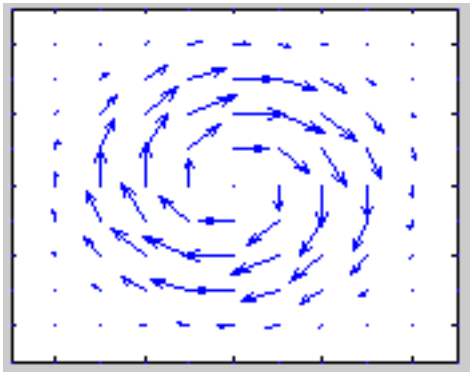
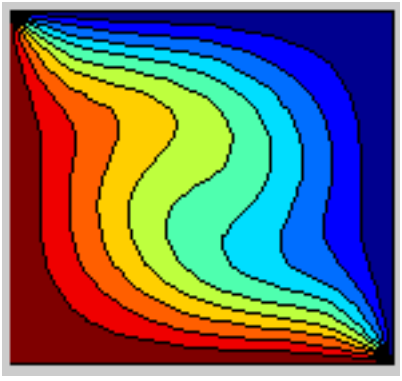
Mechanical Engineering 9620 Computational Fluid Dynamics
University of New South Wales
Assignment #1 April 29, 2003

John Middendorf
Student #3049731





I hereby certify that this document contains
my own original and independent work.

John Middendorf

## Statement of Assignment

The steady state energy transport equation (convection/conduction) for an incompressible Newtonian fluid, for a two-dimensional Cartesian co-ordinate system, can be written as:

$$\frac{\partial u'T'}{dx'} + \frac{\partial v'T'}{dy'} = \alpha\left(\frac{\partial^2 T'}{dx'^2} + \frac{\partial^2 T'}{dy'^2}\right)$$

where the terms on the left represent the advection (transfer of properties by movement of the fluid) terms, and the terms on the right represent the diffusion terms.

The energy equation is applied to a square domain with boundary conditions, with the velocity field given by:

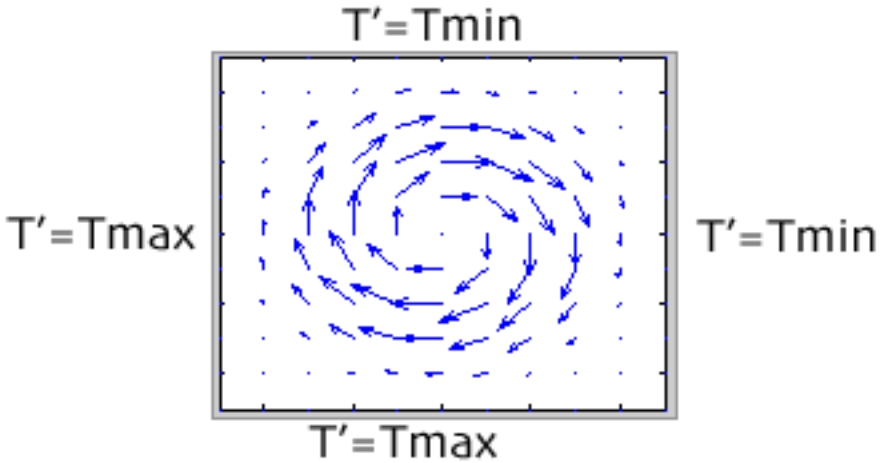
$$\vec{V} = u'\hat{i} + v'\hat{j}$$

where

$$u' = -(0.25 - h)h\left(\frac{y'}{L} - \frac{1}{2}\right)U_o \qquad v' = -(0.25 - h)h\left(\frac{x'}{L} - \frac{1}{2}\right)U_o$$

$$h = f(x,y) = 15 * \max\left[0, \left\{0.25 - \left(\frac{x'}{L} - \frac{1}{2}\right)^2 - \left(\frac{y'}{L} - \frac{1}{2}\right)^2\right\}\right]$$

The boundary conditions and flow field are illustrated below:



## Part A: Non-Dimensional Equations

The first step is to non-dimensionalize the equations, using:

$$\theta = \frac{T' - T\min}{T\max - T\min} = \frac{T' - T\min}{\Delta T} \Rightarrow T' = \theta \Delta T - T\min$$

$$u = \frac{u'}{U_o} \quad v = \frac{v'}{U_o} \Rightarrow u' = uU_o \quad v' = vU_o$$

$$x = \frac{x'}{L} \quad y = \frac{y'}{L} \Rightarrow x' = xL \quad y' = yL$$

Substituting into the general equation, we get:

$$\frac{U_o \Delta T}{L}\left(\frac{\partial u\theta}{dx} + \frac{\partial v\theta}{dy}\right) = \frac{\alpha \Delta T}{L^2}\left(\frac{\partial^2 \theta}{dx^2} + \frac{\partial^2 \theta}{dy^2}\right)$$

Which reduces to:

$$\left(\frac{\partial u\theta}{dx} + \frac{\partial v\theta}{dy}\right) = \frac{1}{Pe}\left(\frac{\partial^2 \theta}{dx^2} + \frac{\partial^2 \theta}{dy^2}\right)$$

Where Pe (the Péclet number) is given by:

$$Pe = \frac{U_o L}{\alpha}$$

which can be interpreted as the ratio of heat transfer by advection to the heat transfer by conduction.

Using a similar process, we can non-dimensionalize the velocity equations to:

$$u = -(0.25 - h)h\left(y - \frac{1}{2}\right) \quad v = -(0.25 - h)h\left(x - \frac{1}{2}\right)$$

$$h == 15 * \max\left[0, \left\{0.25 - \left(x - \frac{1}{2}\right)^2 - \left(y - \frac{1}{2}\right)^2\right\}\right]$$

## Part B: Solving the equation using differencing schemes

Matlab was used to write the programs and display the data. The first step is the discretisation of the equation:

$$\left(\frac{\partial u\theta}{dx}+\frac{\partial v\theta}{dy}\right)=\frac{1}{Pe}\left(\frac{\partial^2\theta}{dx^2}+\frac{\partial^2\theta}{dy^2}\right)$$

Expanding the left side:

$$\frac{\partial u\theta}{dx}+\frac{\partial v\theta}{dy}=u\frac{\partial\theta}{dx}+\theta\frac{\partial u}{dx}+v\frac{\partial\theta}{dy}+\theta\frac{\partial v}{dy}$$

but: $\theta\left(\frac{\partial u}{dx}+\frac{\partial v}{dy}\right)=0$ since div(**V**)=0 for an incompressible fluid (mass conservation).

Therefore we have:

$$\left(u\frac{\partial\theta}{dx}+v\frac{\partial\theta}{dy}\right)=\frac{1}{Pe}\left(\frac{\partial^2\theta}{dx^2}+\frac{\partial^2\theta}{dy^2}\right)$$

Next we consider two differencing schemes:
--one using central differences for both the advection and the diffusion terms
--one using upwind differences for the advection term, and central differences for the second order diffusion terms.

For first order upwind differences:

$$\frac{\partial u}{\partial x}=\frac{u}{\Delta x}\left(T_{i+1,j}-T_{i,j}\right) \quad \text{if u<0} \qquad \frac{\partial u}{\partial x}=\frac{u}{\Delta x}\left(T_{i,j}-T_{i-1,j}\right) \quad \text{if u>0}$$

$$\frac{\partial v}{\partial x}=\frac{v}{\Delta x}\left(T_{i,j+1}-T_{i,j}\right) \quad \text{if v<0} \qquad \frac{\partial v}{\partial x}=\frac{v}{\Delta x}\left(T_{i,j}-T_{i,j-1}\right) \quad \text{if v>0}$$

For first order central differences:

$$\frac{\partial u}{\partial x}=\frac{u}{2\Delta x}\left(T_{i+1,j}-T_{i-1,j}\right) \quad \text{and} \quad \frac{\partial v}{\partial x}=\frac{v}{2\Delta x}\left(T_{i,j+1}-T_{i,j-1}\right)$$

For second order central differences:

$$\frac{\partial^2\theta}{\partial x^2}=\left(\frac{\theta_{i+1,j}-2\theta_{i,j}+\theta_{i-1,j}}{\Delta x^2}\right) \quad \text{and} \quad \frac{\partial^2\theta}{\partial y^2}=\left(\frac{\theta_{i,j+1}-2\theta_{i,j}+\theta_{i,j-1}}{\Delta y^2}\right)$$

## Part B, continued.

In the Matlab programs listed in the Appendix, three different schemes were utilized:
*For Central Advection/Central Diffusion differencing (program 1), an explicit method using a time step was programmed.
*For Upwind Advection/Central Diffusion differencing (program 2), an explicit method was programmed.
*Also for Central Advection/Central Diffusion, an implicit/explicit method using the Samarskii/Andreyev scheme was programmed (program 3). Program 1 and Program 3 gave identical results, verifying the explicit methodology.

An analysis of the program code reveals how the schemes were implemented. In general, matrix arrays representing the grids were initialized, then the velocity field was calculated. A new temperature matrix (t) was calculated based on the previous temperature matrix using one of the schemes mentioned above, then the difference was calculated, determining the continuation or end of the iteration process. For example, in Program 1, the discretisation of the utilized equation :

$$\frac{\partial T}{\partial t}+\left(\frac{\partial u\theta}{dx}+\frac{\partial v\theta}{dy}\right)=\frac{1}{Pe}\left(\frac{\partial^2\theta}{dx^2}+\frac{\partial^2\theta}{dy^2}\right)$$ resulted in solving:

$$T^{n+1}=T^n+\frac{u\Delta t}{2\Delta x}\left(T_{i+1,j}-T_{i-1,j}\right)-\frac{v\Delta t}{2\Delta x}\left(T_{i,j+1}-T_{i,j-1}\right)$$

$$+\left(\frac{\Delta t}{Pe\Delta x^2}\right)\left(T_{i+1,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1}-4T_{i,j}\right) \quad \text{since } \Delta x=\Delta y$$

In Program 1, the timestep was determined by a Von Neumann stability analysis, and error checked to ensure: $\frac{(\max velocity)\Delta t}{\Delta x}<\frac{2\alpha\Delta t}{\Delta x^2}<\frac{1}{2}$

In Program 3, the Samarskii/Andreyev Method was used, with the implicit/explicit value of sigma =1/2. The program successively solves for omega first implicit in x, then implicit in y, then updates the temperature array.

