Computational Methods

Francisco Castillo
Homework 6

April 29, 2018

Problem 1

Consider the differential equation

$$y' = \sin(t)(y^2 - \cos^2(t) - 1), \quad y(0) = 1, \quad t > 0.$$

The exact solution to this nonlinear ODE is $y(t) = \cos(t)$.

Use `rk4` to solve this equation for $t \in [0, 50]$. Plot the error vs. values of Δt (use `loglog`). Does the error decay as $\mathcal{O}(\Delta t^4)$?

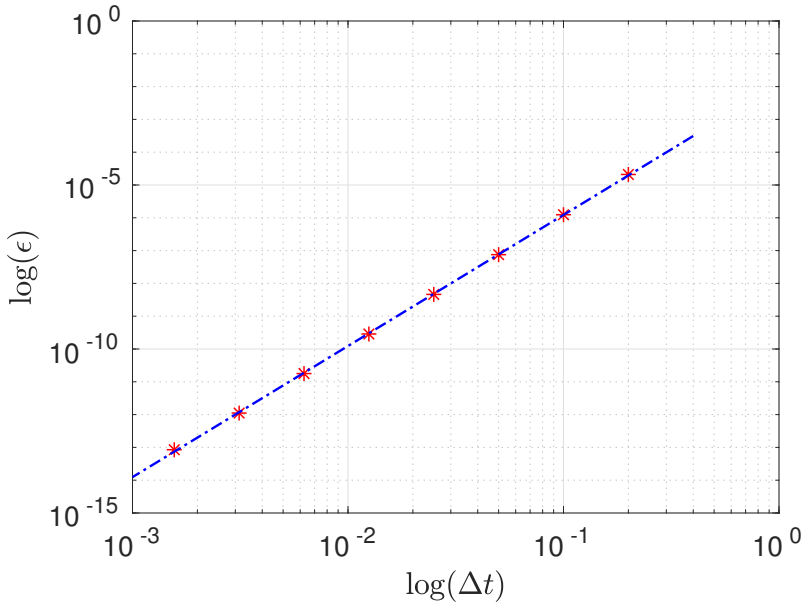


Figure 1: Error Vs. Δt

As we can see in the figure above, we have done a linear fit with the *Matlab* function `polyfit` obtaining a slope of 4. Hence, the figure above shows that the error decays as $\mathcal{O}(\Delta t^4)$.

You will now compare `rk4`, forward-Euler, and modified Euler methods. Solve the problem with these three methods for several Δt. on a `loglog` plot, show the

error vs. the number of function evaluations used by eahc method. Using the number of function evaluations as a measure of cost, is it worth using several stages in RK methods?

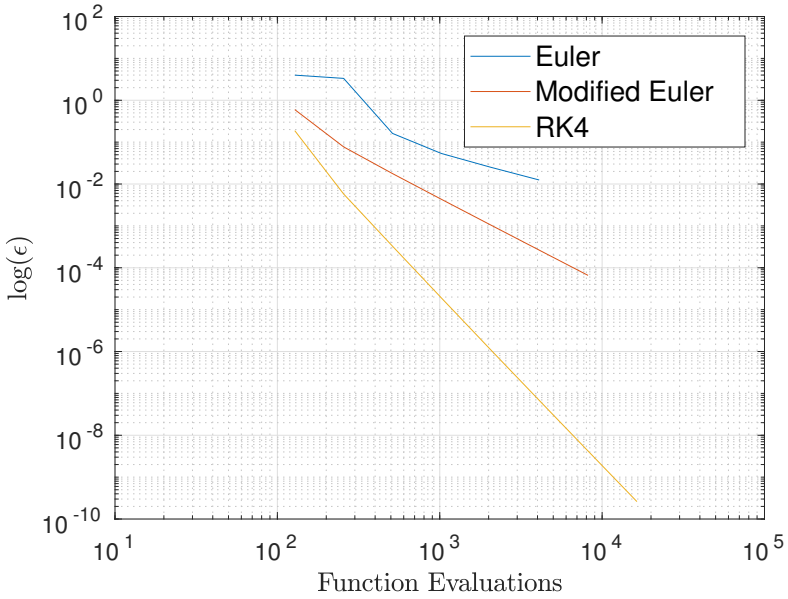


Figure 2: Error Vs. cost

As we can see in the previous figure, it is worth using several stages in RK methods despite their cost. With the same number of funciton evaluations, RK4 reaches much higher level of accuracy.

