MATH-6840
Vignesh Ramakrishnan
Due: Monday April 25, 2022

Problem Set 10

1. (20 pts. extra credit) Consider the inviscid Burgers' equation $u_t + \left(\frac{1}{2}u^2\right)_x = 0$ with initial conditions

$$u(x,0) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$$

(a) Show that the following is a weak solution for $\alpha < 1$

$$u(x,t) = \begin{cases} -1 & \text{if } x < -(1-\alpha)t/2 \\ \alpha & \text{if } -(1-\alpha)t/2 \leq x < 0 \\ -\alpha & \text{if } 0 \leq x < (1-\alpha)t/2 \\ 1 & \text{if } x \geq (1-\alpha)t/2. \end{cases}$$

For $\alpha < 1$, the value $1 - \alpha > 0$. Hence, let us consider each interval separately.

- **Case 1:** Consider the interval $x < -(1-\alpha)t/2$,

$$\partial_t u = 0$$
$$\partial_x u = 0$$

Hence, it solves the Burger's equation in the Differential form. Now, if we consider the Integral conservational form,

$$\frac{d}{dt}\int_{-\infty}^{(1-\alpha)t/2} u\ dx + \left[\frac{1}{2}u^2\right]_{-\infty}^{-(1-\alpha)t/2} = 0$$

Since $u = -1$ in this interval, the flux jump goes away and u is conserved.

- **Case 2:** Consider the interval $-(1-\alpha)t/2 \leq x < 0$,

$$\partial_t u = 0$$
$$\partial_x u = 0$$

Hence, it solves the Burger's equation in the Differential form. Now, if we consider the Integral conservational form,

$$\frac{d}{dt}\int_{-(1-\alpha)t/2}^{0} u\ dx + \left[\frac{1}{2}u^2\right]_{-(1-\alpha)t/2}^{0} = 0$$

Since $u = \alpha$ in this interval, the flux jump goes away and u is conserved.

- **Case 3:** Consider the interval $0 \leq x < (1-\alpha)t/2$,

$$\partial_t u = 0$$
$$\partial_x u = 0$$

Hence, it solves the Burger's equation in the Differential form. Now, if we consider the Integral conservational form,

$$\frac{d}{dt}\int_0^{(1-\alpha)t/2} u \; dx + \left[\frac{1}{2}u^2\right]_0^{(1-\alpha)t/2} = 0$$

Since $u = -\alpha$ in this interval, the flux jump goes away and u is conserved.

- **Case 4:** Consider the interval $x > (1-\alpha)t/2$,

$$\partial_t u = 0$$
$$\partial_x u = 0$$

Hence, it solves the Burger's equation in the Differential form. Now, if we consider the Integral conservational form,

$$\frac{d}{dt}\int_{(1-\alpha)t/2}^{\infty} u \; dx + \left[\frac{1}{2}u^2\right]_{(1-\alpha)t/2}^{\infty} = 0$$

Since $u = 1$ in this interval, the flux jump goes away and u is conserved.

Therefore, this is a weak solution to the Burger's equation.

(b) Plot (or sketch) the solution from part (a) for $\alpha = .5$, and plot (or sketch) the characteristics in the $x - t$ plane. Is this solution an entropy satisfying solution (give a reason as to why or why not)?

The weak solution given in the question is plotted in Fig 2 and the characteristics are plotted in Fig 1. This solution is NOT an entropy satisfying because, there are propagating discontinuities in the solution. A solution where the discontinuities vanish as the solution propagates is a physical solution.

(c) Construct the entropy satisfying weak solution to this problem.

I did not know how to construct an entropy stable solution but this is my attempt. From the characteristics we can observe that, there is no shock wave in this problem but more of a refraction fan. So, I did some research and the only physical weak solution to this problem is of the same form as the weak solution presented, except the speeds of propagation used are the respective left velocity and right velocity of the solution itself.

$$u(x,t) = \begin{cases} u_L, & x < u_L t, \\ x/t, & u_L t \leq x \leq u_R t \\ u_R, & x > u_R t \end{cases}$$

$$u(x,t) = \begin{cases} -1, & x < -t, \\ x/t, & -t \leq x \leq t \\ 1, & x > t \end{cases}$$

