MA 322: Lab Assignment #8

Due on Sunday, November 2, 2015

Jiten Chandra Kalita

Tushar Sircar - 130123038

Contents

PROBLEM 1

To derive explicit and implicit finite difference approximations to the problem

$$Du_{xx} = u_t + bu$$

where $u(0,t) = u(1,t) = 0$ and $u(x,0) = g(x)$. Also, D and b are positive constants.

SOLUTION

(a) Explicit Method

$$D(\frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{\Delta x^2}) = (\frac{U_i^{n+1} - U_i^n}{\Delta t}) + bU_i^n$$

$$\frac{D\Delta t}{\Delta x^2}(U_{i+1}^n - 2U_i^n + U_{i-1}^n) = U_i^{n+1} - U_i^n + bU_i^n \Delta t$$

$$U_i^{n+1} = U_{i+1}^n \frac{D\Delta t}{\Delta x^2} + U_{i-1}^n \frac{D\Delta t}{\Delta x^2} + (1 - b\Delta t - 2\frac{D\Delta t}{\Delta x^2})U_i^n$$

(b) Implicit Method

The implicit finite difference approximation can be obtained by replacing every grid value at the n^{th} time stamp with the average of the n^{th} and $(n+1)^{th}$ time stamp.

$$(\frac{U_i^{n+1} - U_i^n}{\Delta t}) = D(\frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{\Delta x^2}) - bU_i^n$$

$$(\frac{U_i^{n+1} - U_i^n}{\Delta t}) = D(\frac{(\frac{U_{i+1}^{n+1} + U_{i+1}^n}{2}) - 2(\frac{U_i^{n+1} + U_i^n}{2}) + (\frac{U_{i-1}^{n+1} + U_{i-1}^n}{2})}{\Delta x^2}) - b(\frac{U_i^{n+1} + U_i^n}{2})$$

$$(1 + \alpha + \frac{\beta}{2})U_i^{n+1} - \frac{\alpha}{2}U_{i+1}^{n+1} - \frac{\alpha}{2}U_{i-1}^{n+1} = (1 - \alpha - \frac{\beta}{2})U_i^n + \frac{\alpha}{2}U_{i+1}^n + \frac{\alpha}{2}U_{i-1}^n$$

where $\alpha = \frac{D\Delta t}{\Delta x^2}$ and $\beta = b\Delta t$

PROBLEM 2

$g(x) = sin(\pi x)$, $D = 0.1$, $b = 1$. Compute the solution at $t = 1$ using Δt =0.25 , 0.125 , 0.0625. On the same axes plot the exact solution at t = 1 and the three numerical solutions, one for the explicit and the other for the implicit method.

SOLUTION

(a) EXPLICIT METHOD

Code

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

double getG(double x)
{
    return sin((double)atan(1)*(double)4*x);
}

int main()
{
    double xMin = 0, xMax = 1;
    double tMin = 0, tMax = 1;
    double D = 0.1;
    double b = 1.0;
    double deltaX,deltaT;
   // cout<<"Enter size of space interval: ";
   // cin>>deltaX;
    cout<<"Enter size of time interval: ";
    cin>>deltaT;
    deltaX =  sqrt((double)4*D*deltaT/(2.0 - (b*deltaT)));
    cout<<"Space interval: "<<deltaX<<endl;
    int xPoints =  1 + floor(((double)1)/deltaX);
    int tPoints = 1 + floor(((double)1.0)/deltaT);

    double point[xPoints][tPoints];

    double x[xPoints];
    double t[tPoints];

    for(int i=0; i<xPoints; i++)
        x[i] = (double)i * deltaX;
    for(int i=0; i<tPoints; i++)
        t[i] = (double)i * deltaT;

    for(int c=0; c<xPoints; c++)
        point[c][0] = getG(x[c]);
    for(int r=0; r<tPoints; r++)
        point[0][r] = point[xPoints-1][r] = 0;


    double alpha = (D*deltaT)/(deltaX*deltaX);
```

```
    double beta = (b*deltaT);
    //cout<<"c1: "<<c1<<endl<<"c2: "<<c2<<endl;
    for(int r=1; r<tPoints; r++)
    {
        for(int c=1; c<xPoints-1; c++)
        {
            point[c][r] = (alpha * (point[c-1][r-1] + point[c+1][r-1])) + ((1 - (2*alpha) - beta)*(po
            cout<<point[c][r]<<endl;
        }
        cout<<"----------\n";
    }