Matlab code for this part

```
clear variables; close all; format long; clc
path='Latex/FIGURES/';

linewidth=1.2;
markersize=7.5;
legendfontsize=12;
axisfontsize=16;

%% Problem 1
% Part a)
dt = 2e-1;
dydt = @(t,y) sin(t)*(y.^2-(cos(t))^2-1);
err=1;
tol=1e-12;
tspan=[0 50];
y0=1;
i=1;
```

```
while err>tol
    N=(tspan(2)-tspan(1))/dt(i);
    [t,y] = rk4(dydt,tspan,y0,N);
    err(i)=norm(y-cos(t),inf);
    dt(i+1)=dt(i)/2;
    i=i+1;
end
dt(end)=[];

% Linear fit
P=polyfit(log(dt),log(err),1);
dtfit=2*dt(1):-0.001:dt(end)/2;
errfit=exp(P(2))*dtfit.^(P(1));

figure(1)
loglog(dt,err,'r*','markersize',markersize)
grid on
hold on
loglog(dtfit,errfit,'b-.','linewidth',linewidth)
grid on
xlabel('$\log(\Delta t)$','Interpreter','latex')
ylabel('$\log(\epsilon)$','Interpreter','latex')
set(gca,'fontsize',14)
txt=[path,'problem1_a'];
saveas(gcf,txt,'epsc')

% Part b)
N=2.^(4:12);
for i=1:length(N);
    [t1,y1] = euler(dydt,tspan,y0,N(i));
    [t2,y2] = modEuler(dydt,tspan,y0,N(i));
[t4,y4] = rk4(dydt,tspan,y0,N(i));
    err1(i)=norm(y1-cos(t1),inf);
    err2(i)=norm(y2-cos(t2),inf);
    err4(i)=norm(y4-cos(t4),inf);
end
figure
loglog(N,err1,2*N,err2,4*N,err4)
grid on
ylabel('$\log(\epsilon)$','Interpreter','latex')
set(gca,'fontsize',12)
xlabel('Function Evaluations','Interpreter','latex')
legend({'Euler','Modified Euler', 'RK4'},'fontsize',14)
txt=[path,'problem1_b'];
saveas(gcf,txt,'epsc')
```

```
function [t,w] = rk4(dydt,tspan,y0,N)

h = diff(tspan)/N;
t = tspan(1) + h*(0:N)';
w = zeros(length(y0),N+1);
w(:,1) = y0(:).';

for i = 1:N
    k1 = h*dydt(t(i)    ,w(:,i)      );
    k2 = h*dydt(t(i)+h/2, w(:,i)+k1/2);
    k3 = h*dydt(t(i)+h/2, w(:,i)+k2/2);
    k4 = h*dydt(t(i)+h  , w(:,i)+k3  );
    w(:,i+1) = w(:,i) + (k1 + 2*k2 + 2*k3 + k4)/6;
end
w = w.';
end

function [t,w] = modEuler(dydt,tspan,y0,N)

h = diff(tspan)/N;
t = tspan(1) + h*(0:N)';
w = zeros(length(y0),N+1);
w(:,1) = y0(:).';

for i = 1:N
    k1 = h*dydt(t(i)    ,w(:,i)      );
    k2 = h*dydt(t(i)+h, w(:,i)+k1);
    w(:,i+1) = w(:,i) + (k1 + k2)/2;
end
w = w.';
end

function [t,w] = euler(dydt,tspan,y0,N)

h = diff(tspan)/N;
t = tspan(1) + h*(0:N)';
w = zeros(length(y0),N+1);
w(:,1) = y0(:).';

for i = 1:N
    k1 = h*dydt(t(i)    ,w(:,i)      );
w(:,i+1) = w(:,i) + k1;
end
w = w.';
```

```
end
```

Problem 2

Suppose the discretization of a boundary value problem, such as the Poisson equation, leads to the matrix equation

$$uA + Bu = f,$$

that needs to be solved for u. Here u is $n \times m$, A is $m \times m$, B is $n \times n$, and f is $n \times m$.

Assuming both A and B are non-defective matrices, with $A = V_A \Lambda_A V_A^{-1}$ and $B = V_B \Lambda_B V_B^{-1}$, show that the solution u is given by $u = V_B U V_A^{-1}$, where

$$U_{i,j} = \frac{F_{i,j}}{\alpha_j + \beta_i},$$

$F = V_B^{-1} f V_A$, and α_k and β_k are the diagonal entries of Λ_A and Λ_B respectively.