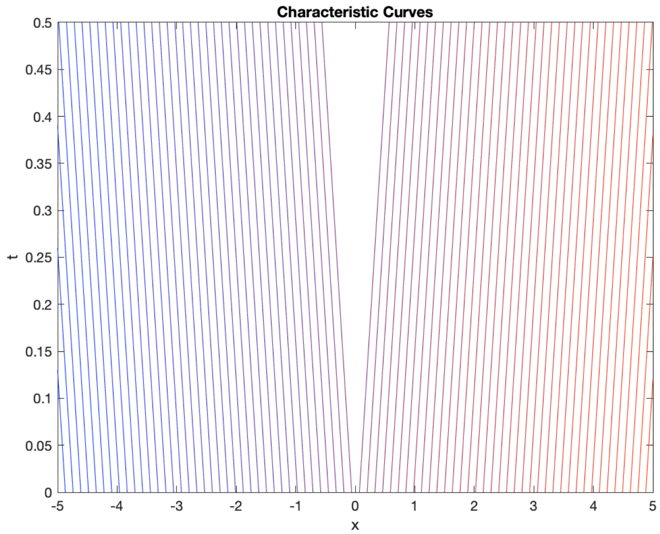

Figure 1: Characteristics of the Burger's Equation

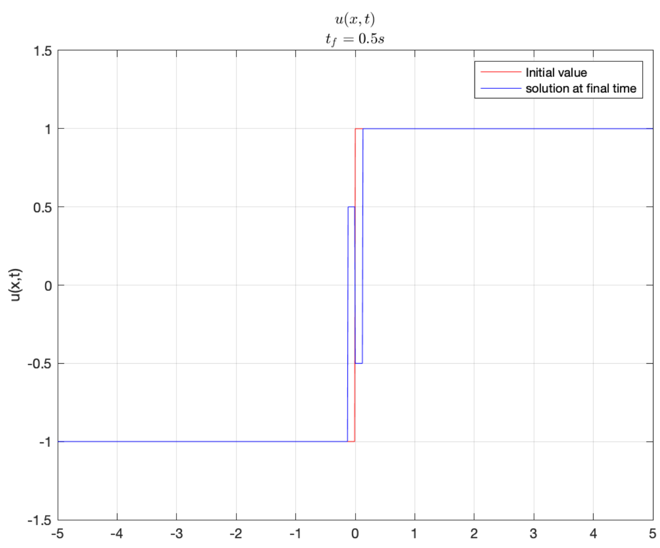

Figure 2: Weak Solution at $t = 0$ s and at $t = 0.5$ s

(d) Create a conservative upwind code to run this case and present computed solution at $t = 0.5$. Compare the numerical solution to the exact solution from part (c) above.

I tried the upwind flux scheme and it was not really converging. Hence, I did some research on my own and the only method I could understand was the Godunov scheme described in the Finite Volume literature. I have tried to attempt the Godunov scheme in my code for multiple initial conditions and it was stable for many initial conditions, even for the one where a shock wave is encountered. The scheme I have written is as follows.

The domain I have considered is $x_1 = -5$ to $x_2 = 5$.

$$\Delta x = (x_2 - x_1)/N;$$
$$x_j = x_1 + (j + \frac{1}{2})\Delta x$$

The solution variables are declared at x_j which is the center of each element. Now, the standard upwind scheme is written as,

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(\hat{F}^n_{j+\frac{1}{2}}\left(u_j^n, u_{j+1}^n\right) - \hat{F}^n_{j-\frac{1}{2}}\left(u_{j+1}^n, u_j^n\right)\right)$$
$$j = 1, 2, 3, ..., N-1$$

$$\hat{F}^n_{j+\frac{1}{2}}\left(u_L, u_R\right) = \begin{cases} u_L, & u_L \geq 0 \text{ and } u_R \geq 0 \\ u_R, & u_L \leq 0 \text{ and } u_R \leq 0 \\ u^*, & u_L \geq 0 \text{ and } u_R \leq 0 \\ 0, & u_L \leq 0 \text{ and } u_R \geq 0 \end{cases}$$

Here,

$$u^* = \begin{cases} u_L, & \frac{f(u_L)-f(u_R)}{u_L-u_R} \geq 0 \\ u_R, & \frac{f(u_L)-f(u_R)}{u_L-u_R} \leq 0 \end{cases}$$