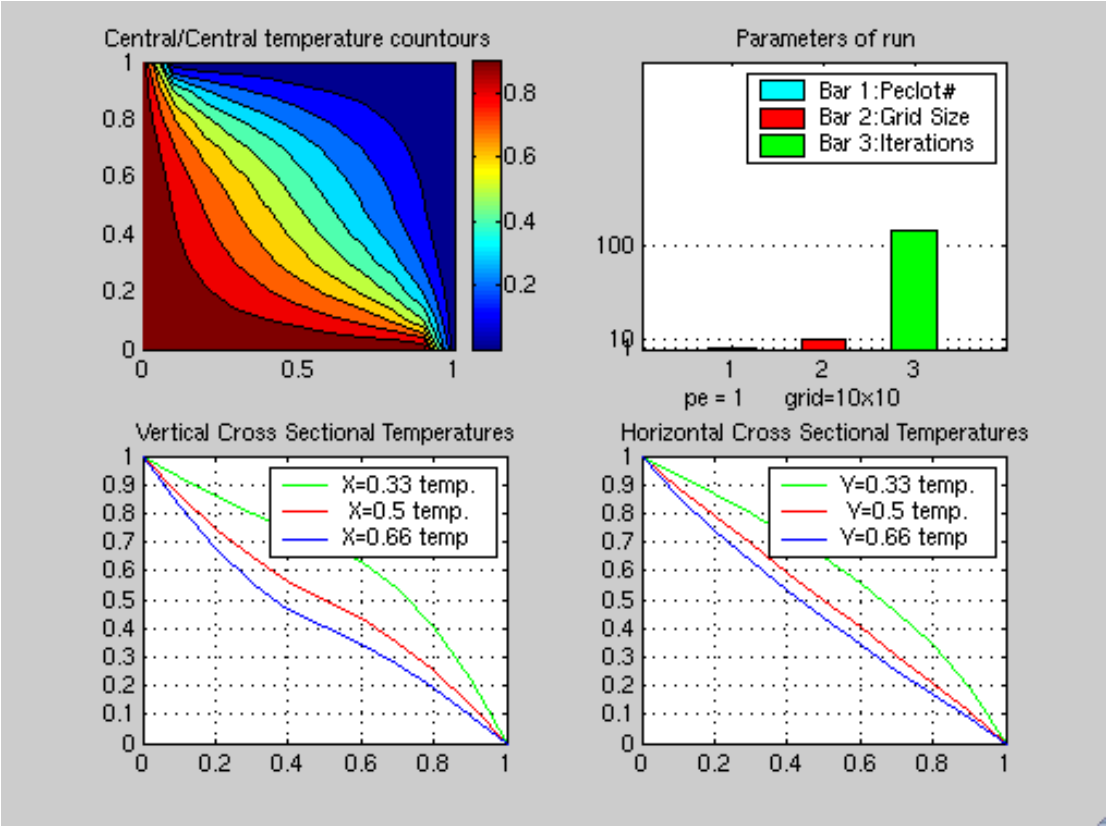
$$\omega^{*}+\sigma\Delta t\,u\delta_x\omega^{*}-\frac{\sigma\Delta t}{Pe}\delta_x^2\omega^{*}=-u\delta_x\theta^n-v\delta_y\theta^n+\frac{1}{Pe}\delta_x^2\theta^n+\frac{1}{Pe}\delta_y^2\theta^n$$

$$\omega^{**}+\sigma\Delta t\,v\delta_y\omega^{**}-\frac{\sigma\Delta t}{Pe}\delta_y^2\omega^{**}=\omega^{*}$$

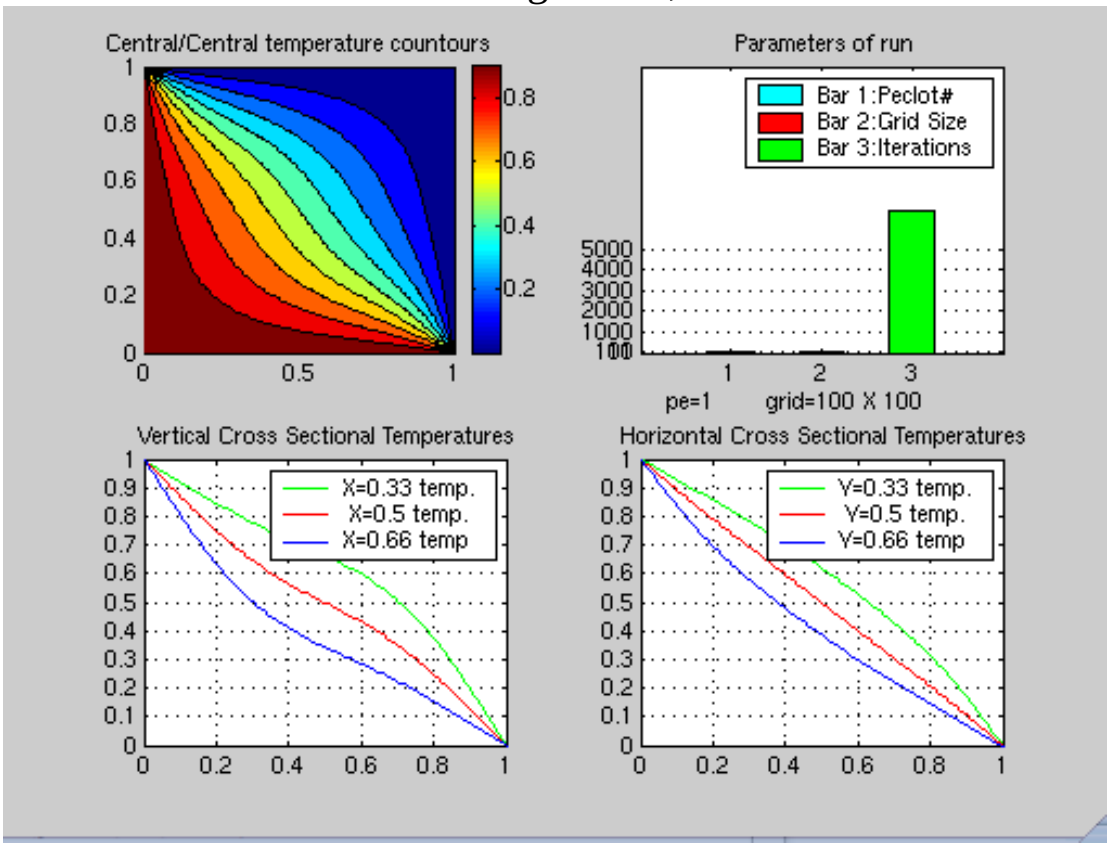
$$\theta^{n+1}=\theta^n+k\omega^{**}$$

--(from Professor Leonardi, 2003, Chapter 7)