     FILE * output = fopen("outputExplicit3.txt","w");
    for(int r=0; r<xPoints; r++)
        fprintf(output,"%f %f\n",x[r],point[r][tPoints-1]);
    fclose(output);

}
```

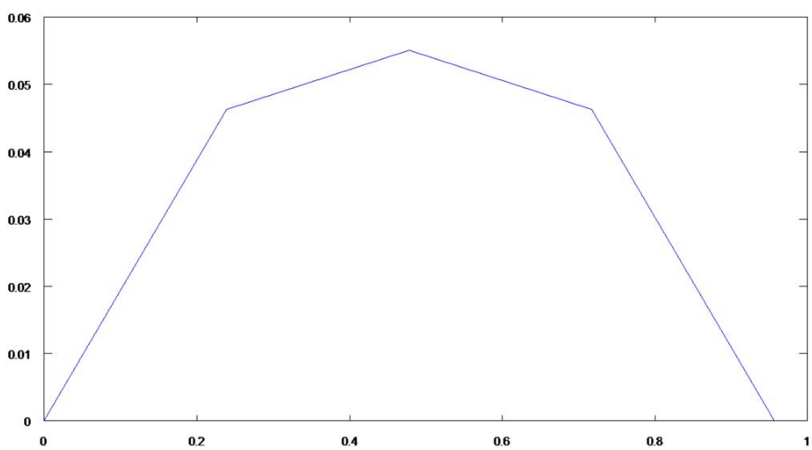
Explanation

The exact solution of the PDE can be obtained by integrating the expression given in the problem.It comes out as $e^{-b-D\pi^2}sin(\pi x)$ at $t = 1$

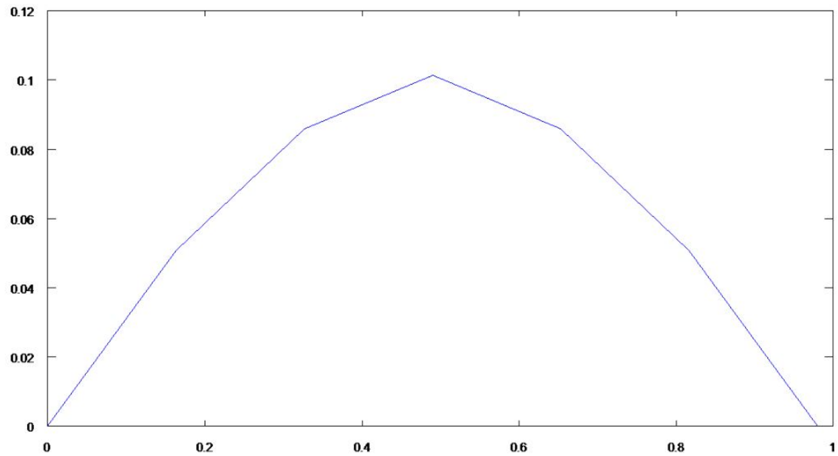
To get the appropriate Δx for the given values of Δt ,the conditions of stability are exploited.

The condition comes out as :