We start with the eigenvalue decomposition of the matrices A and B,

$$uV_A\Lambda_A V_A^{-1} + V_B\Lambda_B V_B^{-1} u = f,$$

and premultiplying by V_B^{-1} multiplying from the right by V_A,

$$V_B^{-1} u V_A \Lambda_A + \Lambda_B V_B^{-1} u V_A = V_B^{-1} f V_A.$$

Then let $U = V_B^{-1} u V_A$ and $F = V_B^{-1} f V_A$ so that. Therefore, we have

$$U\Lambda_A + \Lambda_B U = F,$$

which in matrix form is

$$\begin{bmatrix} \tilde{u}_{11} & \cdots & \tilde{u}_{1m} \\ \tilde{u}_{21} & \cdots & \tilde{u}_{2m} \\ \vdots & & \vdots \\ \tilde{u}_{n1} & \cdots & \tilde{u}_{nm} \end{bmatrix} \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & \alpha_m \end{bmatrix} + \begin{bmatrix} \beta_1 & 0 & \cdots & 0 \\ 0 & \beta_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & \beta_m \end{bmatrix} \begin{bmatrix} \tilde{u}_{11} & \cdots & \tilde{u}_{1m} \\ \tilde{u}_{21} & \cdots & \tilde{u}_{2m} \\ \vdots & & \vdots \\ \tilde{u}_{n1} & \cdots & \tilde{u}_{nm} \end{bmatrix} = \begin{bmatrix} \tilde{f}_{11} & \cdots & \tilde{f}_{1m} \\ \tilde{f}_{21} & \cdots & \tilde{f}_{2m} \\ \vdots & & \vdots \\ \tilde{f}_{n1} & \cdots & \tilde{f}_{nm} \end{bmatrix}.$$

We can infer from the matrix equation above that

$$U_{i,j} = \frac{F_{i,j}}{\alpha_j + \beta_i}.$$

Then, to get back our solution, we use $u = V_B U V_A^{-1}$.

Using part (a), solve the Poisson equation $\Delta u = \sin(x)\cos(100y)$ on $[0,1] \times [0,1]$ (with zero Dirichlet boundary conditions) using 100 points in the x direction and 300 in the y direction. Estimate the accuracy of your solution.

Coding into *Matlab* the previous method, we obtain a solution for u shown in the figure below,

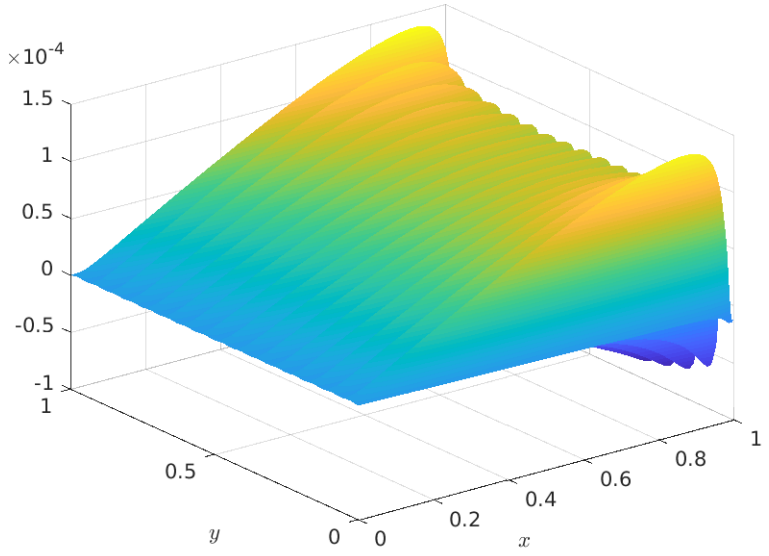


Figure 3: Solution0 u of the poisson equation.

The error, found by comparing $uD2A + D2Bu$ and f, where $D2A$ and $D2B$ are the centered finite difference matrices for x and y respectively, is $2.10806 \cdot 10^{-12}$.