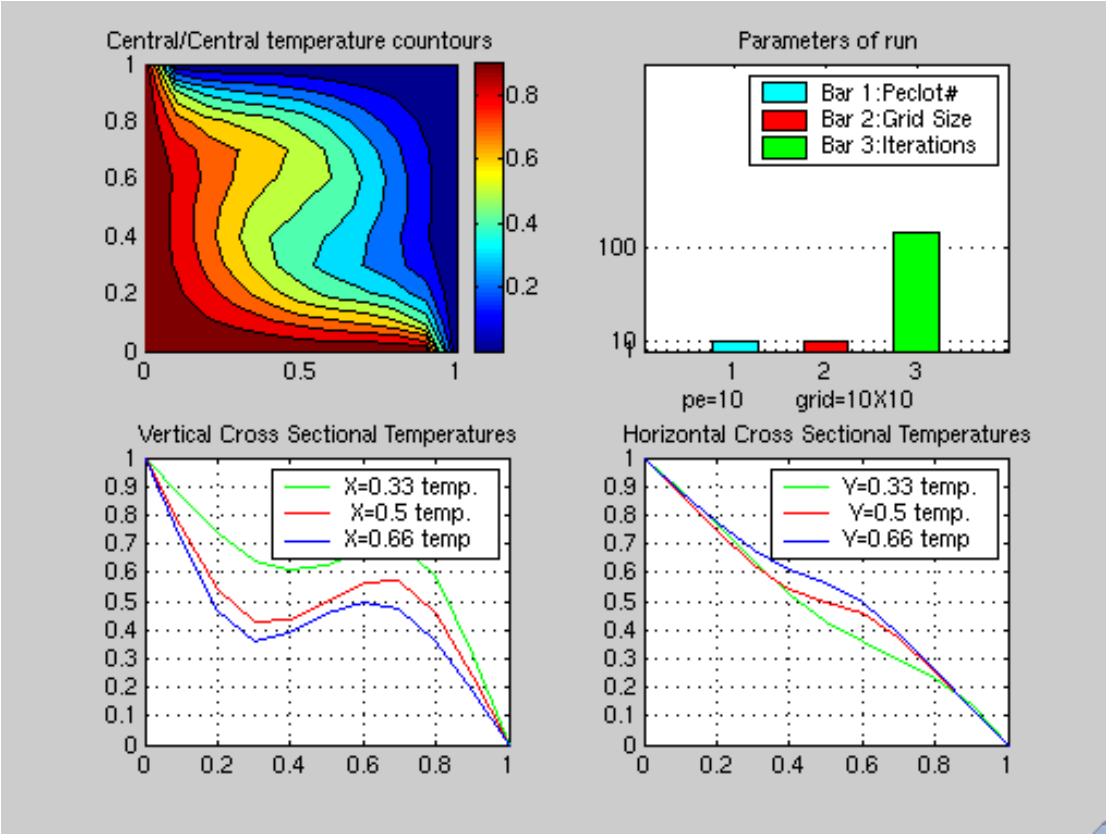
Part C: Results



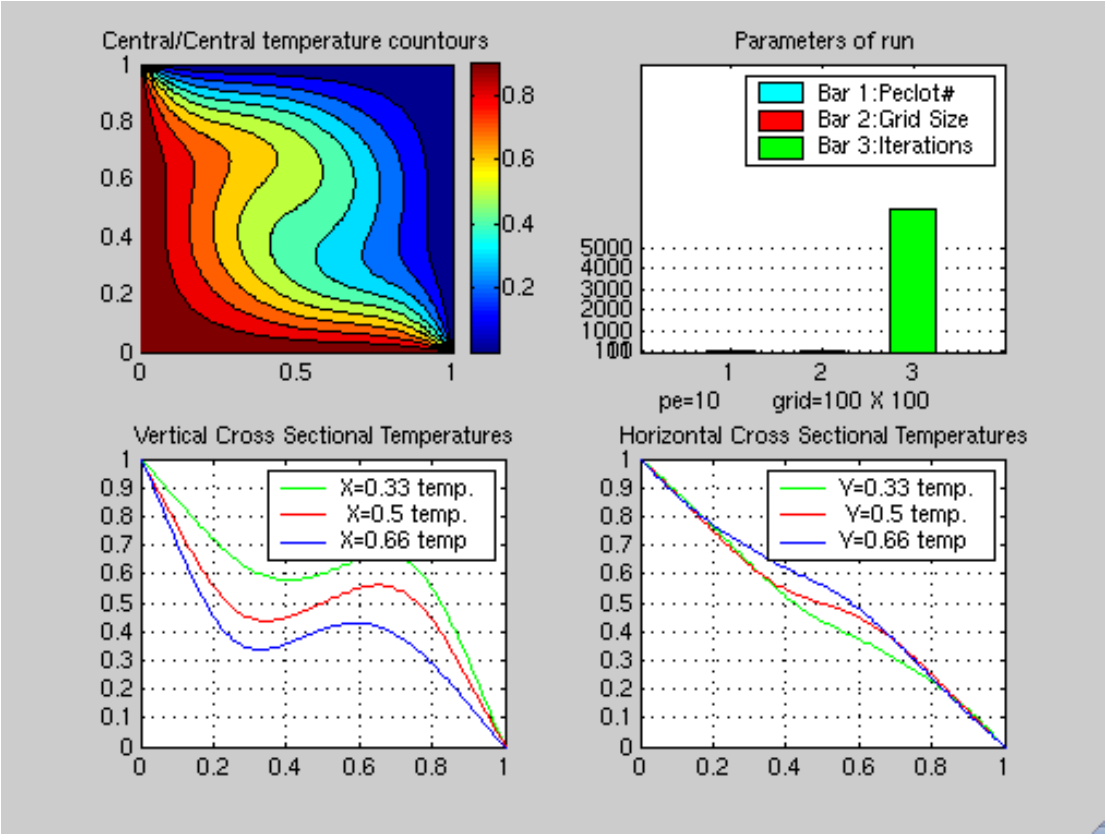

1. Central/Central Differencing Pe=1, Grid= 10x10



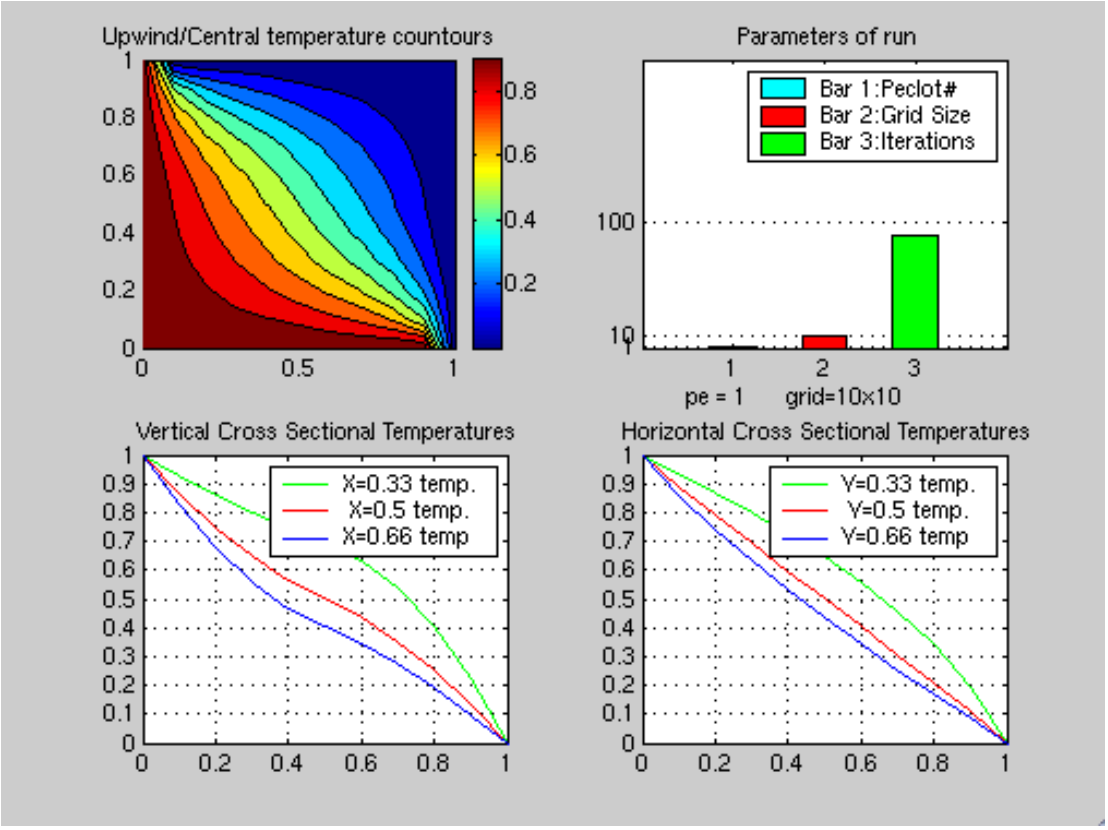

2. Central/Central Differencing Pe=1, Grid= 100x100



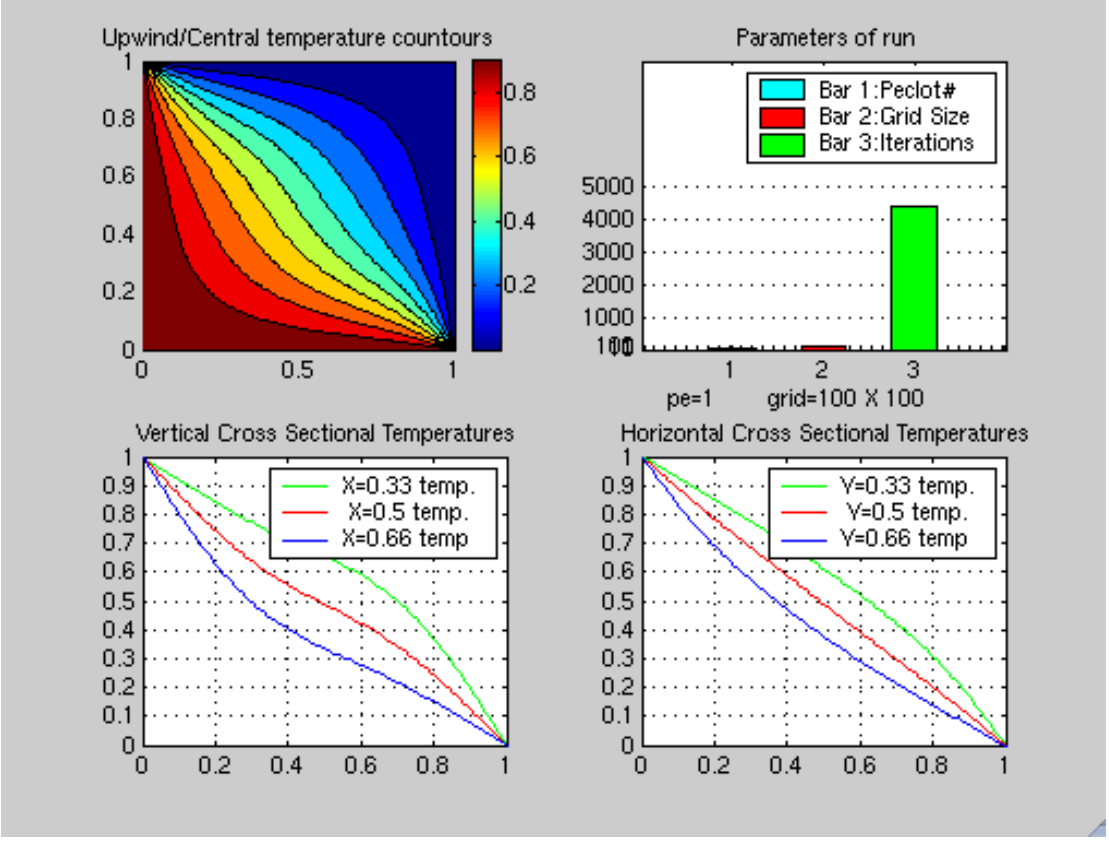

3. Central/Central Differencing Pe=10, Grid= 10x10



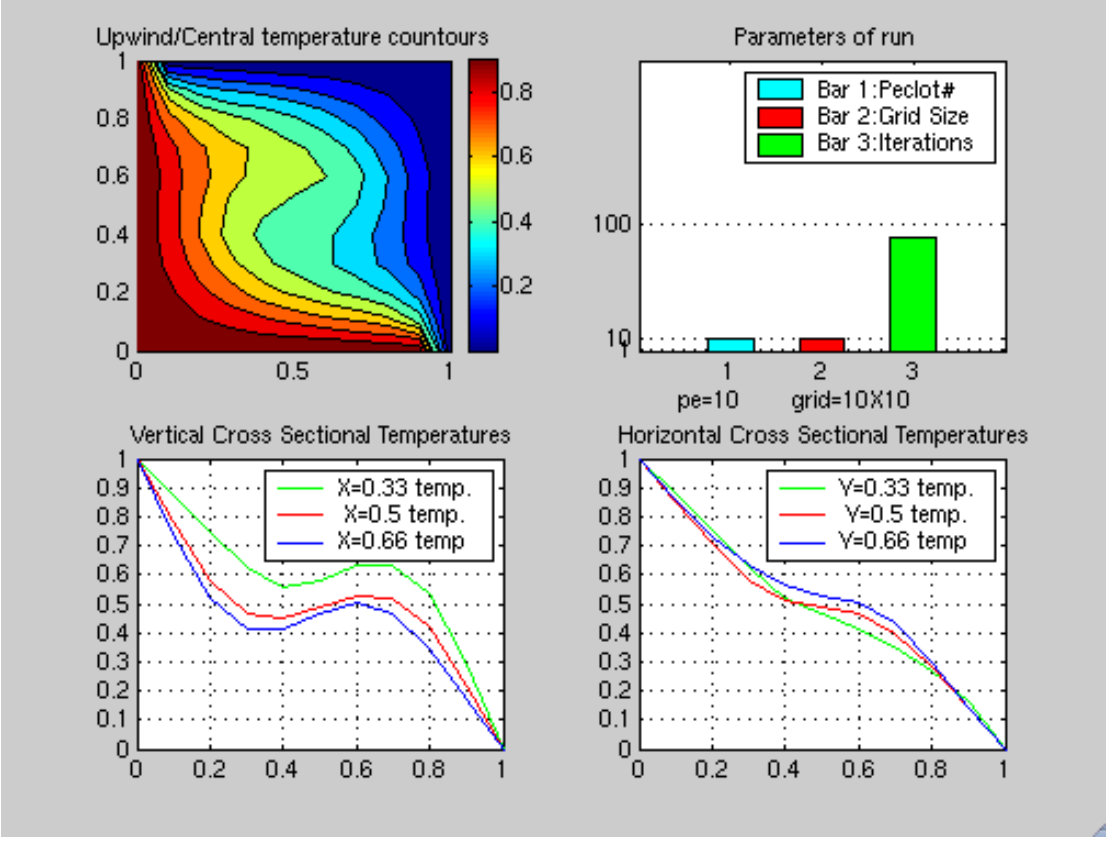

4. Central/Central Differencing Pe=10, Grid= 100x100



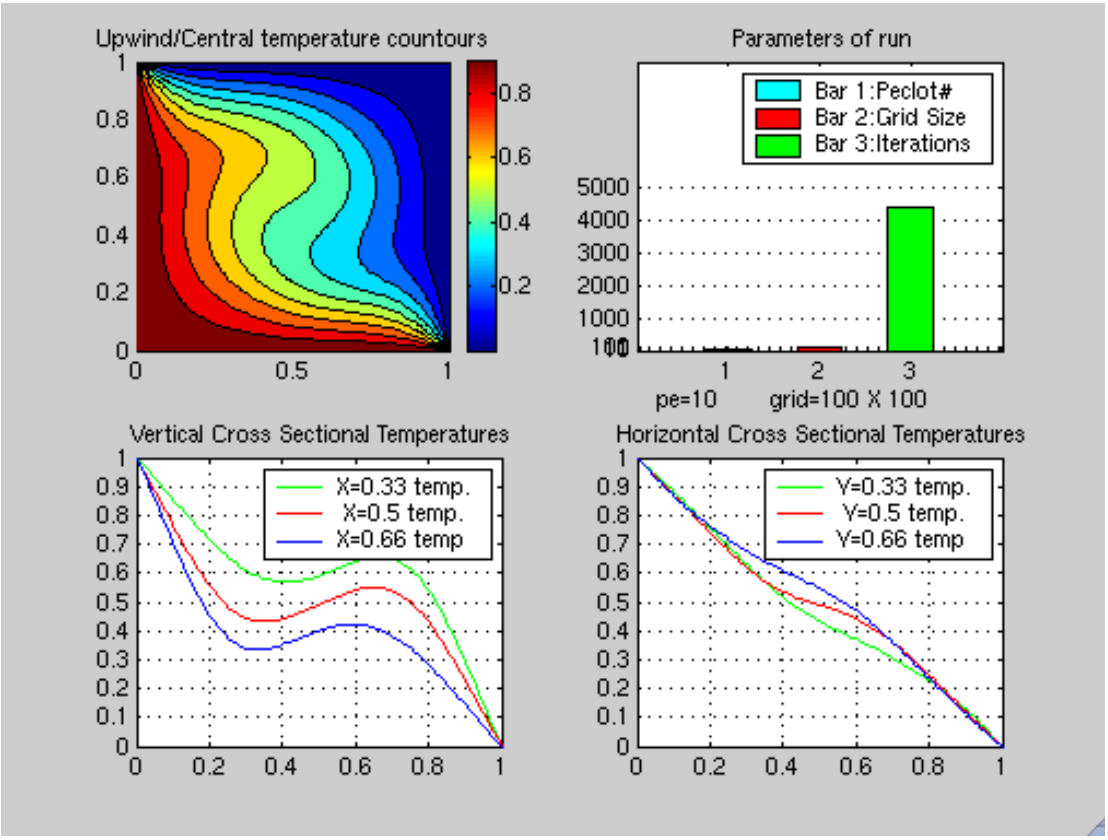

5. Upwind/Central Differencing Pe=1, Grid= 10x10



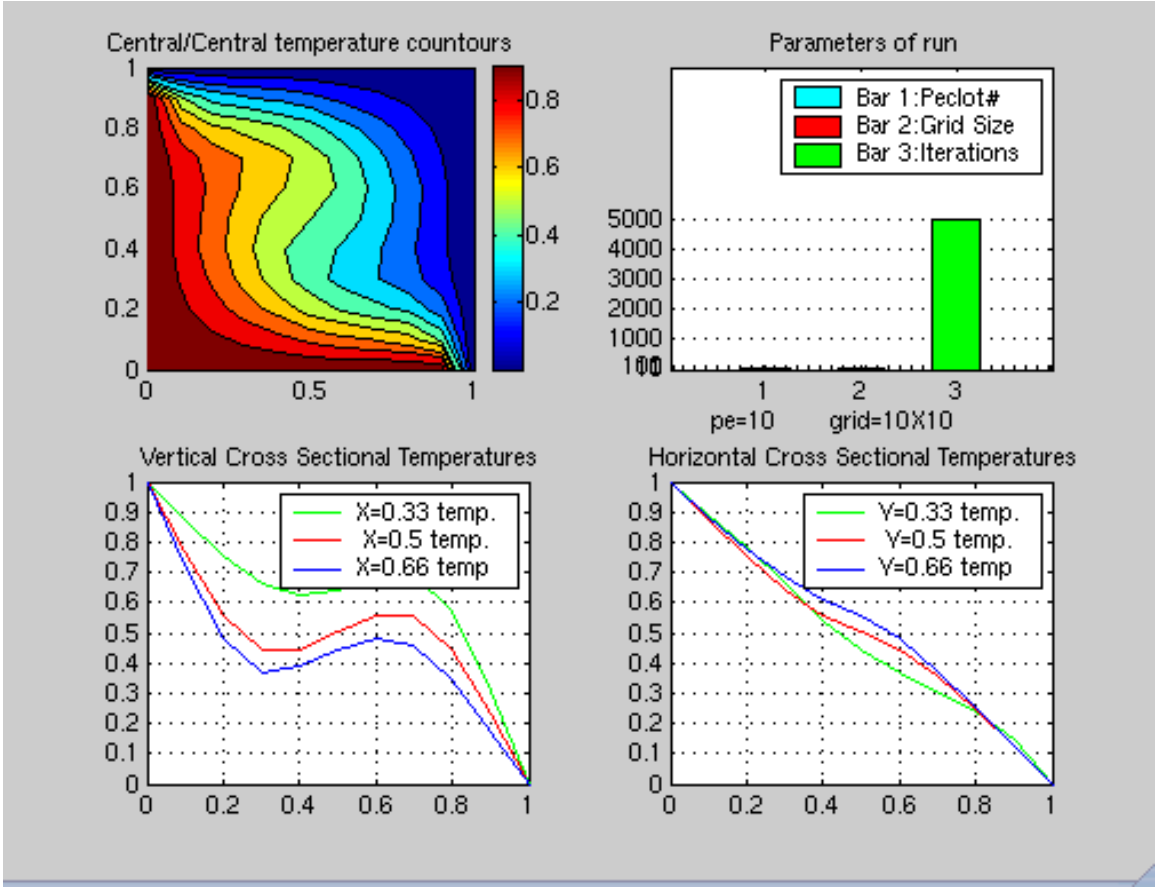

6. Upwind/Central Differencing Pe=1, Grid= 100x100



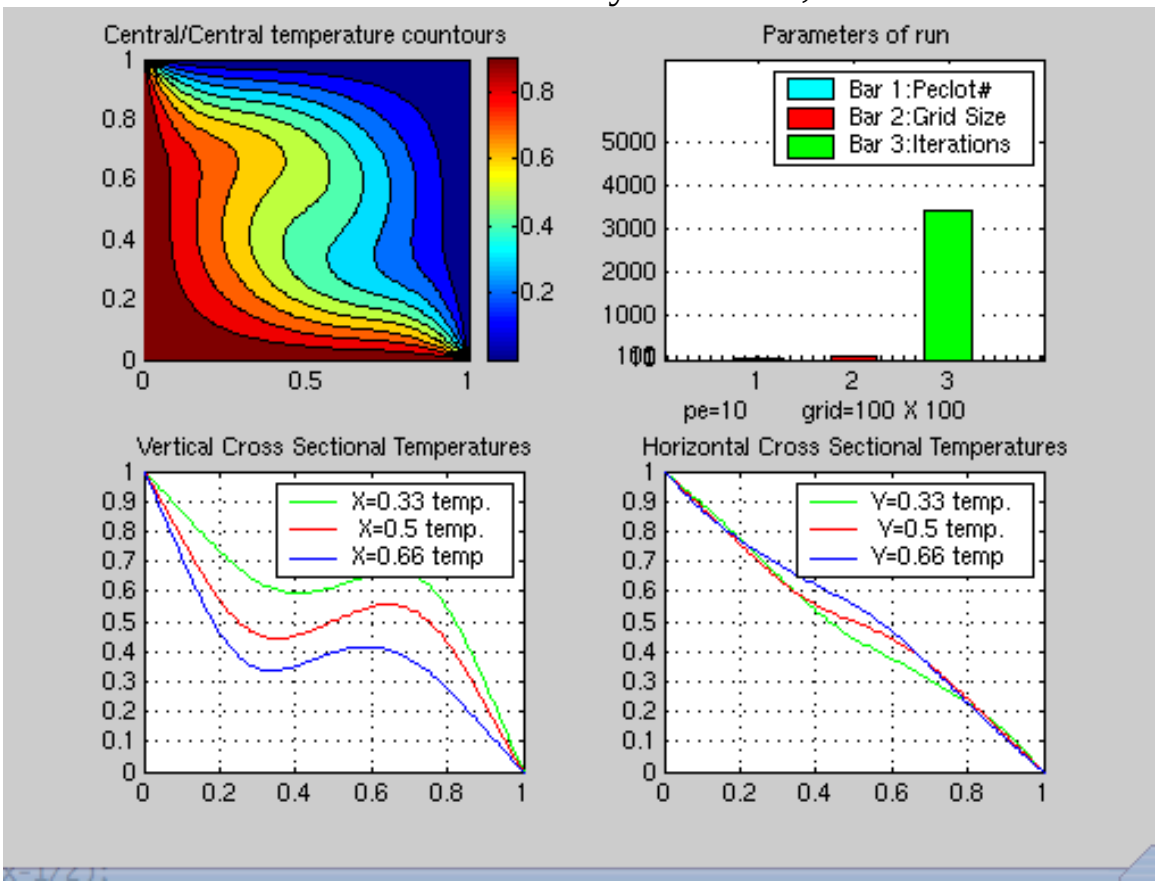

7. Upwind/Central Differencing Pe=10, Grid= 10x10




8. Upwind/Central Differencing Pe=10, Grid= 100x100



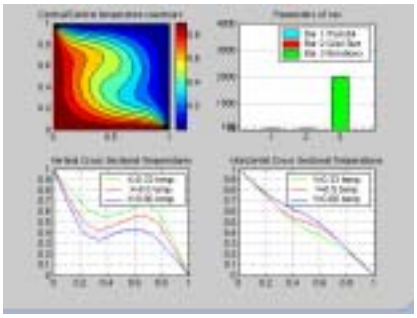

9. Central/Central Samarskii/Andreyev  Pe=10, Grid=10




10. Central/Central Samarskii/Andreyev Pe=10, Grid=100

# Part D: Analysis of Results

For all three algorithms (central/central explicit, upwind/central explicit, and central/central Samarskii/Andreyev method) using a grid size of 100 x 100 (delta x=0.01), we get identical results (charts 2, 6 for Pe=1, and charts 4, 6, 10 for Pe=10). Therefore we can consider the results using the finer grid the "correct solution" using our schemes. Other runs were made with varying grid and Péclet numbers, the "correct solution" appears at a grid size of around 25 x 25.



Pe=10, Grid= 25x25, using Samarskii/Andreyev Central/Central differencing solution scheme.

Before looking at the specific data related to the different schemes and grids, it is worth making an assessment of the schemes in general. Numerical solutions are dependent on a adequately fine mesh. In cases of a coarse mesh, numerical solutions can vary from the exact solution related to the following considerations.

**Conservativeness**

Both the central and the upwind schemes use consistent differencing expressions to evaluate the convective and conductive fluxes through the control volumes, so the results for the different schemes will be conservative in terms of overall distribution of energy flow, which in this case will be indicated by the temperature gradients. Even with a grid of 2x2, both the central and upwind schemes produced conservative results in which the cross sectional temperatures graphed diagonally from one corner of the graph to the other, indicating a evenly distributed non-dimensional temperature.