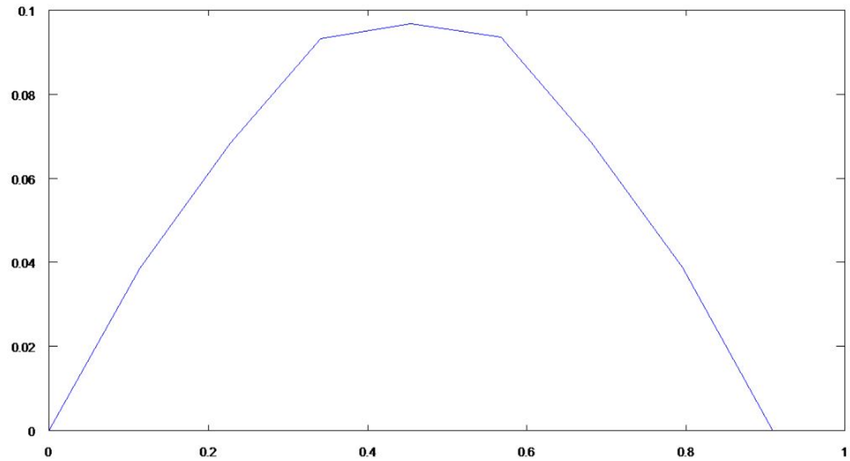
$$\Delta x^2 >= \frac{4D\Delta t}{2 - b\Delta t}$$



deltaT = 0.25



deltaT = 0.125



deltaT = 0.0625

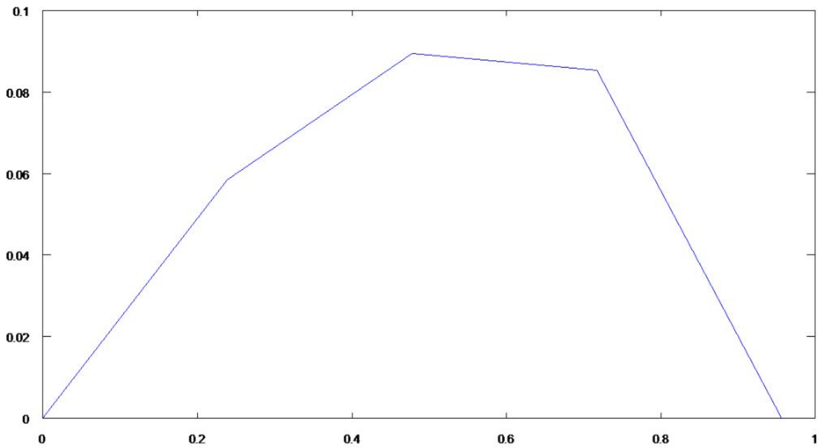
(b) IMPLICIT METHOD

Code

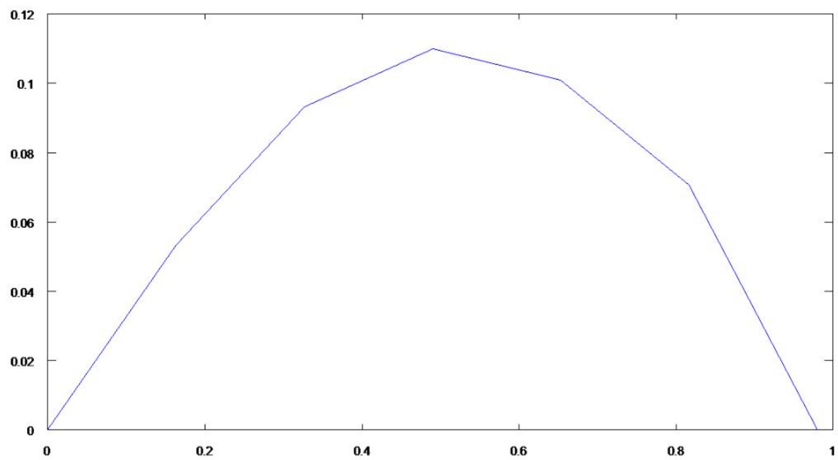
```
#include<iostream>
#include<stdio.h>
#include<cmath>
using namespace std;

double getG(double x)
{
    return sin((double)atan(1)*(double)4*x);
}

void solveGaussSiedel(int xPoints,double alpha,double beta,double arr1[],double arr2[])
{
    int n = xPoints-2;
    double b[n+1];