Matlab code for this part

```
%% Problem 2
Nx=100; Ny=300;
hx=1/(Nx-1); hy=1/(Ny-1);

D2x=gallery('tridiag',Nx,1,-2,1)/(hx^2);
D2y=gallery('tridiag',Ny,1,-2,1)/(hy^2);

[Vx,LambdaX]=eig(full(D2x));
[Vy,LambdaY]=eig(full(D2y));
[x,y]=meshgrid(linspace(0,1,100),linspace(0,1,300));
f=sin(x).*cos(100*y);
F=Vy'*f*Vx;
U=zeros(size(f));
lambdax=diag(LambdaX);
lambday=diag(LambdaY);
% whos U F lambdax lambday
for j=1:Nx
    U(:,j)=F(:,j)./(lambday+lambdax(j));
end

u=Vy*U*Vx';
```

```
% Calculate error
err=norm(u*D2x+D2y*u-f,inf)
figure
surf(x,y,u)
shading interp
xlabel('$x$','Interpreter','latex')
ylabel('$y$','Interpreter','latex')
set(get(gca,'ylabel'),'rotation',0)
txt=[path,'problem2_u'];
saveas(gcf,txt,'png')
```

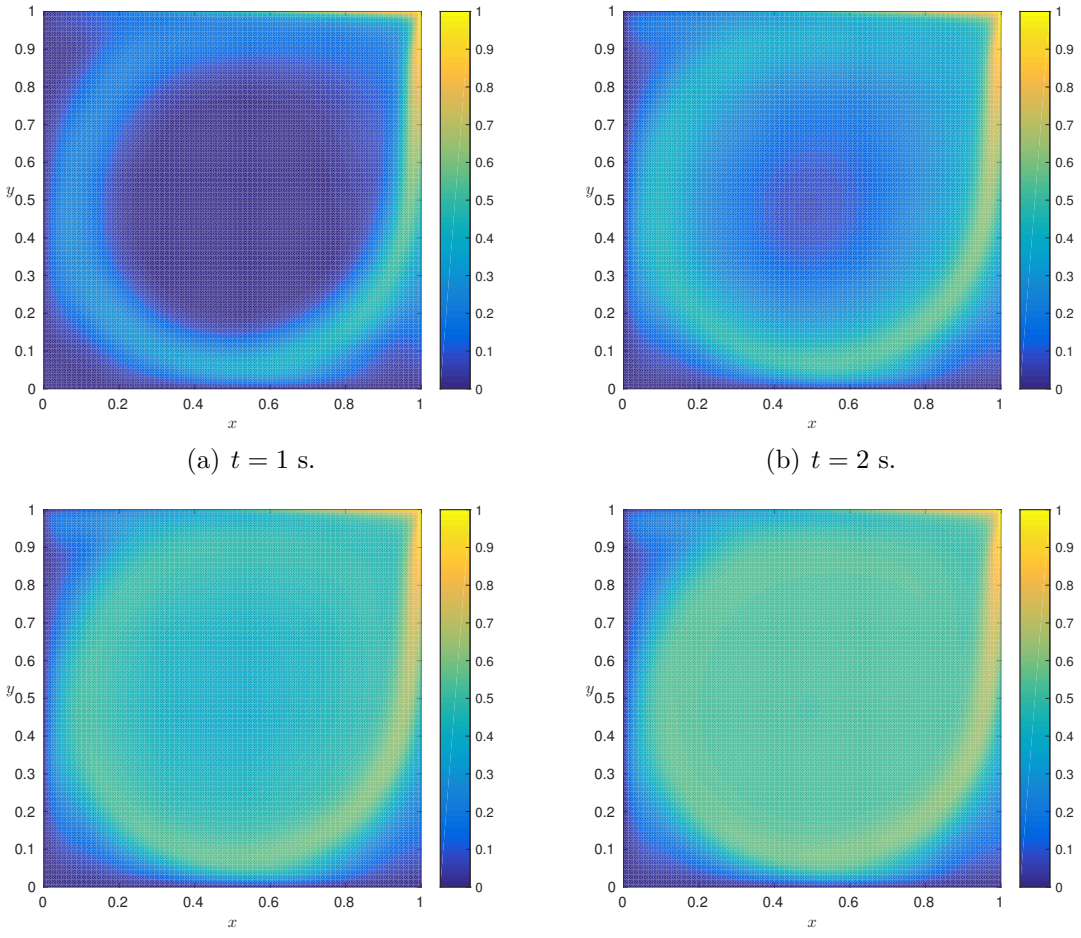
Problem 3

Using finite differences on the given grid and the data in u and v from `velocitydata.mat`, solve the equation of temperature in a fluid flow

$$T_t + uT_x + vT_y = \alpha(T_{xx} + T_{yy}), \quad (x, y) \in (0, 1) \times (0, 1), \quad t > 0,$$

where $\alpha = 0.0005$. Assume that $T = 0$ at $t = 0$ and that $T(t, x, y) = xy \tanh(10t)^2$ on the boundary. use `contourf` to plot the level curves to the temperature at times $t = 10, 50, 150, 300$.

