# Assignment 11 - Finite Difference Method

**SGTB Khalsa College, University of Delhi**

**Preetpal Singh(2020PHY1140)(20068567043)**

**Ankur Kumar(2020PHY1113)(20068567010)**

**Unique Paper Code: 32221401**

**Paper Title: Mathematical Physics III**

**Submitted on: April 29, 2022**

**B.Sc(H) Physics Sem IV**

**Submitted to: Dr. Mamta**

# Finite Difference Method

Theory :

a) Dirichlet Boundary conditions

$$y'' = p(x) y' + q(x) y + r(x) \quad , x \in [a, b]$$

s.t $y(a) = \alpha$, $y(b) = \beta$

$y_i \to$ exact value of the sol^n at $x = x_i$

$w_i \to$ finite difference approximation to $y_i$.

$x_i \to$ grid points , $x_i \in [a, b]$

$w_i$ is obtained by replacing derivatives by the finite difference formula.

Converts ODE into a system of discrete algebraic eqns. for $w_0, w_1 \dots w_N$.

Grid

$$x_i = a + ih \quad \text{where } h = \frac{b-a}{N}$$

$h \to$ step size.

Finite diff. approximation

evaluate diff eq. at each interior grid point

$$\left\{ y'' = p(x) y' + q(x) y + r(x) \right\} \Big|_{x = x_i} \quad 1 \le i \le N-1$$

Date / /  Page no.

Replace derivatives by finite diff approximation

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + O(h^2) = p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i + r_i + O(h^2)$$

$p_i, q_i, r_i \rightarrow p(x_i), q(x_i), r(x_i)$

Drop truncation errors and replace y's with w's.
for $i = 1, 2 \ldots N-1$.

$\{1 \le i \le N-1\}$ $$\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} = p_i \frac{w_{i+1} - w_{i-1}}{2h} + q_i w_i + r_i \quad (1)$$

Two more eqs. come from the boundary conds

$$w_0 = \alpha \quad (2)$$
$$w_N = \beta \quad (3)$$

These 3 equations constitute the finite diff method.

Multiply (1) by $-h^2$ and collect terms to get:

$$\left(-1 - \frac{h}{2} p_i\right) w_{i-1} + \left(2 + h^2 q_i\right) w_i + \left(-1 + \frac{h}{2} p_i\right) w_{i+1} = -h^2 r_i$$

This gives us N+1 equations.
∴ The finite diff eqs. can be written in the form $AW = b$ where
$A \rightarrow (N+1) \times (N+1)$ tridiagonal matrix.

$$A = \begin{bmatrix} 1 & 0 & & & & & \\ l_1 & d_1 & u_1 & & & & \\ & l_2 & d_2 & u_2 & & & \\ & & \vdots & \vdots & \vdots & & \\ & & & l_{n-2} & d_{N-2} & u_{N-1} & \\ & & & & l_{N-1} & d_{N-1} & u_{N-1} \\ & & & & & 0 & 1 \end{bmatrix}$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-2} \\ \cancel{w_{N-1}} \\ w_N \end{bmatrix} \qquad b = \begin{bmatrix} \alpha \\ -h^2 r_1 \\ -h^2 r_2 \\ \vdots \\ -h^2 r_{N-1} \\ \beta \end{bmatrix}$$

Where $d_i = 2 + h^2 q_i$

$u_i = -1 + \frac{h}{2} p_i$

$l_i = -1 - \frac{h}{2} p_i$

It can be proved that above matrix has a unique solution.

$w = A^{-1} \cdot b$

we can find $w$ (matrix) which is the finite difference approximation at each point.

Reference → A friendly introduction to Numerical Analysis (Brian Bradie)

Date / /                                                            Page no.

Non-Dirichlet Boundary Conditions

The most general B.C. are the robin B.C.;

$$y'' = p(x)y' + q(x)y + r(x), \quad x \in [a,b]$$

S.t. $\alpha_1 y(a) + \alpha_2 y'(a) = \alpha_3$

$\beta_1 y(b) + \beta_2 y'(b) = \beta_3$

The grid remains the same.
Notations are the same.
we need N+1 eqs. for $w_0, w_1, \dots w_N$.
N-1 eqs. are obtained similarly as before:

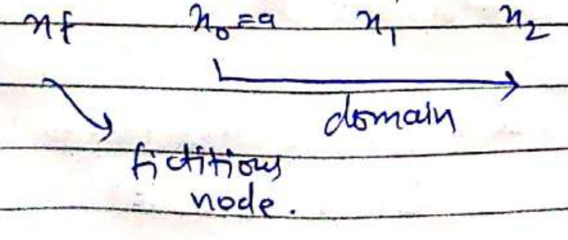
Template $\leftarrow$ $\left(-1-\frac{h}{2}p_i\right)w_{i-1} + \left(2+h^2 q_i\right)w_i + \left(-1+\frac{h}{2}p_i\right)w_{i+1} = -h^2 r_i$

we introduce a fictitious node to the grid.
$\rightarrow w_f$.

