MA 322: End-Semester Assignment

Due on Sunday, November 22, 2015

Jiten Chandra Kalita

Tushar Sircar(130123038) — T.Dinesh Reddy(130123037) — Akepogu Prince(130123004)

PROBLEM 1

A thin rectangular homogeneous thermally conducting plate lies in the xy-plane define by $0 \leq x \leq 4$,$0 \leq y \leq 1$. The edge $y = 0$ is maintained at $200x(x-4)$ while the remaining edges are held at 0 degrees. The other faces are insulated and no sources or sinks are present.

Set the partial difference equation governing the steady state temperature $T(x, y)$ along with the boundary conditions. Then, solve it numerically using a known finite difference scheme. Use the Gauss-Seidel method as the iterative solver. You must experiment with:
(a) $\Delta x = 0.1, 0.05, 0.025$
(b) 3 different grid aspect ratios given by $\beta = \frac{\Delta x}{\Delta y}$ preferably 1, < 1 and > 1

Compare graphically the analytic solution along the vertical center line with the numerical ones obtained for $\Delta x = 0.1, 0.05, 0.025$.
What can you comment about the efficiency of the iterative solvers? Perform further experiments on the Gauss-Seidel iterative solver by using SOR and comment upon the optimum ω.

The PDE governing the steady state temperature is given by:

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

$$T(x, 0) = 200x(x-4), T(0, y) = T(4, y) = T(x, 1) = 0$$

SOLUTION

Finite Difference Equation

We replace T_{xx} and T_{yy} by the second order center difference formula we get FDE that has truncation error of second order in both Δx and Δt

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

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

Replacing the above FDE in the original PDE we get:

$$2(1+\beta^2)T(x, y) = T(x-1, y) + T(x+1, y) + \beta^2[T(x, y-1) + T(x, y+1)]$$

We denote by:
N_x : Number of variables along x $= \frac{4}{\Delta x} - 1$
N_y : Number of variables along y $= \frac{1}{\Delta y} - 1$
So, we have a system of $N_x N_y$ variables. Let $T^{(k)}$ denote the values at the k_{th} iteration. Then,

$$T^{(k+1)} = \frac{T^{(k+1)}_{x-1,y} + T^{(k)}_{x+1,y} + \beta^2 T^{(k+1)}_{x,y-1} + \beta^2 T^{(k)}_{x,y+1}}{2(1+\beta^2)}$$

We stop iterating when relative error of all grid points is less than $\epsilon = 0.000001$