    double c1 = -alpha/2;
    double c2 = ((2*alpha)+beta-2)/(2.0);
```

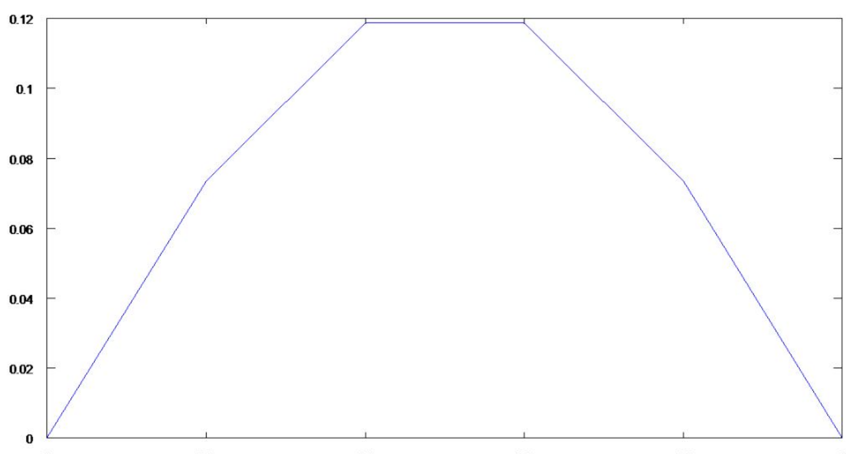
```
    double c3 = alpha/(2.0);
    double c4 = -(2+(2*alpha)+beta)/(2.0);

    cout<<c3<<" "<<c4<<endl;


    for(int i=1; i<=n; i++)
        b[i] = (c1*arr1[i-1]) + (c2*arr1[i]) + (c1*arr1[i+1]);
    double err;
    do
    {
        for(int i=1; i<=n; i++)
        {
            if(i == 1)
            {
                arr2[i] = (b[i] - (c3*arr1[i+1]))/c4;
            }
            else if(i != n)
            {
                arr2[i] = (b[i] - (c3*arr2[i-1]) - (c3*arr1[i+1]))/c4;
            }
            else
                arr2[i] = (b[i] - (c3*arr2[i-1]))/c4;

        }
        double maxErr1 = -1;
        double maxErr2 = -1;
        for(int i=1; i<=n; i++)
        {
            maxErr1 = max(maxErr1,fabs(arr1[i]-arr2[i]));
            maxErr2 = max(maxErr2,fabs(arr1[i]));
            cout<<arr2[i]<<" ";
            arr1[i] = arr2[i];
        }
        cout<<endl;
        err = maxErr1/maxErr2;

    }while(err > 0.0001);
}



int main()
{
    double xMin = 0, xMax = 1;
    double tMin = 0, tMax = 1;
    double D = 0.1;
    double b = 1.0;
    double deltaX,deltaT;
   // cout<<"Enter size of space interval: ";
   // cin>>deltaX;
    cout<<"Enter size of time interval: ";
    cin>>deltaT;
```

```
    deltaX =  sqrt((double)4*D*deltaT/(2.0 - (b*deltaT)));
    cout<<"Space interval: "<<deltaX<<endl;
    int xPoints =  1 + floor(((double)1)/deltaX);
    int tPoints = 1 + floor(((double)1.0)/deltaT);

    cout<<"X-Points: "<<xPoints<<endl;

    double arr1[xPoints];
    double arr2[xPoints];

    double x[xPoints];
    double t[tPoints];

    for(int i=0; i<xPoints; i++)
        x[i] = (double)i * deltaX;
    for(int i=0; i<tPoints; i++)
        t[i] = (double)i * deltaT;

    for(int c=0; c<xPoints; c++)
        arr1[c] = getG(x[c]);


    double alpha = (D*deltaT)/(deltaX*deltaX);
    double beta = (b*deltaT);

    for(int i=0; i<tPoints; i++)
        solveGaussSiedel(xPoints,alpha,beta,arr1,arr2);