We can appreciate that the equation to solve has the same form than the vorticity equation using the frozen values for the velocity given as data. Just adjusting the given code we obtain the following results.



(a) $t = 1$ s. (b) $t = 2$ s.

(c) $t = 3$ s. (d) $t = 5$ s.

Figure 4: Temperature Contours at different values of time.

Matlab code for this part

```
clear variables; close all; clc
```

```
DATA=load('velocitydataPlatte.mat');
u=DATA.u;
v=DATA.v;
xx=DATA.xx;
yy=DATA.yy;
fluidflowFDTemperature(xx,yy,u,v)

% Fluid flow in a cavity (Navier-Stokes equations)
% Stream-function and vorticity formulation
%
% Rodrigo Platte, Arizona State University, April 2013.

function fluidflowFDTemperature(xx,yy,u,v)
% Parameters
path='Latex/FIGURES/';
N = 150;
alpha = 0.0005;
dt = min(0.2/(alpha*N^2),1e-3);

h = 1/N; % N+1 points in each direction

% Initial condition & pre-allocate memory (start with zeros at t = 0)
T = 0*xx;

% boundary condition
ii = 2:N; jj = 2:N;  % index for interior nodes
Ti = zeros(N-1,N-1); % interior values of vorticity

count = 0;
time = 0;
outputTime=[10 50 150 300];
endtime=outputTime(end);
n=1;
% main loop
while time<endtime
if ( time < outputTime(n) && time+dt >= outputTime(n) ) %#ok<ALIGN>
        dt=outputTime(n)-time;
        n=n+1;
    else
        dt = min(0.2/(alpha*N^2),1e-3);
    end
   % advance Temperature with forward Euler
   Ti = Ti + dt*Trhs(u,v,T,alpha);
   T(ii,jj)=Ti;
```

```
    % update Temperature at boundary
    T = BCT(xx,yy,time,T);

    time = time+dt;
    count = count + 1;

    % Plot results every now and then
    if ismember(time,outputTime);
        plotresults(time,T);
    end

end
    % Right-hand side of vorticity equation
    function rhs = Trhs(u,v,T,alpha)
        Tx = (T(ii,jj+1)-T(ii,jj-1))/(2*h);
        Ty = (T(ii+1,jj)-T(ii-1,jj))/(2*h);
        LT = (T(ii,jj+1)+T(ii,jj-1)+T(ii+1,jj)+T(ii-1,jj)-4*T(ii,jj))/h^2;
        rhs = -u(ii,jj).*Tx - v(ii,jj).*Ty + alpha*LT;
    end

    % Enforce boundary condition (temperature)
    function T = BCT(xx,yy,t,T)
        T(1,:)=xx(1,:).*yy(1,:).*tanh(10*t).^2;
        T(N+1,:)=xx(N+1,:).*yy(N+1,:).*tanh(10*t).^2;
        T(:,1)=xx(:,1).*yy(:,1).*tanh(10*t).^2;
        T(:,N+1)=xx(:,N+1).*yy(:,N+1).*tanh(10*t).^2;
    end

    % Plot results
    function plotresults(time,T)
        figure();set(gcf,'Visible', 'off');
        pcolor(xx,yy,T)
        shading interp
        H = colorbar; set(H,'fontsize',16)
        caxis([0 1])
        colorbar
        axis square
        drawnow
        xlabel('$x$','Interpreter','latex')
        ylabel('$y$','Interpreter','latex')
        set(get(gca,'ylabel'),'rotation',0)
        txt=[path,'T_' num2str(time)];
        saveas(gcf,txt,'epsc')
    end
```

```
end
```

Problem 4

Modify the code `fluidflowFD.m` to solve the equations

$$\begin{aligned} w_t + \psi_y w_x - \psi_x w_y &= Pr\Delta w + RaPrT_x, \\ T_t + \psi_y T_x - \psi_x T_y &= \Delta T, \\ \Delta\psi &= -w, \end{aligned}$$

where $(x, y) \in (0, 1) \times (0, 1)$ and $t > 0$. For this exercise, set $Ra = 2 \times 10^5$ and $Pr = 0.71$ (air).