Code

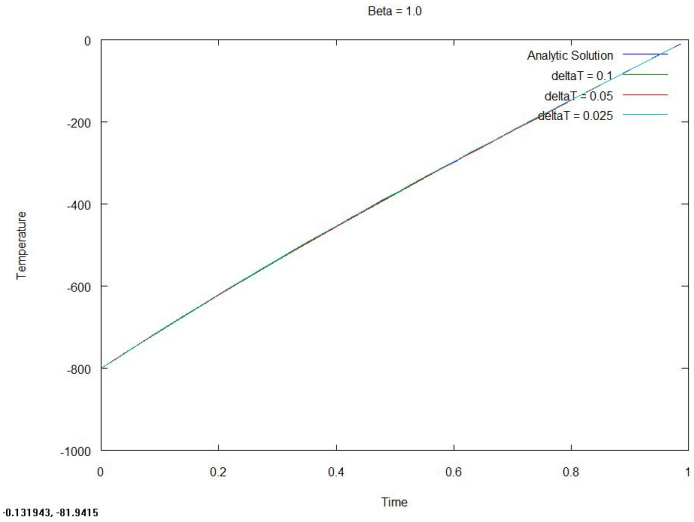
```
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

double getF(double x)
{
    return (double)200.0 * x * (x - (double)4.0);
}
double getAnalytic(double x,double y)
{
    double ans = 0;
    double pi = atan(1);
    for(int i=1; i<=50; i+=2)
        ans += -2.0 * ((double)1/sinh(-(double)i * pi))*
        ((double)200/(pow((double)i*pi,3.0))) *
        sin(((double)i*pi*x)) * sinh((double)i*pi*(y-(double)1));
    return ans;
}


void solvePDE(double DeltaX,double DeltaY)
{
    double beta = DeltaX/DeltaY;
    int Nx = -1 + floor((double)4.0/DeltaX);
    int Ny = -1 + floor((double)1.0/DeltaY);
    printf("Number of variables along X: %d\nNumber of variables
    along Y: %d\nAspect Ratio: %f\n",Nx,Ny,beta);

    double x[Nx+2];
    double y[Ny+2];

    x[0] = y[0] = 0;
    for(int i=1; i<Nx+2; i++)
        x[i] = x[i-1] + DeltaX;
    for(int i=1; i<Ny+2; i++)
        y[i] = y[i-1] + DeltaY;

    double T[Nx+2][Ny+2];
    double tempT[Nx+2][Ny+2];

    for(int i=0; i<Nx+2; i++)
        T[i][0] = tempT[i][0] =  getF(x[i]);
    for(int i=0; i<Ny+2; i++)
        T[0][i] = tempT[0][i] = T[Nx+1][i] = tempT[Nx+1][i] = 0;
    for(int i=0; i<Nx+2; i++)
        T[i][Ny+1] = tempT[i][Ny+1] = 0;

    //initialisation code
```

```
    for(int i=1; i<=Nx; i++)
        for(int j=1; j<=Ny; j++)
            T[i][j] = 0;

    double errorNum,errorDen;
    double epsilon = 0.000001;
    double errorCount = 0;
    long long iterations = 0;
    do
    {
        errorCount = 0;
        iterations++;
        errorNum = errorDen = 0;
        for(int j=1; j<=Ny; j++)
            for(int i=1; i<=Nx; i++)
        {
            tempT[i][j] = tempT[i-1][j] + T[i+1][j] +
            (pow(beta,(double)2)*(tempT[i][j-1] + T[i][j+1]));
            tempT[i][j] /= (double)2.0*((double)1 + pow(beta,
            (double)2));
            errorNum += pow(fabs(tempT[i][j] - T[i][j]),(double)2);
            errorDen += pow(T[i][j],(double)2);
            if(fabs(tempT[i][j] - T[i][j])/fabs(T[i][j]) < epsilon)
                errorCount++;
        }

        for(int i=1; i<=Nx; i++)
            for(int j=1; j<=Ny; j++)
                T[i][j] = tempT[i][j];

    }while(errorCount < Nx*Ny);
    cout<<"Number of iterations: "<<iterations<<endl;


    FILE *output = fopen("outputTest.txt","w");
    FILE *aOutput = fopen("analyticOutput.txt","w");


    for(int j=0; j<=Ny; j++)
    {
        fprintf(output,"%f %f\n",y[j],T[(Nx+1)/2][j]);
        fprintf(aOutput,"%f %f\n",y[j],getAnalytic(x[(Nx+1)/2],y[j]));
    }
    fclose(output);
    fclose(aOutput);
}


int main()
{
    double DeltaX,DeltaY;
```

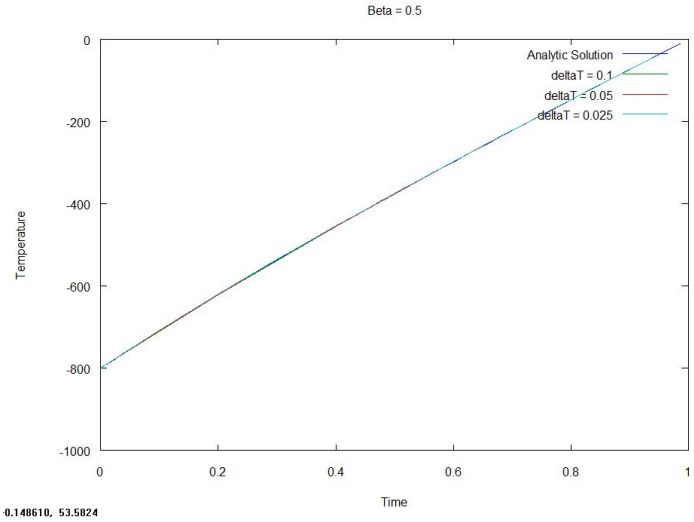
```
    while(1)
    {
        cout<<"\nEnter DeltaX and DeltaY: ";
        scanf("%lf %lf",&DeltaX,&DeltaY);
        solvePDE(DeltaX,DeltaY);
    }

    return 0;
}
```