Neumann like Boundary conditions are considered at both ends and its given by,

$$u(0) = u(1)$$
$$u(N) = u(N-1)$$

The code is attached in the Listing 1.

```
function burgersEqn(N,tf,CFL,iOption,fOption)
% fOption
% 1 = Godunov, entropy stable, 2 = Central flux, 3 = upwind
% Trial runs
% burgersEqn(200,5,0.1,1,1) - step function, Godunov
% burgersEqn(200,5,0.1,2,1) - sine wave, Godunov
% burgersEqn(200,5,0.1,3,1) - gaussian wave, Godunov
% burgersEqn(200,5,0.1,1,2) - step function, average


% set up domain
xlim1 = -5;
xlim2 = 5;
```

```
tlim1 = 0;
tlim2 = tf;

% domain discretization
ng   = 0;
NTot = N+1+2*ng;
ja   = ng+1;
jb   = NTot-ng;

dx   = (xlim2-xlim1)/N;

% spatial points
x = (xlim1:dx:xlim2);

% set initial condition
u  = zeros(N,1);
xh = zeros(N,1);
for j=1:N
    xh(j) = 0.5*(x(j)+x(j+1));
    if iOption==1
        if xh(j)<0
            u(j) = -1;
        elseif xh(j)==0
            u(j) = 0;
        else
            u(j) = 1;
        end
    elseif iOption==2
        u(j) = sin(xh(j));
    elseif iOption==3
        u(j) = exp(-10*(xh(j)^2));
    end
end

% time step size
t  = tlim1;
dt = CFL*dx/(max(abs(u)));

while t < tlim2
    uold = u;
    for j=2:N-1
        u(j) = uold(j)-(dt/dx)*( F(u(j),u(j+1),fOption)-F(u(j-1),u(j),fOption) );
    end
    % set boundary conditions
    u(1) = u(2);
    u(N) = u(N-1);

    plot(xh,u);
    drawnow
    pause(0.01)

    dt = CFL*dx/(max(abs(u)));
    t  = t+dt;
end

if iOption==1
    uex = zeros(length(N),1);
    for j=1:N
```

```matlab
        uex(j) = getEx(xh(j),t-dt);
    end
    figure
    plot(xh,u,'rs-');
    hold on;
    grid on;
    plot(xh,uex,'k');
    xlabel('x');
    ylabel('$u(x,t)$','Interpreter','latex');
    ylim([-1.25 1.25]);
    legend('Numerical Solution (Godunov scheme)','Physical Entropy stable
     solution');
    title('Numerical v Entropy stable solution','$t_f = 0.5s$','
    Interpreter','latex');
end

end

%% functions

function f = F(uL,uR,fOption)
if fOption==1 % Godunov
    if uL >=0 && uR >=0
        us = uL;
    elseif uL <=0 && uR <=0
        us = uR;
    elseif uL >=0 && uR <=0
        fj = 0.5*(uL^2-uR^2);
        uj = uL-uR;
        if (fj/uj>0)
            us = uL;
        else
            us = uR;
        end
    elseif uL <0 && uR >0
        us = 0;
    end
    f = 0.5*us^2;
elseif fOption==2 % average flux
    f = 0.25*(uL^2+uR^2);
elseif fOption==3 % upwind flux
    if uL>=0
        f = 0.5*uL^2;
    else
        f = 0.5*uR^2;
    end
end
end


function uex = getEx(x,t)
if x<-t
    uex = -1;
elseif x>=-t && x<=t
    if t<=1e-14
        uex = 0;
    else
        uex = x/t;
    end
```

```
elseif x>t
    uex = 1;
end
end
```

Listing 1: Burger's Equaiton - Upwind Godunov Scheme

. The comparison of the entropy stable physical solution and the numerical solution is shown in Fig 3.

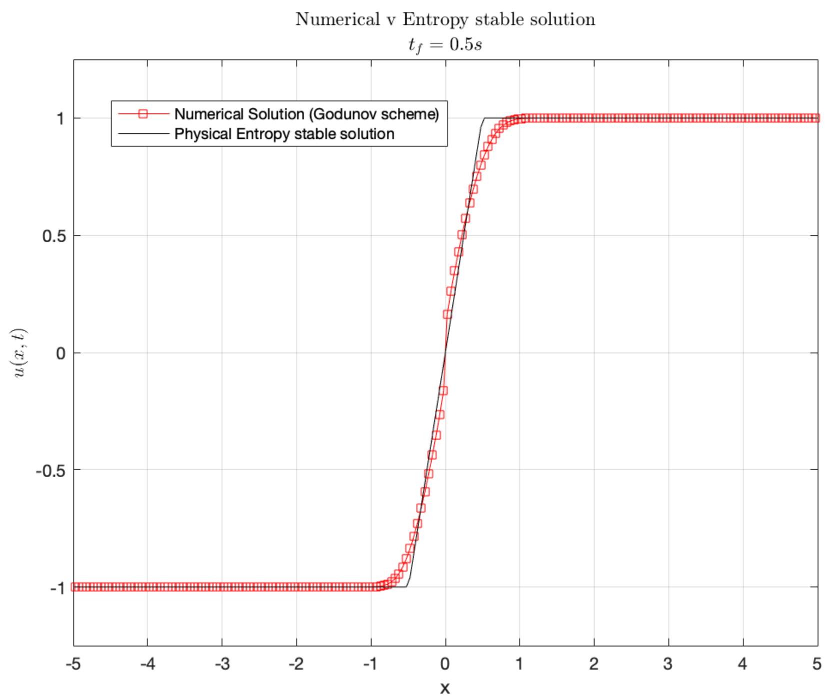

Figure 3: Numerical Solution v Entropy Stable weak Solution

2. (20 pts. extra credit) Now let us revisit the annular section problem from PS7, and consider the steady state of a heat conduction problem in an annular section

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

with boundary conditions

$$\begin{aligned} u_r(1,\theta) &= 0, \qquad u_r(2,\theta) = 0 \\ u(r,0) &= 0 \qquad u(r,\frac{\pi}{2}) = (r-1)^2(r-2)^2. \end{aligned}$$

(a) Starting from your code from PS7 (or the one from SS7), write a second-order accurate code to solve this problem using centered differencing. Hint, as before you may find it useful to use manufactured solutions to verify the accuracy of your code.

$$\begin{aligned} r_j &= r_1 + j\Delta r \qquad & j &= -1, 0, 1, \ldots\ldots N_r + 1 \\ \theta_k &= \theta_1 + k\Delta\theta \qquad & k &= -1, 0, 1, \ldots\ldots N_\theta + 1 \end{aligned}$$

$$\frac{1}{r_{j,k}}\frac{r_{j+\frac{1}{2},k}v_{j+1,k}-\left(r_{j+\frac{1}{2},k}+r_{j-\frac{1}{2},k}\right)v_{j,k}+r_{j-\frac{1}{2},k}v_{j-1,k}}{\Delta r^2}+ \frac{1}{r_{j,k}^2}\frac{v_{j,k+1}-2v_{j,k}+v_{j,k-1}}{\Delta\theta^2}=0$$