**Boundedness**

Boundedness refers to the ability of a differencing scheme to converge rather than diverge, and requires that the sum of the coefficients of nodes surrounding a central node divided by the coefficient of a central node (minus source terms) be less than one. In general cases using a central differencing scheme, when Pe>2, this can be violated. However, since we are considering an incompressible fluid, we have more leeway in this regard since the advection terms are only affected by the velocity of the fluid which in our case is symmetric in both the x and y directions. The upwind solution is always bounded because of the way the discretisation is defined, because the difference in temperature is always defined in terms of the direction of the flow field.

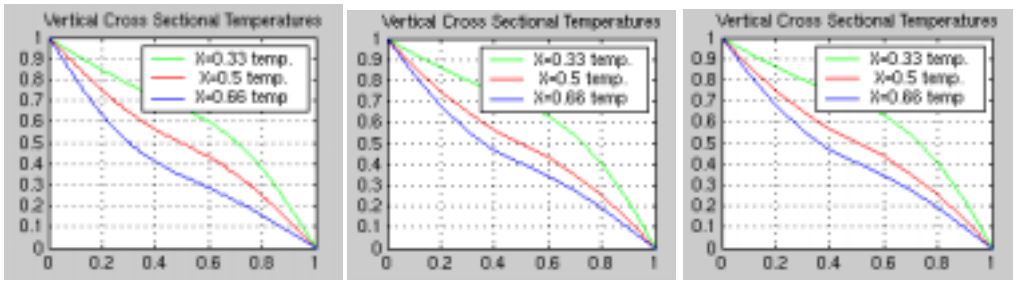
**Transportiveness**

Transportiveness indicates that a property flux will be transported downstream, and in discretisation schemes the relationship between the magnitude of the Péclet number and the directionality of the flow variables be maintained. Since the central differencing scheme calculates independently of flow direction, at higher Pe values, the transportiveness of the solution is not guaranteed. The upwind solution, on the other hand, takes into account flow direction, so transportiveness is assured.

# Accuracy and Grid Refinement

The interesting analysis lies in looking at the coarser 10 x 10 grid (delta x = 0.1), and how the solutions compare with the "correct" solution. For 10 x 10 grids:

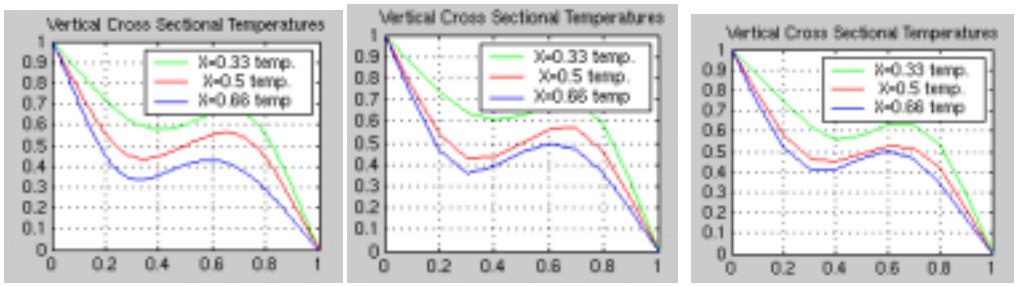
**Péclet Number =1**




"correct solution" Pe=1 (grid=100) | Central/Central Pe =1 (grid=10) | Upwind/Central Pe=1 (grid=10)

With a 10 x10 grid we see that both the central/central and the upwind/central differencing schemes vary from the correct solution by similar amounts. In the analysis of the actual numbers in the temperature matrix (see final Appendix), we find the average difference in temperature values between the central and upwind 10 x 10 schemes is 0.000526, not statistically significant, while the average difference