Results

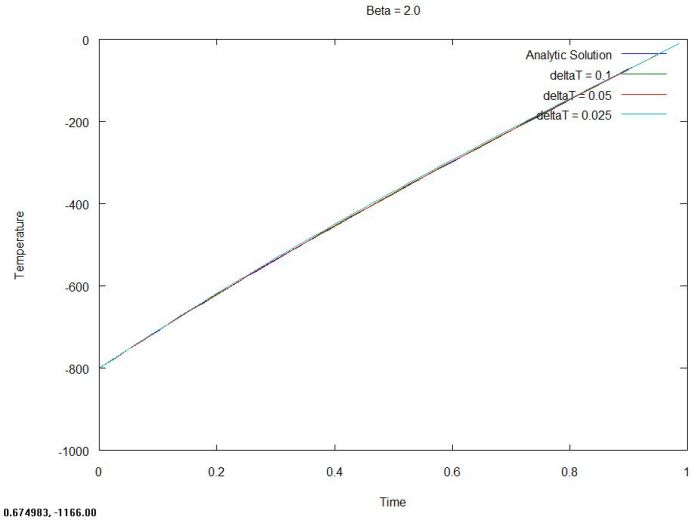
(a) $\beta = 1$



(b) $\beta = 0.5$



(c) $\beta = 2.0$



Observations

(a) We can see that at a particular value of x, T varies linearly with y.
(b) This is because keeping x fixed, $T_{yy} = 0$ which implies that at a particular value of x, $T = ay + b$ that is T varies linearly with y. This is consistent with our observation and matches the analytic solution as well.
(c) We obtain convergence for all the Δx and Δt pairs but the number of iterations can be reduced using SOR method.

Using SOR Method

The FDE derived in the above section can be further written as:

$$T_{x,y}^{k+1} = T_{x,y}^{k} + \omega[\frac{T_{x-1,y}^{(k+1)} + T_{x+1,y}^{(k)} + \beta^2 T_{x,y-1}^{(k+1)} + \beta^2 T_{x,y+1}^{(k)}}{2(1+\beta^2)} - T_{x,y}^{k}]$$

According the Kahan's Theorem, we get convergence for $0 \leq \omega \leq 2$. So we vary ω and plot the number of iterations before convergence.

Code

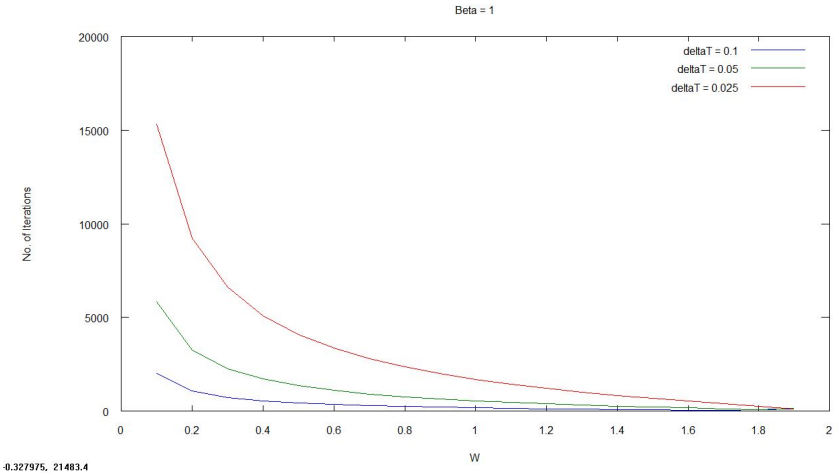
```
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

double getF(double x)
{
    return (double)200.0 * x * (x - (double)4.0);
}

long long solvePDE(double DeltaX,double DeltaY,double W)
{
    double beta = DeltaX/DeltaY;
    int Nx = -1 + floor((double)4.0/DeltaX);
```

```
    int Ny = -1 + floor((double)1.0/DeltaY);
    printf("Number of variables along X: %d\nNumber
    of variables along Y: %d\nAspect Ratio: %f\n",Nx,Ny,beta);

    double x[Nx+2];
    double y[Ny+2];

    x[0] = y[0] = 0;
    for(int i=1; i<Nx+2; i++)
        x[i] = x[i-1] + DeltaX;
    for(int i=1; i<Ny+2; i++)
        y[i] = y[i-1] + DeltaY;

    double T[Nx+2][Ny+2];
    double tempT[Nx+2][Ny+2];

    for(int i=0; i<Nx+2; i++)
        T[i][0] = tempT[i][0] =  getF(x[i]);
    for(int i=0; i<Ny+2; i++)
        T[0][i] = tempT[0][i] = T[Nx+1][i] = tempT[Nx+1][i] = 0;
    for(int i=0; i<Nx+2; i++)
        T[i][Ny+1] = tempT[i][Ny+1] = 0;

    //initialisation code
    for(int i=1; i<=Nx; i++)
        for(int j=1; j<=Ny; j++)
            T[i][j] = 0;

    double errorNum,errorDen;
    long long iterations = 0;
    long long errorCount;
    double epsilon = 0.000001;
    do
    {
        errorCount = 0;
        iterations++;
        errorNum = errorDen = 0;
        for(int j=1; j<=Ny; j++)
            for(int i=1; i<=Nx; i++)
        {
            double v = tempT[i-1][j] + T[i+1][j] + (pow(beta,
            (double)2)*(tempT[i][j-1] + T[i][j+1]));
            v /= (double)2.0*((double)1 + pow(beta,(double)2));
            v -= T[i][j];
            tempT[i][j] = T[i][j] + (v*W) ;

            errorNum += pow(fabs(tempT[i][j] - T[i][j]),(double)2);
            errorDen += pow(T[i][j],(double)2);
            if(fabs(tempT[i][j] - T[i][j])/fabs(T[i][j]) < epsilon)
                errorCount++;
        }

        for(int i=1; i<=Nx; i++)
```