The Boundary conditions are writen as,

$$\frac{v_{2,k}-v_{-1,k}}{2\Delta r}=0, \qquad \frac{v_{N_r+1,k}-v_{N_r-1,k}}{2\Delta r}=0$$
$$\frac{v_{j,-1}+v_{j,2}}{2}=0, \qquad \frac{v_{j,N_\theta-1}+v_{j,N_\theta+1}}{2}=(r_{j,\theta_2}-1)^2(r_{j,\theta_2}-2)^2$$

This is written in a matrix form as follows,

$$\underline{\underline{\mathbf{A}}}\ \underline{v}=\underline{0}$$

This discretization is tested against the manufactured solution,

$$u(r,\theta)=(r-1)^2(r-2)^2\sin(\theta)$$

where, the PDE becomes,

$$\left(\frac{1}{r}\left(ru_r\right)_r+\frac{1}{r^2}u_{\theta\theta}\right)=f(r,\theta)$$

The error convergence can be seen in Fig 4. The code is attached in Listing 2.

```
function [Ud,Xd,Yd,A,b,norm_err] = HeatEqn2DSteady(Nr,Ns,iOption,sOption)

nu = 1;

rlim1 = 1;
rlim2 = 2;
slim1 = 0;
slim2 = pi/2;

mesh = genMesh(rlim1,rlim2,slim1,slim2,Nr,Ns);

NrTot  = mesh.NrTot;
NsTot  = mesh.NsTot;
ng     = mesh.ng;
jar    = mesh.jar;
jbr    = mesh.jbr;
jas    = mesh.jas;
jbs    = mesh.jbs;
dr     = mesh.dr;
ds     = mesh.ds;

r1 = nu/dr^2;
r2 = nu/ds^2;

X      = zeros(NsTot,NrTot);
Y      = zeros(NsTot,NrTot);

for k=1:NsTot
    for j=1:NrTot
```

```
        loc  = mesh.grid{k,j};
        xloc = loc(1)*cos(loc(2));
        yloc = loc(1)*sin(loc(2));
        X(k,j)= xloc;
        Y(k,j)= yloc;
    end
end

uini = zeros(NsTot*NrTot,1);

A    = zeros(NsTot*NrTot);
b    = zeros(NsTot*NrTot,1);
v    = zeros(NsTot*NrTot,1);

i = 1;
for k=1:NsTot
    for j=1:NrTot
        row = mesh.DOF(k,j);
        rjk  = mesh.grid{k,j}(1);
        sjk  = mesh.grid{k,j}(2);
        if (k==ng&&j==ng)||(k==1&&j==NrTot)||...
                (k==NsTot&&j==1)||(k==NsTot&&j==NrTot) % corner check
             col = row;
             A(row,col) = 1;
             b(i,1)     = getEx(rjk,sjk,iOption);
        else
            if k==ng        % bottom boundary (Dirchlet) - BC1
                col = mesh.DOF(jas+1,j);
                A(row,row) = 1;
                A(row,col) = 1;
                b(i,1)     = 2*getBC1(rjk,slim1,iOption);
            elseif j==ng    % left boundary (Neumann) - BC2
                col = mesh.DOF(k,jar+1);
                A(row,row) = -1;
                A(row,col) = 1;
                b(i,1)     = 2*dr*getBC2(rlim1,sjk,iOption);
            elseif j==NrTot % right boundary (Neumann) - BC3
                col = mesh.DOF(k,jbr-1);
                A(row,row) = 1;
                A(row,col) = -1;
                b(i,1)     = 2*dr*getBC3(rlim2,sjk,iOption);
            elseif k==NsTot % top boundary (Dirchlet) - BC4
                col = mesh.DOF(jbs-1,j);
                A(row,row) = 1;
                A(row,col) = 1;
                b(i,1)     = 2*getBC4(rjk,slim2,iOption);
            else
                rjmk = mesh.grid{k,j-1}(1);
                %rjk  = mesh.grid{k,j}(1);
                rjpk = mesh.grid{k,j+1}(1);

                rjmh = 0.5*(rjk+rjmk);
                rjph = 0.5*(rjk+rjpk);

                colm = mesh.DOF(k,j-1);
                colp = mesh.DOF(k,j+1);
                rowm = mesh.DOF(k-1,j);
                rowp = mesh.DOF(k+1,j);