Applying the template at $x = x_0$,

$$\left(-1-\frac{h}{2}p_0\right)w_f + \left(2+h^2 q_0\right)w_0 + \left(-1+\frac{h}{2}p_0\right)w_1 = -h^2 r_0 \quad (1)$$

$w_f$ is to be eliminated.



DELTA My NoteBook

To remove $w_f$ :

$\alpha_1 y(a) + \alpha_2 y'(a) = \alpha_3 \Rightarrow \alpha_1 w_0 + \alpha_2 \frac{w_1 - w_f}{2h} = \alpha_3,$

$\Rightarrow w_f = w_1 - \frac{2h}{\alpha_2}(\alpha_3 - \alpha_1 w_0)$

Putting this ①

$$\left[2 + h^2 q_0 - (2 + h p_0) h \frac{\alpha_1}{\alpha_2}\right] w_0 - 2w_1 = -h^2 r_0 - (2 + h p_0) h \frac{\alpha_3}{\alpha_2},$$

for newmann B.C $\alpha_1 = 0$.

$\Rightarrow (2 + h^2 q_0) w_0 - 2w_1 = -h^2 r_0 - (2 + h p_0) h\alpha$

where, $\alpha = \alpha_3/\alpha_2$

Similarly at $x = b$,

$$-2w_{N-1} + \left[2 + h^2 q_N + (2 - h p_N) h \frac{\beta_1}{\beta_2}\right] w_N$$

$$= -h^2 r_N + (2 - h p_N) h \frac{\beta_3}{\beta_2}$$

for newmann B.C,

$-2w_{N-1} + (2 + h^2 q_N) w_N = -h^2 r_N + (2 - h p_N) h\beta$

where, $\beta = \beta_3/\beta_2$

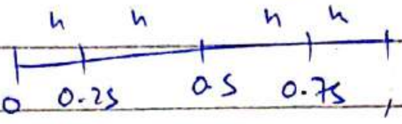
This gives us all $N+1$ eqs.

$W \rightarrow$ unknown matrix can be found by taking inverse of A and multiplying with b.

This will give us the finite diff. approximation.

b) $$-u'' + \pi^2 u = 2\pi^2 \sin(\pi x)$$
$$u(0) = u(1) = 0.$$

$p(x) = 0,$
$q(x) = \pi^2$
$r(x) = -2\pi^2 \sin(\pi x)$
$N = 4 \quad x \in [0, 1]$



$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{(1/4)^2} + O(h^2) + \pi^2 u_i = 2\pi^2 \sin\left(\frac{i\pi}{4}\right)$$

$$\Rightarrow -w_{i-1} + [2 + (\pi/4)^2] w_i - w_{i+1} = 2(\pi/4)^2 \sin(i\pi/4)$$

Writing out eqs. for $i = 1, 2, 3$, and using B.C we get:

$$\begin{bmatrix} 1 & 0 & & & \\ -1 & 2+(\pi/4)^2 & -1 & & \\ & -1 & 2+(\pi/4)^2 & -1 & \\ & & -1 & 2+(\pi/4)^2 & -1 \\ & & & 0 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{2}(\pi/4)^2 \\ 2(\pi/4)^2 \\ \sqrt{2}(\pi/4)^2 \\ 0 \end{bmatrix}$$

$$w = [\,0 \quad 0.725371 \quad 1.025830 \quad 0.725371 \quad 0\,]^T$$

Comparing with actual solution,

$y_{exact} = \sin(\pi x)$

$\uparrow$

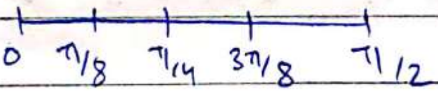
| $x_i$ | Approximate sol$^n$ $w_i$ | Exact sol$_n$ $u_i$ | Absolute error |
|---|---|---|---|
| 0.00 | 0.0000 | 0.0000 | 0 |
| 0.25 | 0.725371 | 0.707107 | 0.018264 |
| 0.50 | 1.025830 | 1.0000 | 0.025830 |
| 0.75 | 0.725371 | 0.707107 | 0.018264 |
| 1.00 | 0.0000 | 0.0000 | 0 |

$$u'' + u = \sin(3x), \quad x \in [0, \pi/2]$$

$$u(0) + u'(0) = -1$$

$$u'(\pi/2) = 1$$

$N = 4$



$p(x) = 0, \; q(x) = -1, \; r(x) = \sin(3x).$

$p_i = 0; \; q_i = -1; \; r_i = \sin(3i\pi/8).$

For Robic B.C at $x = 0$, $\alpha_1 = \alpha_2 = 1$, $\alpha_3 = -1$

For neumann B.C at $x = \pi/2$, $\beta = 1$.