The fluid is at rest at $t = 0$, with $T = \psi = w = 0$. The boundary conditions for the stream function is $\psi_\Gamma = 0$, which implies that there is no mass transfer through the boundary Γ. The value of the vorticity at the walls is expressed as $w_\Gamma = -\Delta\psi_\Gamma$ and the temperature at Γ is defined by

$$T(t, x, y) = \begin{cases} 2^9 \tanh^4(100t)x^5(x-1)^4, & y = 0,\ 0 \le x \le 1,\ t > 0, \\ 0, & (x, y) \in \Gamma,\ y \neq 0,\ t > 0. \end{cases}$$

By replicating the code given to solve the fluid flow, adding the extra step of the temperature, we obtain the following result.

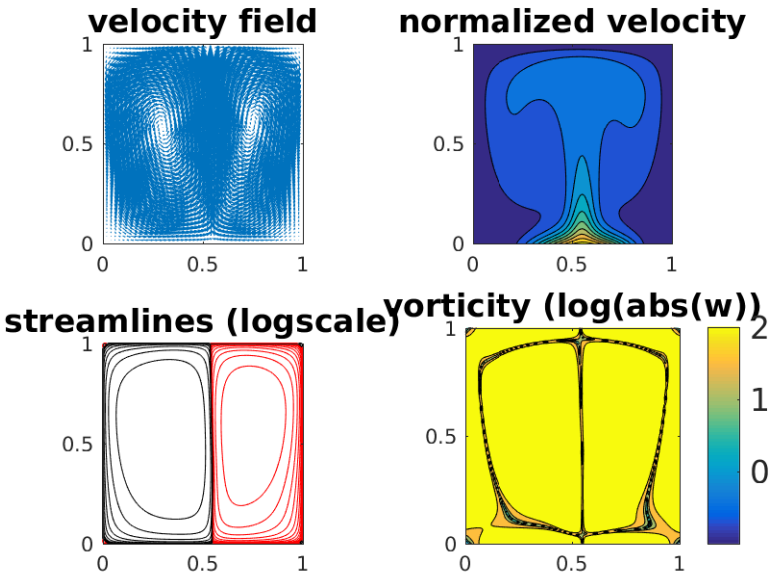


Figure 5: Solution at $t = 1$ second

Matlab code for shit part

```
%% Problem 4
Ra=2e5;
Pr=0.71;
```

```
Problem4(Ra,Pr)

% Fluid flow in a cavity (Navier-Stokes equations)
% Stream-function and vorticity formulation
%
% Rodrigo Platte, Arizona State University, April 2013.

function Problem4(Ra,Pr)
% Parameters
path='Latex/FIGURES/';
N = 150;
dt = 1e-5;

h = 1/N; % N+1 points in each direction
[xx,yy] = meshgrid(0:h:1); % 2D gridpoints

% Laplacian with zero BCs
o = ones(N-1,1);
D2 = (diag(-2*o) + diag(o(1:N-2),-1)+ diag(o(1:N-2),1))/h^2;
% We need evals and evects to solve Poisson equation
[eV,eval] = eig(D2); lam = diag(eval);

% Initial condition & pre-allocate memory (start with zeros at t = 0)
u = 0*xx; v = u; w = u; str = u; T=u;

% boundary condition
ii = 2:N; jj = 2:N;  % index for interior nodes
wi = zeros(N-1,N-1); % interior values of vorticity
Ti = zeros(N-1,N-1); % interior values of temperature

count = 0;
time = 0;
outputTime=[1];
endtime=outputTime(end);
n=1;
% main loop
while time<endtime
if ( time < outputTime(n) && time+dt >= outputTime(n) ) %#ok<ALIGN>
        dt=outputTime(n)-time;
        n=n+1;
    else
        dt = 1e-5;
    end
   % update vorticity at boundary
   w = BCw(u,v,w);
```

```
    % update Temperature at boundary
    T = BCT(xx,time,T);

    % advance vorticity and temperature with forward Euler
     [rhsT,Tx]=Trhs(u,v,T);
     Ti = Ti + dt*rhsT;
     T(ii,jj)=Ti;
     wi = wi + dt*wrhs(u,v,w,Pr,Ra,Tx);
     w(ii,jj) = wi;

    % compute stream function with conjugate gradient
    str(2:end-1,2:end-1) = SolvePoisson(wi);

    % update velocity (interior nodes only)
    u(ii,jj) =  (str(ii+1,jj)-str(ii-1,jj))/(2*h);
    v(ii,jj) = -(str(ii,jj+1)-str(ii,jj-1))/(2*h);

    time = time+dt;
    count = count + 1;

    % Plot results every now and then
    if ismember(time,outputTime);
        plotresults(u,v,str,w,T);
    end

