RAMAKRISHNA MISSION VIVEKANANDA CENTENARY COLLEGE

KOLKATA - 700118

MATHEMATICAL PHYSICS PRACTICAL NOTEBOOK

Core Course : 5

SUBMITTED BY –

ANIK MANDAL

(B.Sc. 3rd semester student of Department Of Physics)

REGESTRATION NO. : A01-1152-111-042-2018

ADMIT NO. :

YEAR : 2019

CONTENT

1. Write a program to determine of the value of nCr and nPr for a n=7 and r =[0,7] :

PROGRAM:

```
# DETERMINATION OF THE VALUE OF nCr AND nPr :

print("\n\tDETERMINATION OF THE VALUE OF nCr AND nPr :\n")
n = 7
r = 0

while r <= n:
    (fn, fr, fnr) = (1, 1, 1)

    for i in range(1, n+1, 1):
        fn = fn*i

    for j in range(1, r+1, 1):
        fr = fr*j

    for k in range(1, n-r+1, 1):
        fnr = fnr*k

    nCr = fn/(fr*fnr)
    nPr = fn/fnr

    print("nCr for n:", n, "r :", r, " equals to :", nCr)
    print("npr for n:", n, "r :", r, " equals to :", nPr)
    r = r+1
```

OUTPUT:

```
      DETERMINATION OF THE VALUE OF nCr AND nPr :

nCr for n: 7 r : 0  equals to : 1.0
nPr for n: 7 r : 0  equals to : 1.0
nCr for n: 7 r : 1  equals to : 7.0
nPr for n: 7 r : 1  equals to : 7.0
nCr for n: 7 r : 2  equals to : 21.0
nPr for n: 7 r : 2  equals to : 42.0
nCr for n: 7 r : 3  equals to : 35.0
```

```
nPr for n: 7 r : 3  equals to : 210.0
nCr for n: 7 r : 4  equals to : 35.0
nPr for n: 7 r : 4  equals to : 840.0
nCr for n: 7 r : 5  equals to : 21.0
nPr for n: 7 r : 5  equals to : 2520.0
nCr for n: 7 r : 6  equals to : 7.0
nPr for n: 7 r : 6  equals to : 5040.0
nCr for n: 7 r : 7  equals to : 1.0
nPr for n: 7 r : 7  equals to : 5040.0
```

2. Write a program to determine the value of sin(x) with error for x= 30°:

Taylor Series of sin(x):

$$\sin(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{(2i+1)}}{(2i+1)!}$$

PROGRAM:

```
# DETERMINATION OF THE VALUE OF sin(30) WITH ERROR APPROXIMATION :


import math
print("\n\tDETERMINATION OF THE VALUE OF sin(30) WITH ERROR APPROXIMATION
:\n")


a = 30.0
x = a*math.pi/180.0
eps = 0.00001


(i, fact, s, err, z, tm) = (0.0, 1.0, 0.0, 1.0, 1.0, 0.0)


while err > eps:
    i = i + 1.0
    fact = fact * i


    if i % 2.0 == 1.0:
        z = z+1.0
        t = (x ** i)*((-1.0)**z) / fact
        s = s + t
```

```
        err = abs(t)
        tm = tm+1.0

print("The value of the sin(", a, ") with maximum error approximation",
eps, " :\t", s)
print("The total number of terms required to reach the accuracy :", tm)
```

OUTPUT:

```
DETERMINATION OF THE VALUE OF sin(30) WITH ERROR APPROXIMATION :

The value of the sin( 30.0 ) with maximum error approximation 1e-05  :
0.4999999918690232
The total number of terms required to reach the accuracy : 4.0
```

3. Write a program to determine the value of cos(x) with error for x= 60°:

Taylor Series of cos(x):

$$\cos(x) = \sum_{i=0}^{\infty}(-1)^i \frac{x^{2i}}{(2i)!}$$

PROGRAM:

```
# DETERMINATION OF THE VALUE OF cos(60) WITH ERROR APPROXIMATION :

import math
print("\n\tDETERMINATION OF THE VALUE OF cos (60) WITH ERROR APPROXIMATION
:\n")

a = 60.0
x = a*math.pi/180
eps = 0.00001

(i, fact, s, err, z, tm) = (0.0, 1.0, 1.0, 1.0, 0.0, 0.0)

while err >= eps:
    i = i + 1.0
    fact = fact * i
```

```
        if i % 2.0 == 0.0:
            z = z+1
            t = (x ** i)*((-1.0)**z) / fact
            s = s + t

            err = abs(t)
            tm = tm+1.0

    print("The value of the cos(", a, ") with maximum error approximation",
    eps, " :\t", s)
    print("The total number of terms required to reach the accuracy :", tm)
```

OUTPUT:

```
      DETERMINATION OF THE VALUE OF cos (60) WITH ERROR APPROXIMATION :


The value of the cos( 60.0 ) with maximum error approximation 1e-05  :
      0.4999999963909432
The total number of terms required to reach the accuracy : 5.0
```

4. Write a program to determine the value of exp(x) with error for x= 5:

Taylor Series of e^x:

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

PROGRAM:

```
# DETERMINATION OF THE VALUE OF exp(5) WITH ERROR APPROXIMATION :


print("\n\tDETERMINATION OF THE VALUE OF exp(5) WITH ERROR APPROXIMATION
:\n")


x = 5.0
eps = 0.000001


(i, fact, s, err, tm) = (0.0, 1.0, 1.0, 1.0, 0.0)
```

```
while err > eps:
    i = i + 1.0
    fact = fact*i
    t = (x**i)/fact
    s = s+t

    err = abs(t)
    tm = tm+1.0

print("The value of the exp(", x, ") with maximum approximation", eps, " :\t", s)
print("The total number of terms required to reach the accuracy :", tm)
```

OUTPUT:

```
DETERMINATION OF THE VALUE OF exp(5) WITH ERROR APPROXIMATION :


The value of the exp( 5.0 ) with maximum approximation 1e-06  : 148.41315898276957
The total number of terms required to reach the accuracy : 23.0
```

5. Write a program to determine the value of $\log_e(x)$ with error for x= 10:

Taylor Series of $\log_e(x)$:

$$\boldsymbol{log_e(x) = -\sum_{i=1}^{\infty} \frac{(1-x)^i}{i}} \qquad\qquad \boldsymbol{for, x < 1}$$

PROGRAM:

```
# DETERMINATION OF THE VALUE OF ln(10) WITH ERROR APPROXIMATION :

print("\n\tDETERMINATION OF THE VALUE OF ln(10) WITH ERROR APPROXIMATION :\n")

a = 10.0

x = 1 - (1/a)
```

```
eps = 0.00001
(err, s, i, tm) = (1.0, 0.0, 1.0, 0.0)

while err > eps:
    t = (x**i)/i
    s = s+t
    i = i+1.0

    err = abs(t)
    tm = tm+1.0

print("The value of the ln(", a, ") :  ", s)
print("The total number of terms required to reach the accuracy :", tm)
```

OUTPUT:

```
      DETERMINATION OF THE VALUE OF ln(10) WITH ERROR APPROXIMATION :

The value of the ln( 10.0 ) :   2.3025137650431016
The total number of terms required to reach the accuracy : 70.0
```

6. Write a program to determine the value of $(1+x)^n$ Taylor expansion with error for x=0.01<<1 and n = 3:

Taylor Series of $(1+x)^n$:

$$(1+x)^n = \sum_{i=0}^{\infty} \binom{n}{i} x^i \qquad\qquad for, x \ll 1$$

PROGRAM:

```
# DETERMINATION OF THE VALUE OF THE (1+x)^n TAYLOR EXPANSION WITH ERROR
APPROXIMATION WHEN x<<1


print("\n\tDETERMINATION OF THE VALUE OF THE (1+x)^n TAYLOR EXPANSION WITH
ERROR APPROXIMATION WHEN x<<1\n")


x = 0.01
n = 3


eps = 0.00001
```

```
(a, s, err, i, f, fn, tm) = (1.0, 1.0, 1.0, 1, 1.0, 1.0, 1)

while err > eps:
    f = f*i
    fn = fn*(n-i+1)

    t = (fn/f)*(x**i)
    s = s+t

    i = i+1

    err = abs(t)
    tm = tm+1

print("The value of (", a, "+", x, ")^", n, " is equals to : ", s)
print("The number of terms to reach the accuracy is :", tm)
```

OUTPUT:

```
    DETERMINATION OF THE VALUE OF THE (1+x)^n TAYLOR EXPANSION WITH ERROR
APPROXIMATION WHEN x<<1


The value of ( 1.0 + 0.01 )^ 3  is equals to :  1.030301
The number of terms to reach the accuracy is : 4
```

7. Write a program to determine the value of the value of π through Ramanujan's Pi formula with error approximation:

Ramanujan's Pi Formula:

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!\,(1103 + 26390k)}{(k!)^4 (396)^{4k}}$$

PROGRAM:

```
# FACTORIAL MODULE:

def factorial(n, i, step):
    fact = 1
    while i <= n:
```

```
        fact = fact*i
        i = i+step
    return fact




# DETERMINATION OF THE VALUE OF π USING RAMANUJAN'S π SERIES

from factorial import *

print("\tDETERMINATION OF THE VALUE OF π USING RAMANUJAN'S π SERIES")

s0 = (8 ** 0.5) / 9801  # constant
(f1, f2, j, k, err, eps, t, c) = (1, 1, 0, 0, 1, 10 ** (-14), 0, 10)

t0 = (1 * (1103 + 26390 * 0)) / (1 * 1)  # 1st term of the summation
# as we know that,
#  0! = 1
# so,
#    (4 * 0)! = 1
#     pow(0!,4) = 1
# and, we also know that,
#  pow(396,4*0)= 1
sum = t0

while err > eps:
    t = t + 1
    for i in range(0, t):
        i = i + 1
        p = 4 * i

        f1 = factorial(p, 1, 1)
        f2 = factorial(i, 1, 1)

        s = (f1 * (1103 + 26390 * i)) / ((f2 ** 4) * (396 ** p))  # every
term of the summation
        sum = sum + s  # summation part

    pi = 1 / (s0 * sum)
```

```
        err = abs(pi - c)
        c = pi

        (f1, f2, sum) = (1, 1, t0)

print("\nUltimate Approximated value of π :", pi)
print("Number of terms required to reach that approximated value :", t +
1)
```