$$\begin{bmatrix} d-\pi/4 & -2 & & & \\ -1 & d & -1 & & \\ & -1 & d & -1 & \\ & & -1 & d & -1 \\ & & & -2 & d \end{bmatrix} \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \begin{bmatrix} \pi/4 \\ -(\frac{\pi}{8})^2 \sin(3\pi/8) \\ -(\frac{\pi}{8})^2 \sin(3\pi/4) \\ -(\pi/8)^2 \sin(9\pi/8) \\ -(\pi/8)^2 + \pi/4 \end{bmatrix}$$

where $d = 2 - (\pi/8)^2$

$$\omega = [-1.023672 \ \ -0.935445 \ \ -0.560486 \ \ 0.0099175 \ \ 0.519840]^T$$

$$y_{exact} = \frac{3}{8}\sin(x) - \cos(x) - \frac{1}{8}\sin(3x)$$

| $x_i$ | Approx. sol$^n$ $w_i$ | Exact sol$^n$ $v_i$ | Absolute error. |
|---|---|---|---|
| 0 | -1.023672 | -1.0000 | 0.023672 |
| $\pi/8$ | -0.935445 | -0.895858 | 0.039587 |
| $\pi/4$ | -0.56048 | -0.530330 | 0.030156 |
| $3\pi/8$ | 0.0099175 | 0.0116068 | 0.001655 |
| $\pi/2$ | 0.519840 | 0.5000 | 0.019840 |

## Programming

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math as m
from scipy.stats import linregress


def finite_diff_method(a,b,alpha1,alpha2,alpha3,beta1,beta2,beta3,N,func_p,func_q,
    func_r):

    def func_ldu(x_arr):
        arr_l=np.array([])
        arr_d=np.array([])
        arr_u=np.array([])
        b_vec=np.array([])

        arr_l=-1-(h/2)*func_p(x_arr)
        arr_d=2+(h**2)*func_q(x_arr)
        arr_u=-1+(h/2)*func_p(x_arr)
        arr_r=func_r(x_arr)
        b_vec=(-1*(h**2)*arr_r)

        if alpha2==0:
            a11=1
            a12=0
            a_n1=1
            a_n2=0
            b1=(alpha3/alpha1)
            b_n1=(beta3/beta1)

        else:
            a11=arr_d[0]+((2*h*arr_l[0]*alpha1)/alpha2)
            a12=-2
            a_n1=arr_d[-1]-((2*h*arr_u[-1]*beta1)/beta2)
            a_n2=-2
            b1=(-1*(h**2))*arr_r[0]+((2*h*arr_l[0]*alpha3)/alpha2)
            b_n1=(-1*(h**2))*arr_r[-1]-((2*h*arr_u[-1]*beta3)/beta2)

        arr_d[0]=a11
        arr_d[-1]=a_n1
        arr_u[0]=a12
        arr_l[-1]=a_n2
        b_vec[0]=b1
        b_vec[-1]=b_n1

        return arr_l[1:],arr_d,arr_u[:-1],b_vec


    def form_tri_matrix(li,di,ui,x_arr):
        N=len(di)

        A_mat=np.zeros((N,N))

        for i in range(N):
            A_mat[i][i]=di[i]
            if i<N-1:
                A_mat[i][i+1]=ui[i]
                A_mat[i+1][i]=li[i]
        return A_mat


    def thomas_algo(A_mat,b_vec):
        N=len(b_vec)
        a=np.zeros((N))
        b=np.zeros((N))
        c=np.zeros((N))
        d=np.zeros((N))

        d=b_vec
        for i in range(N):
            b[i]=A_mat[i][i]
            if i<(N-1) :
                a[i+1]=A_mat[i+1][i]
                c[i]=A_mat[i][i+1]

        cp = np.zeros(N)
        dp = np.zeros(N)
        X = np.zeros(N)

        cp[0] = c[0]/b[0]
        dp[0] = d[0]/b[0]

        for i in np.arange(1,(N),1):
            dnum = b[i] - a[i]*cp[i-1]
            cp[i] = c[i]/dnum
            dp[i] = (d[i]-a[i]*dp[i-1])/dnum

        # Perform Back Substitution
        X[(N-1)] = dp[N-1]  # Obtain last xn

        for i in np.arange((N-2),-1,-1):  # use x[i+1] to obtain x[i]
            X[i] = (dp[i]) - (cp[i])*(X[i+1])

        return(X)


    h=(b-a)/(N+1)
    x_arr=np.linspace(a,b,N+2,float)
    li,di,ui,bv=func_ldu(x_arr)
    A_mat=form_tri_matrix(li,di,ui,x_arr)
    omega=thomas_algo(A_mat,bv)

    return x_arr,omega

def calc_rms_error(y_vec1,y_vec2):
    sum_ele=0
    for i in range(len(y_vec1)):
        ele=(y_vec1[i]-y_vec2[i])**2
        sum_ele=sum_ele+ele
    ans=m.sqrt(sum_ele/len(y_vec1))
    return ans