end

     % Solve Poisson equation (Sylvester equation)
     function S = SolvePoisson(wi)
         ff = -eV'*wi*eV;
         S = ff;
         for j = 1:N-1
             S(:,j) = ff(:,j)./(lam(j)+lam);
         end
         S = eV*S*eV';
     end

     % Right-hand side of vorticity equation
     function rhs = wrhs(u,v,w,Pr,Ra,Tx)
         wx = (w(ii,jj+1)-w(ii,jj-1))/(2*h);
         wy = (w(ii+1,jj)-w(ii-1,jj))/(2*h);
         Lw = (w(ii,jj+1)+w(ii,jj-1)+w(ii+1,jj)+w(ii-1,jj)-4*w(ii,jj))/h^2;
         rhs = -u(ii,jj).*wx - v(ii,jj).*wy + Pr*Lw+Ra*Pr*Tx;
     end
     % Right-hand side of temperature equation
```

```
function [rhs, Tx] = Trhs(u,v,T)
    Tx = (T(ii,jj+1)-T(ii,jj-1))/(2*h);
    Ty = (T(ii+1,jj)-T(ii-1,jj))/(2*h);
    LT = (T(ii,jj+1)+T(ii,jj-1)+T(ii+1,jj)+T(ii-1,jj)-4*T(ii,jj))/h^2;
    rhs = -u(ii,jj).*Tx - v(ii,jj).*Ty + LT;
end

% Enforce boundary condition (vorticity)
function w = BCw(u,v,w)
    % three point sided FD formula (second order accurate)
    w(1,:) = -(-3*u(1,:)+4*u(2,:)-u(3,:))/(2*h);      % = -u_y @ y = 0
    w(N+1,:) = (-3*u(N+1,:)+4*u(N,:)-u(N-1,:))/(2*h); % = -u_y @ y = 1
    w(:,1) = (-3*v(:,1)+4*v(:,2)-v(:,3))/(2*h);       % =  v_x @ x = 0
    w(:,N+1) = -(-3*v(:,N+1)+4*v(:,N)-v(:,N-1))/(2*h);% =  v_x @ x = 1
end
% Enforce boundary condition (temperature)
function T = BCT(xx,t,T)
    T(1,:)=2^9*tanh(100*t)^4*xx(1,:).^5.*(xx(1,:)-1).^4;
end

% Plot results
function plotresults(u,v,str,w,T)
    % Use a subset of gridpoints
   ip = 1:3:N+1;
   up = u(ip,ip); vp = v(ip,ip);
   xp = xx(ip,ip); yp = yy(ip,ip);

   % Velocity
   subplot(2,2,1)
   quiver(xp,yp,up,vp,10)
   axis([0 1 0 1]), axis square
   title('velocity field','fontsize',16)

   % Normalized velocity
   subplot(2,2,2)
   contourf(xx,yy,T)
   axis([0 1 0 1]), axis square
   title('normalized velocity','fontsize',16)

   % Stream function
   subplot(2,2,3)
   mp = max(max(str));
   mm = min(min(str));
   contour(xx,yy,str,-logspace(-10,log10(-mm),30),'r'),  hold on
   contour(xx,yy,str, logspace(-10,log10(mp),20),'k'),   hold off
```

```
        axis square, title('streamlines (logscale)','fontsize',16)

        % Vorticity
        subplot(2,2,4)
        logw = log10(abs(w));    logw(abs(w)==0) = nan;
        contourf(xx,yy,logw,-3:0.5:2),
        H = colorbar; set(H,'fontsize',16)
        set(gca,'position',[0.5703 0.1100 0.3347 0.3412])
        axis square, title('vorticity (log(abs(w))','fontsize',16)
        drawnow
         txt=[path,'solutionProblem4'];
         saveas(gcf,txt,'epsc')
    end

end
```