OUTPUT:

```
      DETERMINATION OF THE VALUE OF π USING RAMANUJAN'S π SERIES
Ultimate Approximated value of π : 3.141592653589793
Number of terms required to reach that approximated value : 3
```

8. Array and matrix operation:

PROGRAM:

```
# ARRAY OR MATRIX OPERATIONS :

import numpy as np
print("\n\tARRAY OR MATRIX OPERATIONS :\n")

xx = [[1.0, 2.0, 3.0], [-5.0, 0.0, 5.0], [0.0, -1.0, 1.0]]
yy = [[0.0, 5.0, 10.0], [1.0, 2.0, 3.0], [-1.0, 0.0, 1.0]]

print("My matrices are \nxx :", xx, "\nyy :", yy)

addxy = np.zeros((3, 3))
sbtxy = np.zeros((3, 3))
mltxy = np.zeros((3,3))
trsx = np.zeros((3, 3))

trx = 0.0

for i in range(3):
    for j in range(3):
```

```python
            addxy[i][j] = xx[i][j]+yy[i][j]
            sbtxy[i][j] = xx[i][j]-yy[i][j]

            for k in range(3):
                mltxy[i][j] += xx[i][k]*yy[k][j]

            trsx[j][i] = xx[i][j]

        trx += xx[i][i]

    print("The addition of xx and yy matrix is : \n", addxy)
    print("The subtraction of xx from yy matrix is : \n", sbtxy)
    print("The multiplication of xx to yy matrix is : \n", mltxy)
    print("The transpose of xx matrix is : \n", trsx)
    print("The tr(xx) : ", trx)
```

OUTPUT:

```
      ARRAY OR MATRIX OPERATIONS :

My matrices are
xx : [[1.0, 2.0, 3.0], [-5.0, 0.0, 5.0], [0.0, -1.0, 1.0]]
yy : [[0.0, 5.0, 10.0], [1.0, 2.0, 3.0], [-1.0, 0.0, 1.0]]
The addition of xx and yy matrix is :
 [[ 1.  7. 13.]
 [-4.  2.  8.]
 [-1. -1.  2.]]
The subtraction of xx from yy matrix is :
 [[ 1. -3. -7.]
 [-6. -2.  2.]
 [ 1. -1.  0.]]
The multiplication of xx to yy matrix is :
 [[ -1.   9.  19.]
 [ -5. -25. -45.]
 [ -2.  -2.  -2.]]
The transpose of xx matrix is :
 [[ 1. -5.  0.]
 [ 2.  0. -1.]
 [ 3.  5.  1.]]
The tr(xx) :  2.0
```

❖ Write a program to determine the square root of a number(x= 3) using Bisection method with error:

PROGRAM:

```
# DETERMINATION OF THE SQUARE ROOT OF A GIVEN NUMBER THROUGH BISECTION :


print("\n\tDETERMINATION OF THE SQUARE ROOT OF A GIVEN NUMBER THROUGH
BISECTION :\n")


x = 3.0


a = 1.0
b = 20.0


eps = 0.00001
err = 1.0
tm = 0.0
while err > eps:
    c = (a+b)/2.0

    if c**2.0 < x:
        a = c
    elif c**2.0 == x:
        err = eps
    else:
        b = c

    d = (a+b)/2.0

    err = abs(c-d)
    tm = tm+1.0

print("The value of the square root of", x, "is equals to :", c)
print("And the number of terms to reach the accuracy is :", tm)
```

OUTPUT:

```
      DETERMINATION OF THE SQUARE ROOT OF A GIVEN NUMBER THROUGH BISECTION :

The value of the square root of 3.0 is equals to : 1.7320585250854492
And the number of terms to reach the accuracy is : 20.0
```

❖ Write a program to determine the square root of a given number (x = 10) using Newton-Raphson method with error:

PROGRAM:

```
# DETERMINATION OF THE VALUE OF SQUARE ROOT OF A GIVEN NUMBER THROUGH NEWTON-RAPHSON METHOD WITH ERROR APPROXIMATION :


n = 2.0

print("\n\tDETERMINATION OF THE VALUE OF SQUARE ROOT OF", n, " THROUGH NEWTON-RAPHSON METHOD WITH ERROR APPROXIMATION :\n")


x = 10.0

eps = 0.00001

(err, i, tm) = (1.0, 1.0, 0.0)


while err > eps:

    x1 = x

    m = 2.0*x1

    y = (x1**2.0)-n

    x = x1-(y/m)

    z = (x**2.0)-n


    err = abs(y-z)

    tm = tm+1.0


print("The value of square root of", n, "with given accuracy", eps, " is : ", x)

print("The number of terms to reach the accuracy is : ", tm)
```

OUTPUT:

```
	DETERMINATION OF THE VALUE OF SQUARE ROOT OF 2.0  THROUGH NEWTON-RAPHSON METHOD WITH ERROR APPROXIMATION :


The value of square root of 2.0 with given accuracy 1e-05  is :  1.4142135623730954

The number of terms to reach the accuracy is :  7.0
```

❖ Write a program to determine the value of integration of a given function through trapezoidal method with error:

PROGRAM:

```
# DETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH TRAPEZOIDAL METHOD :


print("\n\tDETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH TRAPEZOIDAL METHOD :\n")


xi = 0.0
xf = 10.0


n = 10000
h = (xf-xi)/(n-1)


# Suppose,
#   My function is : yi=f(xi)=xi^2


yi = xi**2
yf = xf**2


(i, err, s) = (1, 1.0, 0.0)


while i < (n-1):
    c = xi+i*h
    fc = c**2
    s = s+fc
    i = i+1


s = (yi+yf+2*s)*h/2


print("The value of the integration of my function from", xi, "to", xf, " is : ", s)
```

OUTPUT:

```
	DETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH TRAPEZOIDAL METHOD :


The value of the integration of my function from 0.0 to 10.0  is : 333.3333350003328
```

❖ Write a program to determine the value of integration of a given function through Simpson's 1/3 method of integration with error:

PROGRAM:

```
# DETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH SIMPSON
1/3 METHON :


import numpy as np

print("\n\tDETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH
SIMPSON 1/3 METHON :\n")


xi = 0.0

xf = 1.0


# Suppose,

#    my function is : yi=sin(x)*exp(x^3/3)


yi = np.sin(xi) * np.exp((xi ** 3) / 3)

yf = np.sin(xf) * np.exp((xf ** 3) / 3)


(n, err, eps, sn, t) = (2, 1, 0.0005, 10, 0)

while err > eps:

    h = (xf-xi)/(n-1.0)


    (i, s) = (1, 0.0)


    while i < (n-1):

        c = xi+i*h

        yc = np.sin(c)*np.exp((c**3)/3)


        if i % 2.0 == 1.0:

            s = s+4.0*yc

        else:

            s = s+2.0*yc

        i = i+1


    n = n+1

    s = (yi+yf+s)*h/3.0


    err = abs(sn-s)
```

```
        sn = s
        t += 1

    s = sn
    print("The value of the integration of my function from", xi, "to", xf,
    "is equals to : ", s)
    print("Number of terms to reach the accuracy : ", t)
```

OUTPUT:

```
    DETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH SIMPSON
1/3 METHON :


The value of the integration of my function from 0.0 to 1.0 is equals to :
0.5248062730696056
Number of terms to reach the accuracy :  783
```

- ***<u>Simpson's 1/3 with array operation:</u>***

PROGRAM:

```
# DETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH SIMPSON
1/3 METHOD USING ARRAY:


import numpy as np


print("\n\tDETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH
SIMPSON 1/3 METHOD :")



xi = 0.0
xf = 10.0


n = 10000
h = (xf-xi)/(n-1)


xx = np.linspace(xi, xf, n)


# Suppose,
#    my function is : yi=f(xi)=xi^2


yy = xx**2.0
```

```
(i, s) = (1, 0.0)

while i < n-1.0:
    k = i % 2.0
    if k == 1.0:
        s = s+4.0*yy[i]
    else:
        s = s+2.0*yy[i]
    i = i+1

s = (yy[0]+yy[n-1]+s)*h/3.0

print("\nThe value of the integration of my function from", xi, "to", xf,
"is equals to : ", s)
```

OUTPUT:

```
DETERMINATION OF THE VALUE OF INTEGRATION OF A FUNCTION THROUGH SIMPSON 1/3
METHOD :


The value of the integration of my function from 0.0 to 10.0 is equals to :
333.30000000033306
```

❖ Write a program to determine the value of variables from three linear equation through Gaussian Elimination method :

PROGRAM:

```
# DETERMINATION OF THE VALUE OF VARIABLES FROM THREE LINEAR EQUATION
THROUGH GAUSSIAN ELIMINATION METHOD :


print("\n\tDETERMINATION OF THE VALUE OF VARIABLES FROM THREE LINEAR
EQUATION THROUGH GAUSSIAN ELIMINATION METHOD :\n")


x = 1.0
y = 2.0
z = 3.0
eps = 0.00001
erx = ery = erz = 1.0

# My equations are:
#       12x+3y-5z=1         (1)
#       x+5y+3z=28          (2)
#       3x+7y+13z=76        (3)

while erx and ery and erz > eps:
    x1 = (1.0-3.0*y+5.0*z)/12.0
    y1 = (28.0-x1-3.0*z)/5.0
    z1 = (76.0-3.0*x1-7.0*y1)/13.0

    erx = abs(x-x1)
    ery = abs(y-y1)
    erz = abs(z-z1)

    x = x1
    y = y1
    z = z1
print("So, the general solution of my equations are x =", x, "y =", y, "z
=", z)
```