between the central solutions using a grid of 10 and the grid of 100, is 0.00190, or 0.19% , which we can consider as the error in using the coarser grid. In the graphs above, we see that the error results in a shifting of the x=0.33 and x=0.66 temperature gradients up while the x=0.50 center line remains accurate. Because this happens in both upwind and central, we can assume the upward shift is a error resulting from the coarse grid.

**Péclet number=10**




"correct solution" Pe=10 (grid=100) | Central/Central Pe =10 (grid=10) | Upwind/Central Pe=10 (grid=10)

For Pe=10, things get more interesting. Again, we see a shifting of the x=0.33 and x=0.66 cross sectional temperatures, due to the error from the coarse grid, while again, the x=0.50 cross sectional temperature is accurate in the central 10 x 10 grid differencing solution. But in the upwind solution, all of the cross-sectional ( x=0.33, x=0.50, x= 0.66) temperatures are shifted upward, indicating variation that involves more than the standard error of a coarse grid. The shift, in fact, is influenced by the choice of initial values for the temperature array, in the above cases were set to 0 for all values (except for x=0,y=0 where boundary conditions t=1). The finer 100 x 100 grid, on the other hand, is independent of initial values of the temperature array.

# False Diffusion

Upwind solution schemes suffer from what is known as “false diffusion”, in whch the value of the diffusion constant is essentially increased due to the differencing scheme. In our case:

$$\left(\frac{\partial u\theta}{dx}+\frac{\partial v\theta}{dy}\right)=\frac{1}{Pe}\left(\frac{\partial^2\theta}{dx^2}+\frac{\partial^2\theta}{dy^2}\right)$$

Using difference operators with an upwind scheme results in:

$$\frac{u}{\Delta x}\nabla_x T+\frac{v}{\Delta y}\nabla_y T=\frac{1}{\Delta x^2 Pe}\left(\delta_x^2 T\right)+\frac{1}{\Delta y^2 Pe}\left(\delta_y^2 T\right)$$

Using Taylor and Power series expansions:

$$\frac{u}{\Delta x}\left(\Delta xDT-\frac{\Delta x^2 D^2 T}{2!}+\ldots\right)+\frac{v}{\Delta y}\left(\Delta yDT-\frac{\Delta y^2 D^2 T}{2!}+\ldots\right)=$$
$$\frac{1}{\Delta x^2 Pe}\left(\frac{2\Delta x^2 D^2 T}{2!}+\ldots\right)+\frac{1}{\Delta y^2 Pe}\left(\frac{2\Delta y^2 D^2 T}{2!}+\ldots\right)$$

Which equivalent to:

