MA 322: Endsem Assignment

Due on Sunday, November 22, 2015

Jiten Chandra Kalita

Silvi Pandey (130123045) — Rishi Kumar (130123032) — K Sharath Rushi (130123019)

PROBLEM 1

A thin rectangular homogeneous thermally conducting plate lies in the xy-plane defined by $0 \leq x \leq 4, 0 \leq y \leq 1$. The edge $y = 0$ is maintained at temperature $200x(x-4)$, while the remaining edges are held at 0^0. The other faces are insulated and no sources and sink are present.
Solve numerically for $\Delta x = 0.1, 0.05, 0.025$ with three different grid aspect ratios given by $\beta = \frac{\Delta x}{\Delta y} = 0.5, 1, 1.5$ using Gauss - Seidel Method.
(1) Compare graphically the analytical solution along the vertical centerline i.e $x = 2$ with the numerical ones(one for each β.
(2) Perform further experiment on the Gauss-Seidel iterative solver by using SOR and comment upon the optimum ω for each of the grids.

SOLUTION

PART-1
Finite Difference Equation
The PDE governing the problem is given by

$$T_{xx} + T_{yy} = 0$$

Replacing T_{xx} and T_{yy} by the second-order centered difference approximations at grid point (i,j) yields the following FDE

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0$$

.The truncation error is second order in both Δx and Δy.

$$T_{i,j} = \frac{T_{i+1,j} + \beta^2 T_{i,j+1} + T_{i-1,j} + \beta^2 T_{i,j-1}}{2(1+\beta^2)}$$

where $\beta = \frac{\Delta x}{\Delta y}$
In Gauss-Seidel, first sweep along a row and then go a row above,the equation for the k^{th} iteration is given by

$$T^k_{i,j} = \frac{T^{k-1}_{i+1,j} + \beta^2 T^{k-1}_{i,j+1} + T^k_{i-1,j} + \beta^2 T^k_{i,j-1}}{2(1+\beta^2)}$$

The following are plotted by keeping x fixed at the center value i.e $x = 2$ for $\beta = 0.5, 1, 1.5$ along with the exact value of the temperature at those points.

Code

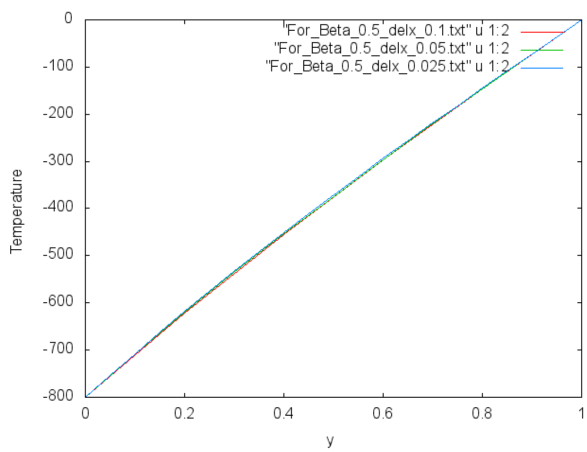
```
#include<bits/stdc++.h>

//beta value 0.5 , 1 , 1.5
using namespace std;
int xmin = 0;
int xmax = 4;
int ymin = 0;
int ymax = 1;
int xrange = xmax - xmin;
int yrange = ymax - ymin;

double boundFunction(double x)
```

```
{
    return 200*x*(x-4);
}
int main()
{
    FILE *fp = fopen("For_Beta_0.5_delx_0.1.txt","w");
    double delx,dely,beta;
    cout<<"Enter delx and beta" ;
    cin>>delx>>beta;
    dely = delx/beta;
    int xPoints = 1 + floor(xrange/delx);
    int yPoints = 1 + floor(yrange/dely);
    double temperature[xPoints][yPoints],temperatureFinal[xPoints][yPoints];
    for(int i=0;i<xPoints;i++)
    {
        temperature[i][0] = boundFunction(i*delx);
        temperature[i][yPoints-1] = 0;
        temperatureFinal[i][0] = boundFunction(i*delx);
        temperatureFinal[i][yPoints-1] = 0;
    }
    for(int i=0;i<yPoints;i++)
    {
        temperature[0][i] = temperature[xPoints-1][i] = 0;
        temperatureFinal[0][i] = temperatureFinal[xPoints-1][i] = 0;
     }
     for(int i=1;i<xPoints-1;i++)
     {
         for(int j=1;j<yPoints-1;j++)
            temperature[i][j] = 0;
     }

     //Gauss Seidel

     long long int count = 0;
     long long int MaxIter = 0;
     double w = 1.5;
    while(1)
    {
        count = 0;
        for(int i=1;i<xPoints-1;i++)
        {
            for(int j=1;j<yPoints-1;j++)
            {
                temperatureFinal[i][j] = (temperature[i+1][j] +
                                          beta*beta*temperature[i][j+1]
                                           + temperatureFinal[i-1][j] +
                                           beta*beta*temperatureFinal[i][j-1])
                                           /(2*(1+beta*beta));
            }
        }
        for(int i=1;i<xPoints-1;i++)
        {
            for(int j=1;j<yPoints-1;j++)
```

