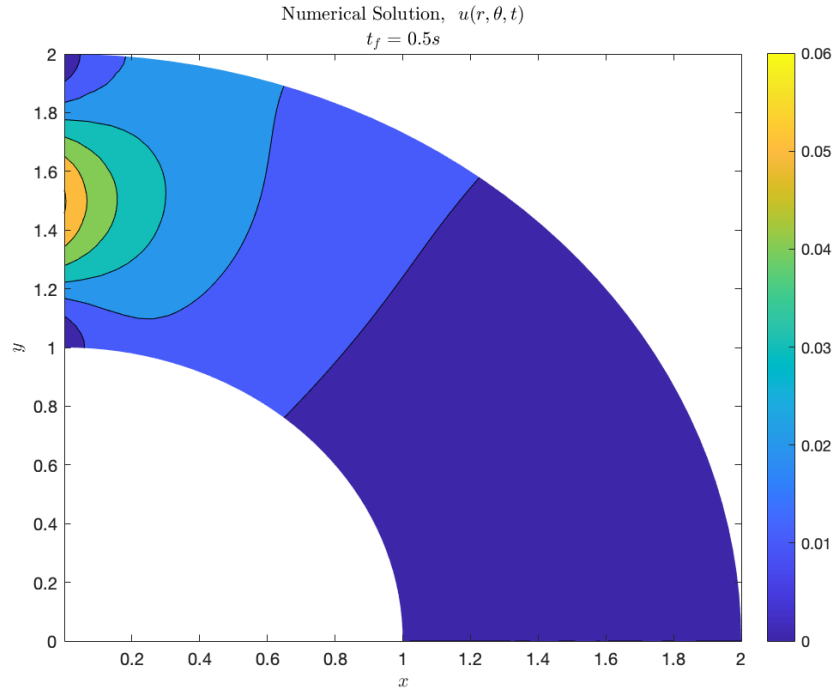


Figure 4: Solution at different final times

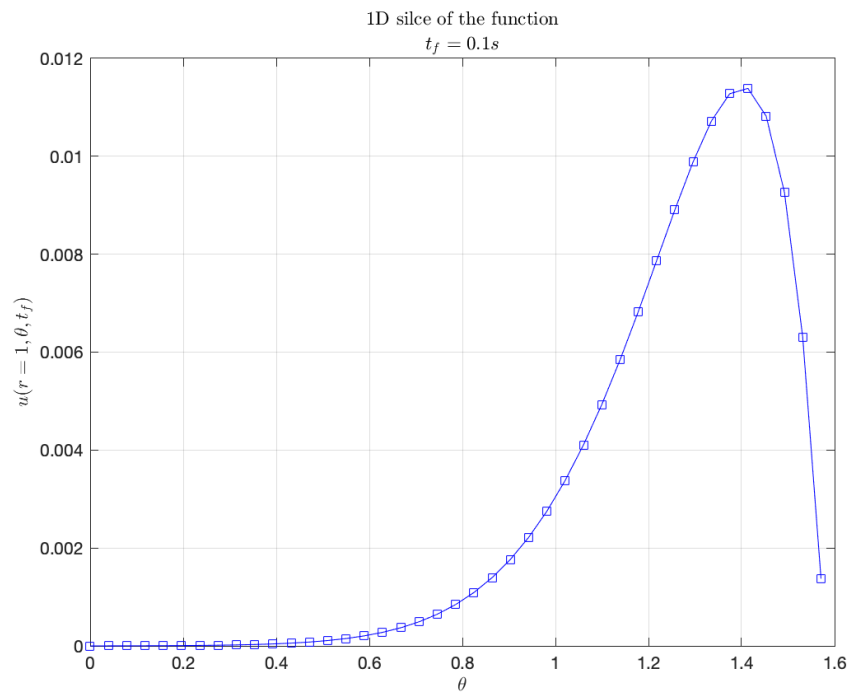
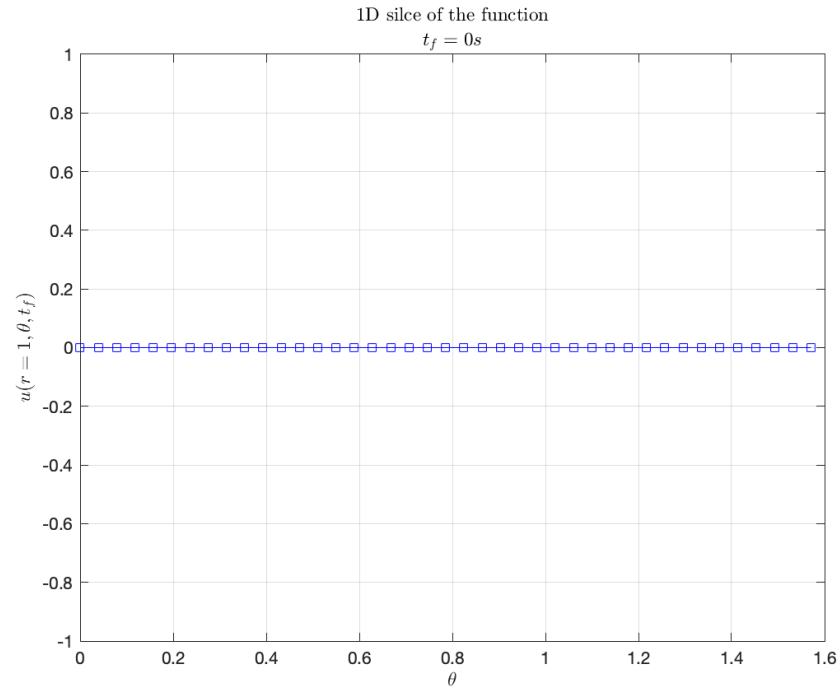