OUTPUT:

```
      DETERMINATION OF THE VALUE OF VARIABLES FROM THREE LINEAR EQUATION THROUGH
GAUSSIAN ELIMINATION METHOD :


So, the general solution of my equations are x = 1.0000035820556061 y =
2.9999977236707336 z = 4.000000399087542
```

❖ Write a program to sort a numerical list in ascending order or descending order through Insertion sort method:

PROGRAM:

```
# SORTING A NUMERICAL LIST IN ASCENDING ORDER OR DESCENDING ORDER THROUGH
INSERTION SORT METHOD :


print("\n\tSORTING A NUMERICAL LIST IN ASCENDING ORDER OR DESCENDING ORDER
THROUGH INSERTION SORT METHOD :\n")


xx = [0, -1, -10, 5, 7, 2, 11, 8, 9, -6]
print("My list :", xx)


for i in range(0, len(xx)):
    for j in range(i+1, len(xx)):
        if xx[i] > xx[j]:               # ASCENDING ORDER
            (xx[i], xx[j]) = (xx[j], xx[i])


print("My list after sorting in ascending order : ", xx)
```

OUTPUT:

```
      SORTING A NUMERICAL LIST IN ASCENDING ORDER OR DESCENDING ORDER THROUGH
INSERTION SORT METHOD :


My list : [0, -1, -10, 5, 7, 2, 11, 8, 9, -6]
My list after sorting in ascending order :  [-10, -6, -1, 0, 2, 5, 7, 8, 9, 11]
```

❖ Write a program to sort a numerical list in ascending order or descending order through Bubble sort method:

PROGRAM:

```
# SORTING A NUMERICAL LIST IN ASCENDING ORDER OR DESCENDING ORDER THROUGH
BUBBLE SORT METHOD :


print("\n\tSORTING A NUMERICAL LIST IN ASCENDING ORDER OR DESCENDING ORDER
THROUGH BUBBLE SORT METHOD :\n")


xx = [0, -1, -10, 5, 7, 2, 11, 8, 9, -6]
print("My list :", xx)


for k in range(100):
    for i in range(0, len(xx)-1):
        if xx[i] < xx[i+1]:               # DESCENDING ORDER
            (xx[i], xx[i+1]) = (xx[i+1], xx[i])


print("My list after sorting in descending order : ", xx)
```

OUTPUT:

```
      SORTING A NUMERICAL LIST IN ASCENDING ORDER OR DESCENDING ORDER THROUGH
BUBBLE SORT METHOD :


My list : [0, -1, -10, 5, 7, 2, 11, 8, 9, -6]
My list after sorting in descending order :  [11, 9, 8, 7, 5, 2, 0, -1, -6, -10]
```

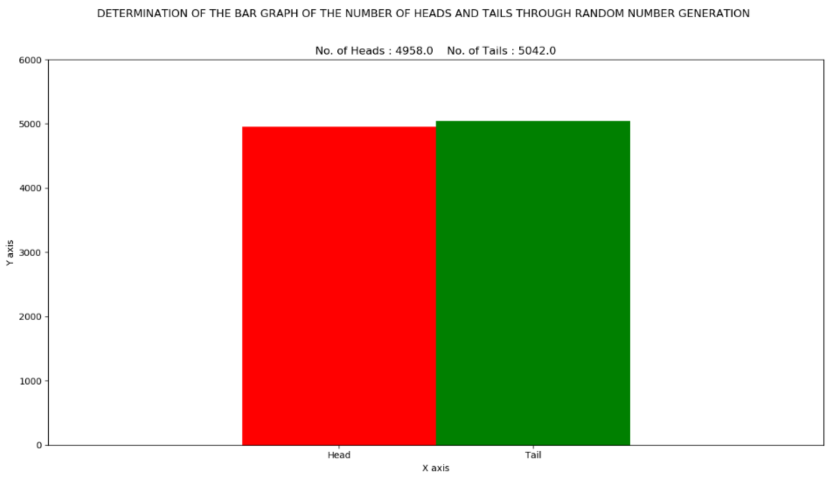
16. Write a program to determine the bar graph of the number of heads and tails through random number generation:

PROGRAM:

```
# DETERMINATION OF THE BAR GRAPH OF THE NUMBER OF HEADS AND TAILS THROUGH
RANDOM NUMBER GENERATION :


import random as rd

import matplotlib.pyplot as plt


i = 1

n = 10000


ctH = 0.0

ctT = 0.0


while i <= n:
    x = rd.random()

    if x < 0.5:
        ctH += 1.0
    else:
        ctT += 1.0

    i += 1


xx = [0.35, 0.65]

yy = [ctH, ctT]

zz = ["Head", "Tail"]


plt.bar(xx, yy, width=0.3, color=('r', 'g'))


plt.suptitle("DETERMINATION OF THE BAR GRAPH OF THE NUMBER OF HEADS AND
TAILS THROUGH RANDOM NUMBER GENERATION")

plt.title("No. of Heads : "+str(ctH)+"    No. of Tails : "+str(ctT))

plt.xticks(xx, zz)

plt.xlim(-0.1, 1.1)

plt.ylim(0, n*0.6)

plt.ylabel("Y axis")

plt.xlabel("X axis")
```

```
plt.show()
```

OUTPUT:



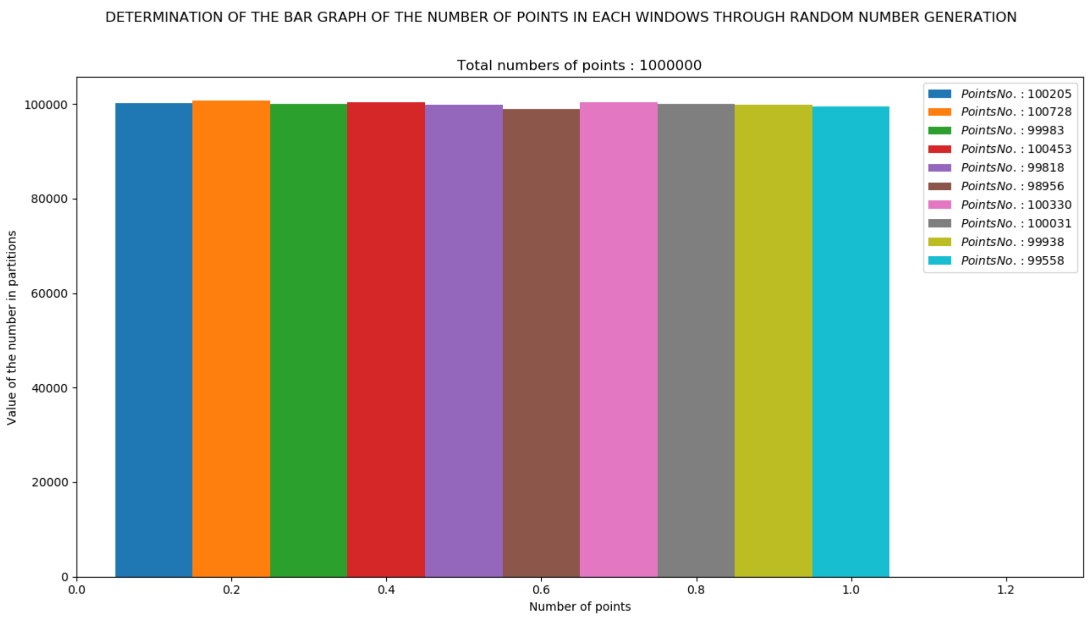
17. Write a program to determine the bar graph of the number of the number of points in each windows between 0 to 1 through random number generation:

PROGRAM:

```
# DETERMINATION OF THE BAR GRAPH OF THE NUMBER OF POINTS IN EACH WINDOWS THROUGH RANDOM NUMBER GENERATION :


import random as rd

import matplotlib.pyplot as plt


n = 1000000


a = 0

xx = []

cc = []

aa = []

ss = []

for i in range(n):

    x = rd.random()

    xx.append(x)
```

```
while a < 0.9:
    c = 0
    for i in range(n):
        if (xx[i] > a) & (xx[i] < (a+0.1)):
            c += 1
    cc.append(c)
    s = "$Points No.:$" + str(c)
    ss.append(s)

    a += 0.1
    aa.append(a)

for i in range(len(aa)):
    plt.bar(aa[i], cc[i], width=0.1)

plt.suptitle("DETERMINATION OF THE BAR GRAPH OF THE NUMBER OF POINTS IN
EACH WINDOWS THROUGH RANDOM NUMBER GENERATION ")
plt.title("Total numbers of points : "+str(n))
plt.legend(ss)
plt.ylabel("Value of the number in partitions")
plt.xlabel("Number of points")
plt.xlim(0, 1.3)
plt.show()
```

OUTPUT:



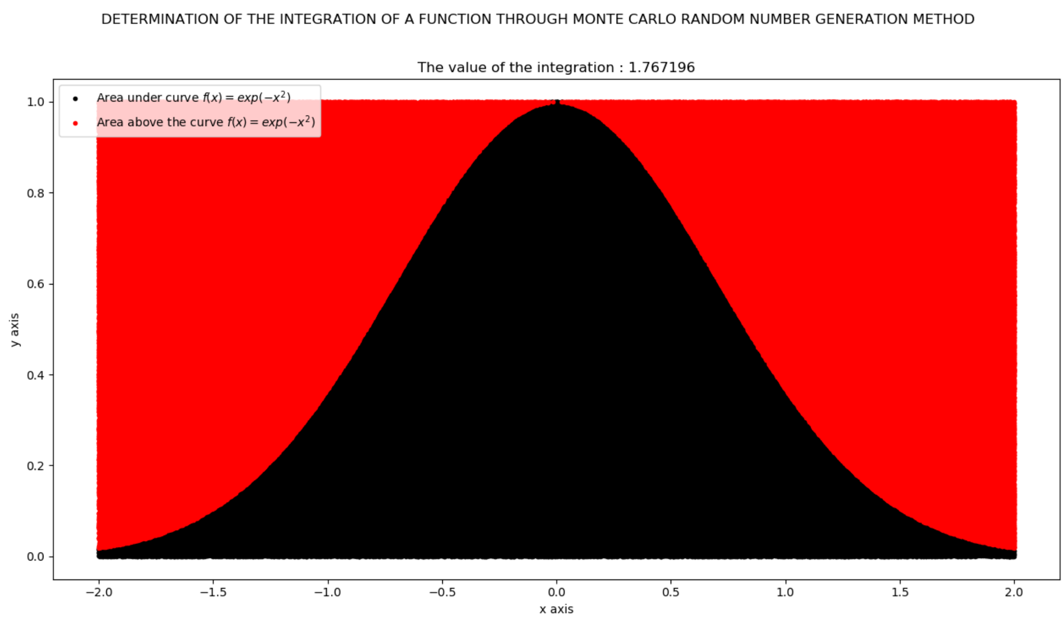
- ❖ **Write a program to determine the integration of a function in a given range using Monte Carlo method:**

PROGRAM:

```
# DETERMINATION OF THE INTEGRATION OF A FUNCTION THROUGH MONTE CARLO
RANDOM NUMBER GENERATION METHOD :


import random as rd
import matplotlib.pyplot as plt
import numpy as np

n = 1000000
ct = 0.0
xx = []
yy = []
ax = []
ay = []
ux = []
uy = []

# Suppose,
#    my function is : yi=exp(-x^2)

for i in range(n):
    x = rd.uniform(-2.0, 2.0)
    y = rd.uniform(0, 1.0)

    xx.append(x)
    yy.append(y)

for i in range(n):
    if np.exp(-(xx[i])**2) >= yy[i]:
        ux.append(xx[i])
        uy.append(yy[i])
        ct += 1.0
    else:
        ax.append(xx[i])
        ay.append(yy[i])
```

```
a = 4.0*1.0
s = (ct/n)*a

plt.plot(ux, uy, '.k', ax, ay, '.r')

plt.suptitle("DETERMINATION OF THE INTEGRATION OF A FUNCTION THROUGH MONTE CARLO RANDOM NUMBER GENERATION METHOD ")
plt.title("The value of the integration : "+str(s))
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.legend(["Area under curve $f(x)=exp(-x^2)$", "Area above the curve $f(x)=exp(-x^2)$"], loc="upper left")
plt.show()
```

OUTPUT:



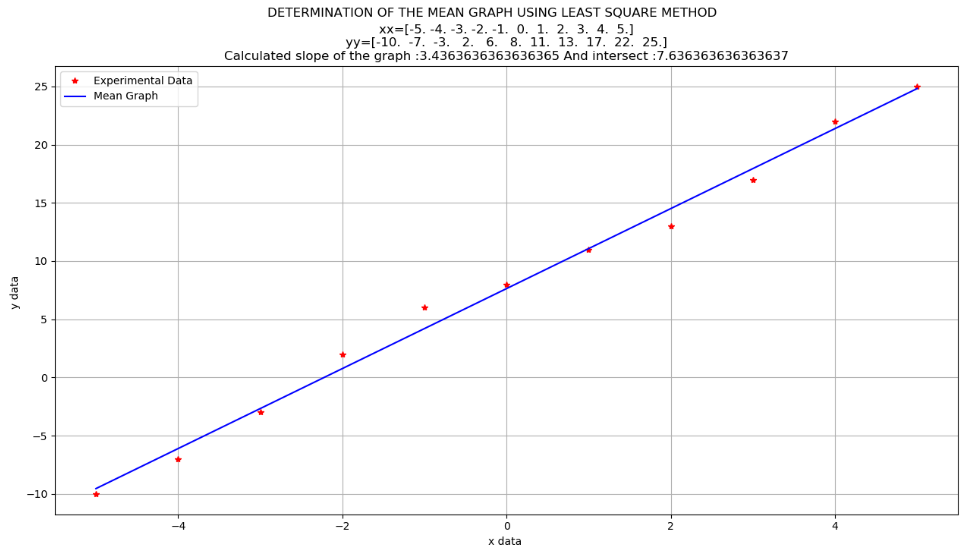
- ## Write a program to determine the mean graph of some experimental data (with linear equation) through least square method:

PROGRAM:

```
# DETERMINATION OF THE MEAN GRAPH USING LEAST SQUARE METHOD :

import matplotlib.pyplot as plt
import numpy as np



xx = np.array([-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0])
yy = np.array([-10.0, -7.0, -3.0, 2.0, 6.0, 8.0, 11.0, 13.0, 17.0, 22.0, 25.0])

(sumx, sumy, p1, p2) = (0.0, 0.0, 0.0, 0.0)

for k in range(len(xx)):
    sumx = sumx+xx[k]
    sumy = sumy+yy[k]

xb = sumx/len(xx)
yb = sumy/len(yy)

for l in range(len(xx)):
    p1 = p1+(xx[l]-xb)*(yy[l]-yb)
    p2 = p2+(xx[l]-xb)**2

m = p1/p2
c = yb-m*xb

xx1 = xx
yy1 = m*xx1+c

plt.plot(xx, yy, "*r", xx1, yy1, "-b")

plt.xlabel("x data ")
plt.ylabel("y data ")
plt.legend(["Experimental Data", "Mean Graph"])
```

```python
plt.suptitle("DETERMINATION OF THE MEAN GRAPH USING LEAST SQUARE METHOD")

plt.title("xx="+str(xx)+"\nyy="+str(yy)+"\nCalculated slope of the graph :"+str(m)+" And intersect :"+str(c))

plt.grid()


plt.show()
```

OUTPUT:



❖ Write a program to determine the value of differentiation of a function at a given point:

PROGRAM:

```
# DETERMINATION OF THE DIFFERENTIATION OF A FUNCTION AT A GIVEN POINT :


print("\n\tDETERMINATION OF THE DIFFERENTIATION OF A FUNCTION AT A GIVEN POINT :")



def f(x):
    return x**3.0-3.0*x



a = 5.0
h = 0.00001
df = (f(a+h)-f(a))/h
print("So, the differentiated value of the function at the point", a, "is : ", df)
```

OUTPUT:

```
	DETERMINATION OF THE DIFFERENTIATION OF A FUNCTION AT A GIVEN POINT :


So, the differentiated value of the function at the point 5.0 is : 72.00014999710902
```

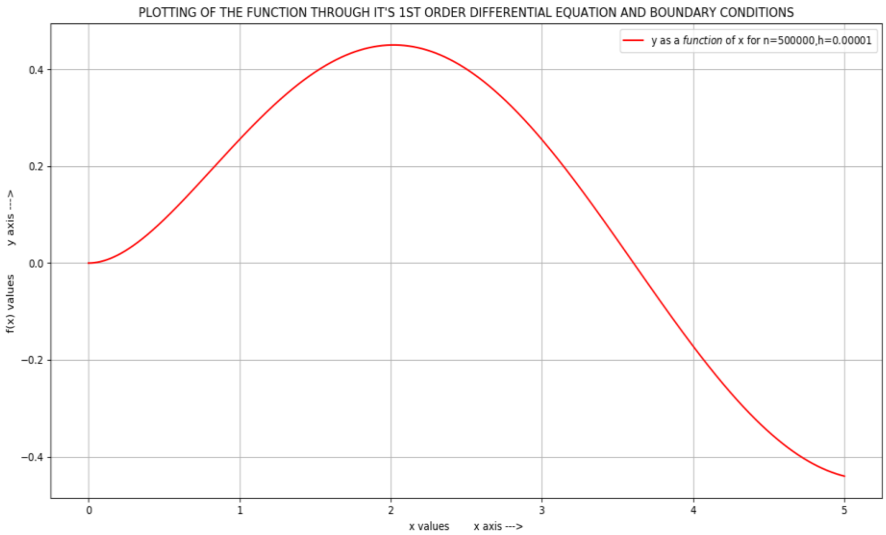
❖ Write a program to determine the function from it's 1st order differential equation using Euler's method:

PROGRAM:

```
# PLOTTING OF THE FUNCTION THROUGH IT'S 1ST ORDER DIFFERENTIAL EQUATION
AND BOUNDARY CONDITIONS :


import matplotlib.pyplot as plt
import numpy as np

# BOUNDARY CONDITIONS :
x = 0.0
y = 0.0

xx = []
yy = []



n = 500000
xs = 0.00001

i = 1

while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = np.sin(x)-2*y

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

xx = np.array(xx)
yy = np.array(yy)
```

```
plt.plot(xx, yy, 'r')


plt.title("PLOTTING OF THE FUNCTION THROUGH IT'S 1ST ORDER DIFFERENTIAL
EQUATION AND BOUNDARY CONDITIONS")

plt.xlabel("x values        x axis --->")

plt.ylabel("f(x) values        y axis --->")

plt.legend(["y as a $function$ of x for n=500000,h=0.00001 "], loc="upper
right")

plt.grid()


plt.show()
```

OUTPUT:



- ***<u>Changes in the graph due to different values of approximated arc length(h):</u>***