```

```
                A(row,colm) = (r1/rjk)*rjmh;
                A(row,row)  = (-r1/rjk)*(rjmh+rjph) - (2*r2/rjk^2);
                A(row,colp) = (r1/rjk)*rjph;
                A(row,rowm) = (r2/rjk^2);
                A(row,rowp) = (r2/rjk^2);

                b(i,1)      = getF(rjk,sjk,iOption);
            end
        end
        i = i+1;
    end
end

v = IterativeSolver(A,b,uini,sOption);
U = reconstructSol(mesh.IDX,v);

Xd = X(jas:jbs,jar:jbr);
Yd = Y(jas:jbs,jar:jbr);
Ud = U(jas:jbs,jar:jbr);
if iOption==1
    uex = zeros(NsTot,NrTot);
    for k=1:NsTot
        for j=1:NrTot
            loc   = mesh.grid{k,j};
            rad   = loc(1);
            theta = loc(2);
            uex(k,j)= getEx(rad,theta,iOption);
        end
    end
    uexd = uex(jas:jbs,jar:jbr);
    err  = abs(Ud-uexd);
    norm_err = max(max(err));
%     figure
%     surf(Xd,Yd,err);
else
    norm_err = 0;
end

end

%% Functions

function uex = getEx(r,theta,iOption)
if iOption==1
    uex = (r-1)^2*(r-2)^2*sin(theta);
else
    uex = 0;
end
end

function Vrr = getVrr(r,theta,iOption)
if iOption==1
    Vrr = (2*sin(theta)/r)*( (r-1)*(r-2)^2 + r*(r-2)^2 + ...
        4*r*(r-1)*(r-2) + (r-1)^2*(r-2) + r*(r-1)^2 );
else
    Vrr = 0;
end
end

```

```matlab
function Vtt = getVtt(r,theta,iOption)
if iOption==1
    Vtt = (-1/r^2)*getEx(r,theta,iOption);
else
    Vtt = 0;
end
end

function F = getF(r,theta,iOption)
if iOption==1
    Vrr = getVrr(r,theta,iOption);
    Vtt = getVtt(r,theta,iOption);
    F   = Vrr+Vtt;
else
    F = 0;
end
end

%% boundary conditions

function ubc1 = getBC1(r,slim1,iOption)
if iOption==1
    ubc1 = (r-1)^2*(r-2)^2*sin(slim1);
else
    ubc1 = 0;
end
end

function ubc2 = getBC2(rlim1,theta,iOption)
if iOption==1
    ubc2 = 2*((rlim1-1)*(rlim1-2)^2 + (rlim1-1)^2*(rlim1-2))*sin(theta);
else
    ubc2 = 0;
end
end

function ubc3 = getBC3(rlim2,theta,iOption)
if iOption==1
    ubc3 = 2*((rlim2-1)*(rlim2-2)^2 + (rlim2-1)^2*(rlim2-2))*sin(theta);
else
    ubc3 = 0;
end
end

function ubc4 = getBC4(r,slim2,iOption)
if iOption==1
    ubc4 = (r-1)^2*(r-2)^2*sin(slim2);
else
    ubc4 = (r-1)^2*(r-2)^2;
end
end
```

Listing 2: Heat Equation 2D - Steady State, Mapping

.

(b) Using 40 grid lines in both the radial and angular coordinate directions, compute a numerical solutions to this problem, and create a surface plots of the solution. In addition, create a single line plot with two curves showing the solution along the inner radius