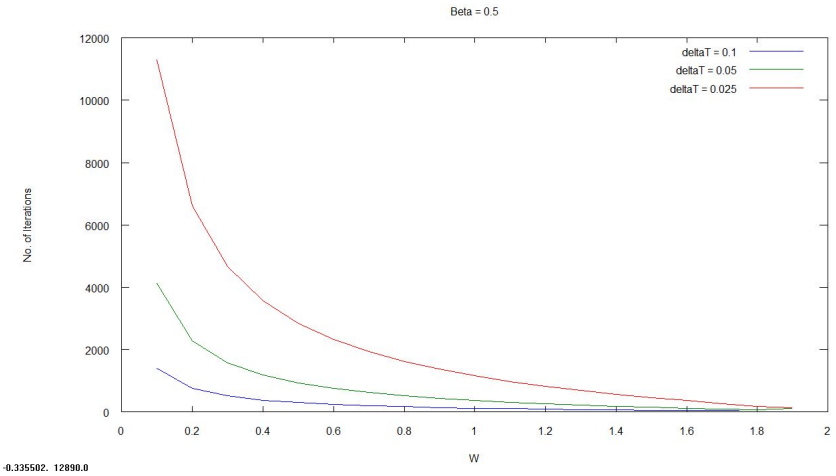
```
            for(int j=1; j<=Ny; j++)
                T[i][j] = tempT[i][j];

    }while(errorCount < Nx*Ny);
    return iterations;
}


int main()
{
    double DeltaX,DeltaY,W;
    cout<<"Enter DeltaX and DeltaY: ";

    scanf("%lf %lf",&DeltaX,&DeltaY);

    FILE *output = fopen("output3.txt","w");

    double Delta = 0.1;
    for(int i=1; i<20; i++)
        fprintf(output,"%f %lld\n",
        (double)i*Delta,solvePDE(DeltaX,DeltaY,(double)i*Delta));
    fclose(output);


    return 0;
}
```