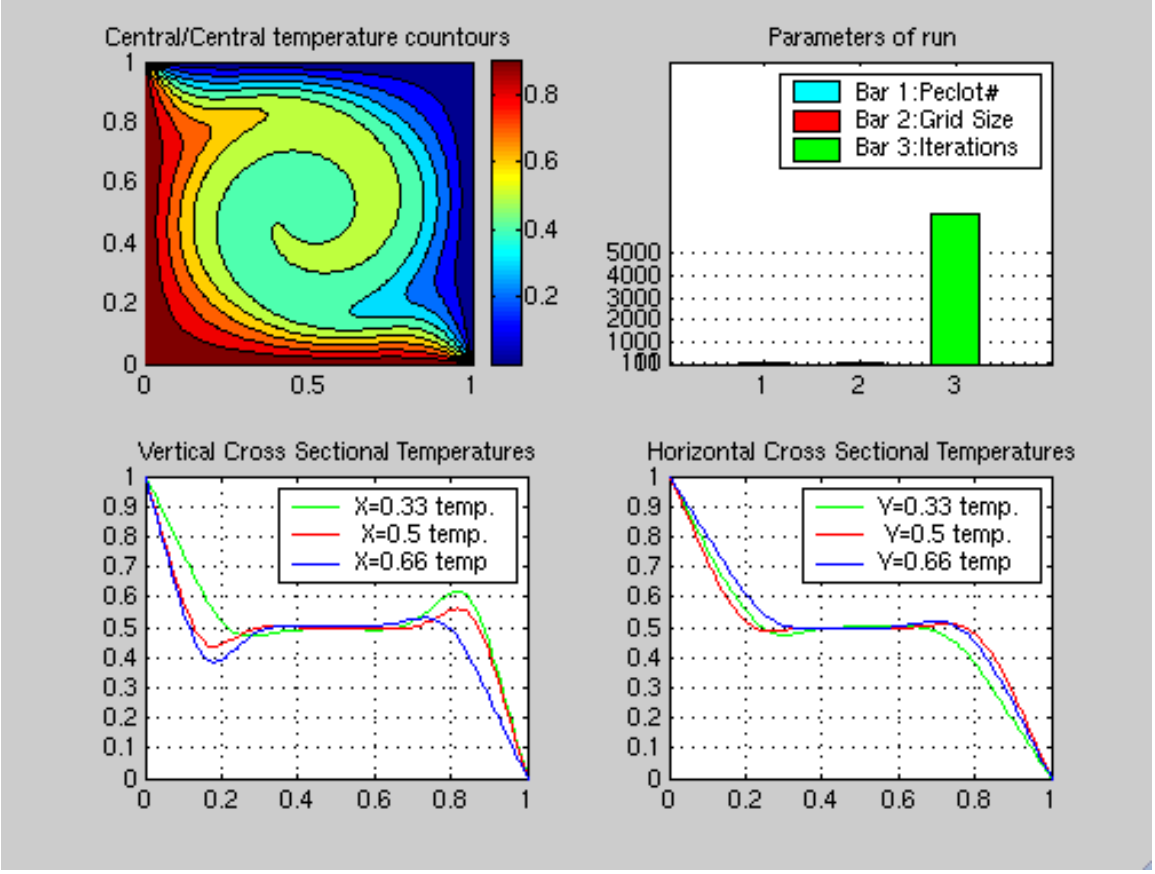
$$\left(\frac{\partial u\theta}{dx}+\frac{\partial v\theta}{dy}\right)=\left(\frac{1}{Pe}+\frac{u\Delta x}{2}\right)\left(\frac{\partial^2\theta}{dx^2}\right)+\left(\frac{1}{Pe}+\frac{v\Delta y}{2}\right)\left(\frac{\partial^2\theta}{dy^2}\right)+(error)$$

So we can see that the effective Péclet number is essentially reduced when $\frac{u\Delta x}{2}$ and $\frac{v\Delta y}{2}$ approach the value of $\frac{1}{Pe}$. In our case, with our equations, u and v may approach the value of 2 in places, therefore the value of effective value of Pe (which influences the diffusion terms) may be halved, since $\frac{u\Delta x}{2}\approx\frac{2(0.1)}{2}$ is roughly equal to $\frac{1}{Pe}$ when Pe=10.

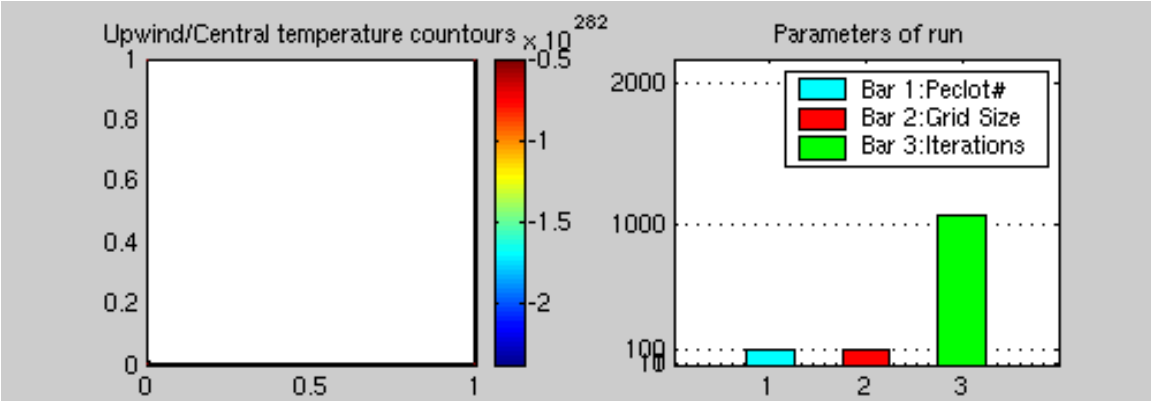
In our results, we can see that the error due to upwind differencing scheme with a 10 x 10 grid are greatest where the velocity gradient is the largest, for example at a point around 0.4,0.4. From our grid values listed in the final appendix, at such a point, the central gives an error from the correct solution of 2.57%, while the upwind error is 7.05%.

For finer grids, the upwind difference gives accurate results, as the ratio of the 1/Pe to the grid distance between nodes is small.

# Part E: Using Péclet Number of 100




Central Differencing, Pe=100 Grid = 100 x 100




Upwind Differencing, Pe=100 Grid = 100 x 100

Results and discussion: With grids of 10 X 10, and a Pe of 100 both the central and upwind schemes are unbounded and the solution goes to infinity. With a grid of 100 x 100, the central solution solves, but for upwind the solution goes to zero after around 1000 iterations. In our code, we can see that high values for Pe would create a situation where the solution will become unstable. Physically, a high Pe refers to a high value of heat transfer from convection relative to the heat transfer from conduction, which would indicate the advection terms would be dominant and the diffusion terms would not contribute properly to the solution. Mathematically, Von Neumann stability analysis indicate that the solution is unstable when cell Péclet numbers are high. In the case of our grids and velocity gradients, when Pe=100, cell Péclet numbers are very high, and differencing conditions, including the Courant-Freidrichs-Levy Condition (CFL criteria), are exceeded. Therefore, the solutions with Pe=100 are of no use.

Appendix: Matlab Programs: Program 1
Central/central explicit scheme using a timestep.

```
function centralcentral %(explict method using delta t)

grid=10;        %gridsize either 10 or 100
  delta=1/grid;
pe=10;          %Peclot number either 1 or 10

%other important numbers
maxdeltaT  = (delta^2)/4*pe; %based on van Neumann Stability analysis
deltaT=0.99*maxdeltaT;
precision=0.000001; %for iteration loop

%initialize arrays t(temperature), u(x velocity), v(y velocity)
t  = 0*ones(grid+1,grid+1);  %allows for variable initialization
u  = zeros(grid+1,grid+1);
v  = zeros(grid+1,grid+1);
t(1:grid+1,1)=1; %initialize Tmax=1 values on boundaries
t(1,1:grid)=1;
t(1:grid+1,grid+1)=0; %initialize Tmin=0 values on boundaries
t(grid+1,1:grid)=0;

%calculate velocity field
        for m=1:(grid+1);
          for n=1:(grid+1);
            x=(m-1)/grid;
            y=(n-1)/grid;
                %velocity at all points inside grid
                if ((x-1/2)^2+(y-1/2)^2)>.25;
                    h=0;
                else h=15*(0.25-(x-1/2)^2-(y-1/2)^2);
                end;
                u(n,m)=-(0.25-h)*h*(y-1/2);
                v(n,m)=(0.25-h)*h*(x-1/2);
           end
        end
 %do a quick Van Neumann Stability check
maxvelocity=0;
for i=2:(grid);
    for j=2:(grid);
        maxu=u(i,j);
        maxv=v(i,j);
        if maxu>maxvelocity;
           maxvelocity=maxu
        end
        if maxv>maxvelocity
           maxvelocity=maxv;
        end
    end
end
if maxvelocity>(2*pe)/delta
    error('warning, unstable solution')
end

maxdiff=2;counter=0;%beginning of an algorithm for convergence testing.
while maxdiff>precision;
    counter=counter+1;
    told=t;% "told" is the old temperature array, before the core
%calculation below.
```

```
    %the core of the program, calculating temperature values
    for i=2:(grid);
        for j=2:(grid);
            %constants a,b,c
            a=v(i,j)*deltaT/(2*delta);
            b=u(i,j)*deltaT/(2*delta);
            c=deltaT/(pe*(delta^2));
          t(i,j)=(1-(4*c))*t(i,j)+(c-a)*t(i+1,j)+(c+a)*t(i-1,j)+...
            (c-b)*t(i,j+1)+(c+b)*t(i,j-1);
        end
    end %ends the core of the program (explicit method).

    %calculate the biggest difference between the old temp array and
%the new one.
    maxdiff=0;
    arraydiff=t-told;
        for i=2:(grid);
            for j=2:(grid);
                diff=arraydiff(i,j);
                if diff>maxdiff;
                    maxdiff=diff;
                end
            end
        end
end %end the while statement.
t
%Create plots of all data  (SEE COMMON PLOTTING ALGORITHMS)
```