     FILE * output = fopen("outputImplicit_3.txt","w");
     fprintf(output,"%f %f\n",x[0],0);
    for(int r=1; r<xPoints-1; r++)
        fprintf(output,"%f %f\n",x[r],arr1[r]);

    fprintf(output,"%f %f\n",x[xPoints-1],0.0);
    fclose(output);

}
```



deltaT = 0.25



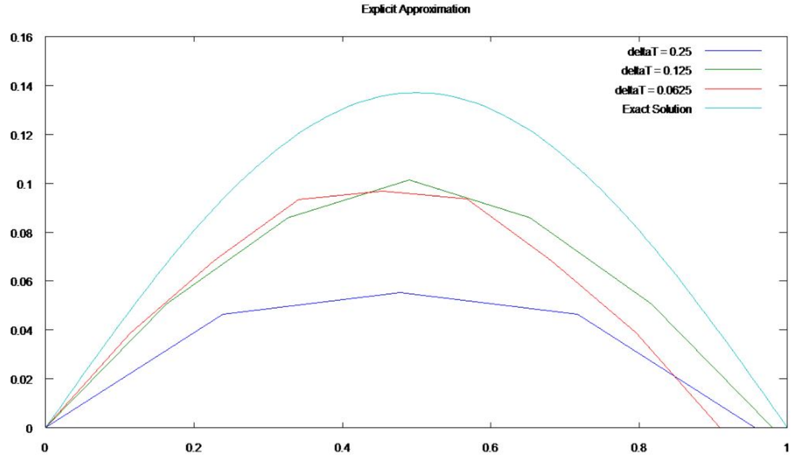


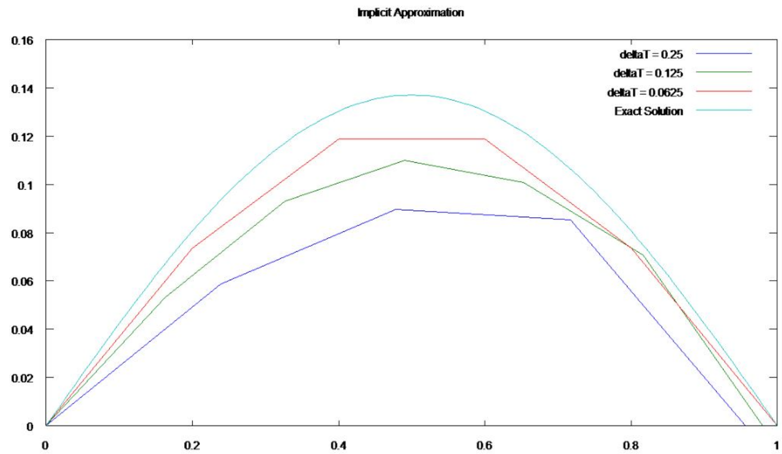
deltaT = 0.125

deltaT = 0.0625

Result

PLOTTING EXACT VS NUMERIC SOLUTIONS





PROBLEM 3

$g(x) = sin(\pi x)$, $D = 0.1$, $b = 1$. Plot the maximum error at $t = 1$ as a function of M using $\Delta t = \frac{1}{M}$ and $\Delta x = \frac{1}{10}$ for M = 4,8,16,32. On the same axes plot the maximum error taking $\Delta x = \frac{1}{20}$. Explain the results.

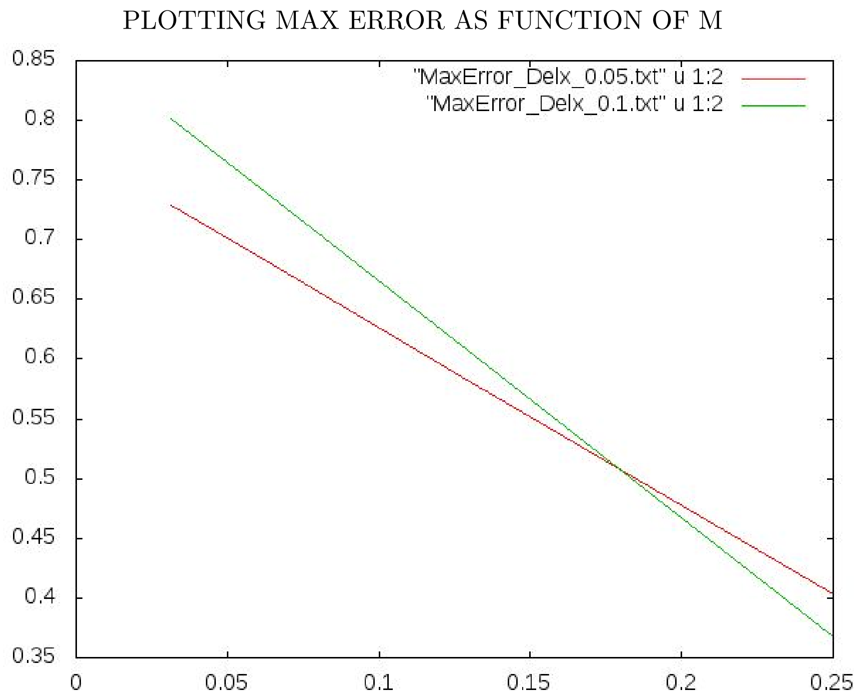
Code to find the maximum errors at time t = 1

Code

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

double getG(double x)
{
    return sin((double)atan(1)*(double)4*x);
}
double getExact(double x,double b,double D)
{
    double pi = atan(1)*(double)4;
    return exp(-(b - (D*pi*pi)))*sin(pi*x);
}

int main()
{
    double xMin = 0, xMax = 1;
    double tMin = 0, tMax = 1;
    double D = 0.1;
    double b = 1.0;
    double deltaX,deltaT;
    double m[] = {4,8,16,32};
    FILE * output = fopen("maxError_1.txt","w");

    for(int index = 0; index<4; index++)
    {



   // cout<<"Enter size of space interval: ";
   // cin>>deltaX;
    cout<<"Enter size of time interval: ";