Results

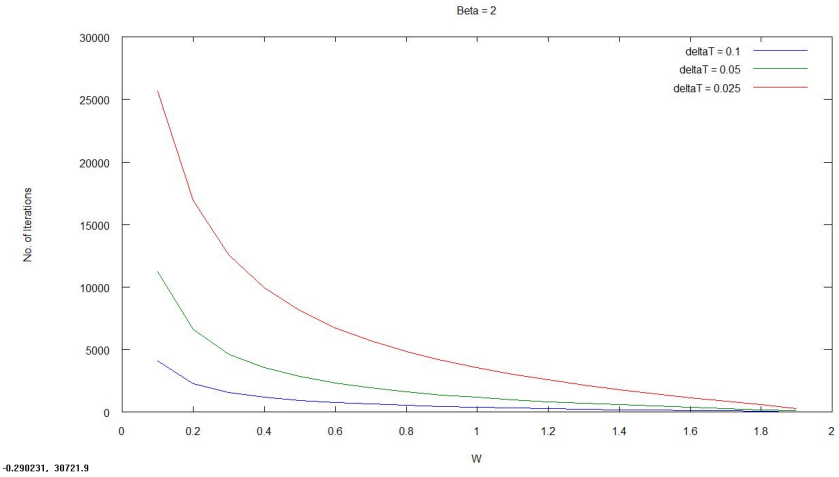
(a) $\beta = 1$



(b) $\beta = 0.5$



(c) $\beta = 2.0$



Observations

(a) $\omega \geq 1$ is over-relaxation and results in less number of iterations whereas $omega < 1$ is under-relaxation and results in more number of iterations.

(b) There is an optimal value of ω lying between 0 and 2

Number of Iterations without SOR

$\beta = 1.0$

$\Delta x = 0.1 \quad Iterations = 224$

$\Delta x = 0.05 \quad Iterations = 782$

$\Delta x = 0.025 \quad Iterations = 2689$

$\beta = 0.5$

$\Delta x = 0.1 \quad Iterations = 152$

$\Delta x = 0.05 \quad Iterations = 521$

$\Delta x = 0.025 \quad Iterations = 1787$

$\beta = 2.0$
$\Delta x = 0.1 \quad Iterations = 505$
$\Delta x = 0.05 \quad Iterations = 1759$
$\Delta x = 0.025 \quad Iterations = 6012$

Minimum number of Iterations with SOR
We also find the theoretically optimal value of ω for the given cases using the below formula:

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

where

$$\epsilon = (\frac{cos(\pi/\Delta x) + \beta^2 cos(\pi/\Delta y)}{1 + \beta^2})^2$$

$\beta = 1.0$
$\Delta x = 0.1 \quad \omega = 1.60 \quad Iterations = 56$
$\Delta x = 0.05 \quad \omega = 1.80 \quad Iterations = 95$
$\Delta x = 0.025 \quad \omega = 1.90 \quad Iterations = 211$

$\beta = 0.5$
$\Delta x = 0.1 = \quad \omega = 1.60 \quad Iterations = 48$
$\Delta x = 0.05 \quad \omega = 1.80 \quad Iterations = 99$
$\Delta x = 0.025 \quad \omega = 1.90 \quad Iterations = 204$

$\beta = 2.0$
$\Delta x = 0.1 = \quad \omega = 1.80 \quad Iterations = 73$
$\Delta x = 0.05 \quad \omega = 1.90 \quad Iterations = 149$
$\Delta x = 0.025 \quad \omega = 1.90 \quad Iterations = 392$

Theoretically Optimal ω
$\beta = 1.0$
$\Delta x = 0.1 \quad \omega = 1.63951$
$\Delta x = 0.05 \quad \omega = 1.80053$
$\Delta x = 0.025 \quad \omega = 1.89484$

$\beta = 0.5$
$\Delta x = 0.1 \quad \omega = 1.57018$
$\Delta x = 0.05 \quad \omega = 1.75504$
$\Delta x = 0.025 \quad \omega = 1.86893$

$\beta = 2.0$
$\Delta x = 0.1 \quad \omega = 1.75438$
$\Delta x = 0.05 \quad \omega = 1.8683$
$\Delta x = 0.025 \quad \omega = 1.93215$

We can see that SOR reduces the number of iterations significantly with the right value of ω! Also, the experimental values of optimal ω is consistent with the theoretical values computed using the above mentioned

formula!

PROBLEM 2
Solve numerically the following convection-diffusion equation for Re = 10, 50 and 100. Use central differences for discretizing the derivatives and then solve the system of the linear algebraic equations by Thomas Algorithm.

$$-u_{xx} + Reu_x = 0, 0 \leq x \leq 1$$

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

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

Use $\Delta x = 0.1$, 0.05 and 0.025. Compare graphically (there should be one graph for each Re. Each figure will have the analytical solution with the numerical ones obtained for $\Delta x = 0.1$, 0.05 and 0.025) the analytical and numeric solutions obtained by you.
What observations can be made based upon your numerical results with increasing Re and decreasing Δx?