#Q1
def func_p1(x):
    ans_arr=np.zeros(len(x))
    for i in range(len(x)):
        ans_arr[i]=0
    return ans_arr

def func_q1(x):
    ans_arr=np.zeros(len(x))
    for i in range(len(x)):
        ans_arr[i]=(np.pi**2)
    return ans_arr

def func_r1(x):
    ans_arr=np.zeros(len(x))
    for i in range(len(x)):
        ans_arr[i]=-2*(np.pi**2)*np.sin(np.pi*x[i])
    return ans_arr

```

```python
def analytic_x1(x):
    return np.sin(np.pi*x)

x_vals11, approximation11 = finite_diff_method(0, 1, 1, 0, 0, 1, 0, 0, 3 ,func_p1,
    func_q1,func_r1)
y_anal11 = analytic_x1(x_vals11)
abs_error11 = np.abs(y_anal11 - approximation11)
err_mat11=np.column_stack((x_vals11,approximation11,y_anal11,abs_error11))
print("N=3")
data_11=pd.DataFrame(err_mat11,columns=["x_i","y_num","y_analytic","error"])
print(data_11)


x_vals12, approximation12 = finite_diff_method(0, 1, 1, 0, 0, 1, 0, 0, 8 ,func_p1,
    func_q1,func_r1)
y_anal12 = analytic_x1(x_vals12)
abs_error12 = np.abs(y_anal12 - approximation12)
err_mat12=np.column_stack((x_vals12,approximation12,y_anal12,abs_error12))
print("N=8")
data_12=pd.DataFrame(err_mat12,columns=["x_i","y_num","y_analytic","error"])
print(data_12)

N1 = []
max_abs_error = []
max_abs_error_ratio = [0]
rms_error = []
rms_error_ratio = [0]

for i in range(1, 7):
    N = 2**i
    x_vals1, approximation1 = finite_diff_method(0, 1, 1, 0, 0, 1, 0, 0, N ,func_p1
    , func_q1,func_r1)
    y_anal1 = analytic_x1(x_vals1)
    abs_error1 = np.abs(y_anal1 - approximation1)
    rms_error1 = calc_rms_error(y_anal1, approximation1)
    max_abs_error1 = np.max(abs_error1)
    plt.plot(x_vals1, approximation1, label = "N={}".format(N), linestyle='dashed')
    plt.scatter(x_vals1, approximation1, s = 10)

    N1.append(N)
    max_abs_error.append(max_abs_error1)
    rms_error.append(rms_error1)

x = np.linspace(0, 1, 100)
plt.plot(x, analytic_x1(x), label = 'Analytic Solution')
plt.title('Variation of solution with N')
plt.xlabel('x')
plt.ylabel('Solution(y)')
plt.legend()
plt.grid()
plt.show()

for i in range(0,5):
    ratio1 = max_abs_error[i]/max_abs_error[i+1]
    max_abs_error_ratio.append(ratio1)

    ratio2 = rms_error[i]/rms_error[i+1]
    rms_error_ratio.append(ratio2)


convergence_data1 =np.column_stack((N1,max_abs_error ,max_abs_error_ratio ,
    rms_error,rms_error_ratio))
convergence_table1=pd.DataFrame(convergence_data1,columns=["N","max_abs_error","
    Error Ratio","Rms Error","Error Ratio"])
print(convergence_table1)

plt.plot(N1, max_abs_error, label = 'Error')
plt.scatter(N1, max_abs_error)
plt.xscale('log')
plt.yscale('log')
plt.title('Log Plot')
plt.xlabel('N')
plt.ylabel('Max absolute error')
plt.legend()
plt.grid()
plt.show()

log_x=np.log10(N1)
log_y=np.log10(max_abs_error)
print("slope,intercept:",linregress(log_x,log_y)[0:2])


#Q2

def func_p2(x):
    ans_arr=np.zeros(len(x))
    for i in range(len(x)):
        ans_arr[i]=0
    return ans_arr

def func_q2(x):
    ans_arr=np.zeros(len(x))
    for i in range(len(x)):
        ans_arr[i]=-1
    return ans_arr

def func_r2(x):
    ans_arr=np.zeros(len(x))
    for i in range(len(x)):
        ans_arr[i]=np.sin(3*x[i])
    return ans_arr

def analytic_x2(x):
    return (3/8)*np.sin(x)-np.cos(x)-(1/8)*np.sin(3*x)


x_vals21, approximation21 = finite_diff_method(0, np.pi/2, 1, 1, -1, 0, 1, 1, 3 ,
    func_p2, func_q2,func_r2)
y_anal21 = analytic_x2(x_vals21)
abs_error21 = np.abs(y_anal21 - approximation21)
err_mat21=np.column_stack((x_vals21,approximation21,y_anal21,abs_error21))
print("N=3")
data_21=pd.DataFrame(err_mat21,columns=["x_i","y_num","y_analytic","error"])
print(data_21)


x_vals22, approximation22 = finite_diff_method(0, np.pi/2, 1, 1, -1, 0, 1, 1, 8 ,
    func_p2, func_q2,func_r2)
y_anal22 = analytic_x2(x_vals22)
abs_error22 = np.abs(y_anal22 - approximation22)
err_mat22=np.column_stack((x_vals22,approximation22,y_anal22,abs_error22))
print("N=8")
data_22=pd.DataFrame(err_mat22,columns=["x_i","y_num","y_analytic","error"])
print(data_22)