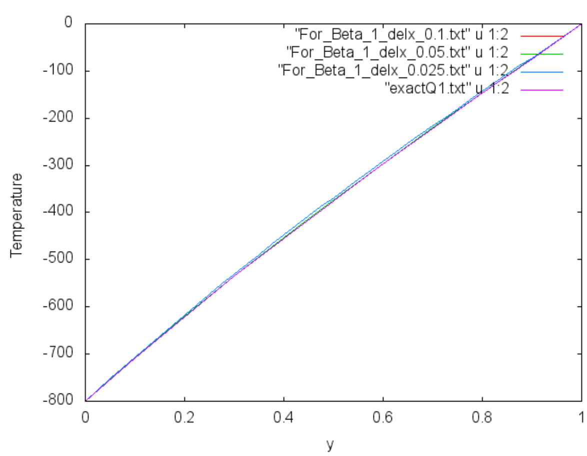
```
            {
                if(fabs(temperature[i][j]-temperatureFinal[i][j])
                   /fabs(temperature[i][j]) <= 0.0001)
                    count++;
                temperature[i][j] = temperatureFinal[i][j];
            }
        }
        MaxIter++;
        if(count == (xPoints-2)*(yPoints-2))
            break;
    }
    for(int i=0;i<yPoints;i++)
        fprintf(fp,"%lf %lf\n",i*dely,temperature[xPoints/2][i]);
    cout<<MaxIter;
}
```

Results

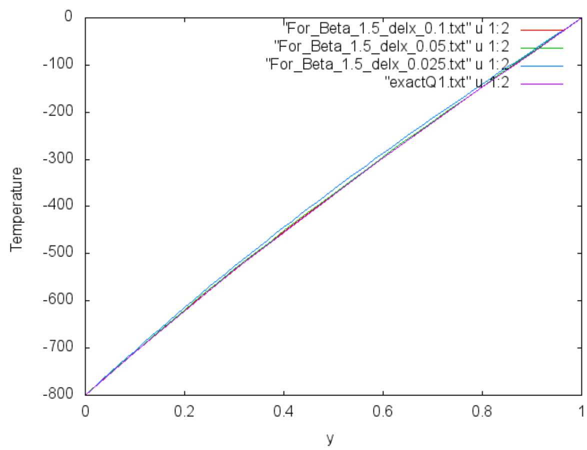
(1) $\beta = 0.5$



(2) $\beta = 1$



(3) $\beta = 1.5$



Observations & Explanation

(1) The plots are similar for all values of β.

(2) The graphs are plotted by keeping the value of x fixed at 2 and the temperature turns out to be linear function of y i.e $T(x, y) = ay + b$ for a given value of x and it is consistent with the PDE (a and b are arbitrary)

(3) Gauss-Seidel shows convergence but at a lower rate.The rate is improved by using SOR(in Part-2 of this problem)

(4) The stopping condition used in the code is for all points i.e for all i and j

```
if(fabs(temperature[i][j]-temperatureFinal[i][j])/fabs(temperature[i][j]) <= 0.0001)
    count++;
temperature[i][j] = temperatureFinal[i][j];
```

PART-2

Finite Difference Equation

According to kahan Theorem ,for $\omega \geq 0$, $\omega \leq 2$ there is convergence in the Gauss-Seidel iteration using SOR.To see the variation of the number of iterations with ω , ω is plotted against the number of iterations using that particular value for each β.In all the cases,a 'knee' value is obtained which is the optimal value of ω with minimum number of iterations

Gauss-Seidel using SOR :

$$T^k_{i,j} = T^{k-1}_{i,j} + \omega \frac{(T^{k-1}_{i+1,j} + \beta^2 T^{k-1}_{i,j+1} + T^k_{i-1,j} + \beta^2 T^k_{i,j-1} - 2(1+\beta^2)T^{k-1}_{i,j})}{2(1+\beta^2)}$$

Code

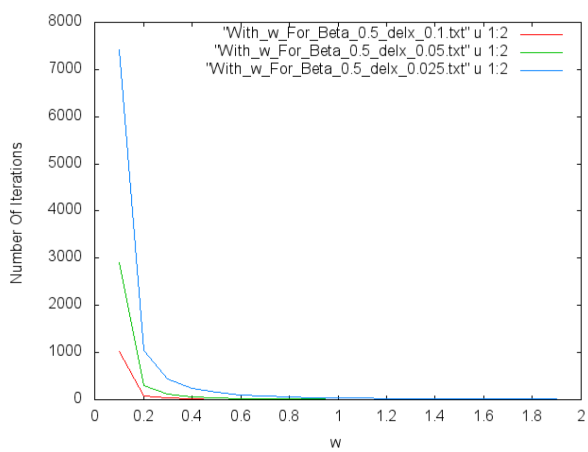
```
#include<bits/stdc++.h>

//beta value 0.5 , 1 , 1.5
using namespace std;
int xmin = 0;
int xmax = 4;
int ymin = 0;
int ymax = 1;
int xrange = xmax - xmin;
int yrange = ymax - ymin;
```

```
double boundFunction(double x)
{
    return 200*x*(x-4);
}
int main()
{
    FILE *fp = fopen("With_w_For_Beta_1.5_delx_0.025.txt","w");
    double delx,dely,beta;
    cout<<"Enter delx and beta" ;
    cin>>delx>>beta;
    dely = delx/beta;
    int xPoints = 1 + floor(xrange/delx);
    int yPoints = 1 + floor(yrange/dely);
    double temperature[xPoints][yPoints],temperatureFinal[xPoints][yPoints];
    for(int i=0;i<xPoints;i++)
    {
        temperature[i][0] = boundFunction(i*delx);
        temperature[i][yPoints-1] = 0;
        temperatureFinal[i][0] = boundFunction(i*delx);
        temperatureFinal[i][yPoints-1] = 0;
    }
    for(int i=0;i<yPoints;i++)
    {
        temperature[0][i] = temperature[xPoints-1][i] = 0;
        temperatureFinal[0][i] = temperatureFinal[xPoints-1][i] = 0;
     }
     for(int i=1;i<xPoints-1;i++)
     {
         for(int j=1;j<yPoints-1;j++)
            temperature[i][j] = 0;
     }

     //Gauss Seidel

     long long int count = 0;
     long long int MaxIter = 0;
     long long MinIter = LLONG_MAX;
     double wOpt;
     double w ;

    for(w=0.1;w<2;w+=0.1)
    {
        MaxIter = 0;
        while(1)
        {
            count = 0;
            for(int i=1;i<xPoints-1;i++)
            {
                for(int j=1;j<yPoints-1;j++)
                {
                    temperatureFinal[i][j] = temperature[i][j] +
                    w*(temperature[i+1][j] + beta*beta*temperature[i][j+1]
                        + temperatureFinal[i-1][j] + beta*beta*temperatureFinal[i][j-1]
```