SOLUTION

Finite Difference Equations

$$u_{xx}|_i = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}$$

$$u_x|_i = \frac{u_{i+1} - u_{i-1}}{2\Delta x}$$

Replacing the above FDE in the original PDE we get,

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

where $\beta = \frac{Re\Delta x}{2}$
We denote by:
N_x: Number of variables along x = $\frac{1}{\Delta x} - 1$
So, we have a system of N_x variables, which we then solve using the Thomas Algorithm.

Code

```
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

double getExactSolution(double x,double Re)
```

```
{
    return (exp(Re*x)-1.0)/(exp(Re)-1.0);
}

void solveThomas(double a[],double b[],double c[],double t[],double f[],int k)
{

    //forward sweep
    f[1] = f[1] - (a[1]*t[0]);
    f[k-1] = f[k-1] - (c[k-1]*t[k]);

    b[1] = b[1]; //unchanged
    for(int i=2; i<=k-1; i++)
    {
        b[i] = b[i] - ((c[i-1]*a[i])/b[i-1]);
        f[i] = f[i] - ((f[i-1]*a[i])/b[i-1]);
    }

    //find solutions using backward sweep
    t[k-1] = f[k-1]/b[k-1];
    for(int i=k-2; i>=1; i--)
        t[i] = (f[i] - (c[i]*t[i+1]))/b[i];


}


void solvePDE(double Re,double DeltaX)
{
    int Nx = -1 + floor((double)1.0/DeltaX);
    cout<<"Number of variables along X-axis: "<<Nx<<endl;

    double beta = (Re*DeltaX)/(double)2.0;
    double solution[Nx+2];
    solution[0] = 0.0;
    solution[Nx+1] = 1.0;

    double a[Nx+2],b[Nx+2],c[Nx+2],f[Nx+2];
    for(int i=0; i<Nx+2; i++)
    {
        a[i] = (double)1.0 + beta;
        b[i] = -(double)2.0;
        c[i] = (double)1.0 - beta;
        f[i] = 0;
    }

    solveThomas(a,b,c,solution,f,Nx+1);

    FILE *output = fopen("output.txt","w");
    FILE *aOutput = fopen("aOutput.txt","w");
    for(int i=0; i<Nx+2; i++)
    {
        fprintf(output,"%f %f\n",DeltaX*(double)i,solution[i]);
```

```
        fprintf(aOutput,"%f %f\n",DeltaX
        *(double)i,getExactSolution((double)i*DeltaX,Re));
    }
    fclose(output);
    fclose(aOutput);
}

int main()
{
    double Re,DeltaX;

    while(1)
    {
        cout<<"\nEnter values of Re and DeltaX: ";
        scanf("%lf %lf",&Re,&DeltaX);
        solvePDE(Re,DeltaX);
    }



    return 0;
}
```

Stability Analysis We use Von Nuemann Fourier Analysis to find the stability condition for the above problem. For this we take $u_i = e^{Ik\Delta x}$ and then express u_{i-1} and u_{i+1} in terms of u_i

$$u_{i-1} = u_i e^{-Ik\Delta x}$$
$$u_{i+1} = u_i e^{Ik\Delta x}$$

Substituting these values in the FDE we get,

$$(1+\beta)e^{-Ik\Delta x} - 2 + (1-\beta)e^{Ik\Delta x} = 0$$

We then denote by G the amplification factor, $G = \frac{u_{i+1}}{u_i} = e^{Ik\Delta x}$ which we substitute into the previous equation to get,

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

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

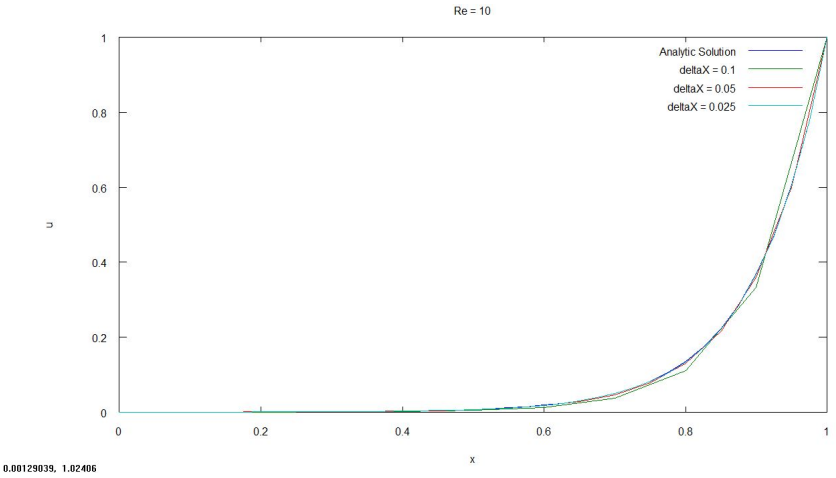
$$G = 1 \quad , G = \frac{1+\beta}{1-\beta}$$