Appendix: Matlab Programs: Program 2
Upwind/Central explicit scheme.

```
function upwindcentral  %using an explicit method

%gridsize either 10 or 100
grid=10;    delta=1/grid;
%Peclot number either 1 or 10
pe=10;

 %other important numbers
maxdeltaT  = (delta^2)/4*pe; %based on van Neumann Stability analysis
deltaT=.99*maxdeltaT;
precision=0.00001; %iteration constant

%initialize arrays t(temperature), u(x velocity), v(y velocity)
t  = zeros(grid+1,grid+1);
u  = zeros(grid+1,grid+1);
v  = zeros(grid+1,grid+1);
t(1:grid+1,1)=1; %initialize Tmax=1 values on boundaries
t(1,1:grid+0)=1;

%calculate velocity field
        for m=1:(grid+1);
          for n=1:(grid+1);
            x=(m-1)/grid;
            y=(n-1)/grid;
                %velocity at all points inside grid
                if ((x-1/2)^2+(y-1/2)^2)>.25;
                    h=0;
                else h=15*(0.25-(x-1/2)^2-(y-1/2)^2);
                end;
                u(n,m)=-(0.25-h)*h*(y-1/2);
                v(n,m)=(0.25-h)*h*(x-1/2);
           end
        end

maxdiff=2;counter=0;%beginning of an algorithm for convergence testing.
while maxdiff>precision;
    counter=counter+1;
    told=t; % "told" is the old temperature array, before the iteration
%below.

    %the core of the program, calculating temperature values
    for i=2:(grid);
        for j=2:(grid);

             %constants a,b,c
            a=v(i,j)/(delta); %v affects the vertical temps
            b=u(i,j)/(delta); %u affects the horizontal temps
            c=1/(pe*(delta^2));
            %constants p,q,r,s for t(i+1),t(i-1),t(j+1),t(j-1)
            if u(i,j)<=0 && v(i,j)<=0;
                d=(4*c)-a-b;
                p=(c-a)/d;
                q=c/d;
                r=(c-b)/d;
                s=c/d;
            elseif u(i,j)>=0 && v(i,j)<=0;
                d=(4*c)+a-b;
                p=c/d;
```

```
                q=(c+a)/d;
                r=(c-b)/d;
                s=c/d;
            elseif u(i,j)<=0 && v(i,j)>=0;
                d=(4*c)-a+b;
                p=(c-a)/d;
                q=c/d;
                r=c/d;
                s=(c+b)/d;
            elseif u(i,j)>=0 && v(i,j)>=0;
                d=(4*c)+a+b;
                p=c/d;
                q=(c+a)/d;
                r=c/d;
                s=(c+b)/d;
            else error('error')
            end
          t(i,j)=p*t(i+1,j)+q*t(i-1,j)+r*t(i,j+1)+s*t(i,j-1);
        end
    end %ends the core of the program.

    %calculate the biggest difference between the old temp array and
%the new one.
    maxdiff=0;
    arraydiff=t-told;
        for i=2:(grid);
            for j=2:(grid);
                diff=arraydiff(i,j);
                if diff>maxdiff;
                    maxdiff=diff;
                end
            end
        end

end %end the while statement.

t


%Create plots of all data  (SEE COMMON PLOTTING ALGORITHMS)
```

## Appendix: Matlab Programs: Program 3
## Central/Central implicit/explicit Samarskii/Andreyev Scheme

```
function centralcentral % Using the Samarskii/Andreyev
%Implicit/Explicit Method

grid=10;        %gridsize either 10 or 100
  delta=1/grid;
pe=10;          %Peclet number either 1 or 10

%other important numbers
deltaT=.001;
precision=0.00001; %for iteration loop
sigma=0.5;

%initialize arrays t(temperature), u(x velocity), v(y velocity)
t  = .5*ones(grid+1,grid+1);
u  = zeros(grid+1,grid+1);
v  = zeros(grid+1,grid+1);
t(1:grid+1,1)=1; %initialize Tmax=1 values on boundaries
t(1,1:grid)=1;
t(1:grid+1,grid+1)=0; %initialize Tmin=0 values on boundaries
t(grid+1,1:grid)=0;

%calculate velocity field
        for m=1:(grid+1);
          for n=1:(grid+1);
            x=(m-1)/grid;
            y=(n-1)/grid;
                %velocity at all points inside grid
                if ((x-1/2)^2+(y-1/2)^2)>.25;
                    h=0;
                else h=15*(0.25-(x-1/2)^2-(y-1/2)^2);
                end;
                u(n,m)=-(0.25-h)*h*(y-1/2);
                v(n,m)=(0.25-h)*h*(x-1/2);
           end
        end
%beginning of the Samarskii/Andreyev central/central implicit method.
maxdiff=2;omegaValue=2;counter=0;
while abs(omegaValue)>precision;
    counter=counter+1;
   told=t; %the old temperature, for comparison calculation

    %the core of the program, calculating temperature values using the
    %Samarskii/Andreyev Method
    b=1-((2*sigma*deltaT)/(pe*(delta^2))); %same value throughout
%calculation

    %calculate d for all points,a grid-1 by grid-1 matrix (9X9 for
%grid=10).
     for i=2:grid
        for j= 2:grid
            d(i-1,j-1)=(-1*v(i,j)/(2*delta))*(t(i+1,j)-t(i-1,j)) ...
            - (u(i,j)/(2*delta))*(t(i,j+1)-t(i,j-1))...
+(1/(pe*(delta^2)))*(t(i-1,j)+t(i+1,j)+t(i,j+1)+t(i,j-1)-(4*t(i,j)));
%note changes in sign and directions are due to the way the array is
stored
        end
     end
```

```
    %create "Matrix X" which will consist of all a,b,c, values for
%omega
   X=zeros(grid-1,grid-1);
   for j=2:grid
       for i=2:grid
           if i==j
           X(i-1,j-1)=b;
           elseif i==j+1
               velocity=v(i,j+1);
           X(i-1,j-1)=((velocity*sigma*deltaT)/(2*delta))...
- sigma*deltaT/(pe*(delta^2));
           elseif i==j-1
               velocity=v(i,j-1);
           X(i-1,j-1)=((velocity*sigma*deltaT)/(2*delta))...
-sigma*deltaT/(pe*(delta^2));
           end
       end
   end

omega=X\d;

%Now for second part of sweep in y direction.
 X2=zeros(grid-1,grid-1);
   for j=2:grid
       for i=2:grid
           if i==j
           X2(i-1,j-1)=b;
           elseif i==j+1
               velocity=u(i,j+1);
           X2(i-1,j-1)=((velocity*sigma*deltaT)/(2*delta))...
-sigma*deltaT/(pe*(delta^2));
           elseif i==j-1
               velocity=u(i,j-1);
           X2(i-1,j-1)=((velocity*sigma*deltaT)/(2*delta))...
-sigma*deltaT/(pe*(delta^2));
           end
       end
   end

omega2=X2\omega;

%update t
for i=2:grid
    for j=2:grid
        t(i,j)=(deltaT*omega2(i-1,j-1)) + t(i,j);
    end
end
```

```
% convergence Calcluations:
maxdiff=0;
    arraydiff=t-told;
        for i=2:(grid);
            for j=2:(grid);
                diff=arraydiff(i,j);
                if diff>maxdiff;
                    maxdiff=diff;
                end
            end
        end
  %calculate the largest value in the omega matrix
    maxOmega=.1;
        for m=1:(grid-1);
            for n=1:(grid-1);
                omegaValue=omega(m,n);
                if abs(omegaValue)>maxOmega;
                    omegaValue=abs(maxOmega);
                end
            end
        end
      if counter==5000
            %error('too many iterations')
            maxdiff=0;
            omegaValue=0;
        end
end %end the while statement.

t
%Create plots of all data  (SEE COMMON PLOTTING ALGORITHMS)
```

Appendix: Matlab Programs: Common Plotting Algorithm for all programs

```
%Create plots of all data
%Temperature contour Plot
[X,Y]=meshgrid(0:delta:1);
subplot(2,2,1),contourf(X,Y,t)
colorbar;
title('<name of scheme> temperature contours')
%Pe, Grid size, and iteration reference
subplot(2,2,2), bar([pe,0,0 ],.5,'-.c*')
hold on;
axis([0 4 0 2*counter+50]) ;
set(gca,'ytick',[1 10 100 1000 2000 3000 4000 5000]);
set(gca,'YGrid','on');
bar([0, grid,0 ],.5,'-.r*');
bar([0,0,counter],.5,'g');
axis([0 4 0 2*counter+50]) ;
legend('Bar 1:Peclot#','Bar 2:Grid Size','Bar 3:Iterations');
if pe==1 && grid==10;
    xlabel('pe = 1      grid=10x10        ');
elseif pe==10 && grid==10;
    xlabel('pe=10        grid=10X10       ');
elseif pe==1 && grid==100;
    xlabel('pe=1        grid=100 X 100       ');
elseif pe==10 && grid==100;
    xlabel('pe=10        grid=100 X 100       ');
end
title(' Parameters of run')
hold off;
end
%XCenterline temperature cross sections
tCx33=t(:,round(grid/3)+1);
tCx5=t(:,(grid/2)+1);
tCx66=t(:,(floor((2*grid)/3))+1);
Xaxis=[0:delta:1];
subplot(2,2,3),plot (Xaxis,tCx33,'g');hold on;
plot(Xaxis,tCx5,'r');
plot(Xaxis,tCx66,'b');
set(gca,'ytick',[0:.1:1]);
set(gca,'xtick',[0:.2:1]);
set(gca,'YGrid','on');
set(gca,'XGrid','on');hold off;
legend('X=0.33 temp.',' X=0.5 temp.', 'X=0.66 temp')
title(' Vertical Cross Sectional Temperatures')
tCy33=t(round(grid/3)+1,:);
tCy5=t((grid/2)+1,:);
tCy66=t((floor((2*grid)/3))+1,:);
Xaxis=[0:delta:1];
subplot(2,2,4),plot (Xaxis,tCy33,'g');hold on;
plot(Xaxis,tCy5,'r');
plot(Xaxis,tCy66,'b');
set(gca,'ytick',[0:.1:1]);
set(gca,'xtick',[0:.2:1]);
set(gca,'YGrid','on');
set(gca,'XGrid','on');hold off;
legend('Y=0.33 temp.',' Y=0.5 temp.', 'Y=0.66 temp')
title(' Horizontal Cross Sectional Temperatures')
```