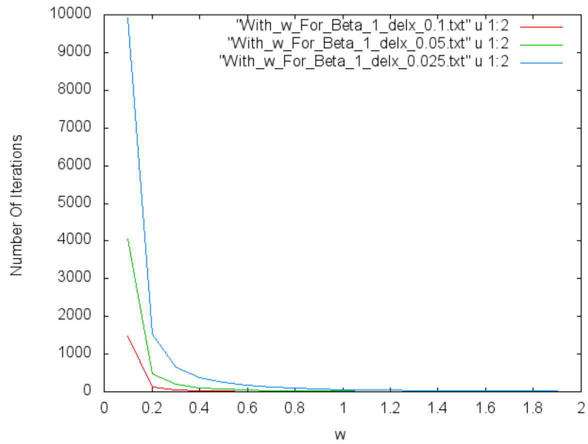
```
                      - temperature[i][j]*(2*(1+beta*beta)))/(2*(1+beta*beta));
                }
            }
            for(int i=1;i<xPoints-1;i++)
            {
                for(int j=1;j<yPoints-1;j++)
                {
                    if(fabs(temperature[i][j]-temperatureFinal[i][j])/
                       fabs(temperature[i][j]) <= 0.0001)
                        count++;
                    temperature[i][j] = temperatureFinal[i][j];
                }
            }
            MaxIter++;
            if(count == (xPoints-2)*(yPoints-2))
                break;
        }
        for(int i=1;i<xPoints-1;i++)
        {
            for(int j=1;j<yPoints-1;j++)
                temperature[i][j] = 0;
        }
        if(MaxIter < MinIter)
        {
             MinIter = MaxIter;
             wOpt = w;
        }
        fprintf(fp,"%lf %lld\n",w,MaxIter);
    }
    cout<<wOpt<<" "<<MinIter<<endl;
}
```

Results

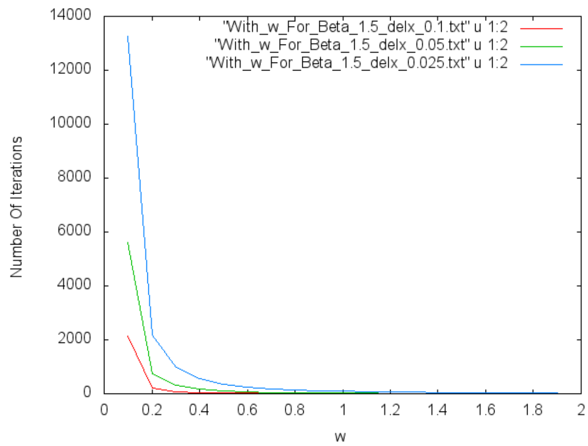
(1) $\beta = 0.5$



(2) $\beta = 1$



(3) $\beta = 1.5$



Observations

(1) Number of iterations without using SOR

$\beta = 0.5$ $\Delta x = 0.1$ Iterations : 29
$\beta = 0.5$ $\Delta x = 0.05$ Iterations : 298
$\beta = 0.5$ $\Delta x = 0.025$ Iterations : 905

$\beta = 1.0$ $\Delta x = 0.1$ Iterations : 136
$\beta = 1.0$ $\Delta x = 0.05$ Iterations : 429
$\beta = 1.0$ $\Delta x = 0.025$ Iterations : 1288

$\beta = 1.5$ $\Delta x = 0.1$ Iterations : 201
$\beta = 1.5$ $\Delta x = 0.05$ Iterations : 629
$\beta = 1.5$ $\Delta x = 0.025$ Iterations : 1855

(2) Minimum number of iterations using SOR and the corresponding optimal ω

$\beta = 0.5$ $\Delta x = 0.1$ ω : 1.6 Iterations : 29
$\beta = 0.5$ $\Delta x = 0.05$ ω : 1.8 Iterations : 61
$\beta = 0.5$ $\Delta x = 0.025$ ω : 1.9 Iterations : 136

$\beta = 1.0 \quad \Delta x = 0.1 \quad \omega : 1.7 \quad$ Iterations : 36
$\beta = 1.0 \quad \Delta x = 0.05 \quad \omega : 1.8 \quad$ Iterations : 63
$\beta = 1.0 \quad \Delta x = 0.025 \quad \omega : 1.9 \quad$ Iterations : 133

$\beta = 1.5 \quad \Delta x = 0.1 \quad \omega : 1.7 \quad$ Iterations : 39
$\beta = 1.5 \quad \Delta x = 0.05 \quad \omega : 1.8 \quad$ Iterations : 92
$\beta = 1.5 \quad \Delta x = 0.025 \quad \omega : 1.9 \quad$ Iterations : 158

(3) The number of iterations reduces to a very small number by using SOR in all the cases(as given above).The optimal value of ω can also be noted by looking into the plots and selecting the 'knee' value in each of the cases.

(4) The optimal value of ω obtained by the formula

$$\omega_{opt} = 2(\frac{1-\sqrt{1-\zeta}}{\zeta})$$

where

$$\zeta = (\frac{cos(\pi/I) + \beta^2 cos(\pi/J)}{1+\beta^2})^2$$

where I and J are the increments along the two co-ordinates x and y respectively.The value of ω obtained by using this formula is consistent with the results obtained in the problem(Point 2 above).The values of ω are incremented by 0.1 in each step so the order of accuracy of the result is to the first decimal place.The values are as follows.
$\beta = 0.5 \quad \Delta x = 0.1 \quad \omega : 1.57018$
$\beta = 0.5 \quad \Delta x = 0.05 \quad \omega : 1.75504$
$\beta = 0.5 \quad \Delta x = 0.025 \quad \omega : 1.86893$

$\beta = 1.0 \quad \Delta x = 0.1 \quad \omega : 1.63951$
$\beta = 1.0 \quad \Delta x = 0.05 \quad \omega : 1.80053$
$\beta = 1.0 \quad \Delta x = 0.025 \quad \omega : 1.89484$

$\beta = 1.5 \quad \Delta x = 0.1 \quad \omega : 1.70461$
$\beta = 1.5 \quad \Delta x = 0.05 \quad \omega : 1.83991$
$\beta = 1.5 \quad \Delta x = 0.025 \quad \omega : 1.91653$

PROBLEM 2

Solve numerically the following convection-diffusion equation for $Re = 10$, 50 and 100.Use $\Delta x = 0.1$, $0,05$ and 0.025. Use central differences for discretizing the derivatives and then solve the system of the linear algebraic equations by Thomas Algorithm.

$$-\frac{\partial^2 u}{\partial x^2} + Re\frac{\partial u}{\partial x} = 0 \qquad 0 \le x \le 1$$

with boundary conditions $u(0) = 0$ and $u(1) = 1$. The analytical solution to this problem is

$$u(x) = \frac{e^{Rex} - 1}{e^x - 1}$$

(1) Compare graphically the analytical solution with the numerical ones(one for each Re.
(2) Observe the changes in numerical solution on increasing Re and decreasing Δx.

SOLUTION

Finite Difference Equation
The PDE governing the problem is

$$-\frac{\partial^2 u}{\partial x^2} + Re\frac{\partial u}{\partial x} = 0$$

Replacing the partial derivative with the central differences formulae yields the following FDE

$$-\frac{(u_{i+1} - 2u_i + u_{i-1})}{\Delta x^2} + Re\frac{(u_{i+1} - u_{i-1})}{2\Delta x} = 0$$

$$(1+\alpha)u_{i-1} - 2u_i + (1-\alpha)u_{i+1} = 0$$

where $\alpha = \frac{Re\Delta x}{2}$. This system of equations is solved using Thomas Method.

Stability Analysis
Let $\frac{u_{i+1}}{u_i} = G$.
For the numerical solution to be stable $|G| \leq 1$.
Expressing u_{i+1} and u_{i-1} in terms of G and u_i

$$(1+\alpha)\frac{u_i}{G} - 2u_i + (1-\alpha)\frac{u_i}{G} = 0$$

$$(1-\alpha)G^2 - 2G + (1+\alpha) = 0$$

$$G = \frac{2 \pm \sqrt{4 - 4(1-\alpha^2)}}{2(1-\alpha)}$$

$$G = 1 \quad G = \frac{1+\alpha}{1-\alpha}$$

. G is an increasing function of α. With decrease in α the solution becomes more stable.
(1) With Re fixed, stability decreases with increase in Δx.
(2) With Δx fixed, stability decreases with increase in Re

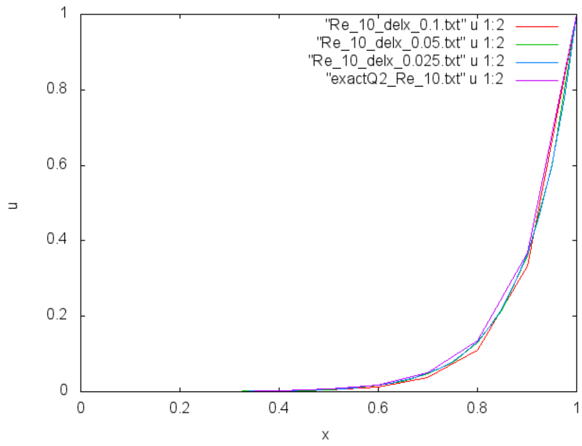
Code

```
#include<iostream>
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

using namespace std;
double low = 0;
double high = 1;
double xrange = high - low;

double function(double x);
double exact(double x,double re);
void ThomasMethod(double u[],int xPoints,double re,double delx);

int main()
{
	FILE *fp = fopen("Re_100_delx_0.05.txt","w");
	double delx,re;
	int xPoints;
```