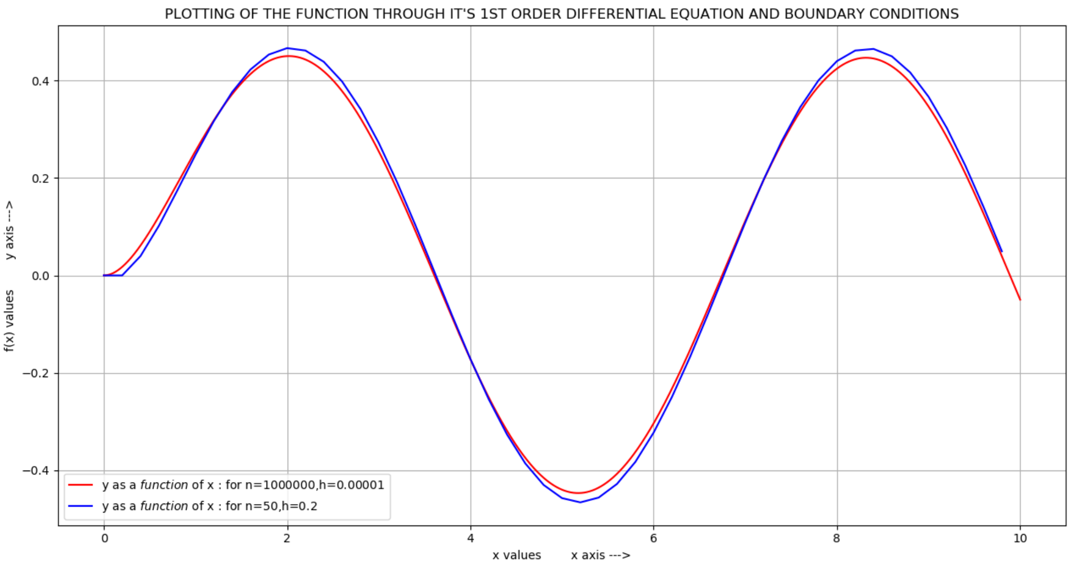
PROGRAM:

```
# PLOTTING OF THE FUNCTION THROUGH IT'S 1ST ORDER DIFFERENTIAL EQUATION
AND BOUNDARY CONDITIONS :


import matplotlib.pyplot as plt

import numpy as np


# BOUNDARY CONDITIONS :

x = x1 = 0.0

y = y1 = 0.0
```

```
xx = []
yy = []

xx1 = []
yy1 = []

n = 1000000
xs = 0.00001

n2 = 50
xs1 = 0.2

i = 1
j = 1
while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = np.sin(x)-2*y

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

while j <= n2:
    xx1.append(x1)
    yy1.append(y1)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m1 = np.sin(x1)-2*y1

    c1 = y1 - m1*x1
    x1 = x1 + xs1
    y1 = m1*x1 + c1

    j = j+1
```

```
xx1 = np.array(xx1)
yy1 = np.array(yy1)

xx = np.array(xx)
yy = np.array(yy)

plt.plot(xx, yy, 'r')
plt.plot(xx1, yy1, 'b')
plt.title("PLOTTING OF THE FUNCTION THROUGH IT'S 1ST ORDER DIFFERENTIAL EQUATION AND BOUNDARY CONDITIONS")
plt.xlabel("x values        x axis --->")
plt.ylabel("f(x) values        y axis --->")
plt.legend(["y as a $function$ of x : for n=1000000,h=0.00001", "y as a $function$ of x : for n=50,h=0.2"], loc="lower left")
plt.grid()
plt.show()
```

OUTPUT:



- ***<u>Determination of the function of radio-active decay from it's differential equation:</u>***

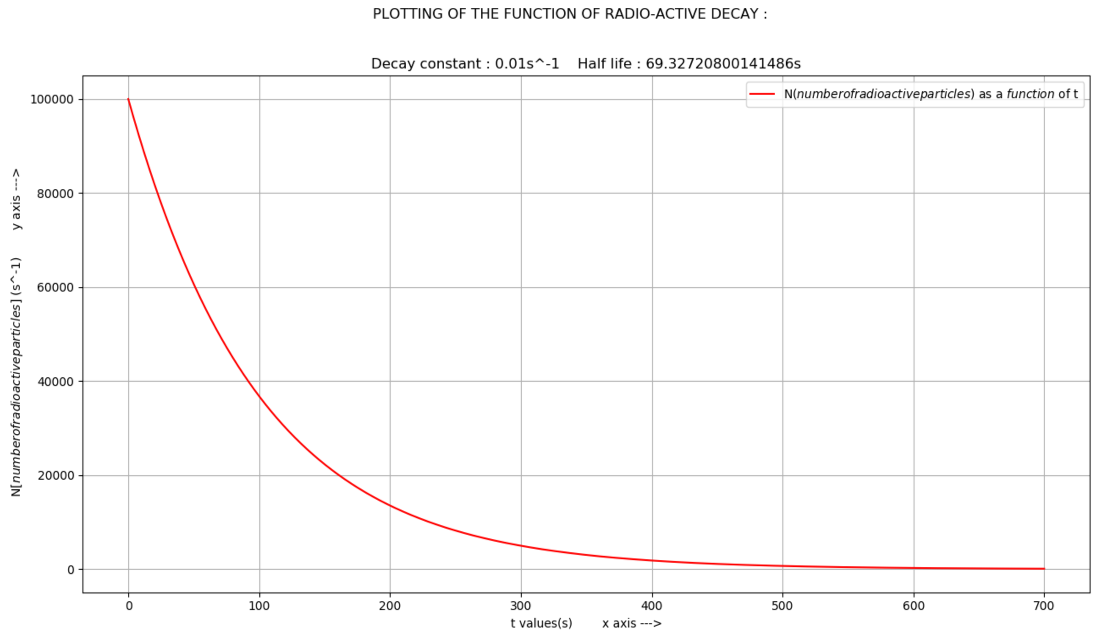
PROGRAM:

```
# PLOTTING OF THE FUNCTION OF RADIO-ACTIVE DECAY :

import matplotlib.pyplot as plt
```

```
import numpy as np

# BOUNDARY CONDITIONS :
x0 = x = 0.0
y0 = y = 10**5
l = 0.01

thf = 2.303*np.log10(2)/l

xx = []
yy = []

n = 70000
xs = 0.01
i = 1

while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = -l*y

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

xx = np.array(xx)
yy = np.array(yy)

plt.plot(xx, yy, 'r')

plt.suptitle("PLOTTING OF THE FUNCTION OF RADIO-ACTIVE DECAY :")
plt.title("Decay constant : "+str(l)+"s^-1    Half life : "+str(thf)+"s")
plt.xlabel("t values(s)        x axis --->")
plt.ylabel(" N[$number of radioactive particles$] (s^-1)       y axis ---
>")
```

```
plt.legend(["N($number of radioactive particles$) as a $function$ of t"])
plt.grid()
plt.show()
```

OUTPUT:



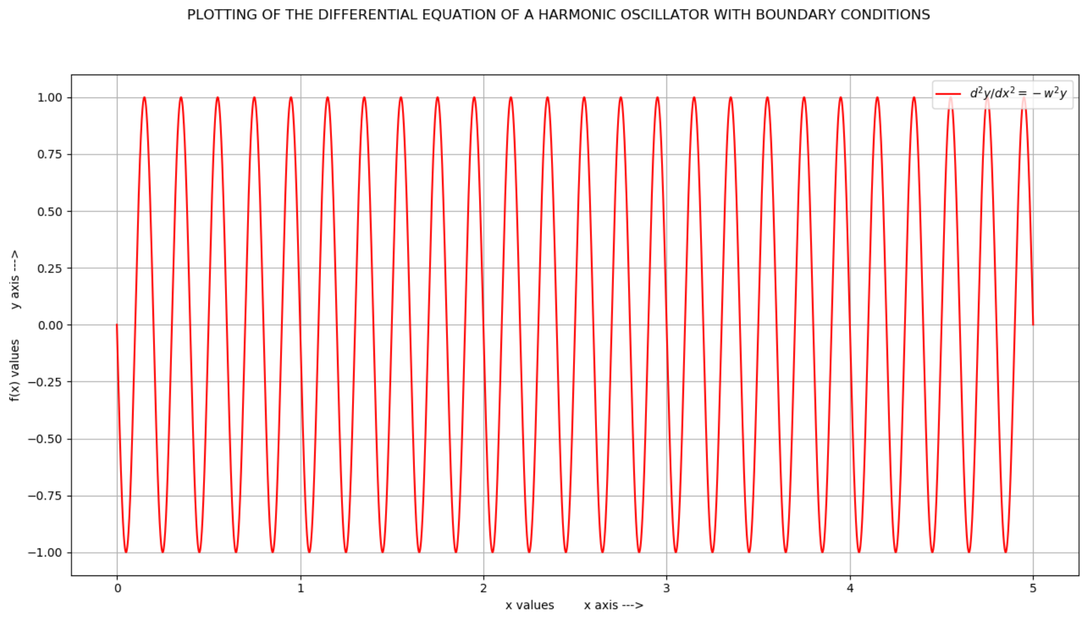
❖ Write a program to determine the function of harmonic oscillator (second order equation):

PROGRAM:

```
# PLOTTING OF THE DIFFERENTIAL EQUATION OF A HARMONIC OSCILLATOR WITH
BOUNDARY CONDITIONS :


import matplotlib.pyplot as plt
import numpy as np

# BOUNDARY CONDITIONS :
x = 0.0
y = 0.0

r = 1
g = np.pi
w = 10*np.pi

xx = []
yy = []

n = 500000
xs = 0.00001
i = 1

while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = r*w*(np.cos(w*x+g))

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

xx = np.array(xx)
```

```
yy = np.array(yy)

plt.plot(xx, yy, 'r')

plt.suptitle("PLOTTING OF THE DIFFERENTIAL EQUATION OF A HARMONIC OSCILLATOR WITH BOUNDARY CONDITIONS")
plt.xlabel("x values        x axis --->")
plt.ylabel("f(x) values        y axis --->")
plt.legend(["$d^2y/dx^2  = -w^2y$"], loc="upper right")
plt.grid()

plt.show()
```