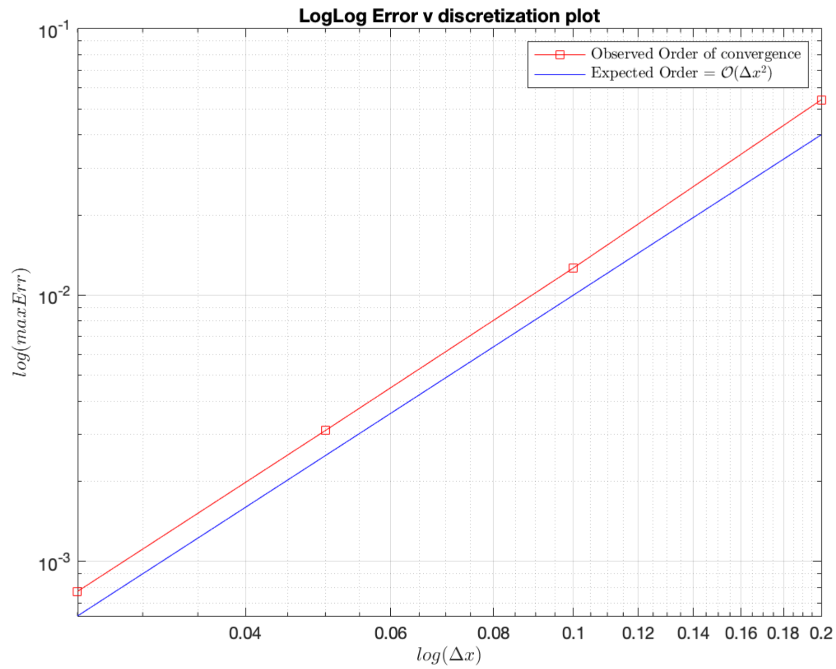

Figure 4: Error Convergence plot

($r = 1$), and the outer radius ($r = 2$), as a function of θ.
I have written down two iterative solvers. Three cases are shown in the following figures Fig 5, Fig 6 and Fig 7. The Iterative solver code is attached in Listing 3. The matrix $\underline{\underline{\mathbf{A}}}$ is first separated into the Lower diagonal ($\underline{\underline{\mathbf{L}}}$), Diagonal ($\underline{\underline{\mathbf{D}}}$) and upper diagonal ($\underline{\underline{\mathbf{U}}}$) matrices.

```
function unew = IterativeSolver(A,b,uini,sOption)
% This function solves the linear system Av = b
% sOption==0 - Jacobi
% sOption==1 - Gauss-Siedal
% sOption==2 - SRC
% sOption==3 - Inverse
[m,~] = size(A);
D      = zeros(m);
L      = zeros(m);
U      = zeros(m);
for i=1:m
    for j=1:m
        if (i==j)
            D(i,i) = A(i,i);
        elseif j>i
            U(i,j) = A(i,j);
        else
            L(i,j) = A(i,j);
        end
    end
end

tol      = 1e-6;
max_iter = 1000;
```

```matlab

if sOption==0
    unew     = uini;
    R        = A*uini-b;
    Dinvb    = D\b;
    DinvLpU  = D\(L+U);
    itr      = 1;
    while norm(R)>tol && itr<max_iter
        uold = unew;
        unew = Dinvb - DinvLpU*uold;
        R    = A*unew-b;
        itr  = itr+1;
    end
elseif sOption==1
    unew     = uini;
    R        = A*uini-b;
    Ls       = L+D;
    Lsinvb   = Ls\b;
    LsinvU   = Ls\U;
    itr      = 1;
    while norm(R)>tol && itr<max_iter
        unew = Lsinvb - LsinvU*unew;
        R    = A*unew-b;
        itr  = itr+1;
    end
elseif sOption==2
elseif sOption==3
    unew = A\b;
end
end
```

Listing 3: Iterative Solver Function Handle

.

- **Jacobi**: Tolerance $\varepsilon = 1e-3$ and Max Iterations $= 3000$

$$\underline{u}^{n+1} = \underline{\underline{\mathbf{D}}}^{-1}\,\underline{b} - \underline{\underline{\mathbf{D}}}^{-1}\left(\underline{\underline{\mathbf{L}}} + \underline{\underline{\mathbf{U}}}\right)$$
$$\underline{\underline{\mathbf{R}}} = \underline{\underline{\mathbf{A}}}\;\underline{u}^{n+1} - \underline{b}$$

  This is done until the norm of the Residual $\underline{\underline{\mathbf{R}}}$ gets below ε or until max iterations are reached.

- **Gauss-Siedel**: Tolerance $\varepsilon = 1e-3$ and Max Iterations $= 3000$

$$\underline{\underline{\mathbf{L}}}^{*} = \underline{\underline{\mathbf{L}}} + \underline{\underline{\mathbf{D}}}$$
$$\underline{u}^{n+1} = \underline{\underline{\mathbf{L}}}^{*-1}\,\underline{b} - \underline{\underline{\mathbf{L}}}^{*-1}\underline{\underline{\mathbf{U}}}$$
$$\underline{\underline{\mathbf{R}}} = \underline{\underline{\mathbf{A}}}\;\underline{u}^{n+1} - \underline{b}$$

  This is done until the norm of the Residual $\underline{\underline{\mathbf{R}}}$ gets below ε or until max iterations are reached.

- **Direct**: A direct solver is used and its computed as $\underline{u} = \underline{\underline{\mathbf{A}}}^{-1}\,\underline{b}$.

(c) Finally, compare the steady state solution you compute here to the solutions of the time-dependent problem from PS7. In particular show the approach of the time-dependent solutions to the steady solution along the inner radius $r = 1$.