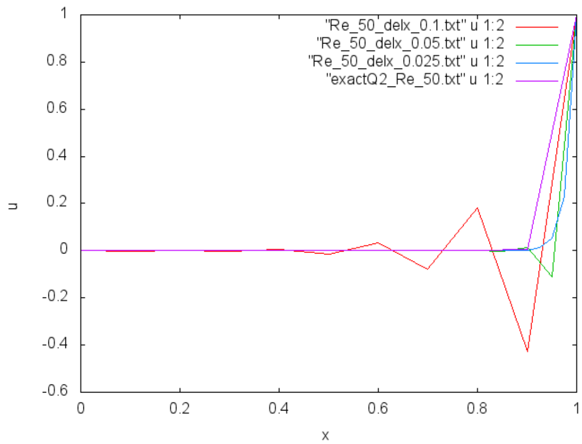
```
    cin>>delx>>re;
    xPoints = 1 + floor(xrange/delx);
    double u[xPoints];
    u[0] = 0;
    u[xPoints-1] = 1;
    ThomasMethod(u,xPoints,re,delx);
    for(int i=0;i<xPoints;i++)
    {
        fprintf(fp,"%lf %lf\n",i*delx,u[i]);
        printf("%0.2lf %0.6lf\n",i*delx,u[i]);
    }
}
double exact(double x,double re)
{
    return (exp(re*x)-1)/(exp(re)-1);
}
void ThomasMethod(double u[],int points,double re,double delx)
{
    double   A[points-2],B[points-2],C[points-2],F[points-2];
    double alpha = re*delx*0.5;
    double a = (double)1+alpha;
    double b = (double)2;
    double c = (double)1-alpha;
    double d = (double)0;
    for(int i=0;i<points-2;i++)
    {
        A[i] = a;
        B[i] = -b;
        C[i] = c;
        F[i] = d;
    }
    F[points-3] = -c;
    for(int i=1;i<points-2;i++)
    {
        B[i] = B[i]-(C[i]*A[i])/B[i-1];
        F[i] = F[i]-(F[i-1]*A[i])/B[i-1];
    }
    u[points-2] = F[points-3]/B[points-3];
    for(int i=points-4;i>=0;i--)
    {
        u[i+1] = (F[i]-(C[i]*u[i+2]))/B[i];
    }
}
```

Results

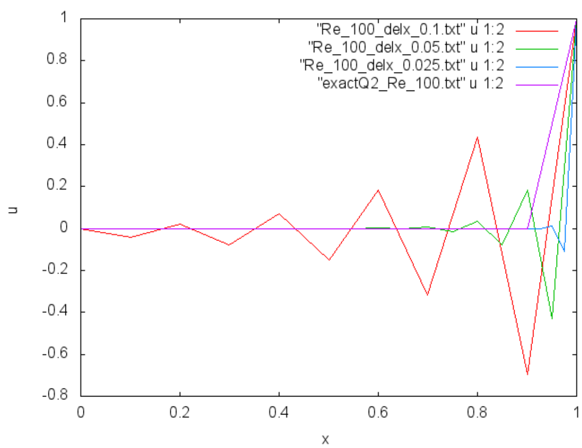
(1) $Re = 10$



(2) $Re = 50$



(1) $Re = 100$



Observations & Explanations

(1) For $Re = 10$, the numerical solutions for all the three values of Δx is almost the same as the analytical solution i.e the error is very less and a somewhat stable solution is obtained.(with very less oscillations - almost equal to 0).

(2) For $Re = 50$, the numerical solutions show oscillations. The oscillations tend to decrease on decreasing the value of Δx.The solution hence is not stable.

(3) For $Re = 100$,again the numerical solution is not stable and the oscillations tend to decrease on decreasing the value of Δx.

(4) Increasing Re decreases the accuracy ,increases the oscillations hence the instability (consistent with the stability analysis done above).

(5) Decreasing Δx increases the accuracy,decreases the oscillations hence the instability (consistent with the stability analysis done above).

(6) According to the stability analysis done , the solution anyways is unconditionally unstable.