OUTPUT:



- ***<u>Determination of the function of damped harmonic oscillator:</u>***

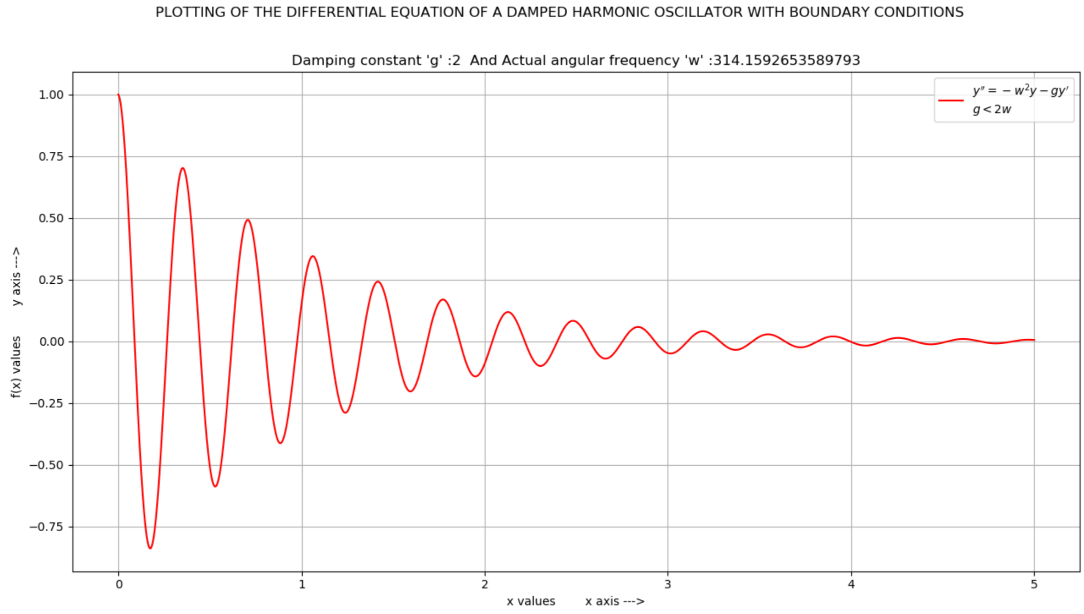
PROGRAM:

```
# PLOTTING OF THE DIFFERENTIAL EQUATION OF A DAMPED HARMONIC OSCILLATOR WITH BOUNDARY CONDITIONS :

import matplotlib.pyplot as plt
import numpy as np

# BOUNDARY CONDITIONS :
x = 0
y = 1
```

```
r = 1
e = 0

w = 100*np.pi
g = 2
w1 = (w-(g**2)/4)**0.5

xx = []
yy = []

n = 500000
xs = 0.00001
i = 1

while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = r*np.exp(-0.5*g*x)*(-0.5*g*np.cos(w1*x+e)-w1*np.sin(w1*x+e))

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

xx = np.array(xx)
yy = np.array(yy)

plt.plot(xx, yy, 'r')


plt.suptitle("PLOTTING OF THE DIFFERENTIAL EQUATION OF A DAMPED HARMONIC
OSCILLATOR WITH BOUNDARY CONDITIONS")
plt.title("Damping constant 'g' :"+str(g)+"  And Actual angular frequency 'w'
:"+str(w))
plt.xlabel("x values        x axis --->")
plt.ylabel("f(x) values        y axis --->")
plt.legend(["$y''  = -w^2y -gy'$\n$g<2w$"], loc="upper right")
plt.grid()
```

```
plt.show()
```

OUTPUT:



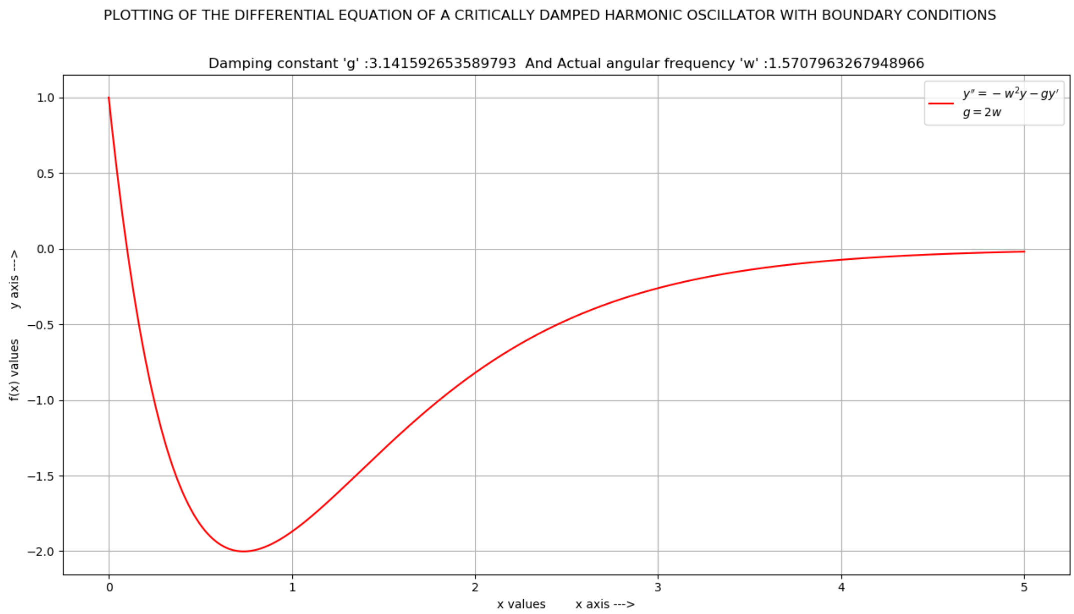
- ***<u>Determination of the function of critically damped harmonic oscillator:</u>***

PROGRAM:

```
# PLOTTING OF THE DIFFERENTIAL EQUATION OF A CRITICALLY DAMPED HARMONIC OSCILLATOR WITH BOUNDARY CONDITIONS :


import matplotlib.pyplot as plt

import numpy as np


# BOUNDARY CONDITIONS :

x = 0

y = 1

a = 1

b = -10


w = np.pi/2

g = 2*w


xx = []

yy = []
```

```
n = 500000
xs = 0.00001
i = 1

while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = np.exp(-0.5*g*x)*(-0.5*g*(a+x*b)+b)

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

xx = np.array(xx)
yy = np.array(yy)

plt.plot(xx, yy, 'r')

plt.suptitle("PLOTTING OF THE DIFFERENTIAL EQUATION OF A CRITICALLY DAMPED HARMONIC OSCILLATOR WITH BOUNDARY CONDITIONS")
plt.title("Damping constant 'g' :"+str(g)+"  And Actual angular frequency 'w' :"+str(w))
plt.xlabel("x values        x axis --->")
plt.ylabel("f(x) values        y axis --->")
plt.legend(["$y''  = -w^2y -gy'$\n$g =2w$"], loc="upper right")
plt.grid()

plt.show()
```

OUTPUT:



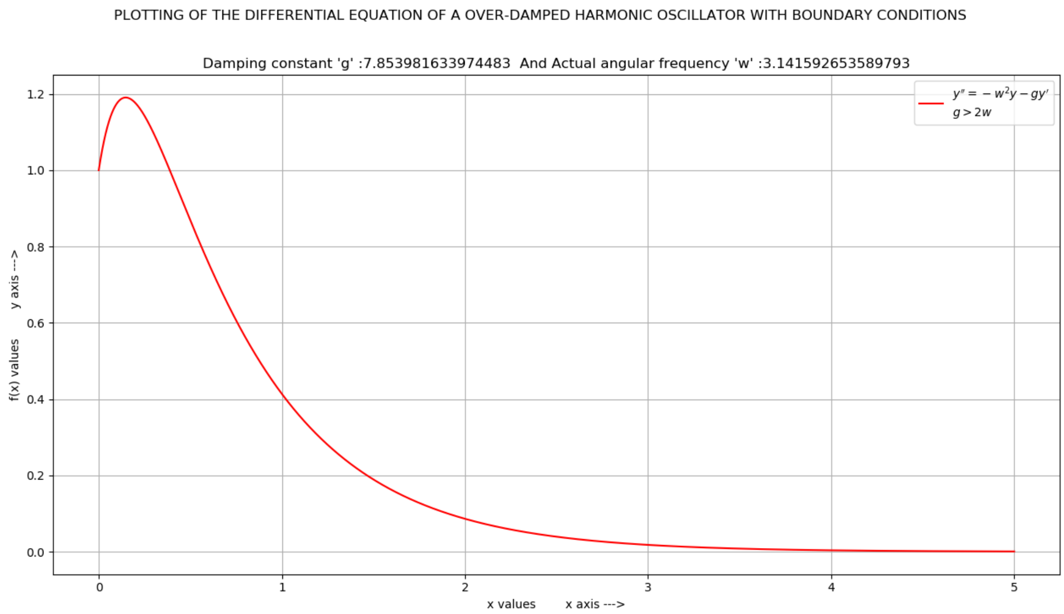
- ***<u>Determination of the function of over-damped harmonic oscillator:</u>***