Thus, G is increases with an increase in β and so reducing the value of β gives us a more stable solution. Since $\beta = \frac{Re\Delta x}{2}$:
(a) For some value of Re, solution become more stable as we decrease the value of Δx

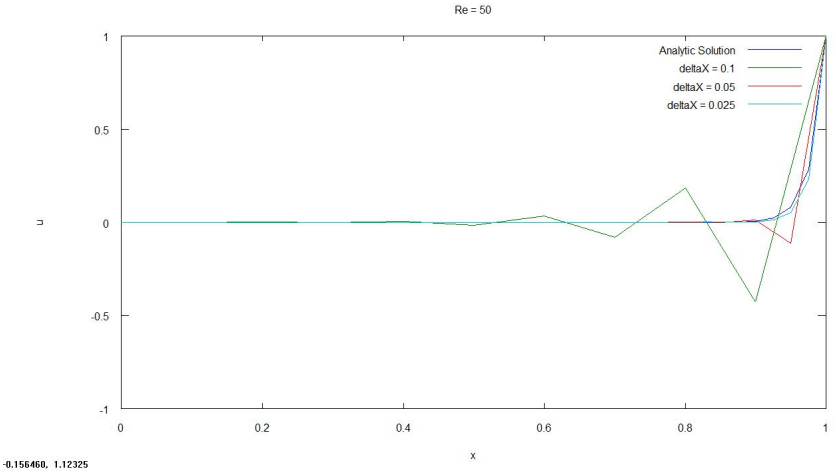
(b) For some value of Δx, the solution becomes more stable as we decresase the value of Re
(c) Thus, smaller values of Re and Δx gives us a more stable solution!

Results

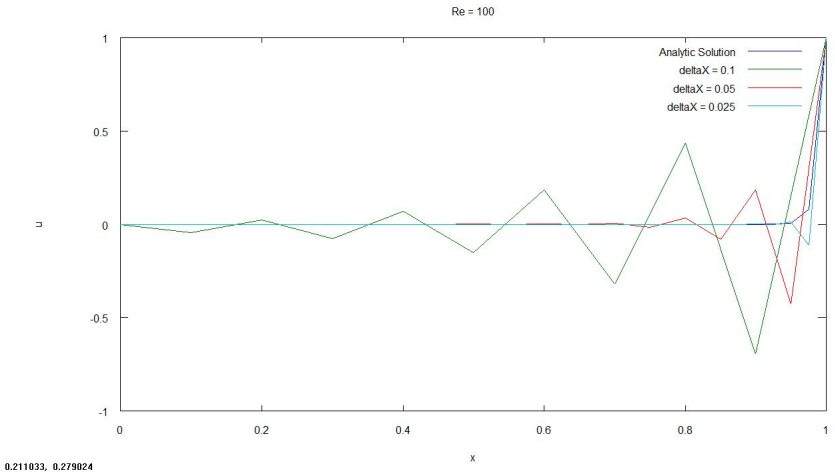
(a) $Re = 10$



(b) $Re = 50$



(c) $Re = 100$



Observations

(a) Re = 10, numerical solution almost exactly matches the numerical solution for all values of Δx.

(b) Re = 50, numerical solution matches analytical solution for $\Delta x = 0.025$. For others, solution is unstable and shows oscillations. Oscillations increase as we increase the value of Δx.

(c) Solution is unstable for all values of Δx. Unstability increases as we increase value of Δx.

(d) We can see that the observations are consistent with our stability analysis. For the smallest value of Re our solution is stable for all values of Δx and for the largest value of Re, the solution is unstable for all values of Δx. For all values of Re, the instability increases as we increase the value of Δx (as seen in the above plots)

(e) Increasing Re decreases stability and decreasing Δx increases the stability of the numerical solution