    deltaT = (double)1.0/m[index];//cin>>deltaT;
    cout<<"Enter size of space interval: ";
    deltaX = 1.0/10.0;//cin>>deltaX;
    int xPoints =  1 + floor(((double)1)/deltaX);
    int tPoints = 1 + floor(((double)1.0)/deltaT);

    double point[xPoints][tPoints];

    double x[xPoints];
    double t[tPoints];

    for(int i=0; i<xPoints; i++)
        x[i] = (double)i * deltaX;
    for(int i=0; i<tPoints; i++)
```

```
        t[i] = (double)i * deltaT;

    for(int c=0; c<xPoints; c++)
        point[c][0] = getG(x[c]);
    for(int r=0; r<tPoints; r++)
        point[0][r] = point[xPoints-1][r] = 0;


    double alpha = (D*deltaT)/(deltaX*deltaX);
    double beta = (b*deltaT);
    //cout<<"c1: "<<c1<<endl<<"c2: "<<c2<<endl;
    for(int r=1; r<tPoints; r++)
    {
        for(int c=1; c<xPoints-1; c++)
        {
            point[c][r] = (alpha * (point[c-1][r-1] + point[c+1][r-1])) + ((1 - (2*alpha) - beta)*(po
            cout<<point[c][r]<<endl;
        }
        cout<<"----------\n";
    }

    double maxError = -1;
    for(int r=0; r<xPoints; r++)
        maxError = max(maxError,abs(point[r][tPoints-1] - getExact(x[r],b,D)));
    fprintf(output,"%f %f\n",m[index],maxError);

    }
    fclose(output);

}
```

Result



The implicit and explicit forms, both have truncation error O(Δt) + O(Δx^2). So fixing Δx we have a linear function in Δt which gives us straigt line for the error