PROGRAM:

```
# PLOTTING OF THE DIFFERENTIAL EQUATION OF A OVER-DAMPED HARMONIC
OSCILLATOR WITH BOUNDARY CONDITIONS :


import matplotlib.pyplot as plt

import numpy as np


# BOUNDARY CONDITIONS :

x = 0

y = 1

a = -1

b = 2


w = np.pi

g = 2.5*w


k1 = -0.5*g-(0.25*g**2-w**2)**0.5

k2 = -0.5*g+(0.25*g**2-w**2)**0.5


xx = []

yy = []
```

```
n = 500000
xs = 0.00001
i = 1

while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = k1*a*np.exp(k1*x)+k2*b*np.exp(k2*x)

    c = y - m*x
    x = x + xs
    y = m*x + c

    i = i+1

xx = np.array(xx)
yy = np.array(yy)

plt.plot(xx, yy, 'r')

plt.suptitle("PLOTTING OF THE DIFFERENTIAL EQUATION OF A OVER-DAMPED
HARMONIC OSCILLATOR WITH BOUNDARY CONDITIONS")
plt.title("Damping constant 'g' :"+str(g)+"  And Actual angular frequency
'w' :"+str(w))
plt.xlabel("x values        x axis --->")
plt.ylabel("f(x) values        y axis --->")
plt.legend(["$y''  = -w^2y -gy'$"], loc="upper right")
plt.grid()

plt.show()
```

OUTPUT:



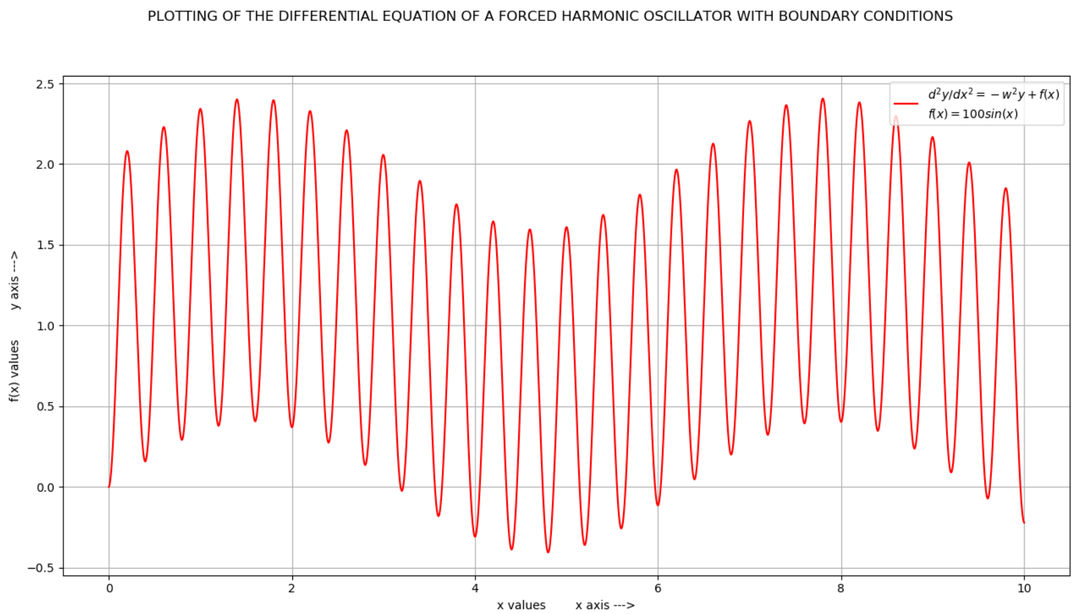
- ***Determination of the function of forced harmonic oscillator for a given function f(x) = 100sin(x):***

PROGRAM:

```
# PLOTTING OF THE DIFFERENTIAL EQUATION OF A FORCED HARMONIC OSCILLATOR
WITH BOUNDARY CONDITIONS :


import matplotlib.pyplot as plt

import numpy as np


# BOUNDARY CONDITIONS :

x = 0

y = 0

a = 1

b = 1


r = 1

e = np.pi

w = 5*np.pi


xx = []

yy = []


n = 1000000
```

```
xs = 0.00001
i = 1
while i <= n:
    xx.append(x)
    yy.append(y)

    # SLOPE OF THE GRAPH AT BOUNDARY CONDITION:
    m = -a*r*w*(np.sin(w*x-e)) + b*100*np.cos(x)/(w**2-1)

    c = y - m*x
    x = x + xs
    y = m*x + c
    i = i+1
xx = np.array(xx)
yy = np.array(yy)

plt.plot(xx, yy, 'r')
plt.suptitle("PLOTTING OF THE DIFFERENTIAL EQUATION OF A FORCED HARMONIC OSCILLATOR WITH BOUNDARY CONDITIONS")
plt.xlabel("x values        x axis --->")
plt.ylabel("f(x) values        y axis --->")
plt.legend(["$d^2y/dx^2  = -w^2y + f(x)$\n$f(x)=100sin(x)$"], loc="upper right")
plt.grid()
plt.show()
```

OUTPUT:



- ## Write a program to determine the Fourier coefficients of a function and the Fourier function :

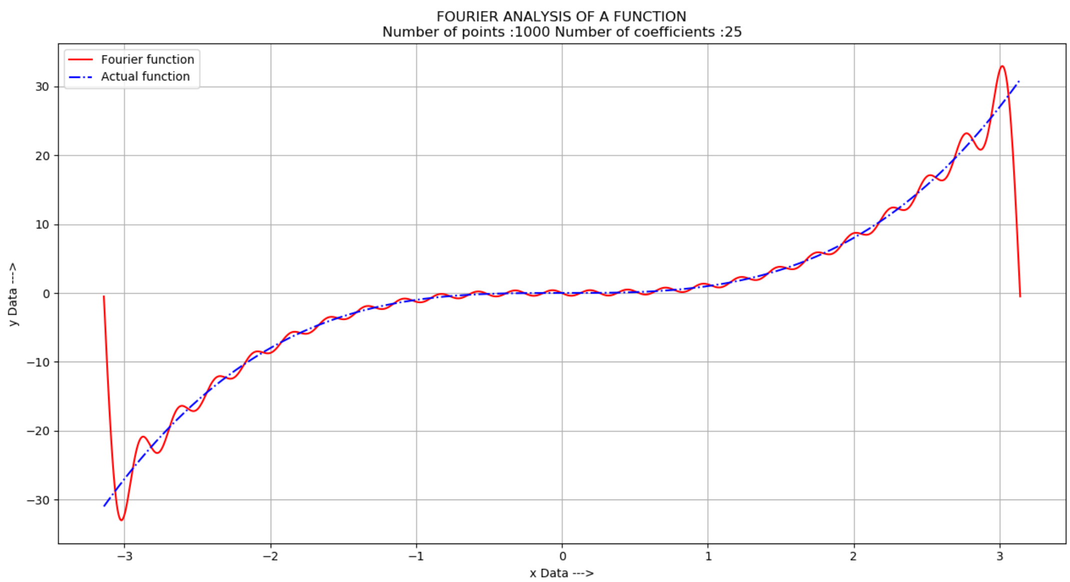
PROGRAM:

```
# DETERMINATION OF THE FOURIER SERIES COEFFICIENTS FOR GIVAN SITUATION AND BOUNDARY CONDITIONS :

# AND PLOTTING OF THE EXACT FOURIER FUNCTION FOR DIFFERENT VALUES OF x :


import matplotlib.pyplot as plt

import numpy as np

print("\n\tDETERMINATION OF THE FOURIER SERIES COEFFICIENTS FOR GIVAN SITUATION AND BOUNDARY CONDITIONS :\n")


xi = -np.pi

xf = np.pi


n = 1000

h = (xf-xi)/(n-1)

nc = 25


xx = np.linspace(xi, xf, n)

# y=xx^3

cc = []

cs = []

yy = []



for i in range(nc):

    yc = (xx**3)*np.cos(i*xx)

    ys = (xx**3)*np.sin(i*xx)


    j = 1

    (sc, ss) = (0, 0)


    while j < (n-1):

        k = j % 2


        if k == 1:

            sc = sc+4*yc[j]

            ss = ss+4*ys[j]
```

```
        else:
            sc = sc+2*yc[j]
            ss = ss+2*ys[j]

        j = j+1

    sc = (yc[0]+yc[n-1]+sc)*h/(3*np.pi)
    ss = (ys[0]+ys[n-1]+ss)*h/(3*np.pi)

    cc.append(sc)
    cs.append(ss)

cc[0] = cc[0]/2
yy = []
zz = xx**3

print("Coefficients of cosine function : ", cc, "\nCoefficients of sine
function : ", cs)

for i in range(n):
    s = 0

    for j in range(nc):
        s = s + cc[j]*np.cos(j*xx[i]) + cs[j]*np.sin(j*xx[i])

    yy.append(s)

plt.plot(xx, yy, '-r', xx, zz, '-.b')

plt.title("FOURIER ANALYSIS OF A FUNCTION"+"\nNumber of points :"+str(n)+"
Number of coefficients :"+str(nc))
plt.xlabel("x Data --->")
plt.ylabel("y Data --->")
plt.legend(["Fourier function", "Actual function"])
plt.grid()

plt.show()
```