Figure 5: 1D slice of solution at final time - 1

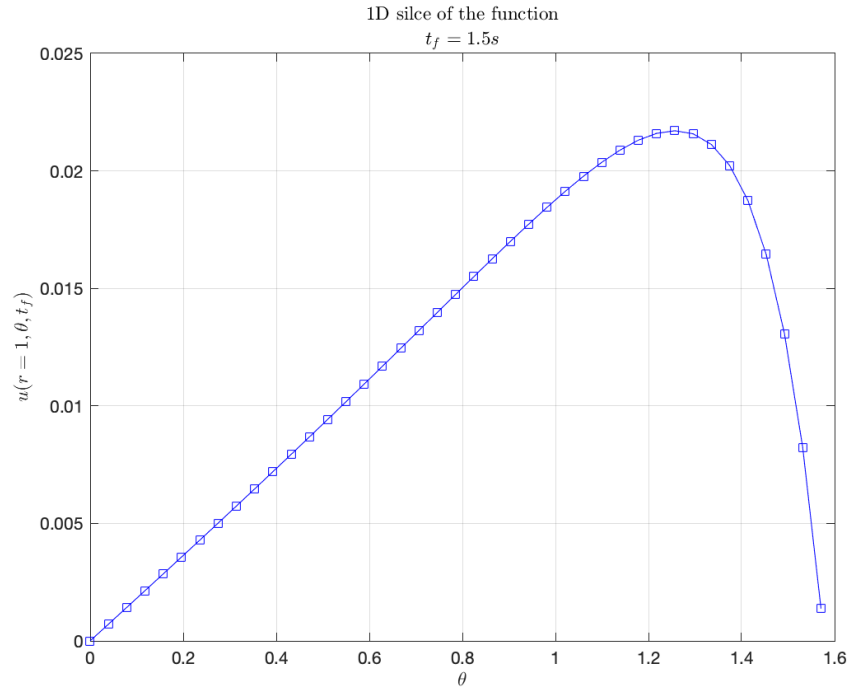
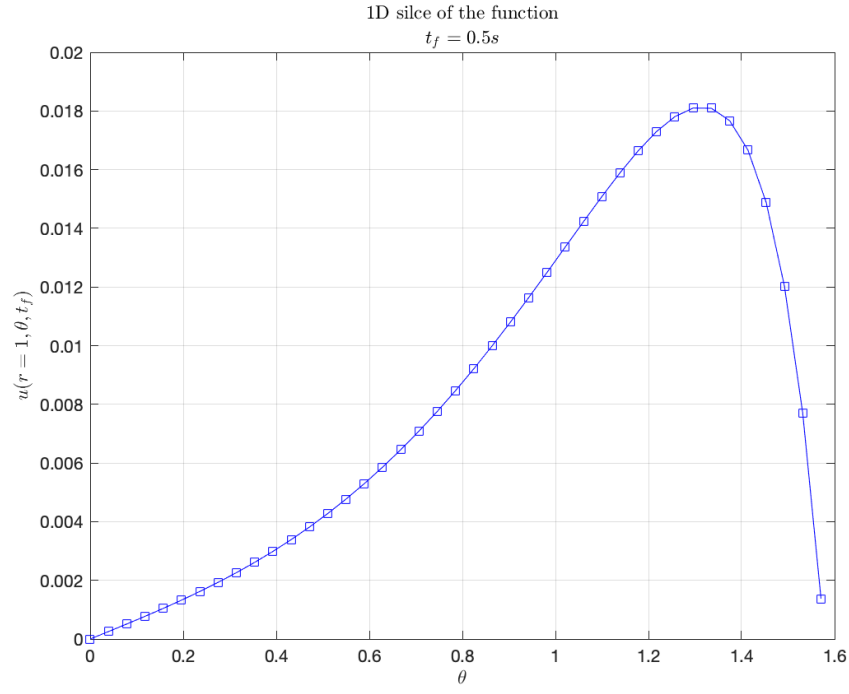


Figure 6: 1D slice of solution at final time - 2