N2 = []
max_abs_error = []
max_abs_error_ratio = [0]
rms_error = []
rms_error_ratio = [0]

for i in range(1, 7):
    N = 2**i
    x_vals2, approximation2 = finite_diff_method(0, np.pi/2, 1, 1, -1, 0, 1, 1, N ,
    func_p2, func_q2,func_r2)
    y_anal2 = analytic_x2(x_vals2)
    abs_error2 = np.abs(y_anal2 - approximation2)
    rms_error2 = calc_rms_error(y_anal2, approximation2)
    max_abs_error2 = np.max(abs_error2)
    plt.plot(x_vals2, approximation2, label = "N={}".format(N), linestyle='dashed')
    plt.scatter(x_vals2, approximation2, s = 10)

    N2.append(N)
    max_abs_error.append(max_abs_error2)
    rms_error.append(rms_error2)

x = np.linspace(0, np.pi/2, 100)
plt.plot(x, analytic_x2(x), label = 'Analytic Solution')
plt.title('Variation of solution with N')
plt.xlabel('x')
plt.ylabel('Solution(y)')
plt.legend()
plt.grid()
plt.show()


for i in range(0,5):
    ratio1 = max_abs_error[i]/max_abs_error[i+1]
    max_abs_error_ratio.append(ratio1)

    ratio2 = rms_error[i]/rms_error[i+1]
    rms_error_ratio.append(ratio2)


convergence_data2 =np.column_stack((N2,max_abs_error ,max_abs_error_ratio ,
    rms_error,rms_error_ratio))
convergence_table2=pd.DataFrame(convergence_data2,columns=["N","max_abs_error","
    Error Ratio","Rms Error","Error Ratio"])
print(convergence_table2)

plt.plot(N2, max_abs_error, label = 'Error')
plt.scatter(N2, max_abs_error)
plt.xscale('log')
plt.yscale('log')
plt.title('Log Plot')
plt.xlabel('N')
plt.ylabel('Max absolute error')
plt.legend()
plt.grid()
plt.show()

log_x=np.log10(N2)
log_y=np.log10(max_abs_error)
print("slope,intercept:",linregress(log_x,log_y)[0:2])
```

## Result and Discussion

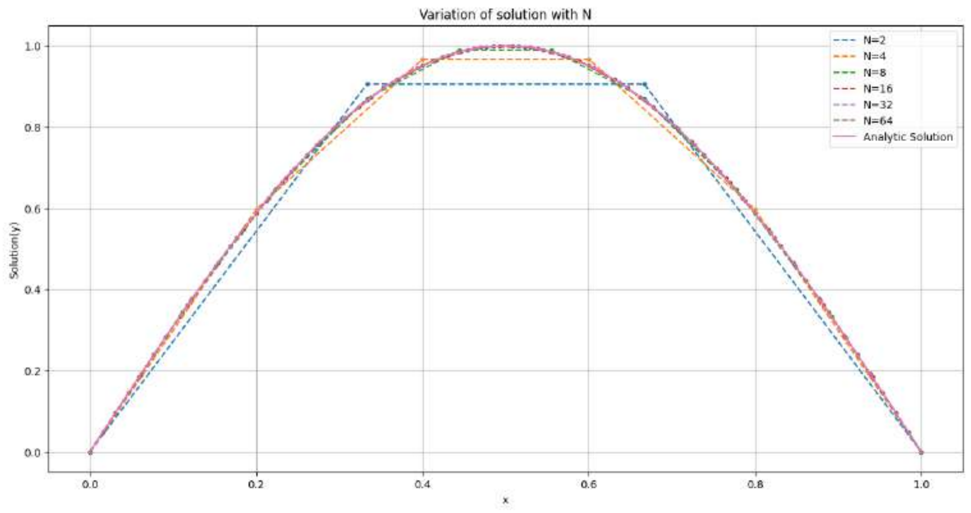


Figure 1: Q1 Variation with N

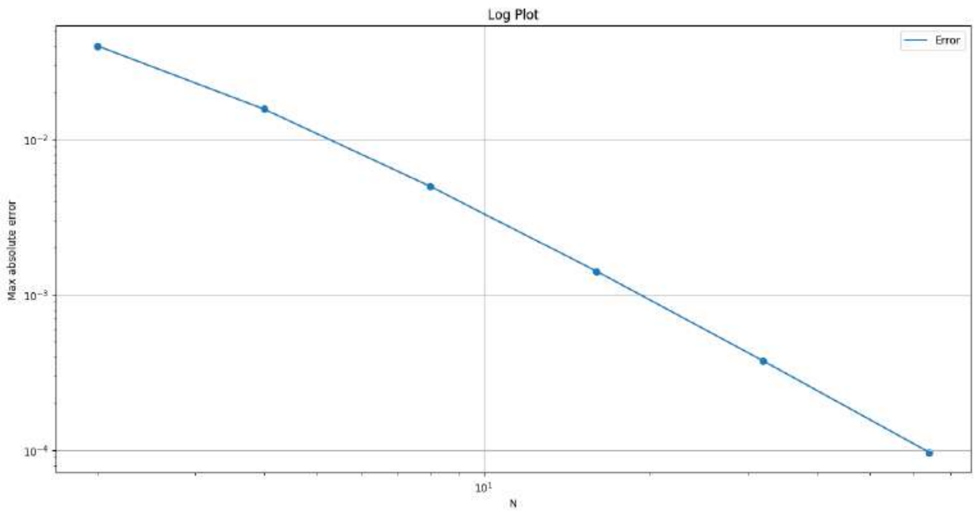


Figure 2: Q1 Log Plot

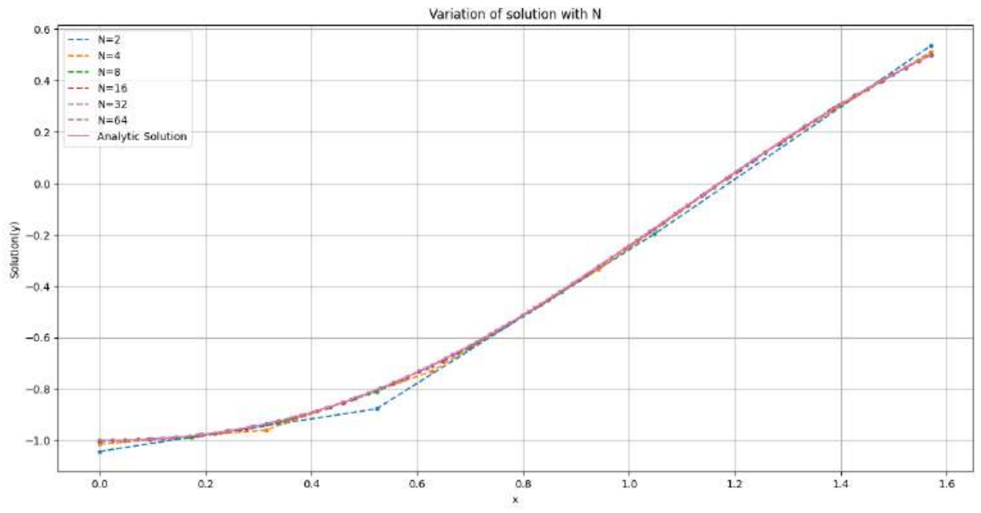


Figure 3: Q2 Variation with N

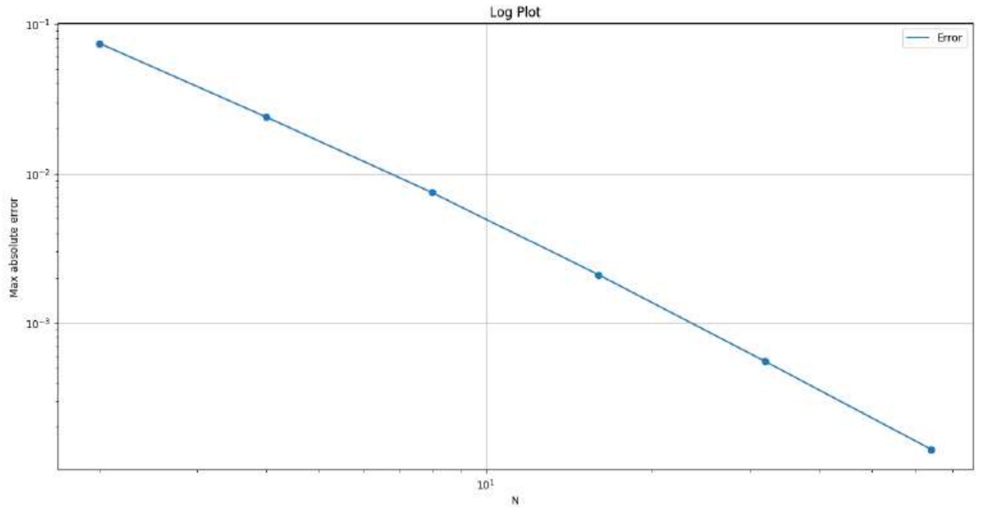
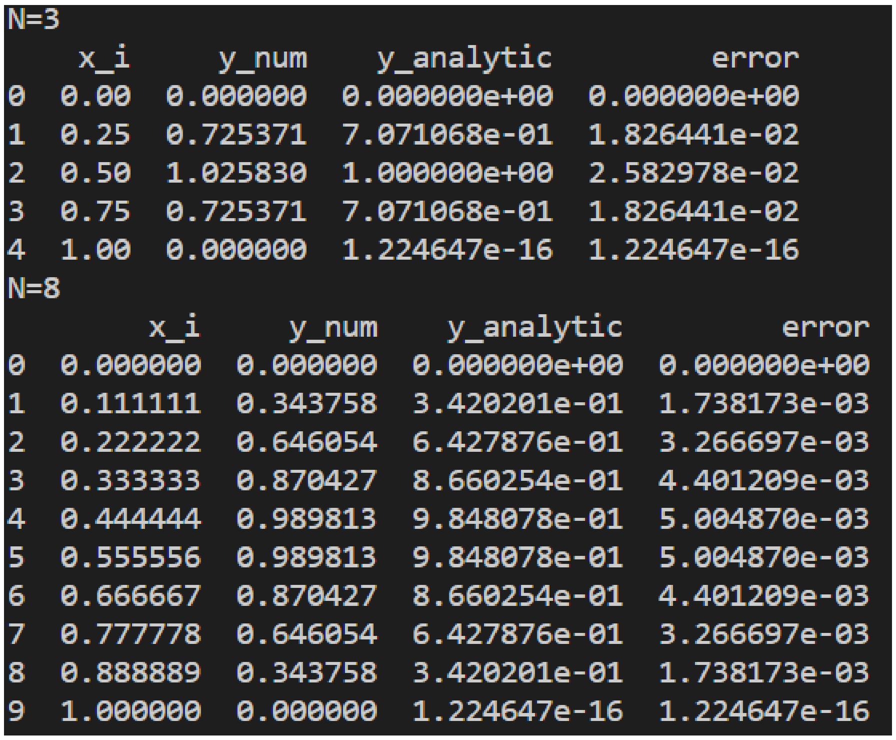


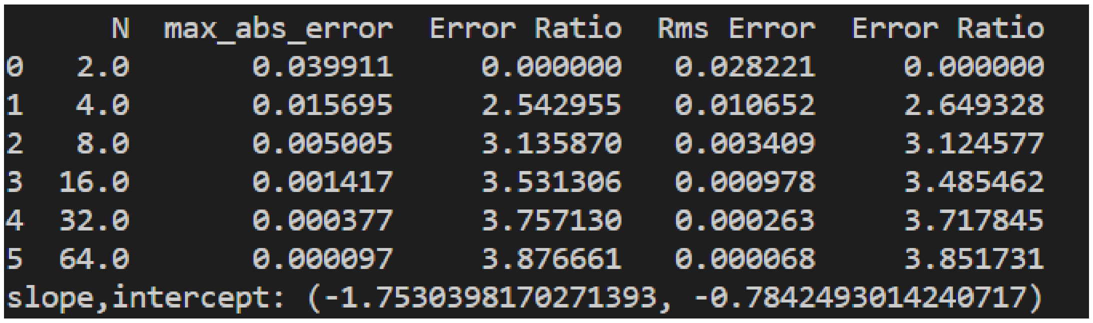
Figure 4: Q2 Log Plot

It can be seen that as the value of N increases the solution approaches the true solution.
From the Log error plot it can be seen that the error decreases contiously as the value N increases.



```
N=3
    x_i     y_num    y_analytic         error
0  0.00  0.000000  0.000000e+00  0.000000e+00
1  0.25  0.725371  7.071068e-01  1.826441e-02
2  0.50  1.025830  1.000000e+00  2.582978e-02
3  0.75  0.725371  7.071068e-01  1.826441e-02
4  1.00  0.000000  1.224647e-16  1.224647e-16
N=8
        x_i     y_num    y_analytic         error
0  0.000000  0.000000  0.000000e+00  0.000000e+00
1  0.111111  0.343758  3.420201e-01  1.738173e-03
2  0.222222  0.646054  6.427876e-01  3.266697e-03
3  0.333333  0.870427  8.660254e-01  4.401209e-03
4  0.444444  0.989813  9.848078e-01  5.004870e-03
5  0.555556  0.989813  9.848078e-01  5.004870e-03
6  0.666667  0.870427  8.660254e-01  4.401209e-03
7  0.777778  0.646054  6.427876e-01  3.266697e-03
8  0.888889  0.343758  3.420201e-01  1.738173e-03
9  1.000000  0.000000  1.224647e-16  1.224647e-16
```

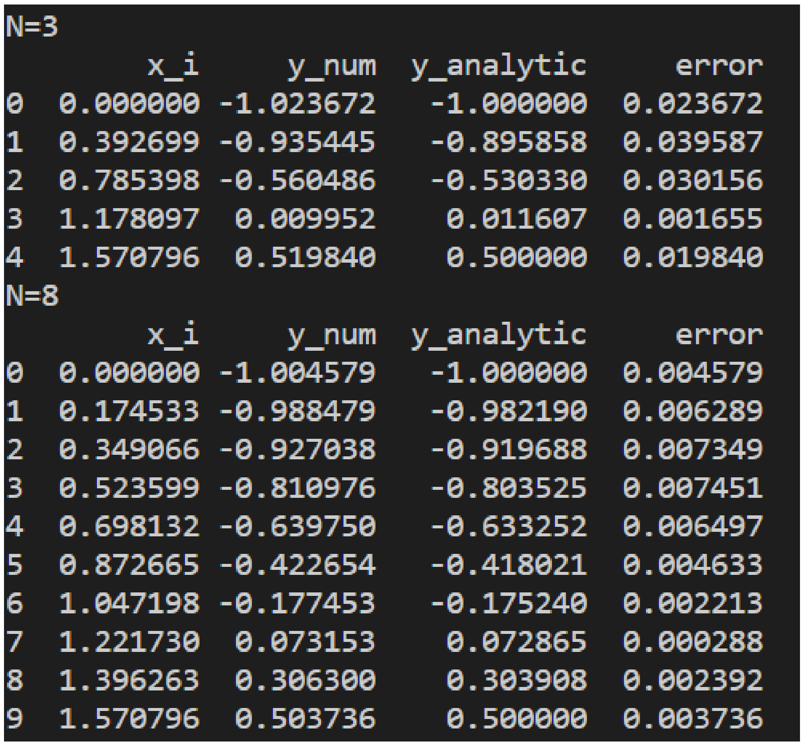
Figure 5: Q1 solution for N=3,8