OUTPUT:

```
      DETERMINATION OF THE FOURIER SERIES COEFFICIENTS FOR GIVAN SITUATION AND BOUNDARY CONDITIONS :


Coefficients of cosine function :  [-0.010314702969813013, 0.020629405325126794, -0.020629403481579073, 0.020629400408861288, -0.020629396106711818, 0.02062939057478826, -0.020629383812656523, 0.02062937581978188, -0.020629366595531017, 0.020629356139174406, -0.020629344449882958, 0.02062933152673188, -0.020629317368695162, 0.020629301974651567, -0.020629285343379735, 0.020629267473561943, -0.02062924836377787, 0.020629228012511092, -0.02062920641814757, 0.020629183578967373, -0.020629159493153484, 0.020629134158790568, -0.020629107573866382, 0.020629079736260627, -0.02062905064375045]

Coefficients of sine function :  [0.0, 7.739143732746151, -8.369474261966861, 6.135096613621219, -4.747041920240474, 3.851516408124934, -3.233922152536394, 2.784446048556541, -2.4434430231892748, 2.1761988613315393, -1.9612701434178348, 1.7847419294680844, -1.6372087155875916, 1.5120926871323432, -1.4046592188342801, 1.3114155122617714, -1.2297295766430356, 1.1575810396020911, -1.0933935987834598, 1.0359195875093448, -0.9841587089905687, 0.9372997128754436, -0.8946778079595052, 0.8557430771308967, -0.8200367191523152]
```



- ***Determination of Fourier coefficients with integration module:***

PROGRAM:

```
# INTEGRATION MODULE (SIMPSON'S 1-THIRD METHOD):

def integrate(yy, xi, xf, n):
    h = (xf-xi)/(n-1)

    (i, s) = (1, 0)
    while i < (n-1):
```

```python
        k = i % 2

        if k == 1:
            s = s+4*yy[i]
        else:
            s = s+2*yy[i]

        i = i+1

    s = (yy[0]+yy[n-1]+s)*h/3

    return s

# DETERMINATION OF THE FOURIER SERIES COEFFICIENTS FOR GIVAN SITUATION AND BOUNDARY CONDITIONS :
# AND PLOTTING OF THE EXACT FOURIER FUNCTION FOR DIFFERENT VALUES OF x :

import matplotlib.pyplot as plt
import numpy as np
from integration import *

print("\n\tDETERMINATION OF THE FOURIER SERIES COEFFICIENTS FOR GIVAN SITUATION AND BOUNDARY CONDITIONS :\n")

xi = -1
xf = 1

xm = -1/2

n = 1000
n1 = int(n*abs((xm-xi)/(xf-xi)))
n2 = int(n*abs((xf-xm)/(xf-xi)))
nc = 25

xx = np.linspace(xi, xf, n)
xx1 = np.linspace(xi, xm, n1)
xx2 = np.linspace(xm, xf, n2)


cc = []
```

```
cs = []
yy = []


for i in range(nc):
    yc1 = -np.cos(np.pi*i*xx1)
    yc2 = np.cos(np.pi*i*xx2)

    ys1 = -np.sin(np.pi*i*xx1)
    ys2 = np.sin(np.pi*i*xx2)

    sc = integrate(yc1, xi, xm, n1) + integrate(yc2, xm, xf, n2)
    ss = integrate(ys1, xi, xm, n1) + integrate(ys2, xm, xf, n2)

    cc.append(sc)
    cs.append(ss)

cc[0] = cc[0]/2
yy = []
zz = np.zeros(n)

print("Coefficients of cosine function : ", cc, "\nCoefficients of sine
function : ", cs)

for i in range(n):
    s = 0

    for j in range(nc):
        s = s + cc[j]*np.cos(np.pi*j*xx[i]) + cs[j]*np.sin(np.pi*j*xx[i])

    yy.append(s)

    if xx[i] <= -1/2:
        zz[i] = -1
    else:
        zz[i] = 1

plt.plot(xx, yy, '-r', xx, zz, '-.b')
```

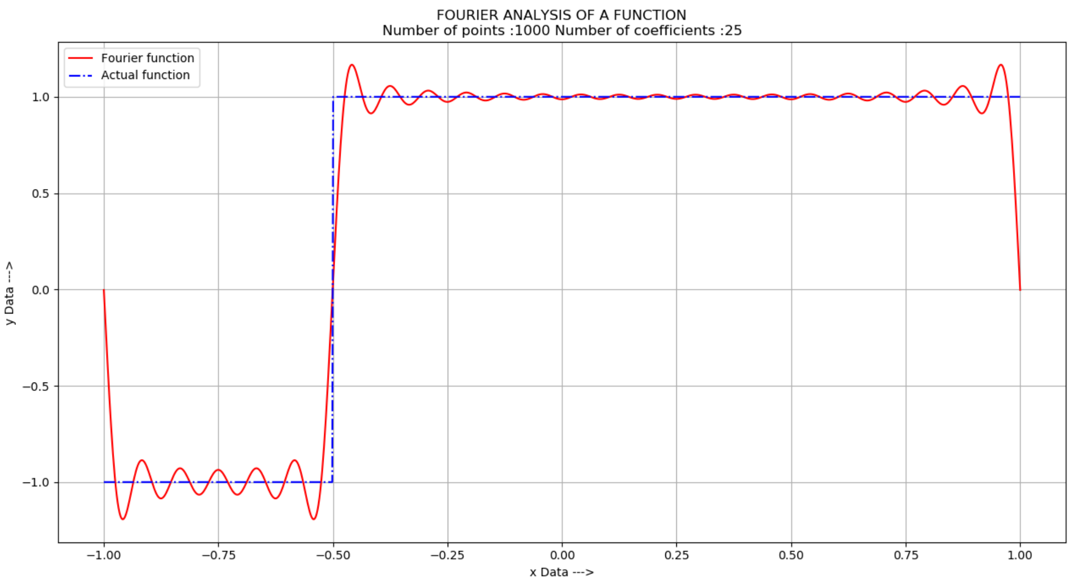
```
plt.title("FOURIER ANALYSIS OF A FUNCTION"+"\nNumber of points :"+str(n)+" Number of coefficients :"+str(nc))

plt.xlabel("x Data --->")

plt.ylabel("y Data --->")

plt.legend(["Fourier function", "Actual function"])

plt.grid()


plt.show()
```

OUTPUT:

DETERMINATION OF THE FOURIER SERIES COEFFICIENTS FOR GIVAN SITUATION AND BOUNDARY CONDITIONS :

Coefficients of cosine function : [0.5000008936502575, 0.6372852178571413, -0.0013369007851609278, -0.21153270025425822, 1.7873005150495329e-06, 0.12798095478040422, -0.0013369007851609582, -0.09026334577458535, 1.787300515065579e-06, 0.0713840847007406, -0.0013369007851610035, -0.057183742349962915, 1.7873005150146216e-06, 0.04961085905302777, -0.0013369007851610124, -0.041742087817690655, 1.787300515083902e-06, 0.03807988034526334, -0.0013369007851609907, -0.032798623478153086, 1.787300515209019e-06, 0.030938434469283935, -0.0013369007851609868, -0.026962987604537837, 1.7873005150837937e-06]

Coefficients of sine function : [0.0, 0.6359483283320411, -0.6366113498599372, 0.21286963482139154, -4.504471568258329e-08, 0.1266441103044902, -0.2121813227313752, 0.09160032540521143, -9.011804719941371e-08, 0.07004728530978864, -0.12728183926070374, 0.058520767094192044, -1.3524866468287407e-07, 0.04827410481150616, -0.09088671672375252, 0.04307915775438066, -1.804653481714296e-07, 0.03674317134645662, -0.07065971164332047, 0.03413573871505965, -2.2579704168480092e-07, 0.029601770835470366, -0.05778184774541635, 0.02830014827852027, -2.7127290947448344e-07]



❖ **Write a program to determine the maximum, minimum, average, rms value and standard deviation of a function $f(x) = x^2 exp(-x^2)$ in a given range$[0,\pi]$:**

PROGRAM:

```
# DETERMINATION OF THE VALUE OF MAXIMUM, AVERAGE, RMS AND STANDARD
DEVIATION OF A GIVEN FUNCTION:


import numpy as np

from integration import *

from summation import *


print("\n\tDETERMINATION OF THE VALUE OF MAXIMUM, AVERAGE, RMS AND
STANDARD DEVIATION OF A GIVEN FUNCTION:\n")


xi = 0

xf = 5

n = 10000

xx = np.linspace(xi, xf, n)

# Suppose,

#    my function is : yi=f(xi)= (x^2)*exp(-x^2)

yy = (xx**2)*np.exp(-xx**2)

zz = np.ones(len(xx))



# MAXIMUM VALUE OF THE FUNCTION:

max = yy[0]

for i in range(len(yy)):

    if max < yy[i]:

        max = yy[i]

print("Maximum value of the function in range ", xi, "to ", xf, " :", max)



# MINIMUM VALUE OF THE FUNCTION:

min = yy[0]

for i in range(len(yy)):
```

```python
        if min > yy[i]:
            min = yy[i]

    print("Minimum value of the function in range ", xi, "to ", xf, " :", min)



    # DETERMINATION OF AVG. VALUE:
    s = integrate(yy, xi, xf, n)
    z = integrate(zz, xi, xf, n)

    avg = s/z
    print("Average value of the function in range ", xi, "to ", xf, " :", avg)



    # DETERMINATION OF THE R.M.S VALUE:
    sum = integrate(yy**2, xi, xf, n)
    rms = (sum/z)**(1/2)

    print("RMS value of the function in range ", xi, "to ", xf, " :", rms)



    # DETERMINATION OF THE STANDARD DEVIATION:
    v = (yy - avg)**2
    add = summation(v,0,n-1,1)

    sd = (add/(n-1))**0.5

    print("Standard deviation of the function in range ", xi, "to ", xf, " :",
    sd)
```

OUTPUT:

```
      DETERMINATION OF THE VALUE OF MAXIMUM, AVERAGE, RMS AND STANDARD DEVIATION OF A
GIVEN FUNCTION:

Maximum value of the function in range  0 to  5  : 0.36787943381262717
Minimum value of the function in range  0 to  5  : 0.0
Average value of the function in range  0 to  5  : 0.08862564702186711
RMS value of the function in range  0 to  5  : 0.15329847846752445
Standard deviation of the function in range  0 to  5  : 0.1250857465091682
```