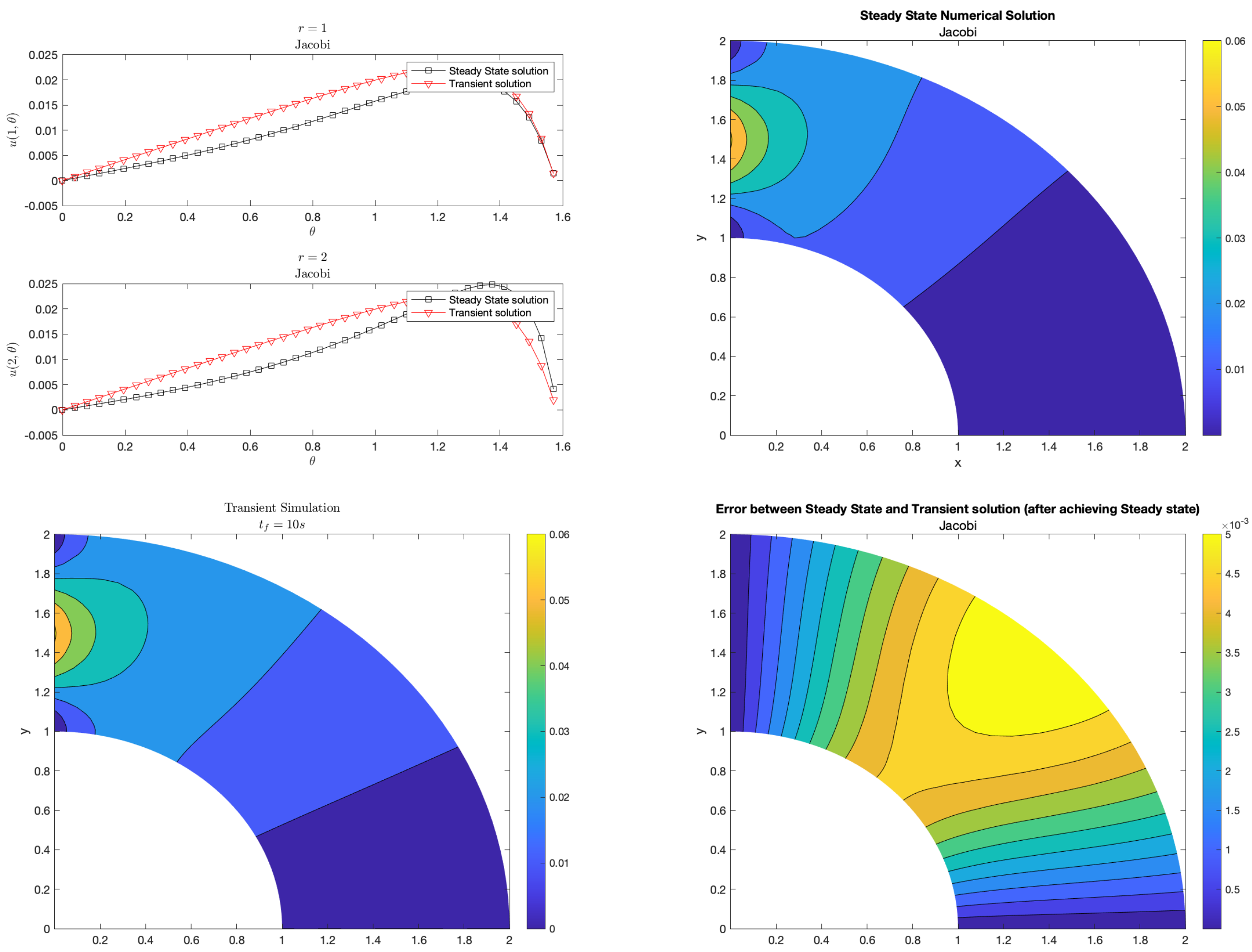

Figure 5: Jacobi Iterative solver results

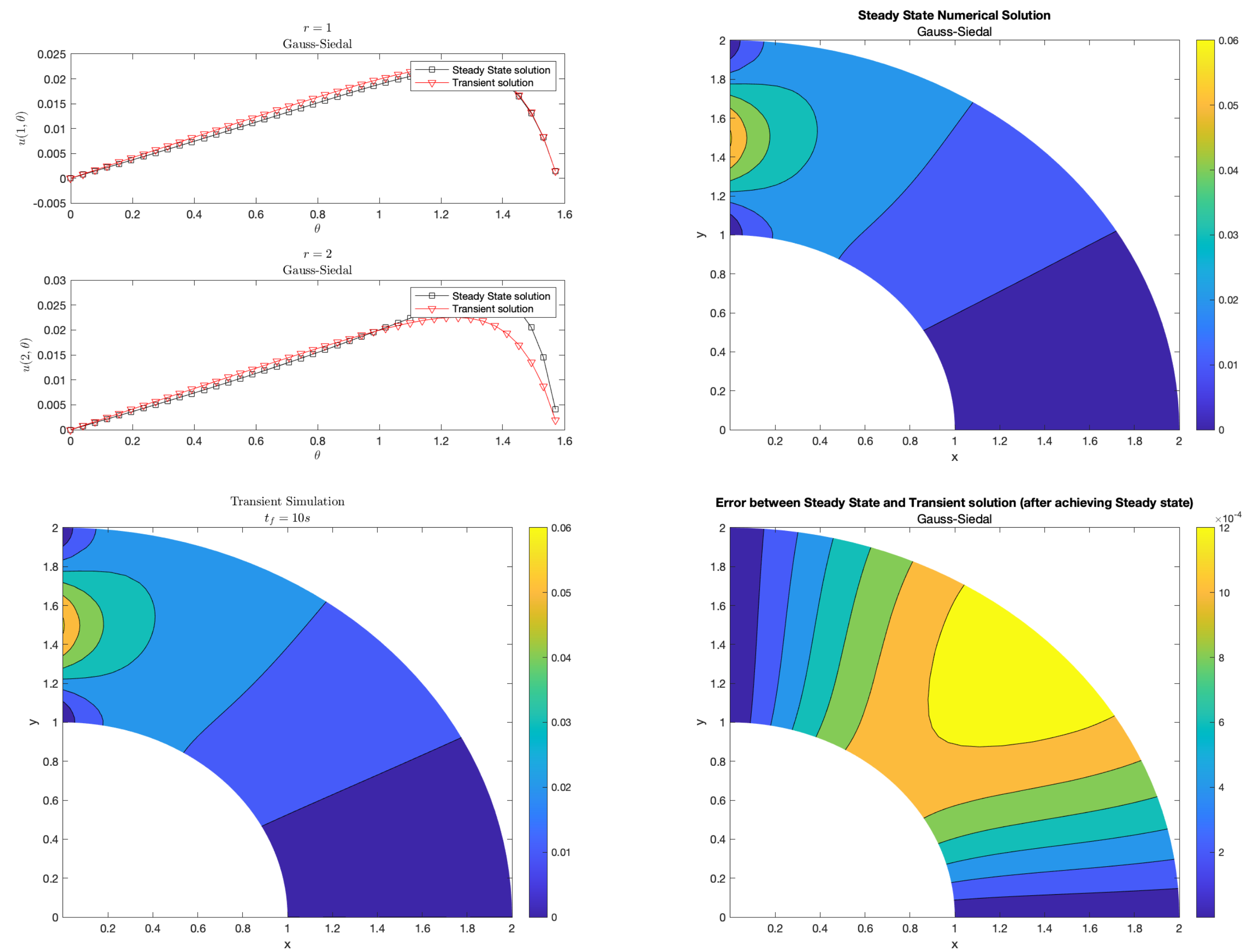

Figure 6: Gauss Siedal Iterative solver results

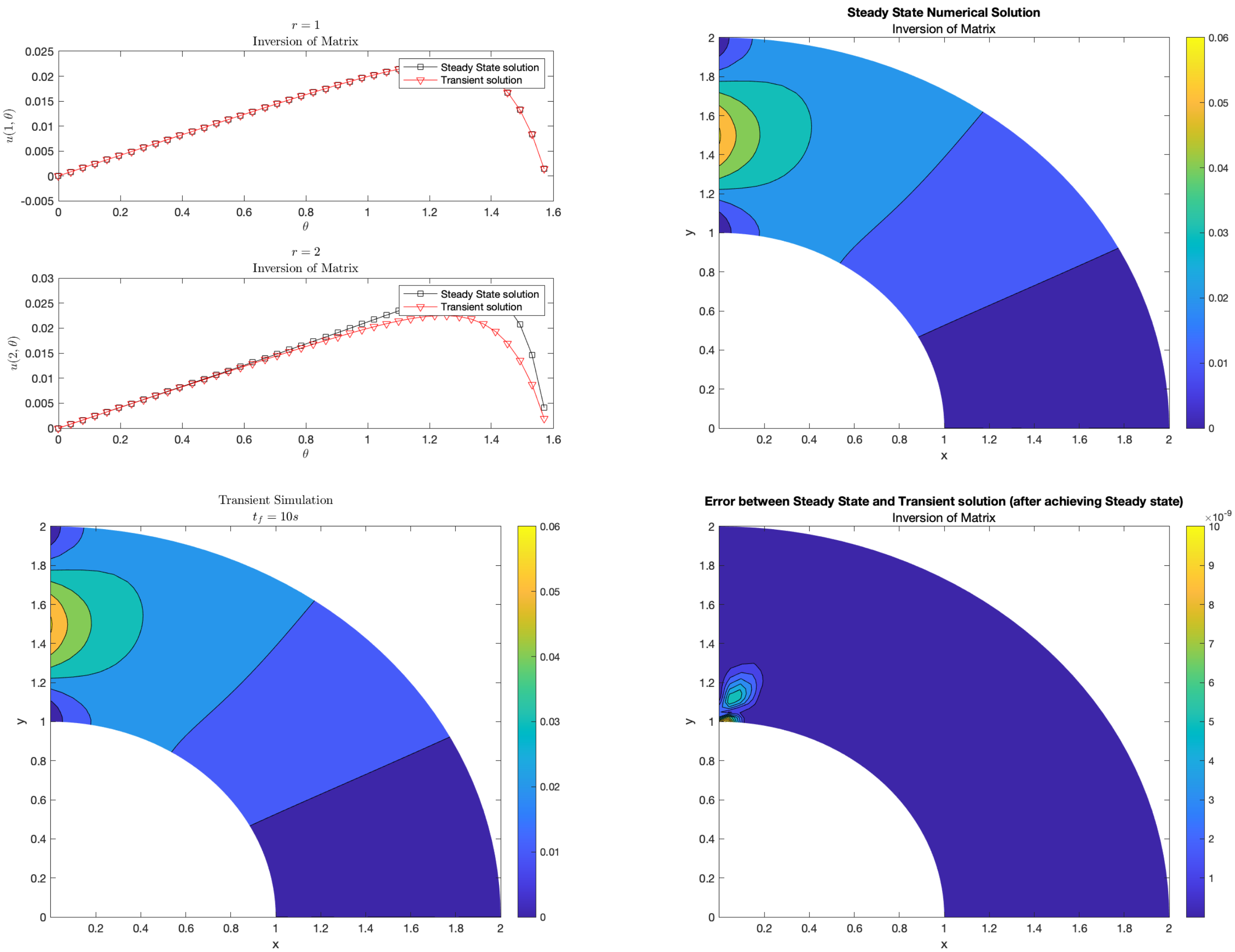

Figure 7: Inversion results

The comparison at $r = 1$ are also plotted in Fig 5, Fig 6 and Fig 7. The results of the transient simulation at $r = 2$ seems to agree better with the steady state results as opposed to the solutions at the inner radius $r = 1$.