```
      N  max_abs_error  Error Ratio  Rms Error  Error Ratio
0   2.0       0.039911     0.000000   0.028221     0.000000
1   4.0       0.015695     2.542955   0.010652     2.649328
2   8.0       0.005005     3.135870   0.003409     3.124577
3  16.0       0.001417     3.531306   0.000978     3.485462
4  32.0       0.000377     3.757130   0.000263     3.717845
5  64.0       0.000097     3.876661   0.000068     3.851731
slope,intercept: (-1.7530398170271393, -0.7842493014240717)
```

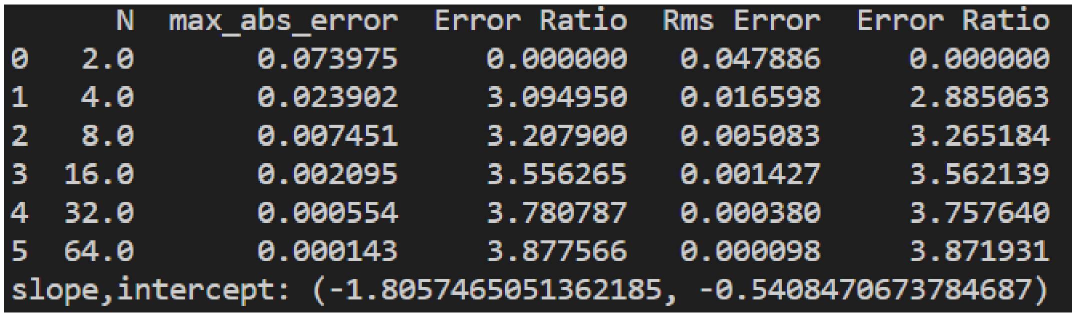
Figure 6: Convergence of Q1 and Slope, Intercept

For higher values of N the solution becomes more accurate.
The slope and intercept come out to be the values shown above.
The error ratio slowly converges to 4 as predicted.



```
N=3
        x_i     y_num  y_analytic     error
0  0.000000 -1.023672   -1.000000  0.023672
1  0.392699 -0.935445   -0.895858  0.039587
2  0.785398 -0.560486   -0.530330  0.030156
3  1.178097  0.009952    0.011607  0.001655
4  1.570796  0.519840    0.500000  0.019840
N=8
        x_i     y_num  y_analytic     error
0  0.000000 -1.004579   -1.000000  0.004579
1  0.174533 -0.988479   -0.982190  0.006289
2  0.349066 -0.927038   -0.919688  0.007349
3  0.523599 -0.810976   -0.803525  0.007451
4  0.698132 -0.639750   -0.633252  0.006497
5  0.872665 -0.422654   -0.418021  0.004633
6  1.047198 -0.177453   -0.175240  0.002213
7  1.221730  0.073153    0.072865  0.000288
8  1.396263  0.306300    0.303908  0.002392
9  1.570796  0.503736    0.500000  0.003736
```

Figure 7: Q2 solution for N=3,8



```
      N  max_abs_error  Error Ratio  Rms Error  Error Ratio
0   2.0       0.073975     0.000000   0.047886     0.000000
1   4.0       0.023902     3.094950   0.016598     2.885063
2   8.0       0.007451     3.207900   0.005083     3.265184
3  16.0       0.002095     3.556265   0.001427     3.562139
4  32.0       0.000554     3.780787   0.000380     3.757640
5  64.0       0.000143     3.877566   0.000098     3.871931
slope,intercept: (-1.8057465051362185, -0.5408470673784687)
```

Figure 8: Convergence of Q2 and Slope, Intercept

For higher values of N the solution becomes more accurate.
The slope and intercept come out to be the values shown above.
The error ratio slowly converges to 4 as predicted.