# Appendix: Temperature Array Values for selected solutions.

upwind/central Pe=10, Grid =10

t =

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 1.0000 | 0.9582 | 0.9155 | 0.8719 | 0.8257 | 0.7807 | 0.7405 | 0.7046 | 0.6431 | 0.4752 | 0 |
| 1.0000 | 0.9175 | 0.8319 | 0.7432 | 0.6513 | 0.5805 | 0.5185 | 0.4622 | 0.3926 | 0.2575 | 0 |
| 1.0000 | 0.8798 | 0.7531 | 0.6273 | 0.5216 | 0.4676 | 0.4139 | 0.3524 | 0.2720 | 0.1624 | 0 |
| 1.0000 | 0.8538 | 0.6983 | 0.5603 | 0.4814 | 0.4526 | 0.4166 | 0.3540 | 0.2556 | 0.1363 | 0 |
| 1.0000 | 0.8537 | 0.7053 | 0.5791 | 0.5128 | 0.4885 | 0.4633 | 0.3987 | 0.2804 | 0.1406 | 0 |
| 1.0000 | 0.8585 | 0.7329 | 0.6288 | 0.5637 | 0.5253 | 0.5023 | 0.4334 | 0.2986 | 0.1438 | 0 |
| 1.0000 | 0.8336 | 0.7197 | 0.6356 | 0.5724 | 0.5188 | 0.4682 | 0.3655 | 0.2426 | 0.1182 | 0 |
| 1.0000 | 0.7399 | 0.6023 | 0.5307 | 0.4742 | 0.4170 | 0.3448 | 0.2534 | 0.1656 | 0.0812 | 0 |
| 1.0000 | 0.5236 | 0.3545 | 0.2922 | 0.2565 | 0.2183 | 0.1728 | 0.1266 | 0.0833 | 0.0411 | 0 |
| 1.0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

central/central Pe = 10, Grid =10

t =

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 1.0000 | 0.9594 | 0.9161 | 0.8689 | 0.8157 | 0.7604 | 0.7153 | 0.6823 | 0.6279 | 0.4678 | 0 |
| 1.0000 | 0.9213 | 0.8362 | 0.7415 | 0.6376 | 0.5404 | 0.4668 | 0.4145 | 0.3614 | 0.2433 | 0 |
| 1.0000 | 0.8898 | 0.7701 | 0.6438 | 0.5227 | 0.4290 | 0.3615 | 0.2986 | 0.2315 | 0.1439 | 0 |
| 1.0000 | 0.8727 | 0.7381 | 0.6068 | 0.5018 | 0.4386 | 0.3906 | 0.3191 | 0.2231 | 0.1189 | 0 |
| 1.0000 | 0.8745 | 0.7467 | 0.6275 | 0.5435 | 0.5000 | 0.4565 | 0.3724 | 0.2533 | 0.1255 | 0 |
| 1.0000 | 0.8811 | 0.7769 | 0.6809 | 0.6094 | 0.5614 | 0.4982 | 0.3932 | 0.2619 | 0.1273 | 0 |
| 1.0000 | 0.8561 | 0.7685 | 0.7014 | 0.6385 | 0.5710 | 0.4772 | 0.3562 | 0.2299 | 0.1102 | 0 |
| 1.0000 | 0.7567 | 0.6386 | 0.5855 | 0.5332 | 0.4596 | 0.3624 | 0.2584 | 0.1638 | 0.0787 | 0 |
| 1.0000 | 0.5322 | 0.3721 | 0.3177 | 0.2847 | 0.2396 | 0.1843 | 0.1311 | 0.0839 | 0.0406 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

central/central Pe=10, Grid = 100 “correct solution” (only each tenth row and column displayed)

t10 =

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 1.0000 | 0.9603 | 0.9181 | 0.8715 | 0.8190 | 0.7652 | 0.7217 | 0.6896 | 0.6365 | 0.4685 | 0 |
| 1.0000 | 0.9232 | 0.8407 | 0.7491 | 0.6496 | 0.5567 | 0.4847 | 0.4314 | 0.3712 | 0.2390 | 0 |
| 1.0000 | 0.8925 | 0.7769 | 0.6548 | 0.5385 | 0.4477 | 0.3794 | 0.3149 | 0.2425 | 0.1440 | 0 |
| 1.0000 | 0.8751 | 0.7437 | 0.6152 | 0.5124 | 0.4479 | 0.3963 | 0.3243 | 0.2282 | 0.1200 | 0 |
| 1.0000 | 0.8748 | 0.7479 | 0.6306 | 0.5467 | 0.4990 | 0.4513 | 0.3677 | 0.2509 | 0.1245 | 0 |
| 1.0000 | 0.8794 | 0.7707 | 0.6741 | 0.6018 | 0.5501 | 0.4857 | 0.3832 | 0.2551 | 0.1243 | 0 |
| 1.0000 | 0.8555 | 0.7565 | 0.6838 | 0.6190 | 0.5507 | 0.4599 | 0.3439 | 0.2221 | 0.1070 | 0 |
| 1.0000 | 0.7607 | 0.6281 | 0.5676 | 0.5141 | 0.4420 | 0.3492 | 0.2499 | 0.1586 | 0.0764 | 0 |
| 1.0000 | 0.5313 | 0.3631 | 0.3099 | 0.2777 | 0.2341 | 0.1804 | 0.1280 | 0.0816 | 0.0395 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# Appendix: Temperature Array Values for selected solutions.

upwind/central Pe=1, Grid=10

t =

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 1.0000 | 0.9774 | 0.9543 | 0.9296 | 0.9018 | 0.8693 | 0.8302 | 0.7773 | 0.6874 | 0.4947 | 0 |
| 1.0000 | 0.9555 | 0.9101 | 0.8620 | 0.8086 | 0.7477 | 0.6782 | 0.5942 | 0.4776 | 0.2915 | 0 |
| 1.0000 | 0.9345 | 0.8686 | 0.8004 | 0.7265 | 0.6453 | 0.5577 | 0.4609 | 0.3452 | 0.1939 | 0 |
| 1.0000 | 0.9140 | 0.8301 | 0.7460 | 0.6582 | 0.5654 | 0.4696 | 0.3706 | 0.2638 | 0.1410 | 0 |
| 1.0000 | 0.8913 | 0.7901 | 0.6939 | 0.5978 | 0.5002 | 0.4030 | 0.3074 | 0.2112 | 0.1093 | 0 |
| 1.0000 | 0.8594 | 0.7368 | 0.6300 | 0.5308 | 0.4346 | 0.3421 | 0.2544 | 0.1704 | 0.0863 | 0 |
| 1.0000 | 0.8063 | 0.6551 | 0.5394 | 0.4423 | 0.3541 | 0.2734 | 0.1997 | 0.1315 | 0.0656 | 0 |
| 1.0000 | 0.7085 | 0.5225 | 0.4058 | 0.3216 | 0.2516 | 0.1911 | 0.1379 | 0.0900 | 0.0445 | 0 |
| 1.0000 | 0.5053 | 0.3126 | 0.2227 | 0.1697 | 0.1303 | 0.0980 | 0.0704 | 0.0457 | 0.0226 | 0 |
| 1.0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

central/central Pe=1, Grid=10

t =

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 1.0000 | 0.9774 | 0.9542 | 0.9295 | 0.9017 | 0.8693 | 0.8300 | 0.7769 | 0.6870 | 0.4945 | 0 |
| 1.0000 | 0.9555 | 0.9100 | 0.8619 | 0.8084 | 0.7475 | 0.6777 | 0.5933 | 0.4767 | 0.2911 | 0 |
| 1.0000 | 0.9345 | 0.8686 | 0.8003 | 0.7263 | 0.6449 | 0.5564 | 0.4590 | 0.3436 | 0.1931 | 0 |
| 1.0000 | 0.9140 | 0.8301 | 0.7460 | 0.6580 | 0.5649 | 0.4681 | 0.3685 | 0.2619 | 0.1400 | 0 |
| 1.0000 | 0.8914 | 0.7902 | 0.6940 | 0.5979 | 0.5000 | 0.4021 | 0.3060 | 0.2098 | 0.1086 | 0 |
| 1.0000 | 0.8600 | 0.7381 | 0.6315 | 0.5318 | 0.4351 | 0.3419 | 0.2539 | 0.1699 | 0.0860 | 0 |
| 1.0000 | 0.8069 | 0.6564 | 0.5410 | 0.4435 | 0.3551 | 0.2737 | 0.1997 | 0.1314 | 0.0655 | 0 |
| 1.0000 | 0.7089 | 0.5233 | 0.4067 | 0.3223 | 0.2525 | 0.1916 | 0.1381 | 0.0899 | 0.0445 | 0 |
| 1.0000 | 0.5055 | 0.3130 | 0.2231 | 0.1700 | 0.1307 | 0.0983 | 0.0705 | 0.0458 | 0.0226 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

central/central Pe=1, grid =100 “correct solution” (only each tenth row and column displayed)

t10 =

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 1.0000 | 0.9776 | 0.9545 | 0.9299 | 0.9024 | 0.8707 | 0.8326 | 0.7815 | 0.6940 | 0.4944 | 0 |
| 1.0000 | 0.9558 | 0.9106 | 0.8627 | 0.8097 | 0.7497 | 0.6808 | 0.5966 | 0.4772 | 0.2839 | 0 |
| 1.0000 | 0.9350 | 0.8694 | 0.8012 | 0.7275 | 0.6467 | 0.5586 | 0.4604 | 0.3422 | 0.1886 | 0 |
| 1.0000 | 0.9147 | 0.8308 | 0.7465 | 0.6584 | 0.5655 | 0.4688 | 0.3685 | 0.2604 | 0.1376 | 0 |
| 1.0000 | 0.8925 | 0.7907 | 0.6935 | 0.5969 | 0.4990 | 0.4012 | 0.3048 | 0.2081 | 0.1069 | 0 |
| 1.0000 | 0.8618 | 0.7385 | 0.6300 | 0.5293 | 0.4325 | 0.3397 | 0.2520 | 0.1680 | 0.0847 | 0 |
| 1.0000 | 0.8109 | 0.6568 | 0.5382 | 0.4398 | 0.3516 | 0.2709 | 0.1974 | 0.1296 | 0.0645 | 0 |
| 1.0000 | 0.7157 | 0.5221 | 0.4024 | 0.3181 | 0.2491 | 0.1892 | 0.1363 | 0.0887 | 0.0438 | 0 |
| 1.0000 | 0.5054 | 0.3056 | 0.2180 | 0.1668 | 0.1287 | 0.0969 | 0.0695 | 0.0451 | 0.0222 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |