Name : Kunal Ghosh
Course : M. Tech. (Course)
Subject : Numerical Solution of
Differential Equations
(DS289)

Assignment No. : 2
S.A.P. No. : 6000007645
S.R. No. : 05-01-00-10-42-22-1-21061

Problem (1) :

Consider a rectangular plate $R = \{(x,y) : 2 \leq x \leq 3, 4 \leq y \leq 6\}$ with the heat conduction equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

The boundary conditions are: $T(2,y) = 30$, $T(3,y) = 60$, $\left.\frac{\partial T}{\partial y}\right|_{(x,4)} = 0$ and

$$\left.\frac{\partial T}{\partial y}\right|_{(x,6)} = (T(x,6) - 60).$$

(a) Approximate the equation using second order central difference and obtain the algebric equation. (Use a 128 X 256 grid)

(b) Write the structure of the coefficient matrix.

(c) Solve the algebraic system using a direct solver. Standard linear solver can be used.

(d) Plot contours of the solution with appropriate x-y coordinates.

Answer (1)(a):

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \quad \text{— (1)}$$

or $$\left(\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2}\right)$$

$$+ \left(\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2}\right) = 0 \quad \text{— (2)}$$

NOTE: $T_{i,j} = T$ at the grid point (x_i, y_j)

$$\Delta x = \frac{x_u - x_l}{N_x - 1}$$

$x_u = 3$, $x_l = 2$,

N_x = Number of the grid points along x-direction.

$N_x = 128$

$$\Delta y = \frac{y_u - y_l}{N_y - 1}, \quad y_u = 6, \; y_l = 4$$

N_y = Number of the grid points along y-direction

$N_y = 256$

So, $\Delta x = 0.007874015 7$

$\Delta y = 0.007843137 3$

Using (2),

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

Let $m = \left(\frac{\Delta x}{\Delta y}\right)^2$

So,

$$T_{i+1,j} - 2T_{i,j} + T_{i-1,j} + m T_{i,j+1} - 2m T_{i,j} + m T_{i,j-1} = 0$$

or $$T_{i+1,j} - 2(m+1) T_{i,j} + T_{i-1,j} + m T_{i,j+1} + m T_{i,j-1} = 0$$

or $$m T_{i,j-1} + T_{i-1,j} - 2(m+1) T_{i,j} + T_{i+1,j} + m T_{i,j+1} = 0 \quad \text{---} (3)$$

This is the equation for all the NODEs which are at the interior and NOT on the boundaries. Rectangular plate.

Boundary conditions:

$T(2, y) = 30$

$T_{i,j} = 30$ (Tal the node number $(1, j)$)

(As $2 \leq x$)

$T(3, y) = 60$

So, $T_{N_x, j} = 60$

(T at the node number (N_x, j))

~~[As $N_x \leq 3$]~~ + [As $x \leq 3$]

$$\left.\frac{\partial T}{\partial y}\right|_{(x,4)} = 0$$

So, $$\left.\frac{\partial T}{\partial y}\right|_{(i,1)} = 0$$

($\frac{\partial T}{\partial y}$ at the node number $(i,1)$)

(As $y \geq 4$)

We will be using One-Sided Scheme, which is second order accurate.

Let $$\left.\frac{\partial T}{\partial y}\right|_{(i,j)} = a_0 T_{i,j} + a_1 T_{i,j+1} + a_2 T_{i,j+2}$$

$$T_{i,j+1} = T_{i,j} + \frac{(\Delta y)^1}{1!}\left(\frac{\partial T}{\partial y}\right)\Big|_{(i,j)} + \frac{(\Delta y)^2}{2!}\left(\frac{\partial^2 T}{\partial y^2}\right)\Big|_{(i,j)} + \text{H.O.T.}$$

$$T_{i,j+2} = T_{i,j} + \frac{(2\Delta y)^1}{1!}\left(\frac{\partial T}{\partial y}\right)\Big|_{(i,j)} + \frac{(2\Delta y)^2}{2!}\left(\frac{\partial^2 T}{\partial y^2}\right)\Big|_{(i,j)}$$

$$+ H.O.T.$$

Neglecting the H.O.T. (Higher Order Terms)

$$\left.\frac{\partial T}{\partial y}\right|_{(i,j)} = a_0 (T_{i,j})$$

$$+ a_1\left(T_{i,j} + \Delta y \left(\frac{\partial T}{\partial y}\right)\Big|_{(i,j)} + \frac{(\Delta y)^2}{2}\left(\frac{\partial^2 T}{\partial y^2}\right)\Big|_{(i,j)}\right)$$

$$+ a_2\left(T_{i,j} + 2\Delta y \left(\frac{\partial T}{\partial y}\right)\Big|_{(i,j)} + 2(\Delta y)^2\left(\frac{\partial^2 T}{\partial y^2}\right)\Big|_{(i,j)}\right)$$

So, $a_0 + a_1 + a_2 = 0$

~~$0a_0 + (\Delta y)a_1 + (2\Delta y)a_2 = 1$~~

$$0a_0 + (\Delta y)a_1 + (2\Delta y)\,a_2 = 1$$

$$0\,a_0 + \frac{(\Delta y)^2}{2}a_1 + 2(\Delta y)^2 a_2 = 0$$

So, $a_1 = -4a_2$

and $(\Delta y)a_1 + (2\Delta y)(a_2) = 1$

$\Rightarrow -(4\Delta y)a_2 + (2\Delta y)a_2 = 1$

$\Rightarrow -(2\Delta y)a_2 = 1$

$\Rightarrow a_2 = \frac{-1}{2\Delta y}$

So, $a_1 = -4 a_2$

$= -4 \left(\frac{-1}{2\Delta y}\right)$

$a_1 = \frac{4}{2\Delta y}$

So, $a_0 + a_1 + a_2 = 0$

$\Rightarrow a_0 = -a_1 - a_2$

$= \frac{-4 + 1}{2\Delta y}$

$\Rightarrow a_0 = \frac{-3}{2\Delta y}$

So, $\left.\frac{\partial T}{\partial y}\right|_{(i,j)} = \left(\frac{-3T_{i,j} + 4T_{i,j+1} - T_{i,j+2}}{2\Delta y}\right)$

So, $\left.\frac{\partial T}{\partial y}\right|_{(i,1)} = 0$

$\Rightarrow \frac{-3T_{i,1} + 4T_{i,2} - T_{i,3}}{2\Delta y} = 0$

$\Rightarrow -3T_{i,1} + 4T_{i,2} - T_{i,3} = 0$

$\Rightarrow 3T_{i,1} - 4T_{i,2} + T_{i,3} = 0$

$$\left.\frac{\partial T}{\partial y}\right|_{(x,6)} = T(x,6) - 60$$

So, $$\left.\frac{\partial T}{\partial y}\right|_{(i,N_y)} = T(x,6) - 60$$

$$= T_{i,N_y} - 60$$

or $$\left.\frac{\partial T}{\partial y}\right|_{(i,N_y)} = T_{i,N_y} - 60$$

($\frac{\partial T}{\partial y}$ at the node number (i,N_y))

[As $y \leq 6$]

We will be using one sided scheme which is second order accurate.

Let $$\left.\frac{\partial T}{\partial y}\right|_{(i,j)} = a_0 T_{i,j} + a_{-1} T_{i,j-1} + a_{-2} T_{i,j-2}$$

~~So, $T_{i,j-1} = T_{i,j} + (-\Delta y)^1 \frac{\partial T}{\partial y}$~~

So, $$T_{i,j-1} = T_{i,j} + \frac{(-\Delta y)^1}{1!}\left(\frac{\partial T}{\partial y}\right)\Big|_{(i,j)}$$

$$+ \frac{(-\Delta y)^2}{2!}\left(\frac{\partial^2 T}{\partial y^2}\right)\Big|_{(i,j)} + H.O.T.$$

So,

$$T_{i,j-2} = T_{i,j} + \frac{(-2\Delta y)'}{1!}\left(\frac{\partial T}{\partial y}\right)\Big|_{(i,j)} + \frac{(-2\Delta y)^2}{2!}\left(\frac{\partial^2 T}{\partial y^2}\right)\Big|_{(i,j)} + H.O.T.$$

Neglecting the H.O.T. (Higher Order Terms)

$$\left.\frac{\partial T}{\partial y}\right|_{(i,j)} = a_0 T_{i,j} + a_{-1}\left(T_{i,j} - (\Delta y)\left.\frac{\partial T}{\partial y}\right|_{(i,j)} + \frac{(\Delta y)^2}{2}\left.\frac{\partial^2 T}{\partial y^2}\right|_{(i,j)}\right) + a_{-2}\left(T_{i,j} - (2\Delta y)\left.\frac{\partial T}{\partial y}\right|_{(i,j)} + 2(\Delta y)^2\left.\frac{\partial^2 T}{\partial y^2}\right|_{(i,j)}\right)$$

So, $a_0 + a_{-1} + a_{-2} = 0$

$0a_0 - (\Delta y)a_{-1} - (2\Delta y)a_{-2} = 1$

$0a_0 + \frac{(\Delta y)^2}{2}a_{-1} + 2(\Delta y)^2 a_{-2} = 0$

So, $a_{-1} = -4a_{-2}$

So, $(-\Delta y)a_{-1} - (2\Delta y)a_{-2} = 1$

$\Rightarrow (-\Delta y)(-4a_{-2}) - (2\Delta y)a_{-2} = 1$

$\Rightarrow (2\Delta y)\, a_{-2} = 1$

$\Rightarrow a_{-2} = \left(\frac{1}{2\Delta y}\right)$

So, $a_{-1} = \frac{-4}{2\Delta y}$

So, $a_0 = -a_{-1} - a_{-2}$

$\Rightarrow a_0 = \frac{4 - 1}{2\Delta y}$

$\Rightarrow a_0 = \frac{3}{2\Delta y}$

So, $\left.\frac{\partial T}{\partial y}\right|_{(i,j)} = \frac{3T_{i,j} - 4T_{i,j-1} + T_{i,j-2}}{(2\Delta y)}$

So, $\left.\frac{\partial T}{\partial y}\right|_{(i,N_y)} = \frac{3T_{i,N_y} - 4T_{i,N_y-1} + T_{i,N_y-2}}{2\Delta y}$

$= T_{i,N_y} - 60$

$\Rightarrow 3T_{i,N_y} - 4T_{i,N_y-1} + T_{i,N_y-2} = (2\Delta y)\, T_{i,N_y} - 120\Delta y$

$\Rightarrow (3 - 2\Delta y)\, T_{i,N_y} - 4T_{i,N_y-1} + T_{i,N_y-2} = -120\Delta y$

Algebraic Equations:

$$m\,T_{i,j-1} + T_{i-1,j} - 2(m+1)\,T_{i,j} + T_{i+1,j} + m\,T_{i,j+1} = 0$$

$$T_{1,j} = 30$$

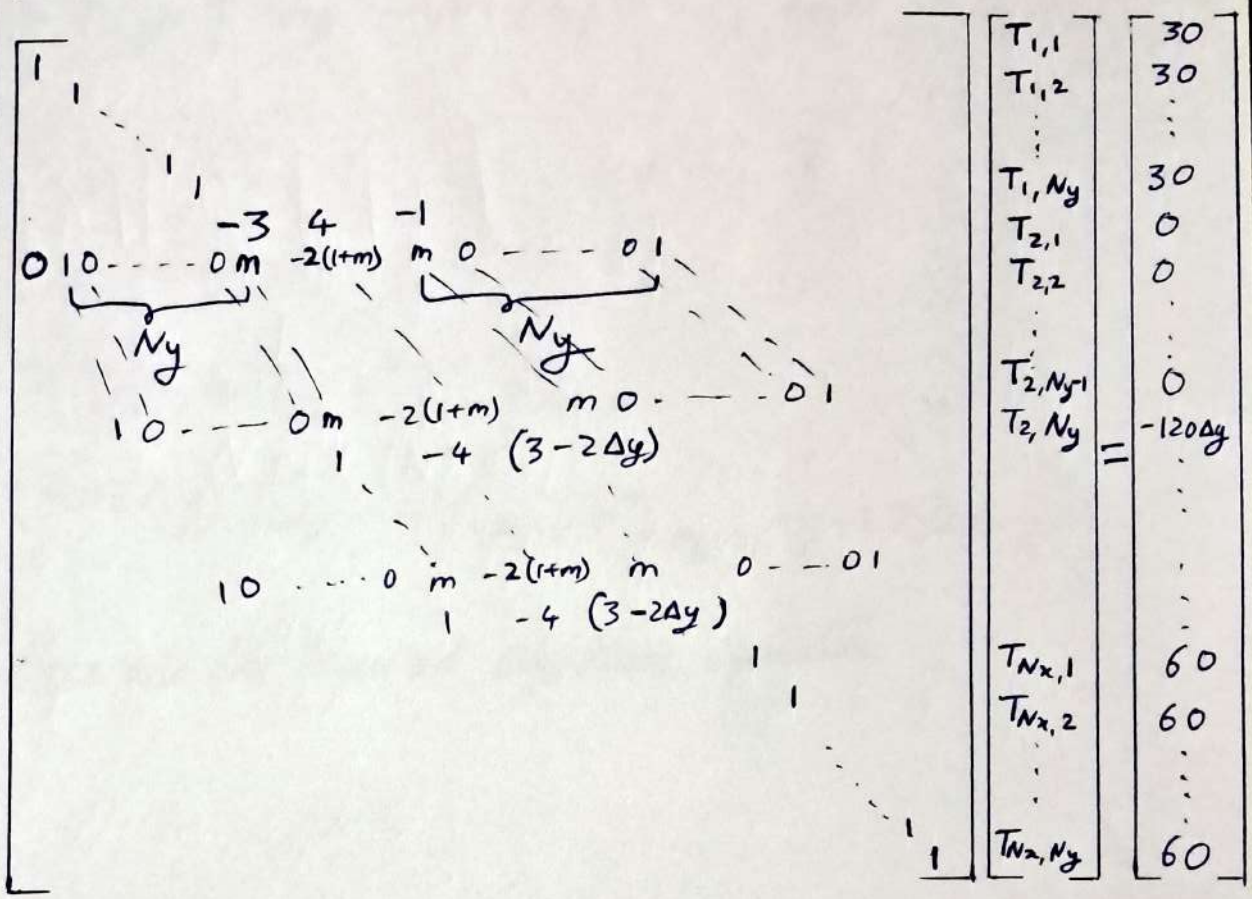
$$T_{N_x,j} = 60$$

$$3\,T_{i,1} - 4\,T_{i,2} + T_{i,3} = 0$$

$$(3 - 2\Delta y)\,T_{i,N_y} - 4\,T_{i,N_y-1} + T_{i,N_y-2} = -120\Delta y$$

These are the required algebraic equations.

Answer (1)(b):



This is the structure of the Coefficient Matrix.

Answer (1)(c):

We have solved the ~~eqn~~ algebraic system using LAPACK library. Please check the file 01.cpp.

Post_Processing

April 11, 2023

1 Question (1) (d):

1.1 Plot contours of the solution with appropriate x-y coordinates.

2 Answer (1) (d):

3 Import the necessary libraries

[1]:
```
import numpy as np;
import matplotlib.pyplot as plt;
import seaborn as sns;
import pandas as pd;
```

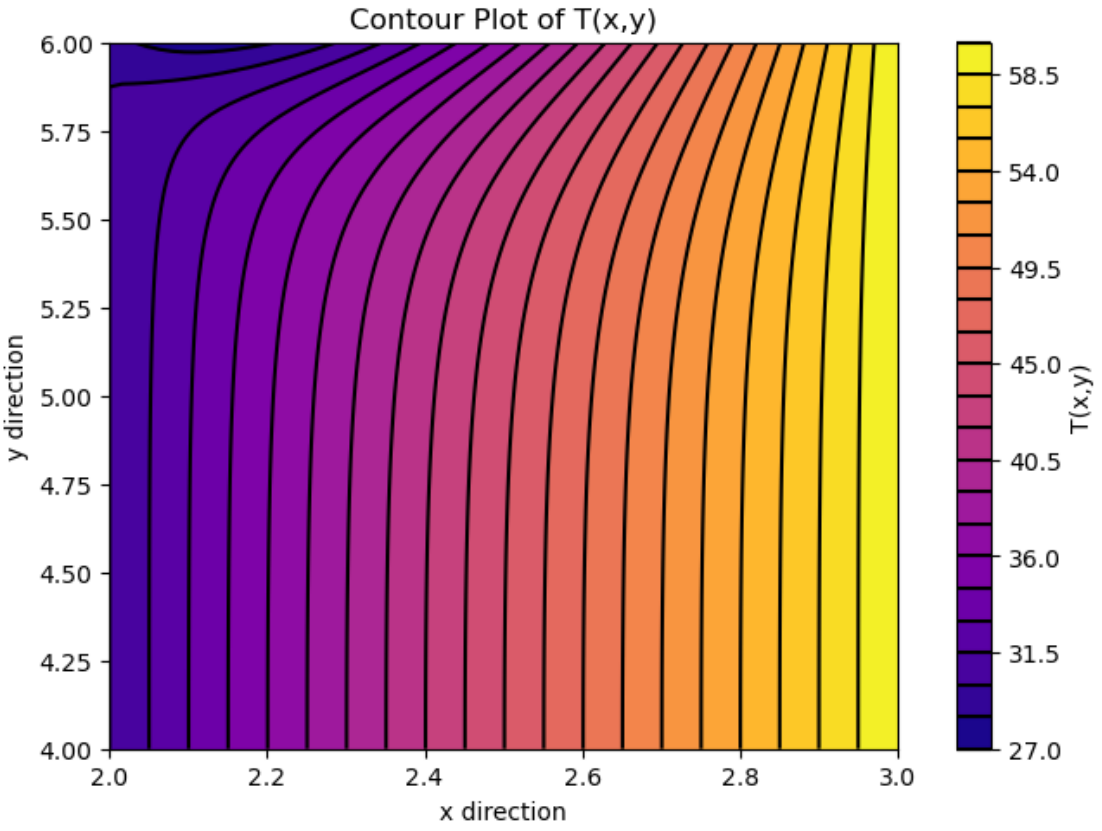
4 Reading the solution files

[2]:
```
Sol = pd.read_csv("Question_1_Solution.csv",header=None);
```

5 Plotting the contour plot

[3]:
```
xlist = np.linspace(2.0, 3.0, Sol.shape[1])
ylist = np.linspace(4, 6.0, Sol.shape[0])
X, Y = np.meshgrid(xlist, ylist)
fig1, ax2 = plt.subplots(layout='constrained')
CS = ax2.contourf(X, Y, Sol, 25, cmap='plasma')
CS2 = ax2.contour(CS, levels=CS.levels[::1], colors='k')

ax2.set_title('Contour Plot of T(x,y)')
ax2.set_xlabel('x direction')
ax2.set_ylabel('y direction')
cbar = fig1.colorbar(CS)
cbar.ax.set_ylabel('T(x,y)')
cbar.add_lines(CS2)
fig1.savefig("Question_1_Contour_Plot.png",dpi= 1000)
```



5.1 In some regions of the computational domain, T(x,y) is less than 30 (in the upper left corner of the contour plot). This is happening because of the negative value of $\frac{\partial T}{\partial y}$ in that region.

Problem (2):

Show that a two level scheme with $\theta \geq 0.5$ for time derivative and a second order central difference for space derivative results in an unconditionally stable method for diffusion equation.

Answer (2):

Diffusion Equation :

$$u_t = \alpha u_{xx} \quad \text{---} (1)$$

So, $$\frac{u_i^{n+1} - u_i^n}{\Delta t}$$

$$= \alpha \left[(1-\theta) \left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} \right) + (\theta) \left(\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} \right) \right] \quad \text{---} (2)$$

Using Von-Neumann Stability,

Let $u_j^n = v^n e^{ikx_j}$ ---- (3)

Using ② and ③,

$$\left(\frac{V^{n+1}e^{ikx_j} - V^n e^{ikx_j}}{\Delta t}\right)$$

$$= \alpha\left[(1-\theta)\left(\frac{V^n e^{ikx_{j+1}} - 2V^n e^{ikx_j} + V^n e^{ikx_{j-1}}}{(\Delta x)^2}\right)\right.$$

$$\left. + (\theta)\left(\frac{V^{n+1}e^{ikx_{j+1}} - 2V^{n+1}e^{ikx_j} + V^{n+1}e^{ikx_{j-1}}}{(\Delta x)^2}\right)\right]$$

[NOTE: $x_{j+1} = x_j + \Delta x$
$x_{j-1} = x_j - \Delta x$]

So, $\frac{V^n e^{ikx_j}}{(\Delta t)}(V-1)$

$$= \frac{\alpha V^n e^{ikx_j}}{(\Delta x)^2}\left[(1-\theta)\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\right.$$
$$\left. + (\theta)(V)\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\right] \quad \text{④}$$

As, $e^{ik\Delta x} - 2 + e^{-ik\Delta x}$

$= \cos(k\Delta x) + i\sin(k\Delta x) - 2 + \cos(k\Delta x) - i\sin(k\Delta x)$

$= 2(\cos(k\Delta x) - 1)$

So, $e^{ik\Delta x} - 2 + e^{-ik\Delta x} = 2\left(1 - 2\sin^2\left(\frac{k\Delta x}{2}\right) - 1\right)$

or $e^{ik\Delta x} - 2 + e^{-ik\Delta x} = -4\sin^2\left(\frac{k\Delta x}{2}\right)$ — ⑤

Using ④ & ⑤,

$$\frac{v^n e^{ikx_j}}{(\Delta t)}(v-1)$$

$$= \frac{\alpha v^n e^{ikx_j}}{(\Delta x)^2}\left[(1-\theta)\left(-4\sin^2\left(\frac{k\Delta x}{2}\right)\right) + (v)(\theta)\left(-4\sin^2\left(\frac{k\Delta x}{2}\right)\right)\right]$$

or $v - 1 = \left(\frac{\alpha \Delta t}{(\Delta x)^2}\right)\left[(1-\theta)\left(-4\sin^2\left(\frac{k\Delta x}{2}\right)\right) + (v)(\theta)\left(-4\sin^2\left(\frac{k\Delta x}{2}\right)\right)\right]$

Let $m = \frac{\alpha \Delta t}{(\Delta x)^2}$

So,

$$v - 1 = -4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right) - 4m\theta v \sin^2\left(\frac{k\Delta x}{2}\right)$$

or $v + 4m\theta v \sin^2\left(\frac{k\Delta x}{2}\right) = 1 - 4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)$

or $v\left[1 + 4m\theta \sin^2\left(\frac{k\Delta x}{2}\right)\right] = 1 - 4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)$

or $v = \left[\frac{\left[1 - 4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)\right]}{\left[1 + 4m\theta \sin^2\left(\frac{k\Delta x}{2}\right)\right]}\right]$

For the scheme to be unconditionally stable

$$|v| \leq 1$$

So, $-1 \leq v \leq 1$

So, $$-1 \leq \left[\frac{1-4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)}{1+4m\theta\sin^2\left(\frac{k\Delta x}{2}\right)}\right] \leq 1$$

or $$\frac{1-4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)}{1+4m\theta\sin^2\left(\frac{k\Delta x}{2}\right)} \leq 1$$

or $$1-4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right) \leq 1+4m\theta\sin^2\left(\frac{k\Delta x}{2}\right)$$

or $$-4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right) \leq 4m\theta\sin^2\left(\frac{k\Delta x}{2}\right)$$

or $$(\theta-1)\left(4m\sin^2\left(\frac{k\Delta x}{2}\right)\right) \leq \theta\left(4m\sin^2\left(\frac{k\Delta x}{2}\right)\right)$$

or $\theta \geq (\theta-1)$ (Always True)

[NOTE: $\alpha > 0$, So, $m = \frac{\alpha\Delta t}{(\Delta x)^2} > 0$]

Similarly,

$$-1 \leq \left[\frac{1-4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)}{1+4m\theta\sin^2\left(\frac{k\Delta x}{2}\right)}\right]$$

or $$-1-4m\theta\sin^2\left(\frac{k\Delta x}{2}\right) \leq 1-4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right)$$

So, $-1-1 \leq -4m(1-\theta)\sin^2\left(\frac{k\Delta x}{2}\right) + 4m\theta\sin^2\left(\frac{k\Delta x}{2}\right)$

or $-2 \leq 4m(\theta-(1-\theta))\sin^2\left(\frac{k\Delta x}{2}\right)$

or $-2 \leq 4m(2\theta-1)\sin^2\left(\frac{k\Delta x}{2}\right)$

or $-2 \leq \left[4m\sin^2\left(\frac{k\Delta x}{2}\right)\right][2\theta-1]$

[As $\alpha > 0$

So, $m = \frac{\alpha\Delta t}{(\Delta x)^2} > 0$

So, $4m\sin^2\left(\frac{k\Delta x}{2}\right) > 0$]

So, If the R.H.S of the above inequality is positive then the inequality holds.

As $4m\sin^2\left(\frac{k\Delta x}{2}\right) \geq 0$

So, $(2\theta-1)$ need to be positive or zero.

So,

$2\theta-1 \geq 0$

or $\theta \geq \frac{1}{2}$

So, the scheme is unconditionally stable if and only if $\theta \geq 0.5$.

Problem ③ :

ⓐ Use the explicit Euler and second order central difference schemes to discretize the diffusion equation ($u_t = \alpha u_{xx}$) and obtain the modified equation

ⓑ Comment on the (dissipative and dispersive) nature of errors.

Answer ③ ⓐ :

Diffusion Equation :

$$u_t = \alpha u_{xx} \qquad \text{——— (1)}$$

Explicit Euler Scheme :

$$u_t = \frac{u_i^{n+1} - u_i^n}{\Delta t} \qquad \text{——— (2)}$$

Second Order Central Difference Scheme :

$$u_{xx} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} \qquad \text{——— (3)}$$

Using ①, ② and ③,

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} \right)$$

or $u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{(\Delta x)^2} \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n \right)$ — ④

This is the discretized diffusion equation using explicit Euler and second order central difference schemes.

[NOTE :- u_j^m = Value of u at mth time step and jth grid point]

Let $r = \frac{\alpha \Delta t}{(\Delta x)^2}$ — ⑤

So, $u_i^{n+1} = u_i^n + r \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n \right)$ — ⑥

Using Taylor Series,

$$u_i^{n+1} = u_i^n + \frac{\dot{u}_i^n (\Delta t)^1}{1!} + \frac{\ddot{u}_i^n (\Delta t)^2}{2!} + \frac{\dddot{u}_i^n (\Delta t)^3}{3!} + \frac{\ddddot{u}_i^n (\Delta t)^4}{4!} + \text{H.O.T.}$$ — ⑦

NOTE: $\dot{u} = \frac{\partial u}{\partial t}$, $\ddot{u} = \frac{\partial^2 u}{\partial t^2}$

$\dddot{u} = \frac{\partial^3 u}{\partial t^3}$, $\ddddot{u} = \frac{\partial^4 u}{\partial t^4}$, ...

Similarly,

$$u_{i+1}^n = u_i^n + \frac{(u_i^n)'(\Delta x)^1}{1!} + \frac{(u_i^n)''(\Delta x)^2}{2!} + \frac{(u_i^n)'''(\Delta x)^3}{3!} + \frac{(u_i^n)''''(\Delta x)^4}{4!} + H.O.T. \quad \text{—— (8)}$$

$$u_{i-1}^n = u_i^n - \frac{(u_i^n)'(\Delta x)^1}{1!} + \frac{(u_i^n)''(\Delta x)^2}{2!} - \frac{(u_i^n)'''(\Delta x)^3}{3!} + \frac{(u_i^n)''''(\Delta x)^4}{4!} - H.O.T.. \quad \text{—— (9)}$$

NOTE: $u' = \frac{\partial u}{\partial x}$, $u'' = \frac{\partial^2 u}{\partial x^2}$,

$u''' = \frac{\partial^3 u}{\partial x^3}$, $u'''' = \frac{\partial^4 u}{\partial x^4}$, ...

We will be substituting the Taylor Series expansion of u_i^{n+1}, u_{i+1}^n and u_{i-1}^n into the equation (6).

So,

$$u_i^n + \frac{\dot{u}_i^n (\Delta t)^1}{1!} + \frac{\ddot{u}_i^n (\Delta t)^2}{2!} + \frac{\dddot{u}_i^n (\Delta t)^3}{3!}$$

$$+ \frac{\ddddot{u}_i^n (\Delta t)^4}{4!} + H.O.T.$$

$$= u_i^n$$

$$+ \nu \left[u_i^n + \frac{(u_i^n)' (\Delta x)^1}{1!} + \frac{(u_i^n)'' (\Delta x)^2}{2!} + \frac{(u_i^n)''' (\Delta x)^3}{3!} + \right.$$

$$+ \frac{(u_i^n)'''' (\Delta x)^4}{4!} + H.O.T.$$

$$- 2u_i^n$$

$$\left. + u_i^n - \frac{(u_i^n)' (\Delta x)^1}{1!} + \frac{(u_i^n)'' (\Delta x)^2}{2!} - \frac{(u_i^n)''' (\Delta x)^3}{3!} \right.$$

$$\left. + \frac{(u_i^n)'''' (\Delta x)^4}{4!} - H.O.T. \right]$$

$$\text{or } \frac{\dot{u}_i^n (\Delta t)^1}{1!} + \frac{\ddot{u}_i^n (\Delta t)^2}{2!} + \frac{\dddot{u}_i^n (\Delta t)^3}{3!} + \frac{\ddddot{u}_i^n (\Delta t)^4}{4!} + H.O.T.$$

$$= \nu \left[2 \left(\frac{(u_i^n)'' (\Delta x)^2}{2!} + \frac{(u_i^n)'''' (\Delta x)^4}{4!} + H.O.T \right) \right]$$

[NOTE: Only the even derivatives of space remains in R.H.S. Odd derivatives of space get cancelled out]

$$\text{or } \dot{u}_i^n(\Delta t) + \frac{\ddot{u}_i^n(\Delta t)^2}{2!} + \frac{\dddot{u}_i^n(\Delta t)^3}{3!} + \frac{\ddddot{u}_i^n(\Delta t)^4}{4!} + H.O.T.$$

$$= \nu\left[(u_i^n)''(\Delta x)^2 + 2\frac{(u_i^n)''''(\Delta x)^4}{4!} + H.O.T.\right]$$

$$\text{or } \dot{u}_i^n + \frac{\ddot{u}_i^n(\Delta t)}{2!} + \frac{\dddot{u}_i^n(\Delta t)^2}{3!} + \frac{\ddddot{u}_i^n(\Delta t)^3}{4!} + H.O.T.$$

$$= \frac{(\Delta x)^2}{(\Delta t)}\nu\left[(u_i^n)'' + 2\frac{(u_i^n)''''(\Delta x)^2}{4!} + H.O.T.\right]$$

Using (5), $\nu = \frac{\alpha \Delta t}{(\Delta x)^2}$

$$\text{or } \dot{u}_i^n + \frac{\ddot{u}_i^n(\Delta t)}{2!} + \frac{\ddot{u}_i^n(\Delta t)^2}{3!} + \frac{\dddot{u}_i^n(\Delta t)^3}{4!} + H.O.T.$$

$$= \alpha\left[(u_i^n)'' + \frac{2(u_i^n)''''(\Delta x)^2}{4!} + H.O.T.\right]$$

$$\text{or } \dot{u}_i^n - \alpha(u_i^n)'' = -\frac{\ddot{u}_i^n(\Delta t)}{2!} - \frac{\dddot{u}_i^n(\Delta t)^2}{3!}$$

$$- \frac{\dddot{u}_i^n(\Delta t)^3}{4!} - H.O.T$$

$$+ \alpha\left[\frac{2(u_i^n)''''(\Delta x)^2}{4!} + H.O.T\right]$$

Considering $u_t = \alpha u_{xx}$

So, $\dot{u} = \alpha u''$

or $\frac{\partial}{\partial t}(\dot{u}) = \frac{\partial}{\partial t}(\alpha u'')$

or $\ddot{u} = \frac{\partial}{\partial t}\left(\alpha \frac{\partial^2 u}{\partial x^2}\right)$

$$= \alpha \frac{\partial^3 u}{\partial t \partial x^2}$$

$$= \alpha \frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial t}\right)$$

$$= \alpha \frac{\partial^2}{\partial x^2}(\dot{u})$$

$$= \alpha \frac{\partial^2 (\alpha u'')}{\partial x^2}$$

or $\ddot{u} = \alpha^2 u''''$ ——— (11)

So, $\ddot{u}_i^n = \alpha^2 (u_i^n)''''$ ——— (12)

Using (10) and (12),

$$\dot{u}_i^n - \alpha (u_i^n)'' = -\frac{\alpha^2 (u_i^n)'''' (\Delta t)}{2!} - \frac{\dddot{u}_i^n (\Delta t)^2}{3!}$$

$$- \frac{\ddddot{u}_i^n (\Delta t)^3}{4!}$$

$$+ \alpha \left[\frac{2 (u_i^n)'''' (\Delta x)^2}{4!} + H.O.T \right]$$

$$\text{or } u_i^n - \alpha (u_i^n)''$$

$$= -\frac{\alpha^2 (u_i^n)''''(\Delta t)}{2!} + O((\Delta t)^2)$$

$$+ \alpha \left[\frac{2 (u_i^n)''''(\Delta x)^2}{4!} + H.O.T \right]$$

$$= (u_i^n)'''' \left[-\frac{\alpha^2 (\Delta t)}{2!} + \frac{2\alpha (\Delta x)^2}{4!} \right] + O((\Delta t)^2) + O((\Delta x)^4)$$

$$\text{or } u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[O(\Delta t) + O((\Delta x)^2) \right]$$

(Neglecting the higher order terms of (Δt) and (Δx))

So the modified equation is

$$u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[-\frac{\alpha^2 (\Delta t)}{2!} + \frac{2\alpha (\Delta x)^2}{4!} \right]$$

Answer (3)(b):

We are having a even order derivative of u in space (4th order derivative, $(u_i^n)''''$) with the leading order terms of Δt and Δx in the modified equation.

So, the nature of the error will be dissipative.

Considering the equation (10) of answer (3)(a).

$$\dot{u}_i^n - \alpha (u_i^n)'' = -\frac{\ddot{u}_i^n (\Delta t)}{2!} - \frac{\dddot{u}_i^n (\Delta t)^2}{3!} - \frac{\ddddot{u}_i^n (\Delta t)^3}{4!} + O((\Delta t)^4) + \alpha \left[\frac{2 (u_i^n)'''' (\Delta x)^2}{4!} + H.O.T. \right]$$

In this equation we have only even derivative of space.

Consider any derivative of time from the R.H.S.

So, $\frac{\partial^m u_i^n}{\partial t^m} = \frac{\partial^{m-1}}{\partial t^{m-1}} \left(\frac{\partial u}{\partial t} \right)$ $\quad m \in \{2, 3, \dots\}$

So, $\frac{\partial^m u_i^n}{\partial t^m} = \frac{\partial^{m-1}}{\partial t^{m-1}} \left(\alpha (u_i^n)'' \right)$

[As, $\dot{u} = \alpha u''$]

So, $\frac{\partial^m u_i^n}{\partial t^m} = \alpha \frac{\partial^2}{\partial x^2} \left(\frac{\partial^{m-1}}{\partial t^{m-1}} u_i^n \right)$

We can continue this process.

So, we will be getting,

$$\frac{\partial^m (u_i^n)}{\partial t^m} = \alpha \frac{\partial^{2m}}{\partial x^{2m}} (u_i^n)$$

(So, we are have an even order derivative of u_i^n in space, for any $m = 2, 3, 4, \ldots$)

So, all the terms in the R.H.S. of the equation (10) of answer (3)(a) can be expressed as a even derivative of space.

So, in the modified equation there is NO odd order derivative of space.

So, the nature of the errors will NOT be dispersive.

Problem (4):

(a) Derive the analytical solution for the equation in question (3) (use $\alpha = 1.5$). The initial condition is $u(x,0) = \sin(4x) + \sin(x)$. The domain size is 2π with a periodic boundary condition.

(b) Perform numerical experiments with a constant r_d of $\frac{1}{2}$ and $\frac{1}{6}$ ($\in 0.166667$) and grid sizes $N = \{32, 64, 128, 256\}$ Here r_d is the stability parameter. In each simulation, evolve the solution to the end time $t_{end} = 0.4$.

(c) Use the analytical solution to compute the average of absolute error ($E(N)$) in each similation at $t = t_{end}$.

(d) In a graph with logarithmic scale plot (N) vs $E(N)$ for $r_d = \frac{1}{2}$ and $\frac{1}{6}$. Obtain the order of accuracy for the two r_d cases.

(e) Explain the abnormal order of accuracy observed in $r_d = \frac{1}{6}$ case.

Answer (4)(a):

$$u_t = \alpha u_{xx} \quad \text{---} (1)$$

Let the analytical solution be of the form

$$u(x,t) = \sum_{n=1}^{\infty} a_n \sin(nx) f(t) \quad \text{---} (2)$$

Using ① and ②

$$\sum_{n=1}^{\infty} a_n \sin(nx)\, \dot{f}(t) = \left[\sum_{n=1}^{\infty} a_n \sin(nx)\right]\left[\dot{f}(t)\right]$$

This is the L.H.S. of equation ①.

Considering the RHS of equation ①,

$$\alpha u_{xx}$$

$$= \alpha \frac{\partial^2}{\partial x^2}\left[\sum_{n=1}^{\infty} a_n \sin(nx) f(t)\right]$$

$$= -n^2 \alpha \left[\sum_{n=1}^{\infty} a_n \sin(nx) f(t)\right]$$

$$\text{or } \alpha u_{xx} = \left[\alpha f(t)\right]\left[\sum_{n=1}^{\infty}\left[a_n\left(-n^2 \sin(nx)\right)\right]\right]$$

$$\text{or } \alpha u_{xx} = -\alpha f(t)\left[\sum_{n=1}^{\infty}\left(a_n n^2 \sin(nx)\right)\right]$$

Equating the LHS and R.H.S. of equation ①

$$\left[\sum_{n=1}^{\infty} a_n \sin(nx)\right]\left[\dot{f}(t)\right] = -\alpha f(t)\left[\sum_{n=1}^{\infty}\left(a_n n^2 \sin(nx)\right)\right]$$

$$\text{or } \sum_{n=1}^{\infty}\left(a_n \dot{f}(t) \sin(nx)\right) = \sum_{n=1}^{\infty}\left(-a_n n^2 \alpha f(t) \sin(nx)\right)$$

$$\text{or } \sum_{n=1}^{\infty}\left(\left(\dot{f}(t)\right) a_n \sin(nx)\right).$$

$$= \sum_{n=1}^{\infty}\left(\left(-n^2\alpha f(t)\right) a_n \sin(nx)\right)$$

Equating the coefficients of $a_n \sin(nx)$ on LHS and RHS.

$$\dot{f}(t) = -n^2\alpha f(t) \qquad (n = 1, 2, \cdots, \infty)$$

$$\text{or } \frac{df(t)}{dt} = -n^2\alpha f(t)$$

$$\text{or } \frac{df(t)}{f(t)} = -n^2\alpha\, dt$$

$$\text{or } \ln\left(\frac{f(t)}{f(0)}\right) = -n^2\alpha(t-0)$$

$$\text{or } f(t) = f(0)\, e^{-n^2\alpha t}$$

Using ②,

$$u(x,t) = \sum_{n=1}^{\infty}\left[\left(a_n \sin(nx)\right)\left(f(0)e^{-n^2\alpha t}\right)\right]$$

Using the initial conditions

$$u(x,0) = \sin(4x) + \sin(x)$$

$$\text{So, } \sum_{n=1}^{\infty}\left[\left(a_n \sin(nx)\right)\left(f(0)e^{0}\right)\right] = \sin(4x) + \sin(x)$$

So, $$\sum_{n=1}^{\infty} \left[(f(0)\, a_n) \sin(nx) \right] = \sin(4x) + 8\sin(x)$$

On comparing the coefficients of $\sin(nx)$ on both sides.

$f(0)\, a_4 = 1$

$f(0)\, a_1 = 1$

$f(0)\, a_n = 0 \qquad n \neq 1, n \neq 4$.

So,

$$u(x,t) = \sum_{n=1}^{\infty} \left[(a_n \sin(nx)) \left(f(0) e^{-n^2 \alpha t} \right) \right]$$

$$u(x,t) = \sum_{n=1}^{\infty} \left[a_n f(0) (\sin(nx))\, e^{-n^2 \alpha t} \right]$$

So,

$$u(x,t) = (\sin(4x))\, e^{-4^2 \alpha t} + (\sin(x))\, e^{-\alpha t}$$

or $$u(x,t) = e^{-16\alpha t} \sin(4x) + e^{-\alpha t} \sin(x)$$

As $\alpha = 1.5$

So,

$$u(x,t) = e^{-24t} \sin(4x) + e^{-1.5t} \sin(x)$$

This is the analytical solution.

Answer (4)(b);

$$u_t = \alpha u_{xx}$$

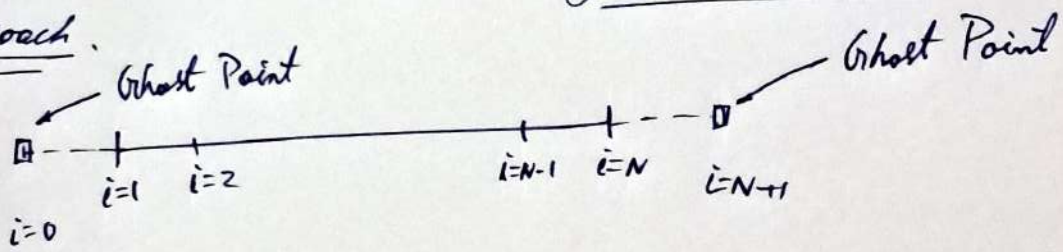
$$r_d = \frac{\alpha \Delta t}{(\Delta x)^2}$$

As derived in Answer (3)(a) (Equation -(4))

$$u_i^{n+1} = u_i^n + r_d \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)$$

$$\Rightarrow u_i^{n+1} = r_d u_{i-1}^n + (1-2r_d) u_i^n + r_d u_{i+1}^n$$

As we are having periodic boundary condition. So, we will be using ghost point approach.



~~$u_N = u_1$~~

$u_0 = u_{N-1}$

$u_1 = u_N$

$u_2 = u_{N+1}$ (Using the ghost point approach)

We will be marching from nth time step to $(n+1)$th step using the above equations.

Post_processing

April 11, 2023

1 Question (4) (b):

1.1 Perform numerical experiments with a constant rd of 1/2 and 1/6(= 0.166667), and grid sizes N = {32, 64, 128, 256}. Here, rd is the stability parameter. In each simulation, evolve the solution to an end time $t_{end} = 0.4$.

2 Answer (4) (b):

3 Import the necessary libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

4 Reading the name of the files generated

[2]:
```
Output_files = pd.read_csv("Output_file_names.csv",delimiter=",",header=None).
 ↪to_numpy()
Output_files = np.squeeze(Output_files)
Output_files = Output_files.tolist()
```

4.1 Output files generated

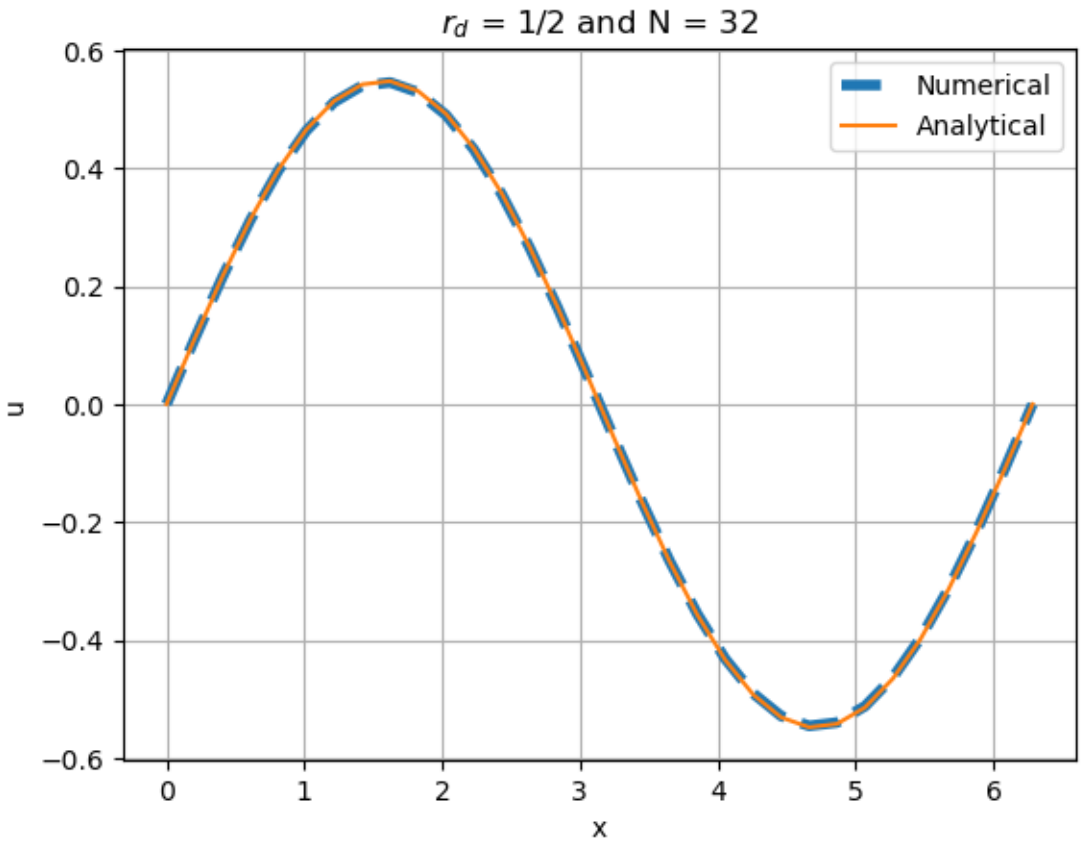
[3]:
```
Output_files
```

[3]:
```
['Question_4_Explicit_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Question_4_Explicit_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Question_4_Explicit_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Question_4_Explicit_u_n_t_0.400000_N_64_rd_0.166667_.csv',
```

```
'Analytical_Solution_u_n_t_0.400000_N_64_rd_0.166667_.csv',
'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_64_rd_0.166667_.csv',
'Question_4_Explicit_u_n_t_0.400000_N_128_rd_0.500000_.csv',
'Analytical_Solution_u_n_t_0.400000_N_128_rd_0.500000_.csv',
'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_128_rd_0.500000_.csv',
'Question_4_Explicit_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Analytical_Solution_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Question_4_Explicit_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Analytical_Solution_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Question_4_Explicit_u_n_t_0.400000_N_256_rd_0.166667_.csv',
'Analytical_Solution_u_n_t_0.400000_N_256_rd_0.166667_.csv',
'Question_4_Explicit_Average_Error_u_n_t_0.400000_N_256_rd_0.166667_.csv']
```

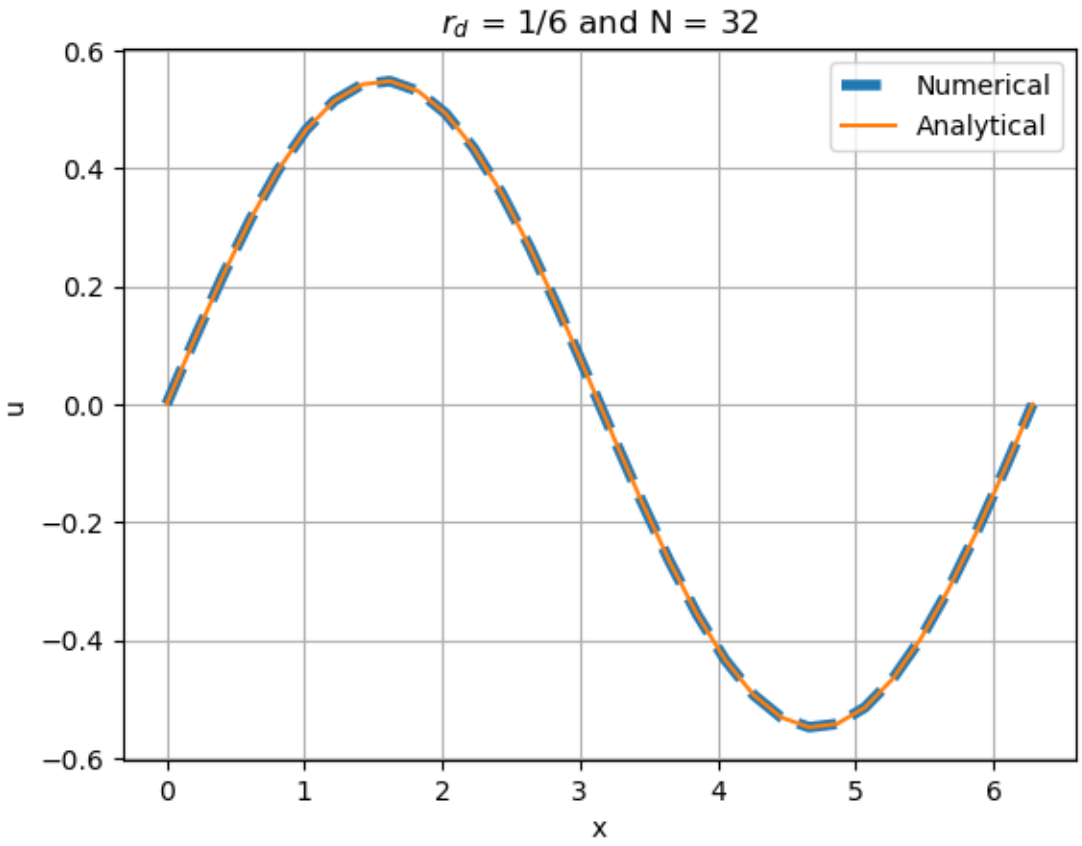
4.2 Plotting the results for rd = 1/2 and N = 32 at $t_{end} = 0.4$

[4]:
```
Numerical_Solution_N_32_rd_half = pd.read_csv(Output_files[0], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_32_rd_half = pd.read_csv(Output_files[1], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_32_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_32_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_32_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 32")
plt.grid()
plt.savefig("r_d_1_2_and_N_32.png",dpi = 1000)
plt.show()
```



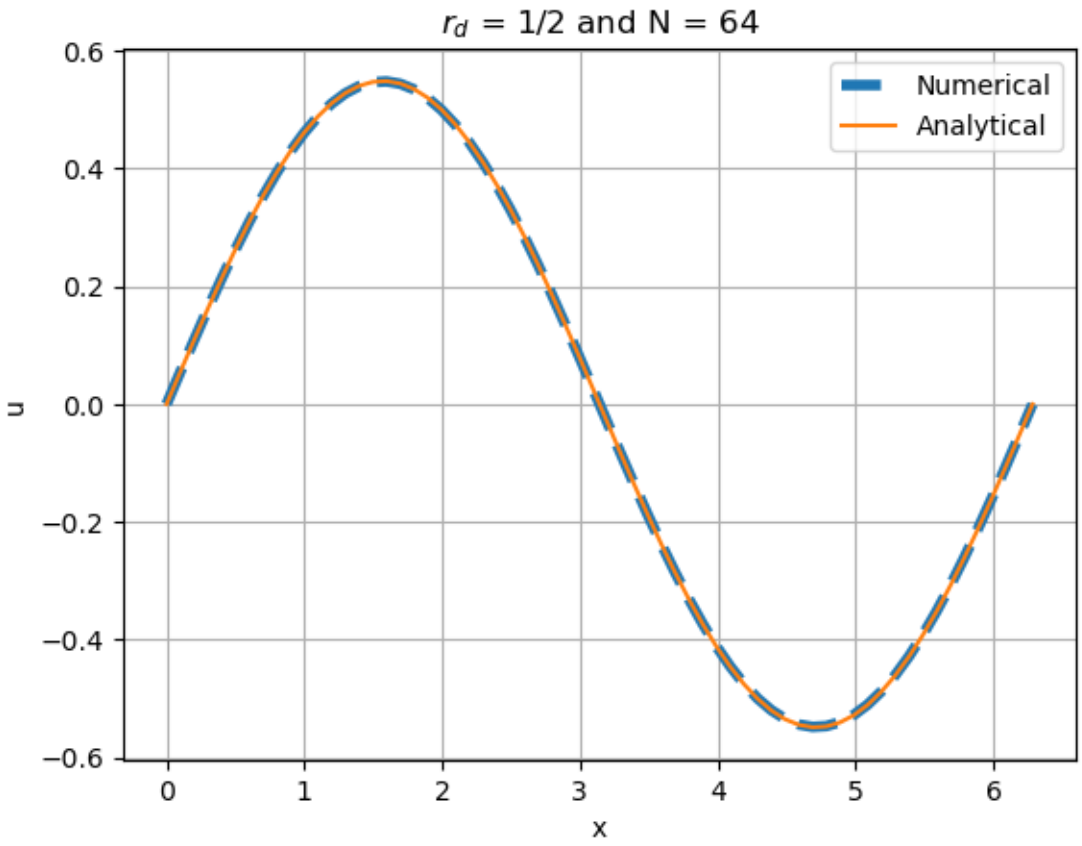
4.3 Plotting the results for rd = 1/6 and N = 32 at $t_{end} = 0.4$

[5]:
```
Numerical_Solution_N_32_rd_1_6 = pd.read_csv(Output_files[3], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_32_rd_1_6 = pd.read_csv(Output_files[4], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_32_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_32_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_32_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 32")
plt.grid()
plt.savefig("r_d_1_6_and_N_32.png",dpi = 1000)
plt.show()
```



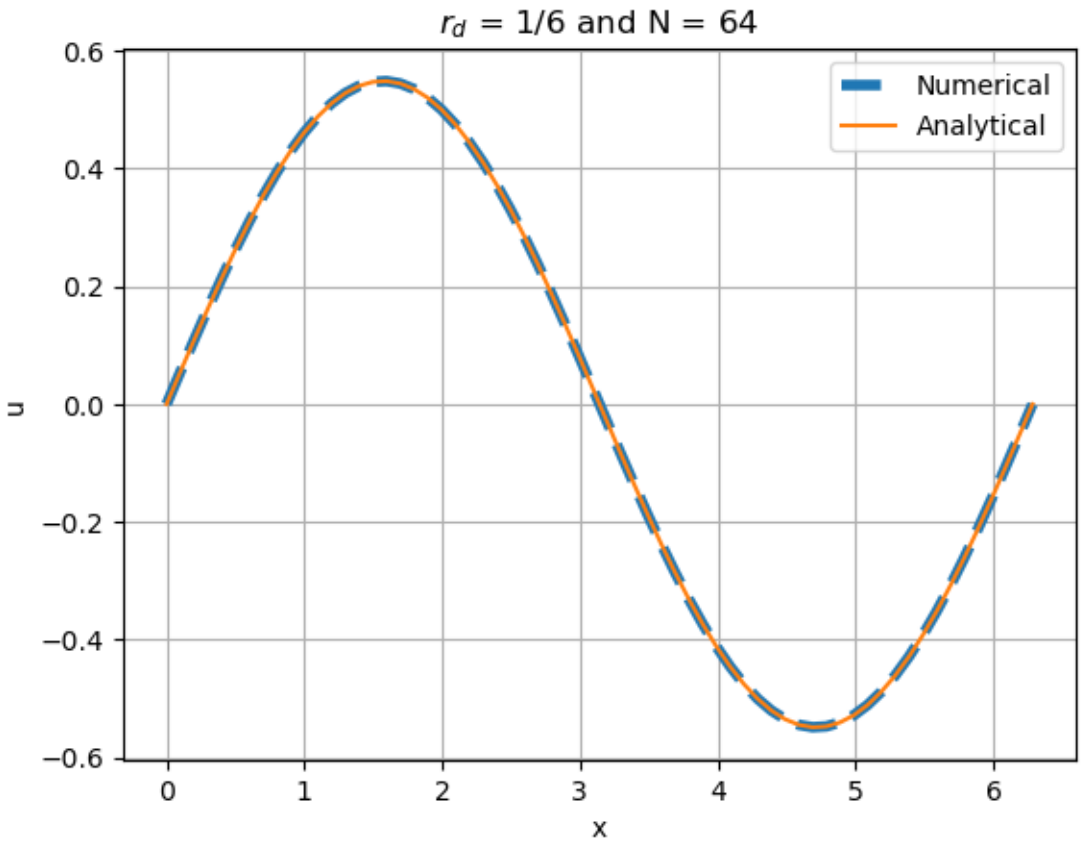
4.4 Plotting the results for rd = 1/2 and N = 64 at $t_{end} = 0.4$

[6]:
```
Numerical_Solution_N_64_rd_half = pd.read_csv(Output_files[6], delimiter =␣
↪",",header=None).to_numpy()
Analytical_Solution_N_64_rd_half = pd.read_csv(Output_files[7], delimiter =␣
↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_64_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_64_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_64_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 64")
plt.grid()
plt.savefig("r_d_1_2_and_N_64.png",dpi = 1000)
plt.show()
```



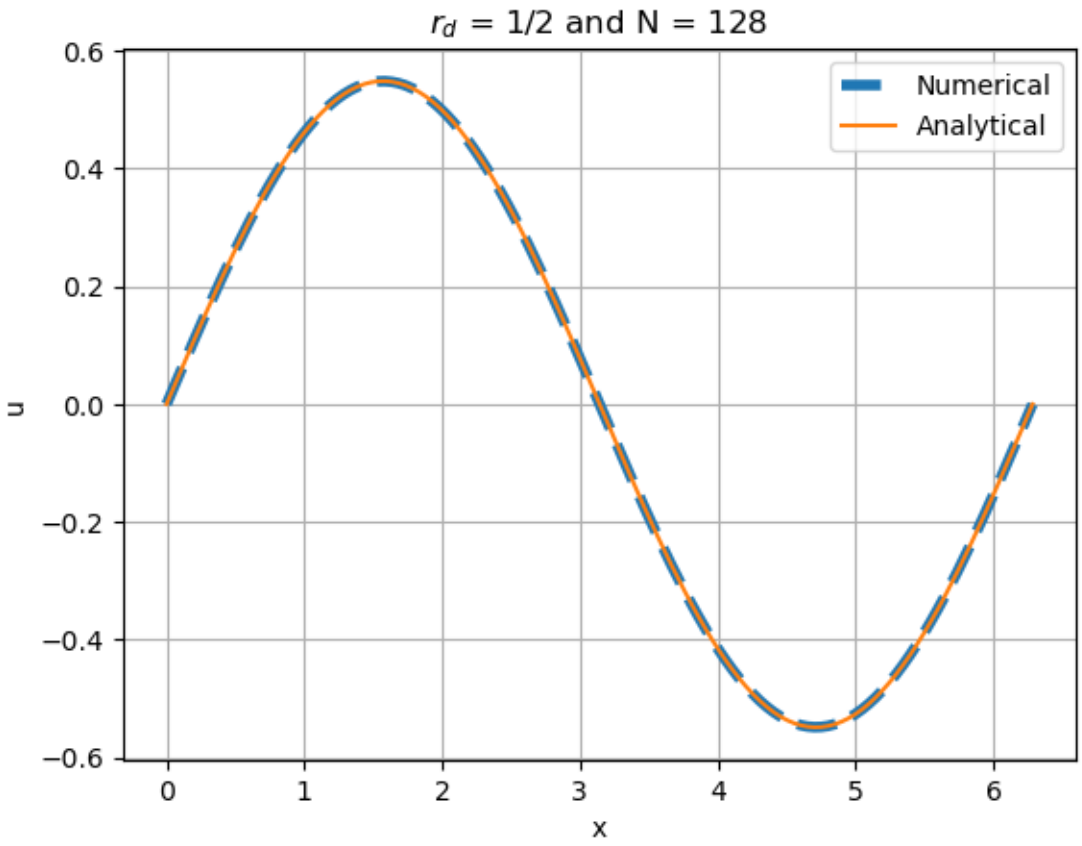
4.5 Plotting the results for rd = 1/6 and N = 64 at $t_{end} = 0.4$

[7]:
```
Numerical_Solution_N_64_rd_1_6 = pd.read_csv(Output_files[9], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_64_rd_1_6 = pd.read_csv(Output_files[10], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_64_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_64_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_64_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 64")
plt.grid()
plt.savefig("r_d_1_6_and_N_64.png",dpi = 1000)
plt.show()
```



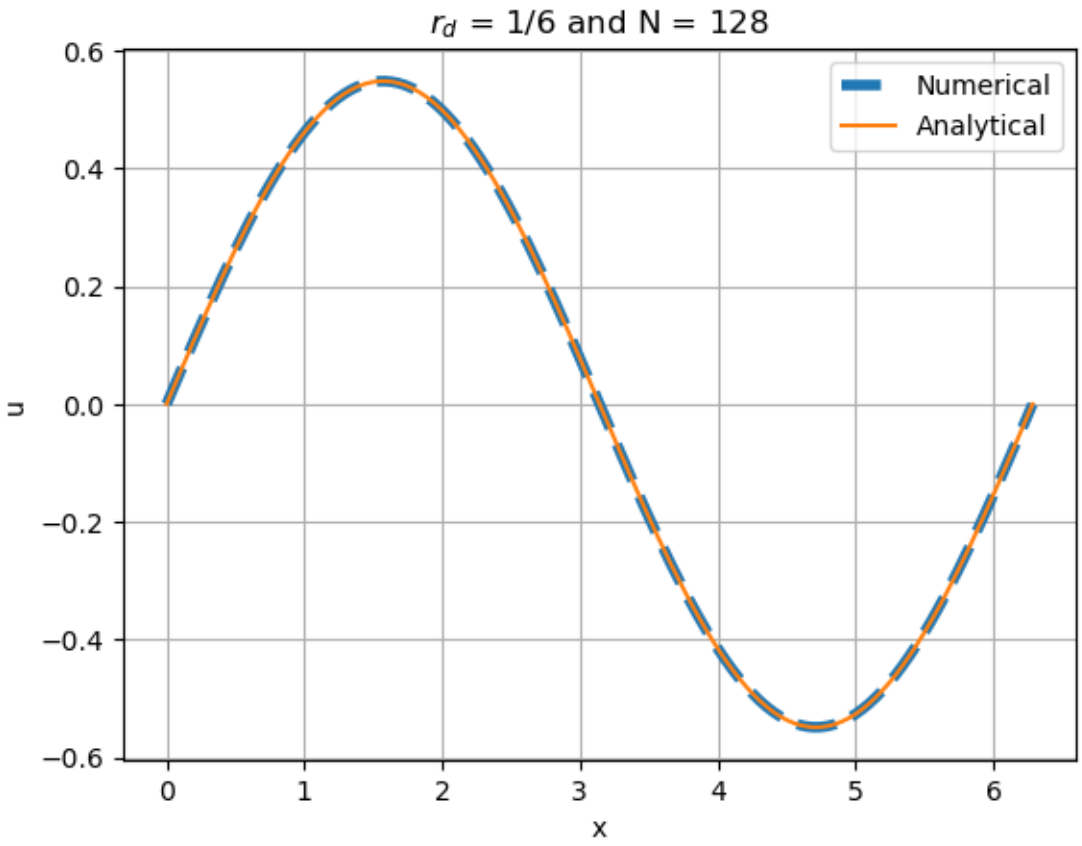
4.6 Plotting the results for rd = 1/2 and N = 128 at $t_{end} = 0.4$

[8]:
```
Numerical_Solution_N_128_rd_half = pd.read_csv(Output_files[12], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_128_rd_half = pd.read_csv(Output_files[13], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_128_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_128_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_128_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 128")
plt.grid()
plt.savefig("r_d_1_2_and_N_128.png",dpi = 1000)
plt.show()
```



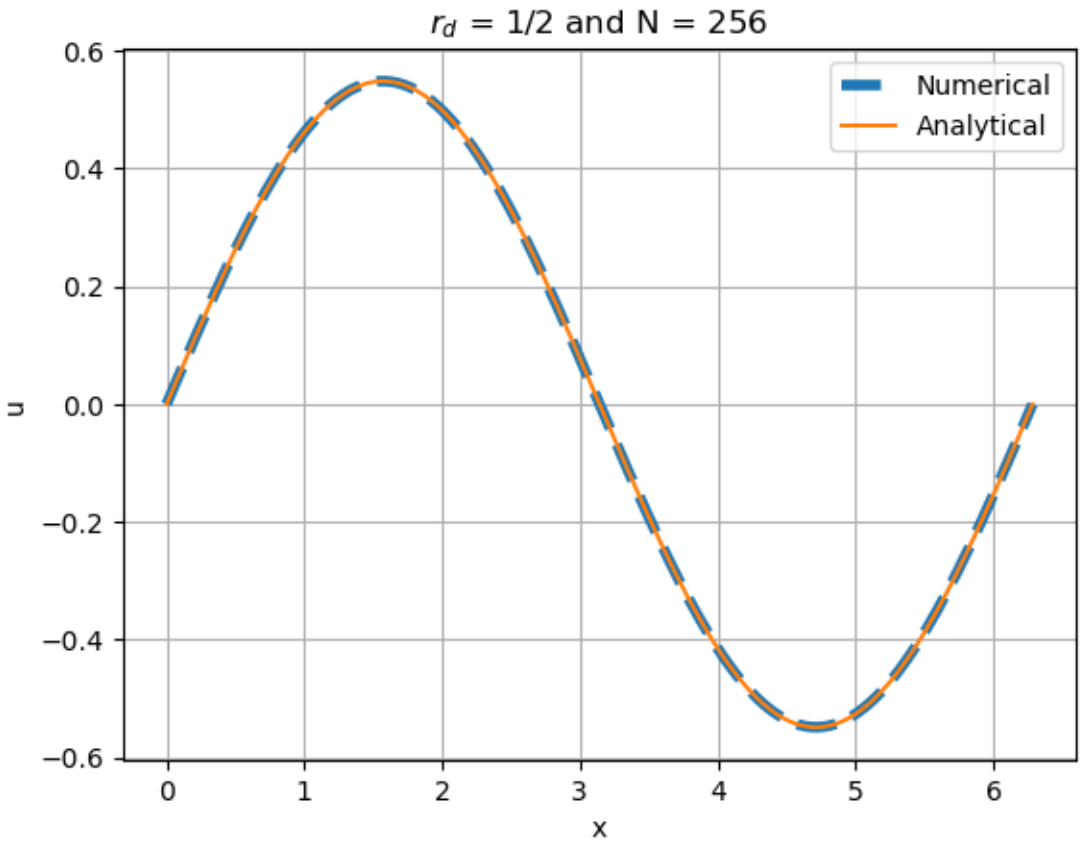
4.7 Plotting the results for rd = 1/6 and N = 128 at $t_{end} = 0.4$

[9]:
```
Numerical_Solution_N_128_rd_1_6 = pd.read_csv(Output_files[15], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_128_rd_1_6 = pd.read_csv(Output_files[16], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_128_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_128_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_128_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 128")
plt.grid()
plt.savefig("r_d_1_6_and_N_128.png",dpi = 1000)
plt.show()
```



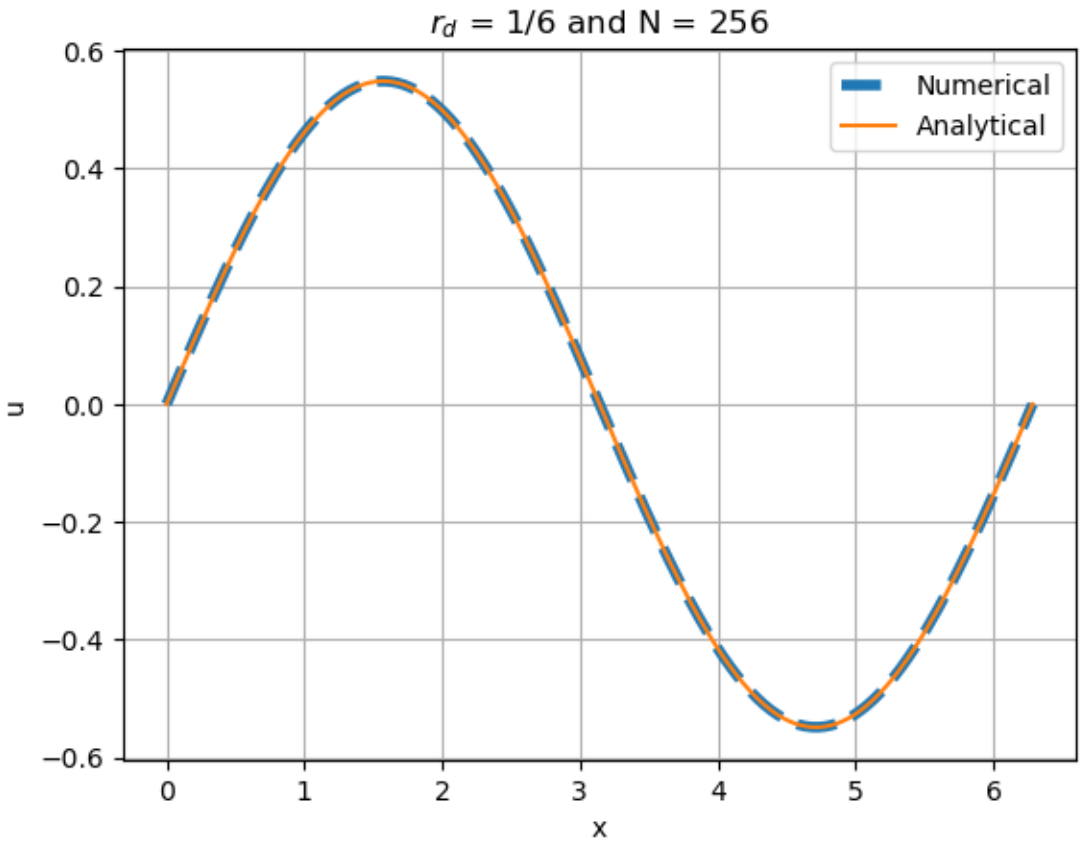
4.8 Plotting the results for rd = 1/2 and N = 256 at $t_{end} = 0.4$

[10]:
```
Numerical_Solution_N_256_rd_half = pd.read_csv(Output_files[18], delimiter =␣
  ↪",",header=None).to_numpy()
Analytical_Solution_N_256_rd_half = pd.read_csv(Output_files[19], delimiter =␣
  ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_256_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_256_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_256_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 256")
plt.grid()
plt.savefig("r_d_1_2_and_N_256.png",dpi = 1000)
plt.show()
```



4.9 Plotting the results for rd = 1/6 and N = 256 at $t_{end} = 0.4$

[11]:
```
Numerical_Solution_N_256_rd_1_6 = pd.read_csv(Output_files[21], delimiter =␣
↪",",header=None).to_numpy()
Analytical_Solution_N_256_rd_1_6 = pd.read_csv(Output_files[22], delimiter =␣
↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_256_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_256_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_256_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 256")
plt.grid()
plt.savefig("r_d_1_6_and_N_256.png",dpi = 1000)
plt.show()
```



5 Question (4) (c):

5.1 Use the analytical solution to compute the average of absolute error (E(N)) in each simulation at t = tend.

6 Answer (4) (c):

6.1 The average of absolute error (E(N)) in each simulation at t = t_{end} is being calculated and stored in the following files.

[12]:
```
for file in Output_files:
    if "Average_Error" in file:
        print(file)
```

```
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_32_rd_0.500000_.csv
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_32_rd_0.166667_.csv
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_64_rd_0.500000_.csv
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_64_rd_0.166667_.csv
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_128_rd_0.500000_.csv
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_128_rd_0.166667_.csv
```

```
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_256_rd_0.500000_.csv
Question_4_Explicit_Average_Error_u_n_t_0.400000_N_256_rd_0.166667_.csv
```

7 Question (4) (d):

7.1 In a graph with logarithmic scale, plot N vs E(N) for rd = 1/2 and 1/6. Obtain the order of accuracy for the two rd cases.

8 Answer (4) (d):

9 Reading the average of absolute error from the files

[13]:
```
rd = []
for i in range(2,len(Output_files),3):
    temp = pd.read_csv(Output_files[i], delimiter = ",",header=None).to_numpy().
 ↪squeeze()
    rd.append(temp.item())
rd_half = []
for i in range(0,len(rd),2):
    temp = rd[i]
    rd_half.append(temp)
rd_one_sixth = []
for i in range(1,len(rd),2):
    temp = rd[i]
    rd_one_sixth.append(temp)
```

9.1 Average of absolute error for rd = 1/2

[14]:
```
rd_half
```

[14]:
```
[0.00138949, 0.000341756, 8.48383e-05, 2.11276e-05]
```

9.2 Average of absolute error for rd = 1/6

[15]:
```
rd_one_sixth
```

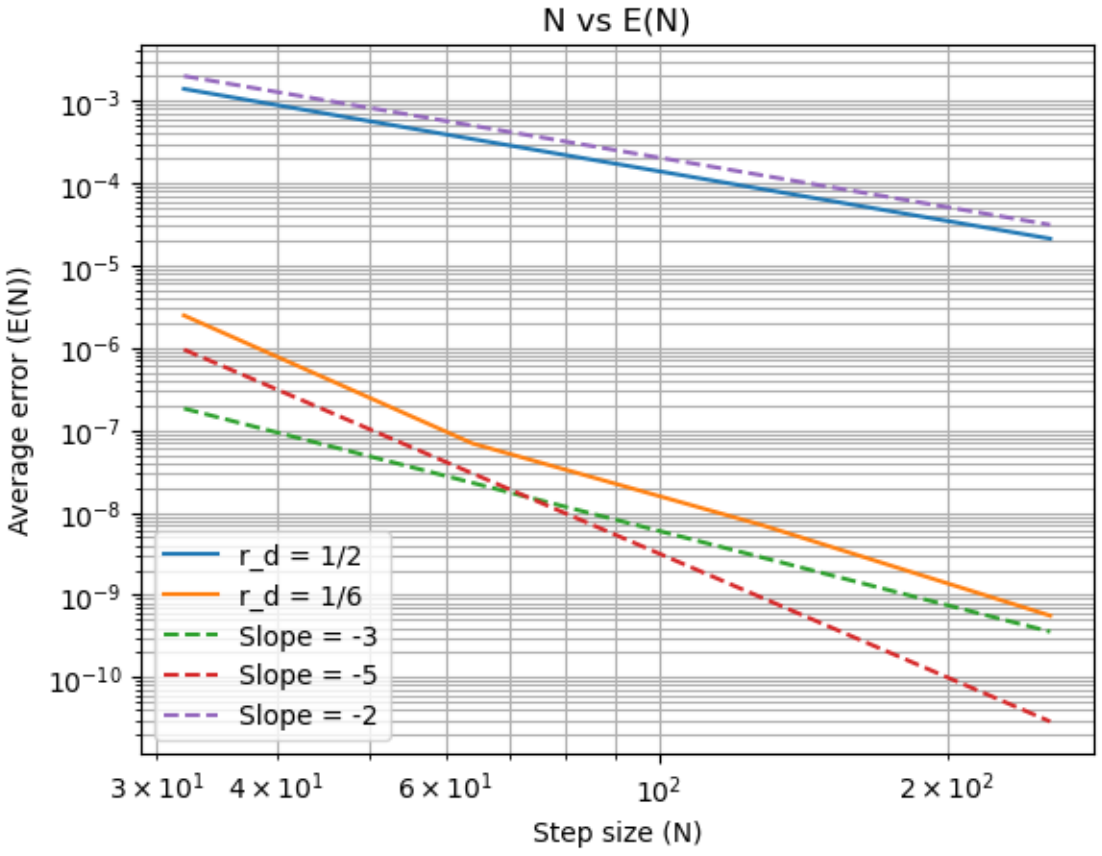
[15]:
```
[2.47444e-06, 6.89476e-08, 7.08266e-09, 5.61005e-10]
```

9.3 Values of N

[16]:
```
N = [32,64,128,256]
```

10 Average error vs step size for different values of rd in log-log scale

[17]:
```
fig = plt.figure()
plt.loglog(N,rd_half)
plt.loglog(N,rd_one_sixth)
plt.plot(N,(5.5*np.array(N))**(-3),"--")
plt.plot(N,(0.5*np.array(N))**(-5),"--")
plt.plot(N,(0.70*np.array(N))**(-2),"--")
plt.legend(["r_d = 1/2", "r_d = 1/6", "Slope = -3","Slope = -5","Slope = -2"])
plt.title("N vs E(N)")
plt.xlabel("Step size (N)")
plt.ylabel("Average error (E(N))")
#plt.xticks(ticks=Step_size_num,rotation=90)
plt.grid(which='both')
plt.show()
fig.savefig("E_N_vs_N.png",dpi = 500, bbox_inches="tight")
```



[ ]:

Answer (4) (e) :-

Modified Equation :-

$$u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[-\frac{\alpha^2 \Delta t}{2!} + 2\frac{\alpha (\Delta x)^2}{4!} \right]$$

$$+ H.O.T.$$

If $\gamma_d = \frac{1}{6}$

then $\gamma_d = \gamma = \frac{\alpha \Delta t}{(\Delta x)^2} = \frac{1}{6}$

or $\alpha \Delta t = \frac{(\Delta x)^2}{6}$

So, $u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[\frac{-\alpha (\Delta x)^2}{(6)(2!)} + \frac{2\alpha (\Delta x)^2}{4!} \right]$

$$+ H.O.T.$$

$$= (u_i^n)'''' \left[-\frac{\alpha (\Delta x)^2}{12} + \frac{\alpha (\Delta x)^2}{12} \right]$$

$$+ H.O.T.$$

or $u_i^n - \alpha (u_i^n)'' = H.O.T.$

So the leading order terms in the RHS of modified equation ~~can~~ cancels each other if $r_d = \frac{1}{6}$.

This reduces the error further.

This is also evident from the N vs E(N) graph. The error for $r_d = \frac{1}{6}$ is less than that of $r_d = \frac{1}{2}$. This is because the leading order terms for $r_d = \frac{1}{2}$ does NOT cancel each other.

If $r_d = \frac{1}{2}$,

then $u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[-\frac{\alpha^2 (\Delta t)^2}{2!} + \frac{2\alpha (\Delta x)^2}{4!} \right]$

$+ H.O.T.$

$\left[r_d = \frac{\alpha \Delta t}{(\Delta x)^2} = \frac{1}{2} \text{ or } \alpha \Delta t = \frac{(\Delta x)^2}{2} \right]$

So, $u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[-\frac{\alpha (\Delta x)^2}{4} + \frac{\alpha (\Delta x)^2}{12} \right]$

$+ H.O.T.$

$u_i^n - \alpha (u_i^n)'' = (u_i^n)'''' \left[-\frac{2\alpha (\Delta x)^2}{12} \right]$

$+ H.O.T.$

So, the leading order terms on the R.H.S. of the modified equation does NOT cancel each other for $\alpha_d = \frac{1}{2}$.

This explains the abnormal order of accuracy observed in $\alpha_d = \frac{1}{6}$ case.

Problem (5):

(a) Perform numerical experiments with implicit Euler scheme. Use the same equation and parameters provided in questions 3 and 4. Solve the linear system at each time step using the Jacobi method. (Tolerance $= 10^{-4}$)

(b) In a N vs E(N) graph, compare the errors with explicit method for $r_d = \frac{1}{2}$. Provide an explanation for your observation.

Answer (5)(a):

Diffusion Equation:

$$u_t = \alpha u_{xx}$$

Implicit Euler Scheme with second order central difference.

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \left(\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2} \right)$$

Let $m = \dfrac{\alpha \Delta t}{(\Delta x)^2}$

So, $u_i^{n+1} - u_i^n = m\left(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right)$

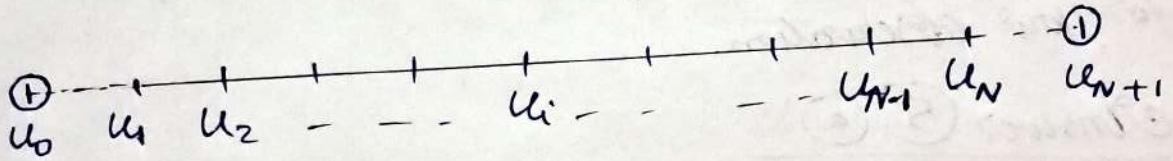
So,

$$-m u_{i+1}^{n+1} + (1+2m) u_i^{n+1} - m u_{i-1}^{n+1} = u_i^n$$

So, we have

$$[A][u^{n+1}] = [u^n]$$

As we are having periodic boundary condition. So, we will be using ghost point approach to model the boundary condition.



$u_N = u_1$

So, $A = \begin{bmatrix} (1+2r_d) & -r_d & 0 & \cdots & 0 & -r_d \\ -r_d & (1+2r_d) & -r_d & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & -r_d & \\ & & & & & \\ -r_d & 0 & \cdots & 0 & -r_d & (1+2r_d) \end{bmatrix}$

$$u^n = \begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_{N-1}^n \end{bmatrix}, \quad u^{n+1} = \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \end{bmatrix}$$

Jacobi Method :

Equations :

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1K}x_K = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2K}x_K = b_2$$
$$\vdots$$
$$a_{K1}x_1 + a_{K2}x_2 + \cdots + a_{KK}x_K = b_K$$

Step (1) :

Assume some seed values for $x_1, x_2, \cdots, x_K$

Step (2) :

$$x_i^{new} = \left(\frac{b_i - \left(\sum_{\substack{j=1 \\ j \neq i}}^{K} a_{ij} b_j \right)}{a_{ii}} \right)$$

$i = 1, 2, \cdots, K$

Step (3) :

Check for the tolerance.

$$\text{Tolerance} = \sum_{i=1}^{K} \left| x_i - x_i^{new} \right|$$

Step ④:

Replace all x_i with x_i^{new}

$(i = 1, 2, \cdots, K.)$

Step ⑤:

Stop if tolerance is less than the required tolerance.

Otherwise, we need to go back to step ② with the new values of x_i

$(i = 1, 2, \cdots, K)$

Post_processing

April 11, 2023

1 Question (5) (a)

1.1 Perform numerical experiments with implicit Euler scheme. Use the same equation and parameters provided in questions 3 and 4. Solver the linear system at each time step using the Jacobi method (tolerance= 10^{-4}; do not use any libraries).

2 Answer (5) (a):

3 For the tolerance value 1e-4

4 Import the necessary libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

5 Reading the name of the files generated

[2]:
```
Output_files = pd.read_csv("Output_file_names.csv",delimiter=",",header=None).
  ↪to_numpy()
Output_files = np.squeeze(Output_files)
Output_files = Output_files.tolist()
```

5.1 Output files generated

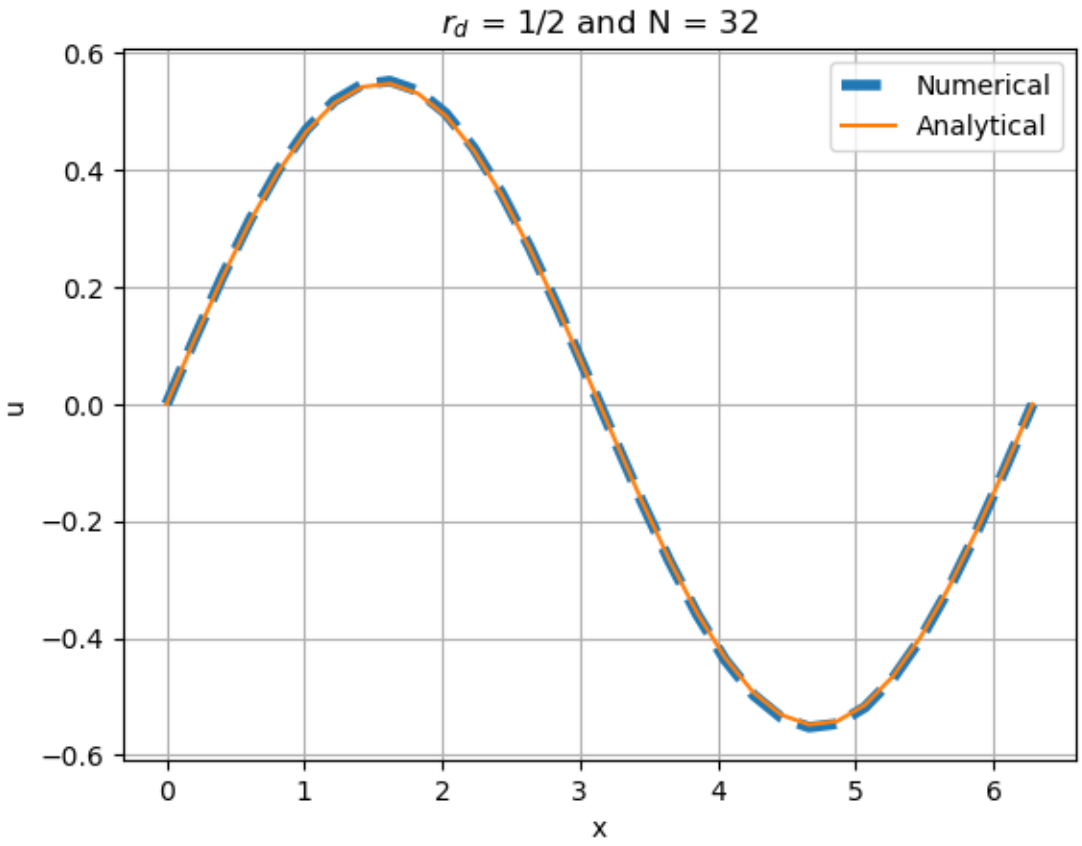
[3]:
```
Output_files
```

[3]:
```
['Question_5_Implicit_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_64_rd_0.500000_.csv',
```

```
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_64_rd_0.500000_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_64_rd_0.166667_.csv',
'Analytical_Solution_u_n_t_0.400000_N_64_rd_0.166667_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_64_rd_0.166667_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_128_rd_0.500000_.csv',
'Analytical_Solution_u_n_t_0.400000_N_128_rd_0.500000_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_128_rd_0.500000_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Analytical_Solution_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Analytical_Solution_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_256_rd_0.166667_.csv',
'Analytical_Solution_u_n_t_0.400000_N_256_rd_0.166667_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_256_rd_0.166667_.csv']
```

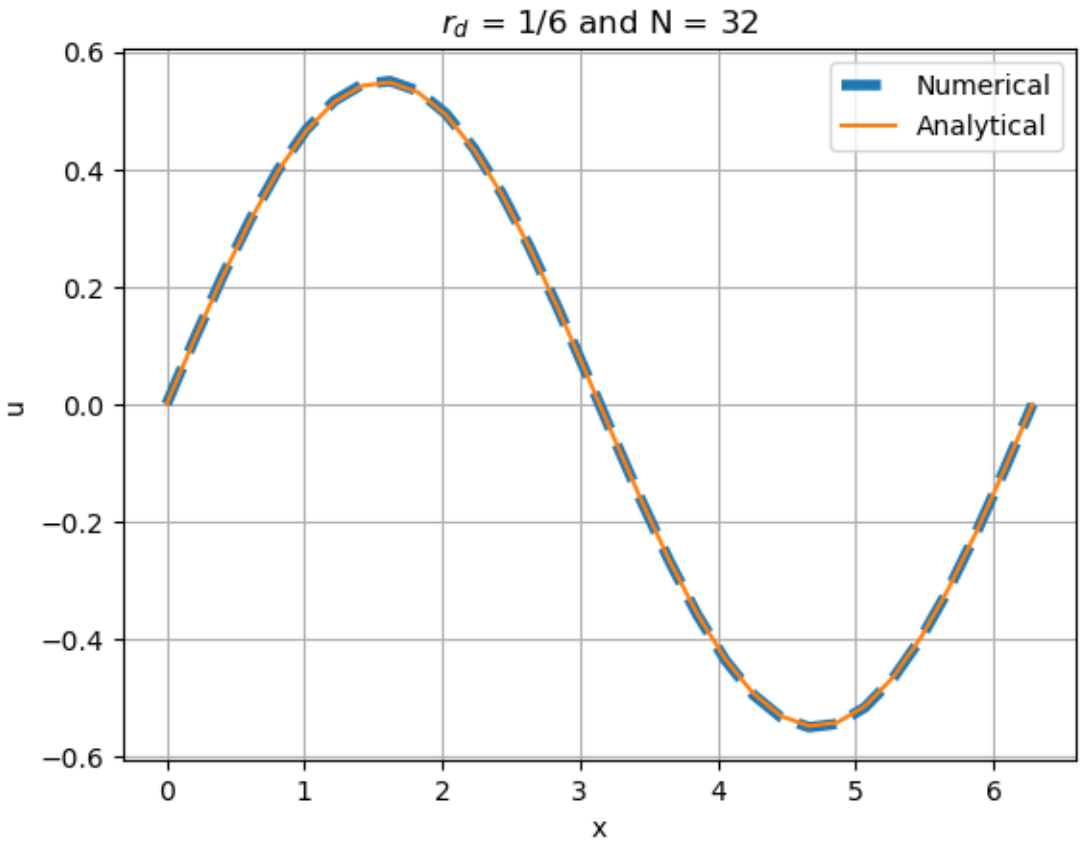
5.2 Plotting the results for rd $= 1/2$ and N $= 32$ at $t_{end} = 0.4$

[4]:
```
Numerical_Solution_N_32_rd_half = pd.read_csv(Output_files[0], delimiter =␣
↪",",header=None).to_numpy()
Analytical_Solution_N_32_rd_half = pd.read_csv(Output_files[1], delimiter =␣
↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_32_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_32_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_32_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 32")
plt.grid()
plt.savefig("r_d_1_2_and_N_32.png",dpi = 1000)
plt.show()
```



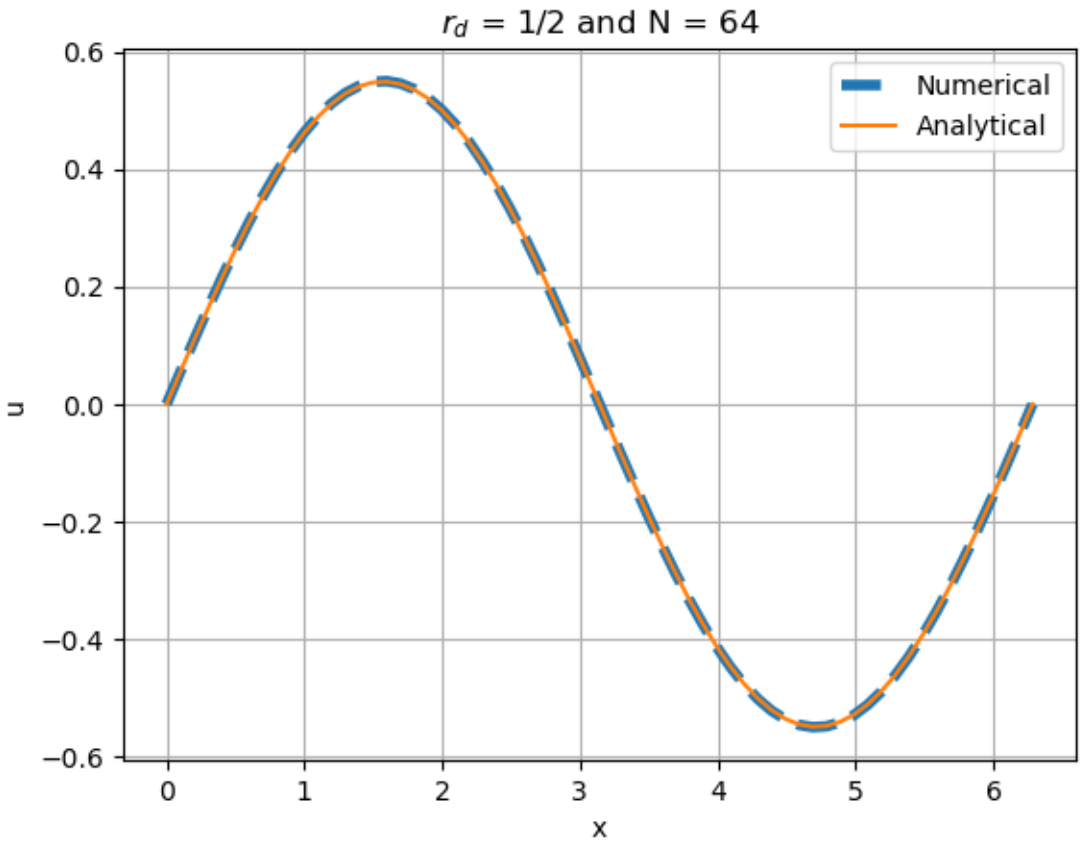
5.3 Plotting the results for rd = 1/6 and N = 32 at $t_{end} = 0.4$

[5]:
```
Numerical_Solution_N_32_rd_1_6 = pd.read_csv(Output_files[3], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_32_rd_1_6 = pd.read_csv(Output_files[4], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_32_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_32_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_32_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 32")
plt.grid()
plt.savefig("r_d_1_6_and_N_32.png",dpi = 1000)
plt.show()
```



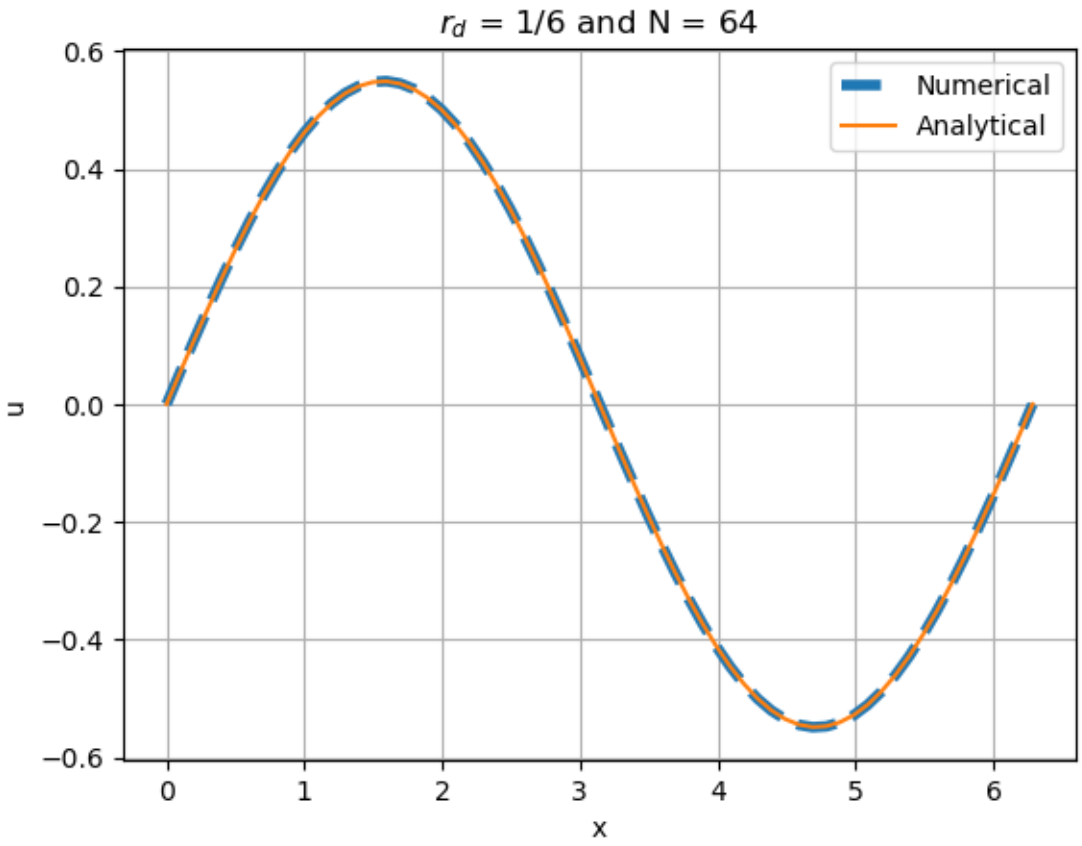
5.4 Plotting the results for rd = 1/2 and N = 64 at $t_{end} = 0.4$

[6]:
```
Numerical_Solution_N_64_rd_half = pd.read_csv(Output_files[6], delimiter =␣
↪",",header=None).to_numpy()
Analytical_Solution_N_64_rd_half = pd.read_csv(Output_files[7], delimiter =␣
↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_64_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_64_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_64_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 64")
plt.grid()
plt.savefig("r_d_1_2_and_N_64.png",dpi = 1000)
plt.show()
```



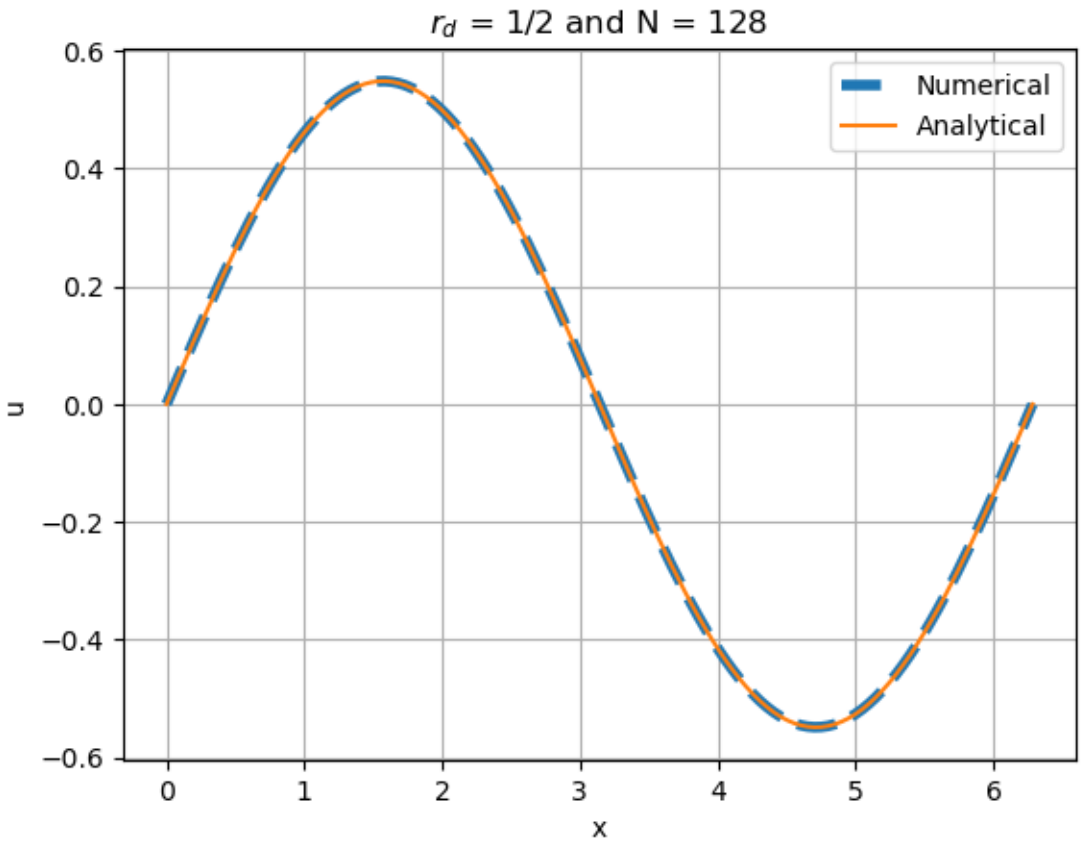
5.5 Plotting the results for rd = 1/6 and N = 64 at $t_{end} = 0.4$

[7]:
```
Numerical_Solution_N_64_rd_1_6 = pd.read_csv(Output_files[9], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_64_rd_1_6 = pd.read_csv(Output_files[10], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_64_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_64_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_64_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 64")
plt.grid()
plt.savefig("r_d_1_6_and_N_64.png",dpi = 1000)
plt.show()
```



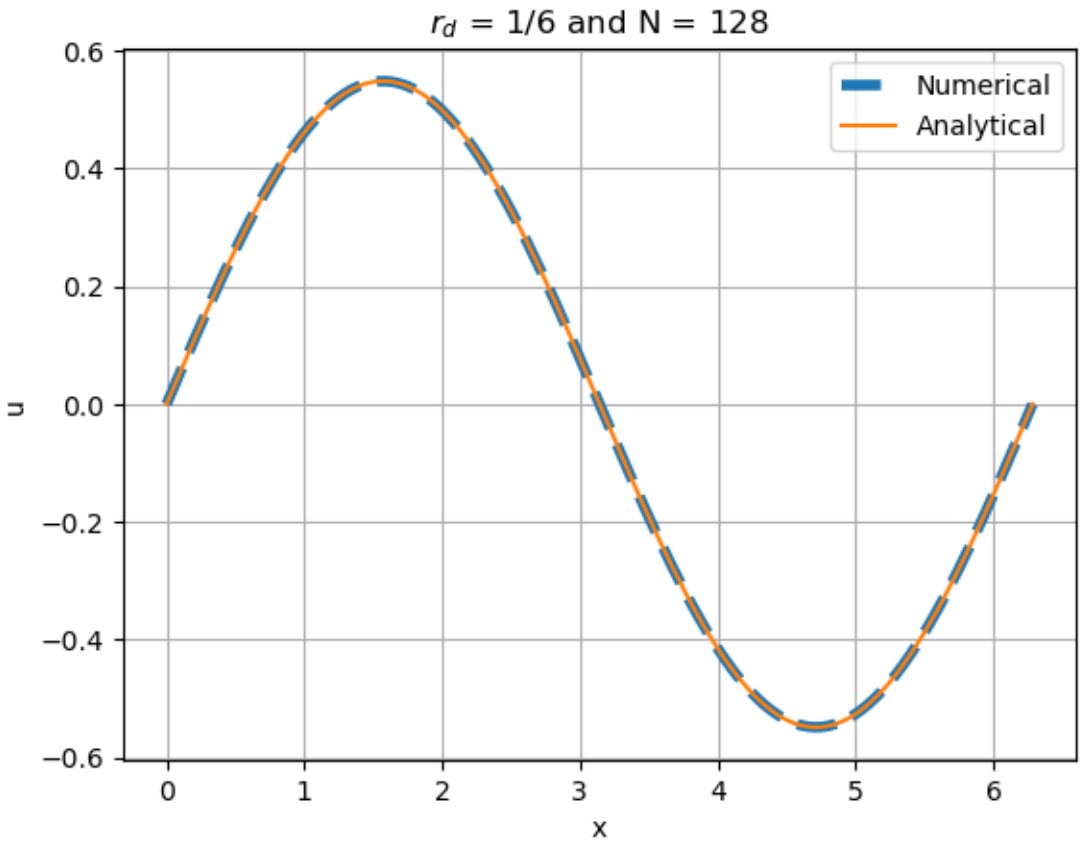
5.6 Plotting the results for rd = 1/2 and N = 128 at $t_{end} = 0.4$

[8]:
```
Numerical_Solution_N_128_rd_half = pd.read_csv(Output_files[12], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_128_rd_half = pd.read_csv(Output_files[13], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_128_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_128_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_128_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 128")
plt.grid()
plt.savefig("r_d_1_2_and_N_128.png",dpi = 1000)
plt.show()
```



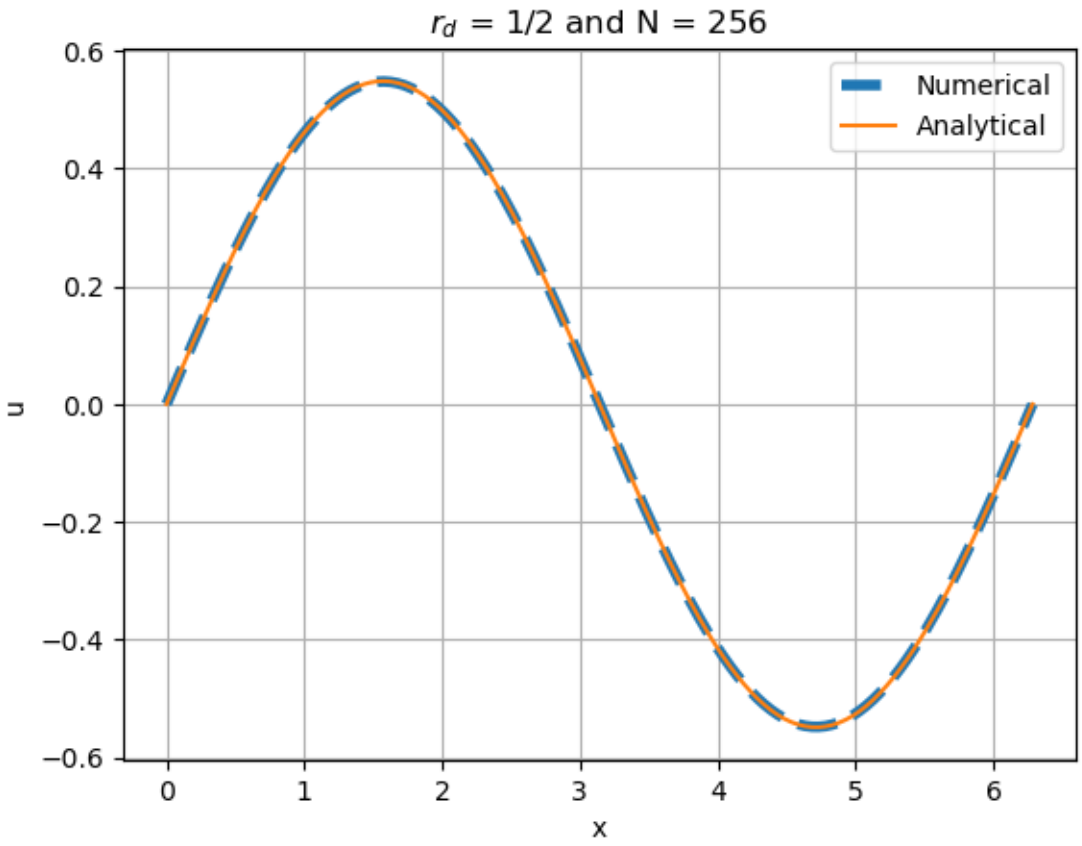
5.7 Plotting the results for rd = 1/6 and N = 128 at $t_{end} = 0.4$

[9]:
```
Numerical_Solution_N_128_rd_1_6 = pd.read_csv(Output_files[15], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_128_rd_1_6 = pd.read_csv(Output_files[16], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_128_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_128_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_128_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 128")
plt.grid()
plt.savefig("r_d_1_6_and_N_128.png",dpi = 1000)
plt.show()
```



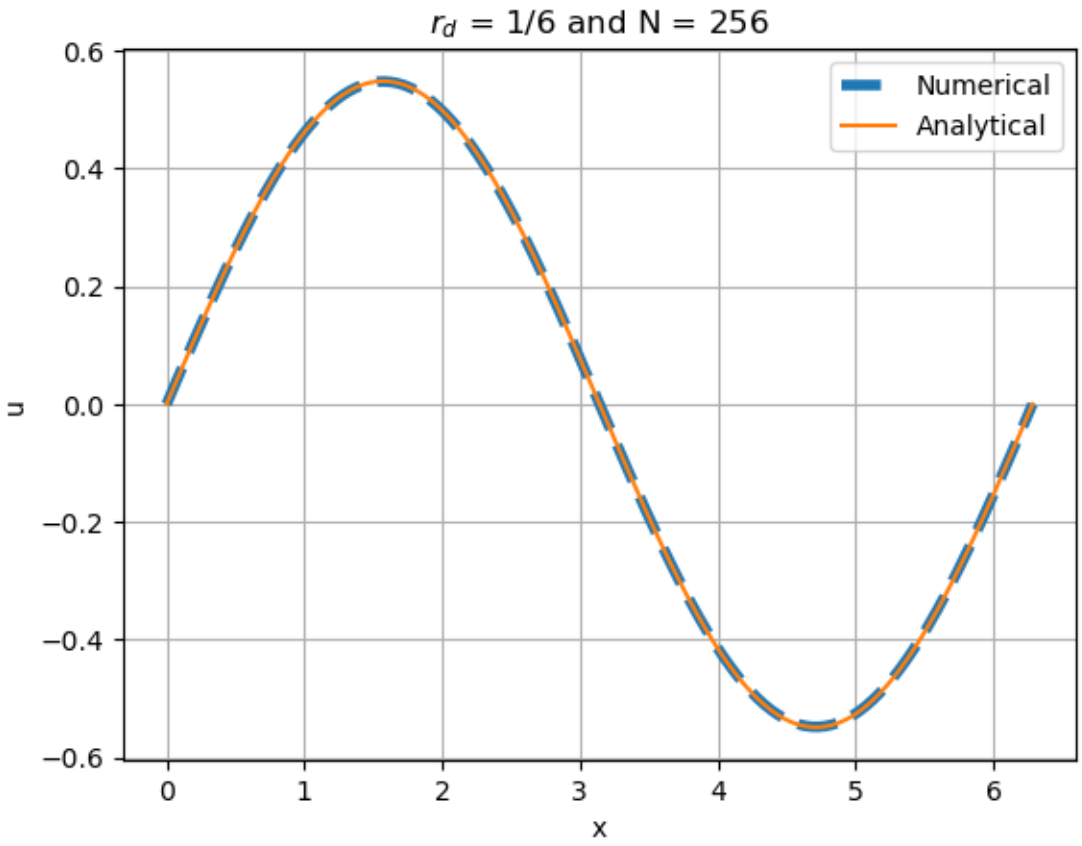
5.8 Plotting the results for rd = 1/2 and N = 256 at $t_{end} = 0.4$

[10]:
```
Numerical_Solution_N_256_rd_half = pd.read_csv(Output_files[18], delimiter =␣
  ↪",",header=None).to_numpy()
Analytical_Solution_N_256_rd_half = pd.read_csv(Output_files[19], delimiter =␣
  ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_256_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_256_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_256_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 256")
plt.grid()
plt.savefig("r_d_1_2_and_N_256.png",dpi = 1000)
plt.show()
```



5.9 Plotting the results for rd = 1/6 and N = 256 at $t_{end} = 0.4$

[11]:
```
Numerical_Solution_N_256_rd_1_6 = pd.read_csv(Output_files[21], delimiter =␣
  ↪",",header=None).to_numpy()
Analytical_Solution_N_256_rd_1_6 = pd.read_csv(Output_files[22], delimiter =␣
  ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_256_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_256_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_256_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 256")
plt.grid()
plt.savefig("r_d_1_6_and_N_256.png",dpi = 1000)
plt.show()
```



6 Average error vs step size for different values of rd in log-log scale

[12]:
```
rd = []
for i in range(2,len(Output_files),3):
    temp = pd.read_csv(Output_files[i], delimiter = ",",header=None).to_numpy().
 ↪squeeze()
    rd.append(temp.item())
rd_half = []
for i in range(0,len(rd),2):
    temp = rd[i]
    rd_half.append(temp)
rd_one_sixth = []
for i in range(1,len(rd),2):
    temp = rd[i]
    rd_one_sixth.append(temp)
```

[13]:
```
rd
```

[13]:
```
[0.00268342,
 0.00135759,
 0.000577935,
 0.000241834,
 3.84373e-05,
 3.5107e-05,
 0.000375391,
 0.000379308]
```

[14]:
```
rd_half
```

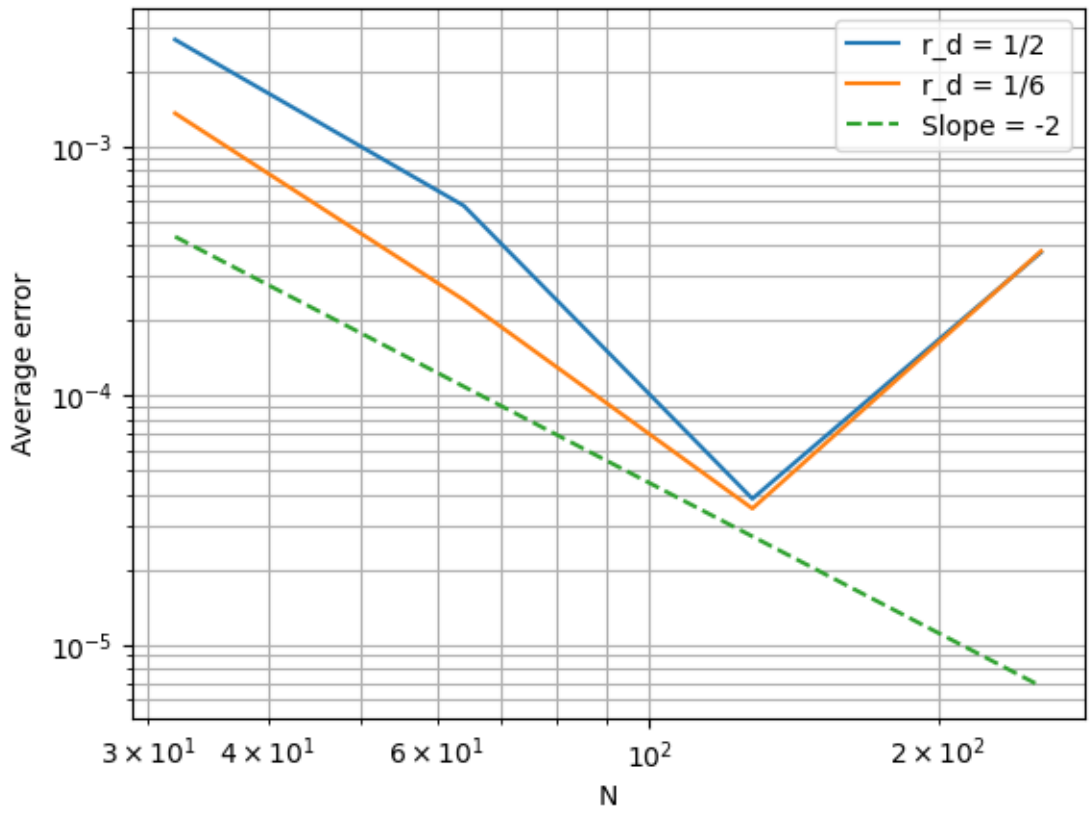
[14]:
```
[0.00268342, 0.000577935, 3.84373e-05, 0.000375391]
```

[15]:
```
rd_one_sixth
```

[15]:
```
[0.00135759, 0.000241834, 3.5107e-05, 0.000379308]
```

[16]:
```
N = [32,64,128,256]
```

[17]:
```
fig = plt.figure()
plt.loglog(N,rd_half)
plt.loglog(N,rd_one_sixth)
plt.plot(N,(1.50*np.array(N))**(-2),"--")
plt.legend(["r_d = 1/2", "r_d = 1/6", "Slope = -2"])
plt.xlabel("N")
plt.ylabel("Average error")
plt.grid(which='both')
plt.show()
fig.savefig("Average_Error_log.png",dpi = 500, bbox_inches="tight")
```



7 Question (5) (b):

7.1 In a N vs E(N) graph, compare the errors with explicit method for rd = 1/2. Provide an explanation for your observations.

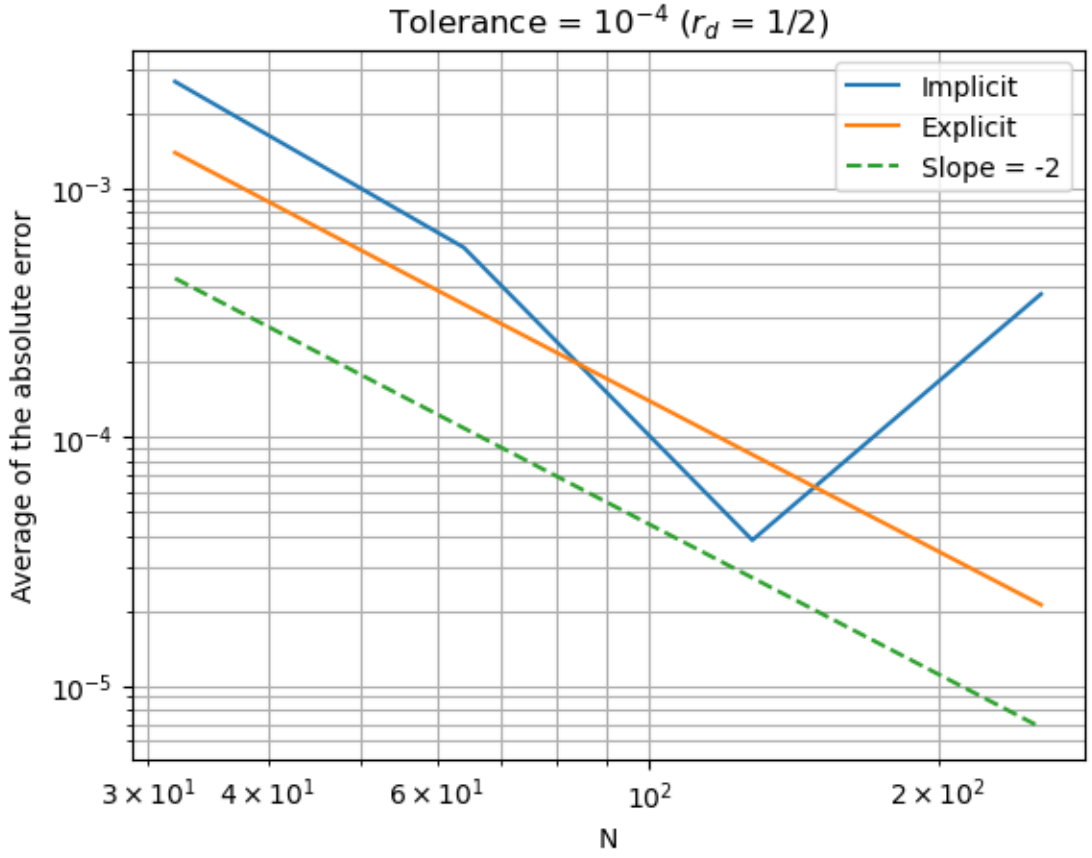
8 Answer (5) (b):

[18]:
```
Q_4_Error_Data = pd.read_csv("Question_4_Error_Data.csv",header=None)
```

9 Average error vs step size for different methods in log-log scale

[19]:
```
fig = plt.figure()
plt.loglog(N,rd_half)
plt.loglog(N,Q_4_Error_Data)
plt.plot(N,(1.50*np.array(N))**(-2),"--")
plt.legend(["Implicit", "Explicit", "Slope = -2"])
plt.xlabel("N")
plt.ylabel("Average of the absolute error")
plt.grid(which='both')
```

```
plt.title(R"Tolerance = $10^{-4}$ ($r_{d}$ = 1/2)")
plt.show()
fig.savefig("Average_Error_log_for_Different_Methods.png",dpi = 500,␣
 ↪bbox_inches="tight")
```



9.1 In case of the implicit method, the truncation error dominates, when we are using lesser number of grid points but when we are using a finer grid the propagation error keep on accumulating over iterations and eventually dominates the truncation error. Hence, the error increases on using a finer grid, for a fixed tolerance.

9.2 We have found that on decreasing the tolerance from the 10^{-4} to 10^{-6} the error keeps on reducing.

Post_processing

April 11, 2023

1 For the tolerance value 1e-6

2 Import the necessary libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

3 Reading the name of the files generated

[2]:
```
Output_files = pd.read_csv("Output_file_names.csv",delimiter=",",header=None).
  ↪to_numpy()
Output_files = np.squeeze(Output_files)
Output_files = Output_files.tolist()
```

3.1 Output files generated

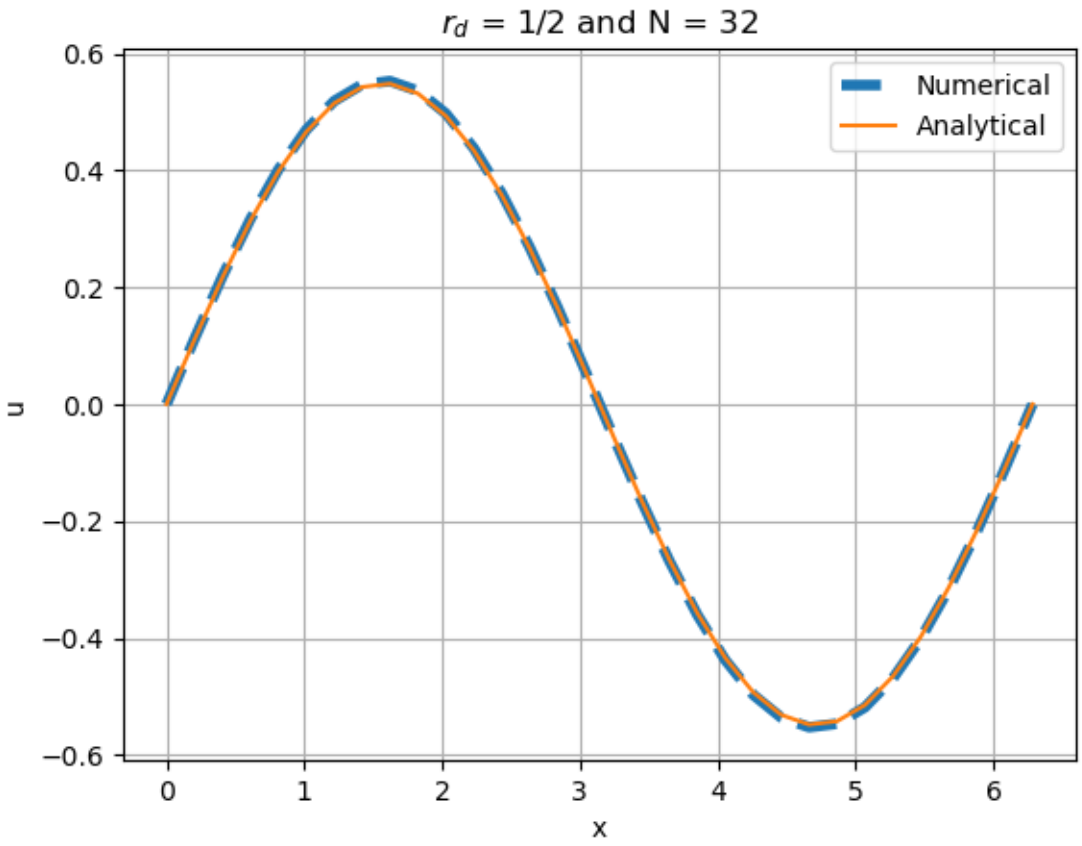
[3]:
```
Output_files
```

[3]:
```
['Question_5_Implicit_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_32_rd_0.500000_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_32_rd_0.166667_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_64_rd_0.500000_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_64_rd_0.166667_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_64_rd_0.166667_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_64_rd_0.166667_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_128_rd_0.500000_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_128_rd_0.500000_.csv',
 'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_128_rd_0.500000_.csv',
 'Question_5_Implicit_u_n_t_0.400000_N_128_rd_0.166667_.csv',
 'Analytical_Solution_u_n_t_0.400000_N_128_rd_0.166667_.csv',
```

```
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_128_rd_0.166667_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Analytical_Solution_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_256_rd_0.500000_.csv',
'Question_5_Implicit_u_n_t_0.400000_N_256_rd_0.166667_.csv',
'Analytical_Solution_u_n_t_0.400000_N_256_rd_0.166667_.csv',
'Question_5_Implicit_Average_Error_u_n_t_0.400000_N_256_rd_0.166667_.csv']
```

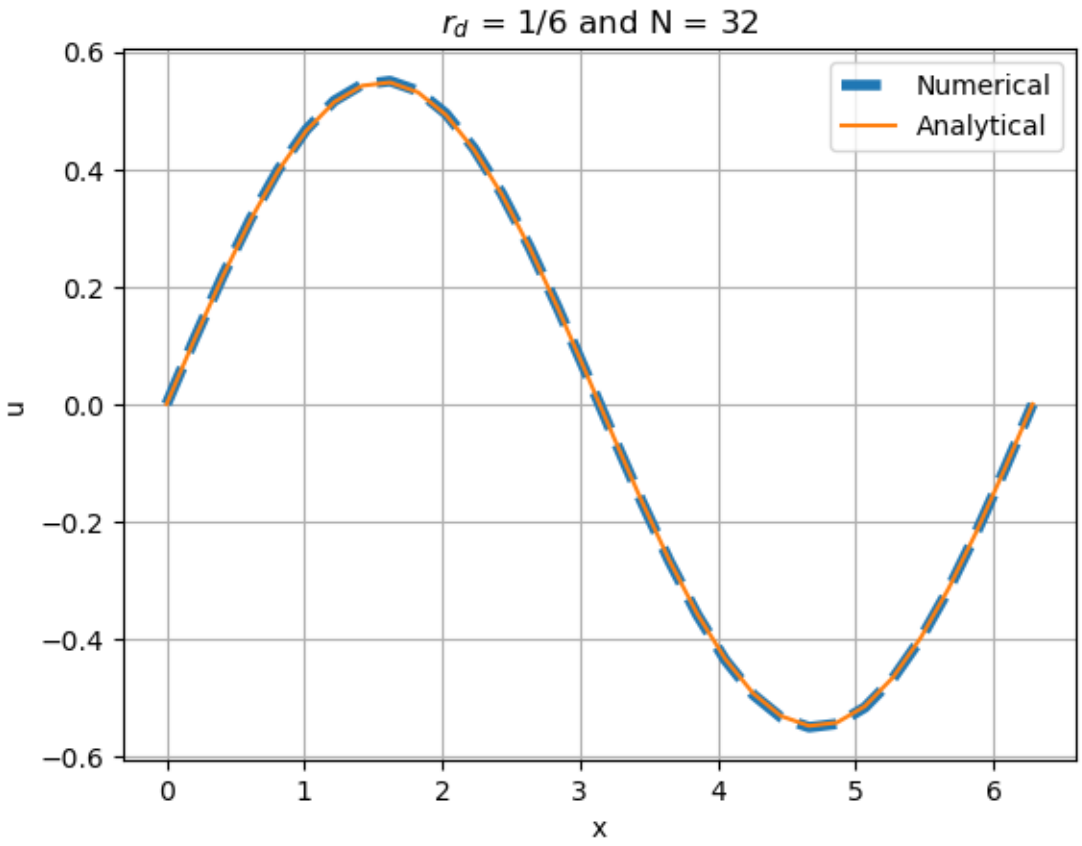
3.2 Plotting the results for rd = 1/2 and N = 32 at $t_{end} = 0.4$

[4]:
```
Numerical_Solution_N_32_rd_half = pd.read_csv(Output_files[0], delimiter =␣
↪",",header=None).to_numpy()
Analytical_Solution_N_32_rd_half = pd.read_csv(Output_files[1], delimiter =␣
↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_32_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_32_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_32_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 32")
plt.grid()
plt.savefig("r_d_1_2_and_N_32.png",dpi = 1000)
plt.show()
```



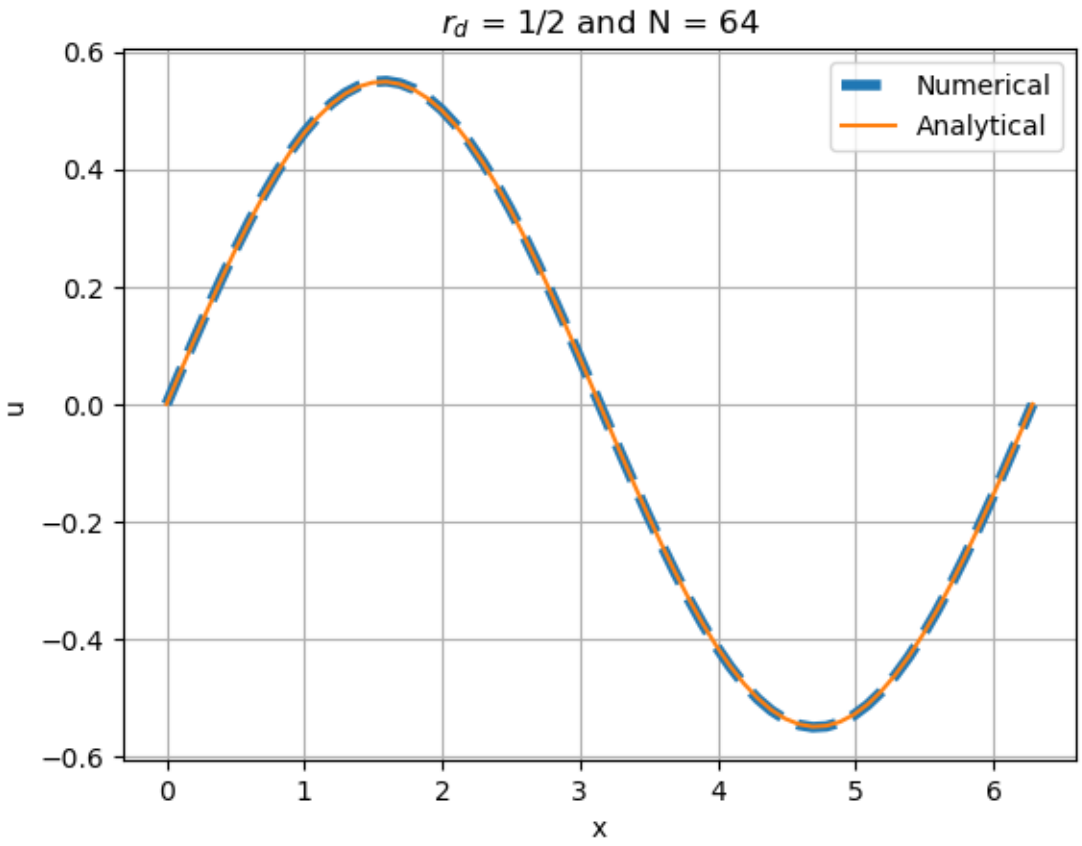
3.3 Plotting the results for rd = 1/6 and N = 32 at $t_{end} = 0.4$

[5]:
```
Numerical_Solution_N_32_rd_1_6 = pd.read_csv(Output_files[3], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_32_rd_1_6 = pd.read_csv(Output_files[4], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_32_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_32_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_32_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 32")
plt.grid()
plt.savefig("r_d_1_6_and_N_32.png",dpi = 1000)
plt.show()
```



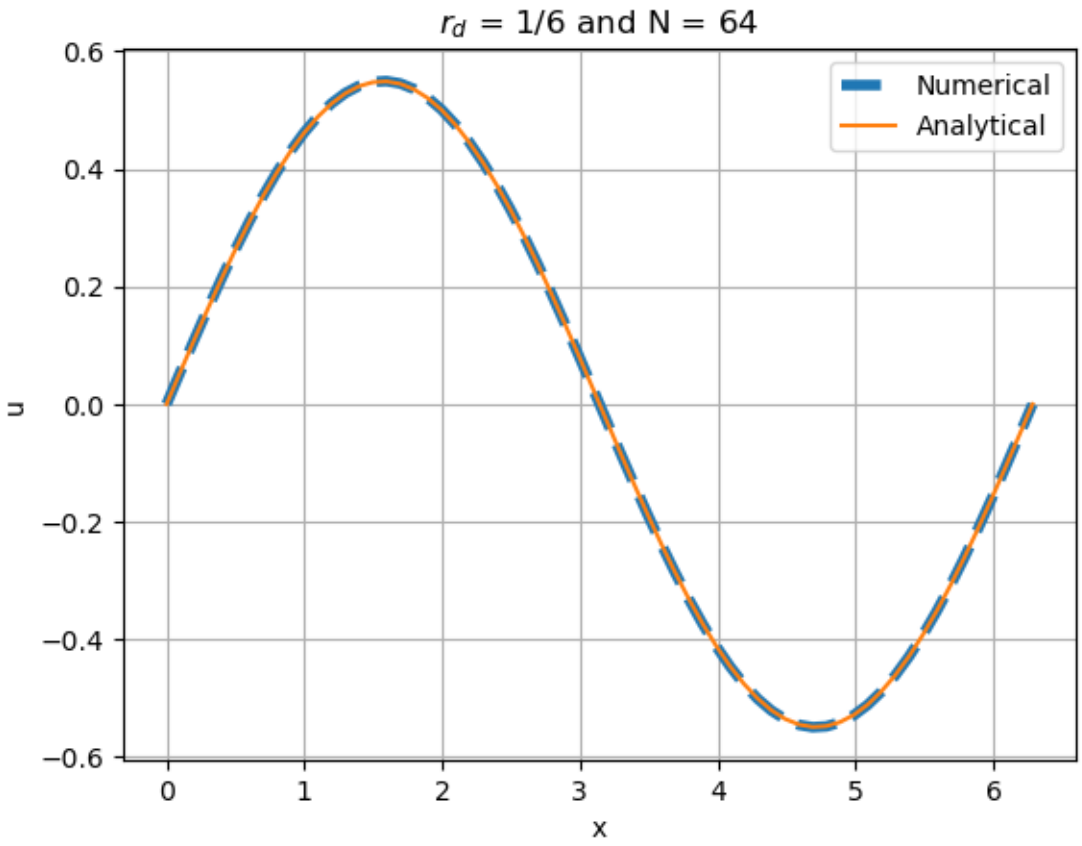
3.4 Plotting the results for rd = 1/2 and N = 64 at $t_{end} = 0.4$

[6]:
```
Numerical_Solution_N_64_rd_half = pd.read_csv(Output_files[6], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_64_rd_half = pd.read_csv(Output_files[7], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_64_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_64_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_64_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 64")
plt.grid()
plt.savefig("r_d_1_2_and_N_64.png",dpi = 1000)
plt.show()
```



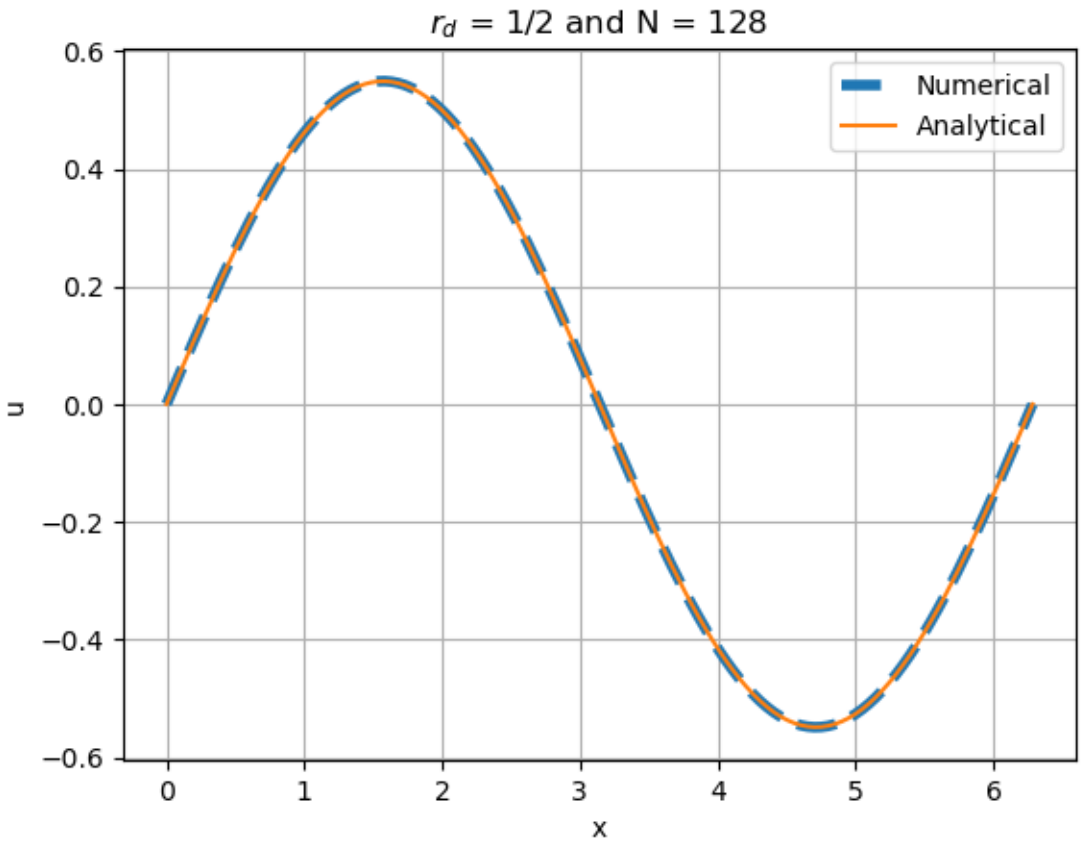
3.5 Plotting the results for rd = 1/6 and N = 64 at $t_{end} = 0.4$

[7]:
```
Numerical_Solution_N_64_rd_1_6 = pd.read_csv(Output_files[9], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_64_rd_1_6 = pd.read_csv(Output_files[10], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_64_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_64_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_64_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 64")
plt.grid()
plt.savefig("r_d_1_6_and_N_64.png",dpi = 1000)
plt.show()
```



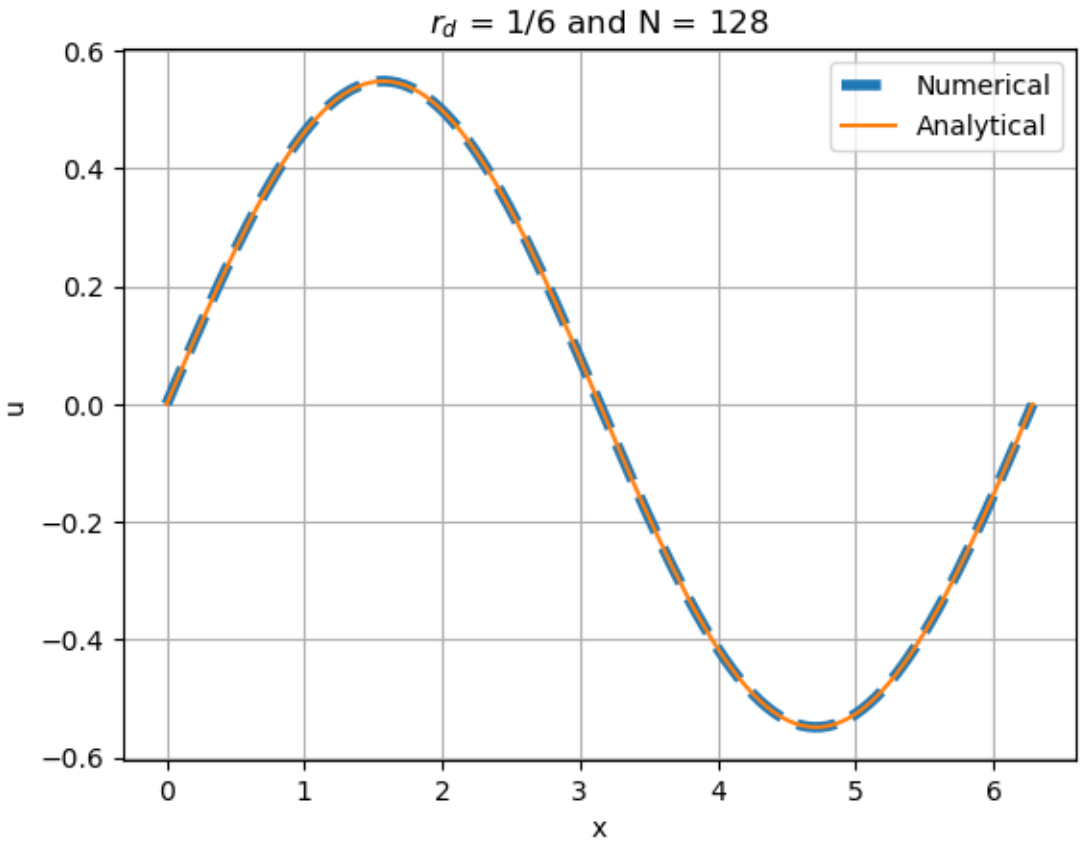
3.6 Plotting the results for rd = 1/2 and N = 128 at $t_{end} = 0.4$

[8]:
```
Numerical_Solution_N_128_rd_half = pd.read_csv(Output_files[12], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_128_rd_half = pd.read_csv(Output_files[13], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_128_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_128_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_128_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 128")
plt.grid()
plt.savefig("r_d_1_2_and_N_128.png",dpi = 1000)
plt.show()
```



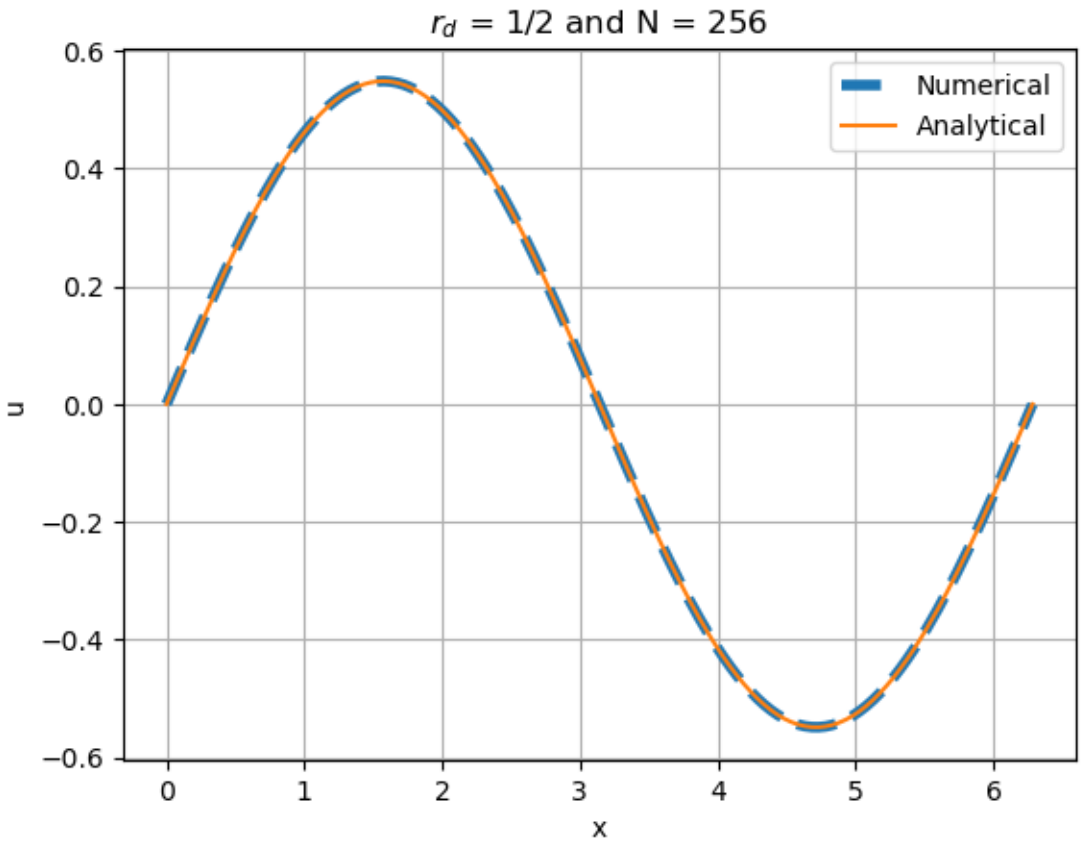
3.7 Plotting the results for rd = 1/6 and N = 128 at $t_{end} = 0.4$

[9]:
```
Numerical_Solution_N_128_rd_1_6 = pd.read_csv(Output_files[15], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_128_rd_1_6 = pd.read_csv(Output_files[16], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_128_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_128_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_128_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 128")
plt.grid()
plt.savefig("r_d_1_6_and_N_128.png",dpi = 1000)
plt.show()
```



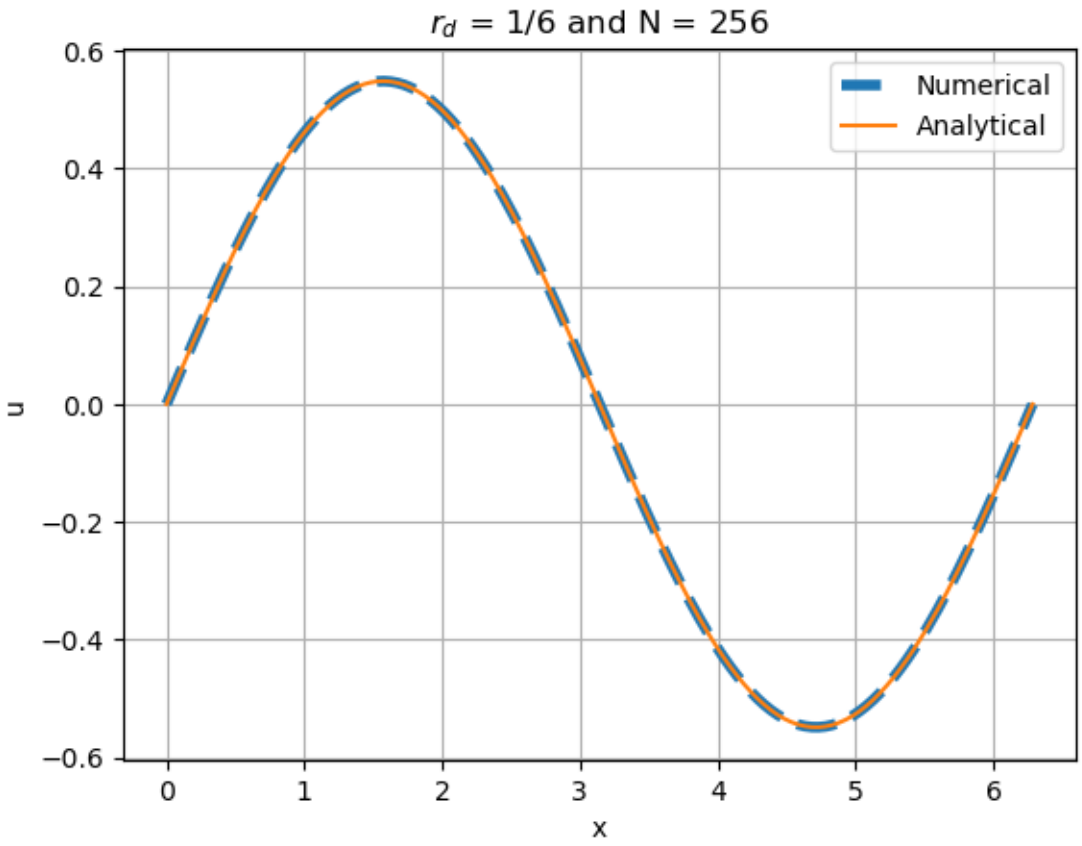
3.8 Plotting the results for rd = 1/2 and N = 256 at $t_{end} = 0.4$

[10]:
```
Numerical_Solution_N_256_rd_half = pd.read_csv(Output_files[18], delimiter =␣
 ↪",",header=None).to_numpy()
Analytical_Solution_N_256_rd_half = pd.read_csv(Output_files[19], delimiter =␣
 ↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_256_rd_half.shape[0])
plt.plot(x,Numerical_Solution_N_256_rd_half,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_256_rd_half)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/2 and N = 256")
plt.grid()
plt.savefig("r_d_1_2_and_N_256.png",dpi = 1000)
plt.show()
```



3.9 Plotting the results for rd = 1/6 and N = 256 at $t_{end} = 0.4$

[11]:
```
Numerical_Solution_N_256_rd_1_6 = pd.read_csv(Output_files[21], delimiter =␣
↪",",header=None).to_numpy()
Analytical_Solution_N_256_rd_1_6 = pd.read_csv(Output_files[22], delimiter =␣
↪",",header=None).to_numpy()
x = np.linspace(0,2*np.pi,Numerical_Solution_N_256_rd_1_6.shape[0])
plt.plot(x,Numerical_Solution_N_256_rd_1_6,linestyle='dashed',linewidth=4)
plt.plot(x,Analytical_Solution_N_256_rd_1_6)
plt.xlabel("x")
plt.ylabel("u")
plt.legend(["Numerical","Analytical"])
plt.title(R"$r_d$ = 1/6 and N = 256")
plt.grid()
plt.savefig("r_d_1_6_and_N_256.png",dpi = 1000)
plt.show()
```



4 Average error vs step size for different values of rd in log-log scale

[12]:
```
rd = []
for i in range(2,len(Output_files),3):
    temp = pd.read_csv(Output_files[i], delimiter = ",",header=None).to_numpy().
 ↪squeeze()
    rd.append(temp.item())
rd_half = []
for i in range(0,len(rd),2):
    temp = rd[i]
    rd_half.append(temp)
rd_one_sixth = []
for i in range(1,len(rd),2):
    temp = rd[i]
    rd_one_sixth.append(temp)
```

[13]:
```
rd
```

[13]:
```
[0.00273252,
 0.0013804,
 0.000680045,
 0.000341009,
 0.000167387,
 8.29671e-05,
 3.79356e-05,
 1.9136e-05]
```

[14]:
```
rd_half
```

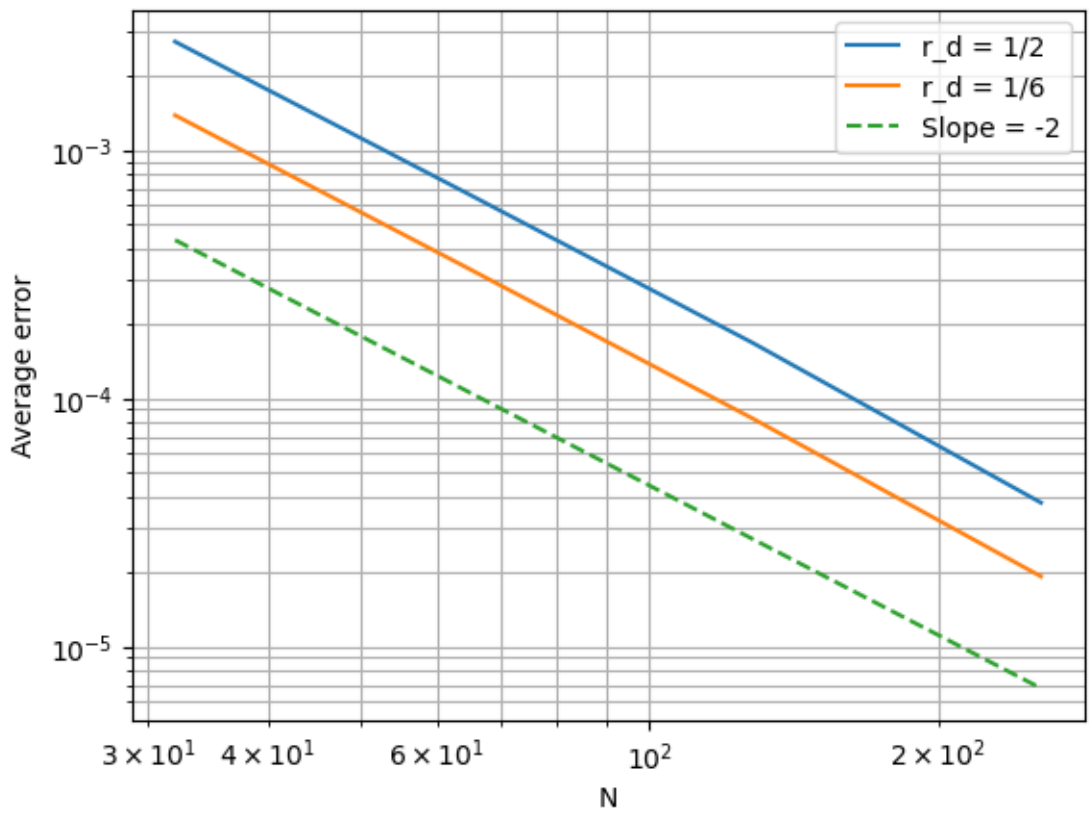
[14]:
```
[0.00273252, 0.000680045, 0.000167387, 3.79356e-05]
```

[15]:
```
rd_one_sixth
```

[15]:
```
[0.0013804, 0.000341009, 8.29671e-05, 1.9136e-05]
```

[16]:
```
N = [32,64,128,256]
```

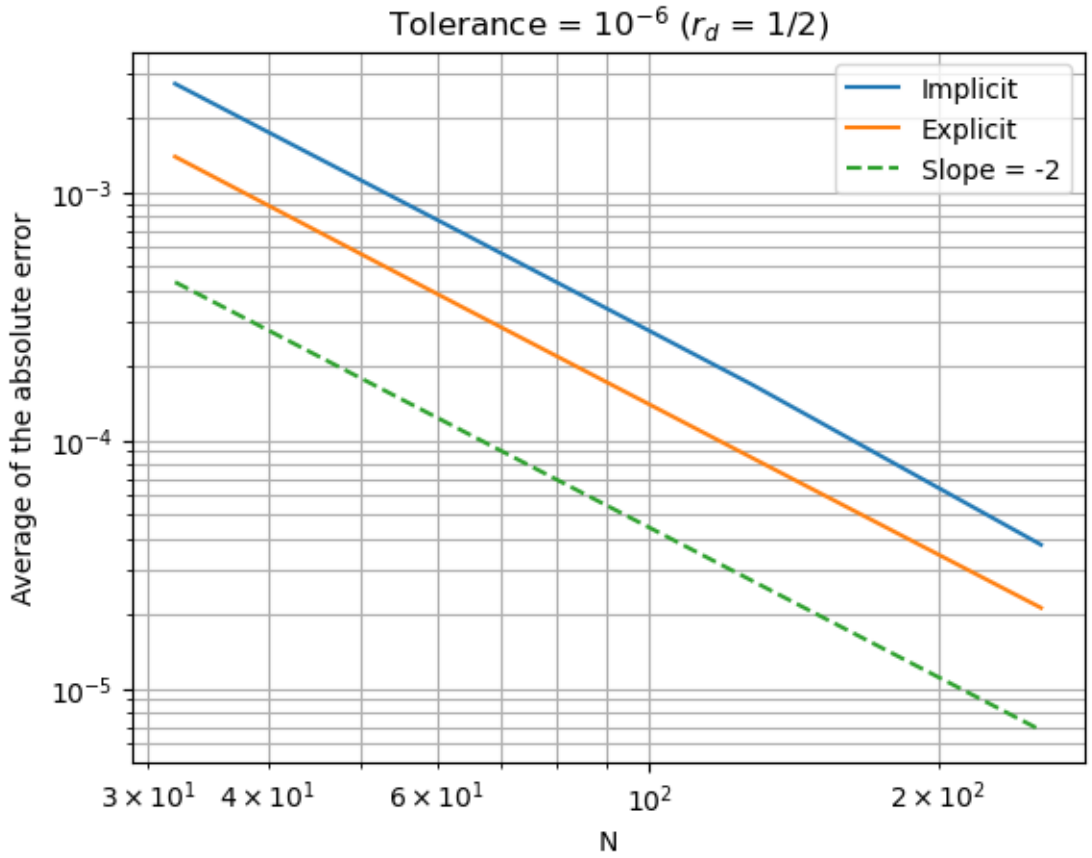
[17]:
```
fig = plt.figure()
plt.loglog(N,rd_half)
plt.loglog(N,rd_one_sixth)
plt.plot(N,(1.50*np.array(N))**(-2),"--")
plt.legend(["r_d = 1/2", "r_d = 1/6", "Slope = -2"])
plt.xlabel("N")
plt.ylabel("Average error")
plt.grid(which='both')
plt.show()
fig.savefig("Average_Error_log.png",dpi = 500, bbox_inches="tight")
```



[18]:
```
Q_4_Error_Data = pd.read_csv("Question_4_Error_Data.csv",header=None)
```

5 Average error vs step size for different methods in log-log scale

[19]:
```
fig = plt.figure()
plt.loglog(N,rd_half)
plt.loglog(N,Q_4_Error_Data)
plt.plot(N,(1.50*np.array(N))**(-2),"--")
plt.legend(["Implicit", "Explicit", "Slope = -2"])
plt.xlabel("N")
plt.ylabel("Average of the absolute error")
plt.grid(which='both')
plt.title(R"Tolerance = $10^{-6}$ ($r_{d}$ = 1/2)")
plt.show()
fig.savefig("Average_Error_log_for_Different_Methods.png",dpi = 500,␣
↪bbox_inches="tight")
```



5.1 **In case of the implicit method, the truncation error dominates, when we are using lesser number of grid points but when we are using a finer grid the propagation error keep on accumulating over iterations and eventually dominates the truncation error. Hence, the error increases on using a finer grid, for a fixed tolerance.**

5.2 **We have found that on decreasing the tolerance from the 10^{-4} to 10^{-6} the error keeps on reducing.**

Problem ⑥ :

The transient 1D heat conduction problem is modelled as

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

where $T(x,t)$ is the temperature and α (a constant) is the thermal diffusivity. This equation is approximated using

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \alpha \left(\frac{T_{i+1}^{n-K} - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \right)$$

where $T_i^n = T(x_i, t_n)$, Δx is the grid spacing, Δt is the time step, and K is a constant ($\neq 0$).

ⓐ Find the expression for truncation error.
ⓑ What is the order of accuracy?
ⓒ Is this a consistent finite difference approximation?

Answer ⑥ ⓐ :

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \alpha \left(\frac{T_{i+1}^{n-K} - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \right) \quad \text{——— (1)}$$

or $$T_i^{n+1} - T_i^n = \frac{\alpha \Delta t}{(\Delta x)^2} \left(T_{i+1}^{n-K} - 2T_i^n + T_{i-1}^n \right)$$

or $$T_i^{n+1} = T_i^n + \left(\frac{\alpha \Delta t}{(\Delta x)^2} \right) \left(T_{i+1}^{n-K} - 2T_i^n + T_{i-1}^n \right)$$

Let $m = \frac{\alpha \Delta t}{(\Delta x)^2}$ — (2)

So, $T_i^{n+1} = T_i^n + m\left(T_{i+1}^{n-k} - 2T_i^n + T_{i-1}^n\right)$ — (3)

Using Taylor Series Expansion,

$$T_i^{n+1} = T_i^n + \frac{\dot{T}_i^n (\Delta t)^1}{1!} + \frac{\ddot{T}_i^n (\Delta t)^2}{2!} + \frac{\dddot{T}_i^n (\Delta t)^3}{3!} + \ldots$$

[NOTE: $\dot{T} = \frac{\partial T}{\partial t}$, $\ddot{T} = \frac{\partial^2 T}{\partial t^2}$, $\dddot{T} = \frac{\partial^3 T}{\partial t^3}$, ...] — (4)

$$T_{i-1}^n = T_i^n + \frac{(T_i^n)'(-\Delta x)^1}{1!} + \frac{(T_i^n)''(-\Delta x)^2}{2!} + \frac{(T_i^n)'''(-\Delta x)^3}{3!} + \frac{(T_i^n)''''(-\Delta x)^4}{4!} + \ldots$$

$$T_{i-1}^n = T_i^n - \frac{(T_i^n)'(\Delta x)^1}{1!} + \frac{(T_i^n)''(\Delta x)^2}{2!} - \frac{(T_i^n)'''(\Delta x)^3}{3!} + \frac{(T_i^n)''''(\Delta x)^4}{4!} - \ldots$$ — (5)

[NOTE: $T' = \frac{\partial T}{\partial x}$, $T'' = \frac{\partial^2 T}{\partial x^2}$, $T''' = \frac{\partial^3 T}{\partial x^3}$, ...]

$$T_{i+1}^{n-k} = T_i^n - \frac{\dot{T}_i^n (k\Delta t)^1}{1!} + \frac{(T_i^n)'(\Delta x)^1}{1!}$$

$$+ \frac{\ddot{T}_i^n (-k\Delta t)^2}{2!} + \frac{(T_i^n)''(\Delta x)^2}{2!}$$

$$- \frac{(\dot{T}_i^n)'(k\Delta t)^1(\Delta x)^1}{1!\,1!} + \cdots \quad - (6)$$

[NOTE: $(\dot{T})' = \frac{\partial}{\partial x}\left(\frac{\partial T}{\partial t}\right) = \frac{\partial^2 T}{\partial x \partial t}$]

Using (3), (4), (5) and (6),

$$T_i^{n+1} = T_i^n + m\left(T_{i+1}^{n-k} - 2T_i^n + T_{i-1}^n\right)$$

or $T_i^n + \frac{\dot{T}_i^n (\Delta t)^1}{1!} + \frac{\ddot{T}_i^n (\Delta t)^2}{2!} + \frac{\dddot{T}_i^n (\Delta t)^3}{3!} + \cdots$

$$= T_i^n + m\Bigg(T_i^n - \frac{\dot{T}_i^n (k\Delta t)^1}{1!} + \frac{(T_i^n)'(\Delta x)^1}{1!} + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!}$$

$$+ \frac{(T_i^n)''(\Delta x)^2}{2!} - \frac{(\dot{T}_i^n)'(k\Delta t)^1(\Delta x)^1}{1!\,1!} + \frac{(T_i^n)'''(\Delta x)^3}{3!} + \cdots$$

$$- 2T_i^n$$

$$+ T_i^n - \frac{(T_i^n)'(\Delta x)^1}{1!} + \frac{(T_i^n)''(\Delta x)^2}{2!} - \frac{(T_i^n)'''(\Delta x)^3}{3!}$$

$$+ \frac{(T_i^n)''''(\Delta x)^4}{4!} - \cdots\Bigg)$$

So,

$$\frac{\dot{T}_i^n (\Delta t)^1}{1!} + \frac{\ddot{T}_i^n (\Delta t)^2}{2!} + \frac{\dddot{T}_i^n (\Delta t)^3}{3!} + \cdots$$

$$= m \left(-\frac{\dot{T}_i^n (k\Delta t)^1}{1!} + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!} + 2\left(\frac{(T_i^n)'' (\Delta x)^2}{2!} \right) \right.$$

$$\left. - \frac{(\dot{T}_i^n)' (k\Delta t)(\Delta x)}{1!\,1!} + 2\left(\frac{(T_i^n)'''' (\Delta x)^4}{4!} \right) + \cdots \right)$$

$$\text{or } \dot{T}_i^n + \frac{\ddot{T}_i^n (\Delta t)}{2!} + \frac{\dddot{T}_i^n (\Delta t)^2}{3!} + \cdots$$

$$= \frac{m(\Delta x)^2}{\Delta t} \left((T_i^n)'' + \frac{2 (T_i^n)'''' (\Delta x)^2}{4!} + \cdots \right)$$

$$+ \frac{m}{\Delta t} \left(-\frac{\dot{T}_i^n (k\Delta t)^1}{1!} + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!} - \frac{(\dot{T}_i^n)' (k\Delta t)(\Delta x)}{1!\,1!} \right.$$

$$\left. + \cdots \right)$$

$$\text{or } \dot{T}_i^n + \frac{\ddot{T}_i^n (\Delta t)}{2!} + \frac{\dddot{T}_i^n (\Delta t)^2}{3!} + \cdots$$

$$= \alpha (T_i^n)'' + \frac{m}{\Delta t} \left(-\frac{\dot{T}_i^n (k\Delta t)^1}{1!} + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!} - \frac{(\dot{T}_i^n)' (k\Delta t)(\Delta x)}{1!\,1!} \right.$$

$$\left. + \frac{2 (T_i^n)'''' (\Delta x)^4}{4!} + \cdots \right)$$

Using (2)

So,

$$\dot{T}_i^n - \alpha (T_i^n)''$$

$$= -\frac{\ddot{T}_i^n (\Delta t)}{2!} - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{m}{\Delta t}\left(-\frac{\dot{T}_i^n (k\Delta t)}{1!} + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!} - \frac{(\dot{T}_i^n)'(k\Delta t)'(\Delta x)'}{1!\,1!} + 2\frac{(T_i^n)''''(\Delta x)^4}{4!} + \cdots \right)$$

As, $m = \frac{\alpha \Delta t}{(\Delta x)^2}$

So, $\dot{T}_i^n - \alpha (T_i^n)''$

$$= -\frac{\ddot{T}_i^n (\Delta t)}{2!} - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{\alpha \Delta t}{(\Delta t)(\Delta x)^2}\left(-\frac{\dot{T}_i^n (k\Delta t)}{1!} + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!} - \frac{(\dot{T}_i^n)'(k\Delta t)'(\Delta x)'}{1!\,1!} + 2\frac{(T_i^n)''''(\Delta x)^4}{4!} + \cdots \right)$$

$$= -\frac{\ddot{T}_i^n (\Delta t)}{2!} - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{\alpha}{(\Delta x)^2}\left(-\dot{T}_i^n (k\Delta t) + \frac{\ddot{T}_i^n (k\Delta t)^2}{2!} - \frac{(\dot{T}_i^n)'(k\Delta t)'(\Delta x)'}{1!\,1!} + 2\frac{(T_i^n)''''(\Delta x)^4}{4!} + \cdots \right)$$

So,

$$\dot{T}_i^n - \alpha (T_i^n)''$$

$$= -\frac{\ddot{T}_i^n (\Delta t)}{2!} - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{2\alpha (T_i^n)'''' (\Delta x)^2}{4!}$$

$$- \frac{\dot{T}_i^n (K\alpha \Delta t)}{(\Delta x)^2} + \ddot{T}_i^n K^2 \alpha \left(\frac{\Delta t}{\Delta x}\right)^2$$

$$- \frac{(\dot{T}_i^n)' (K \Delta t) \alpha}{(\Delta x)} + \cdots \quad \text{—— (7)}$$

So, as $\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$

So, $\dot{T}_i^n - \alpha (T_i^n)'' = 0$

So, the expression for the Truncation Error is. T.E.

$$T.E. = -\ddot{T}_i^n (\Delta t) - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{2\alpha (T_i^n)'''' (\Delta x)^2}{4!}$$

$$- \frac{\dot{T}_i^n (K\alpha \Delta t)}{(\Delta x)^2} + \ddot{T}_i^n K^2 \alpha \left(\frac{\Delta t}{\Delta x}\right)^2$$

$$- \frac{(\dot{T}_i^n)' (K \Delta t) \alpha}{(\Delta x)} + \cdots$$

As $r_d = m = \frac{\alpha \Delta t}{(\Delta x)^2}$

So,

$$T.E. = -\ddot{T}_i^n (\Delta t) - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{2\alpha (\Delta x)^2 (T_i^n)''''}{4!}$$

$$- \frac{\dot{T}_i^n (K \alpha \Delta t)}{(\Delta x)^2} + \ddot{T}_i^n K^2 \alpha \left(\frac{\Delta t}{\Delta x}\right)^2$$

$$- (\ddot{T}_i^n)' K \alpha \frac{\Delta t}{\Delta x} + \cdots$$

So,

$$T.E. = -\ddot{T}_i^n (\Delta t) - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{2\alpha (\Delta x)^2 (T_i^n)''''}{4!}$$

$$- K \dot{T}_i^n r_d + \ddot{T}_i^n K^2 r_d \Delta t$$

$$- (\dot{T}^n)' K r_d (\Delta x) + \cdots$$

Answer (6)(b):

Order of accuracy of the given scheme is 0. This is because of the term $-K \dot{T}_i^n r_d$ in the above expression for the truncation error.

Answer (6)(c):

In equation (7) of answer (6)(a), we derived the modified equation.

$$\dot{T}_i^n - \alpha (T_i^n)''$$

$$= -\frac{\ddot{T}_i^n (\Delta t)}{2!} - \frac{\dddot{T}_i^n (\Delta t)^2}{3!} - \cdots$$

$$+ \frac{2\alpha (T_i^n)'''' (\Delta x)^2}{4!}$$

$$- \frac{\dot{T}_i^n (K\Delta t)\alpha}{(\Delta x)^2} + \ddot{T}_i^n K^2 \alpha \left(\frac{\Delta t}{\Delta x}\right)^2$$

$$- \frac{(\dot{T}_i^n)' (K\Delta t)\alpha}{(\Delta x)} + \cdots$$

In the truncation error, there are terms involving $\frac{\Delta t}{(\Delta x)^2}$, $\left(\frac{\Delta t}{\Delta x}\right)^2$, $\left(\frac{\Delta t}{\Delta x}\right)$ etc. as a factor.

So, the truncation error does NOT approaches 0 if Δt and Δx independently approaches 0.

So, the given scheme is a INCONSISTENT finite difference approximation.