LAB-01

Python Basics

1. Variables and Data Types

In Python, variables are used to store data values. The type of the variable is determined by the value assigned to it. Python is dynamically typed, meaning you don't need to explicitly declare the type of a variable.

Primitive Data Types:

- `int` - Integer numbers (10, -5, 0)
- `float` - Floating point numbers (3.14, -0.001)
- `str` - Strings, which represent text ("Hello, World!")
- `bool` - Boolean values (`True` or `False`)

2. Operators

Operators are used to perform operations on variables and values. Python has various types of operators:

Types of Operators:

- **Arithmetic Operators:** +, -, *, /,
- **Comparison Operators:** ==, !=, ¿, ¡, ¿=, ¡=
- **Logical Operators:** `and`, `or`, `not`
- **Assignment Operators:** =, +=, -=, *=, /=, %=, `**=`, `//=`

3. Conditional Statements

Conditional statements are used to execute code based on certain conditions.

Syntax:

- `if` statement checks the condition.
- `else` statement is executed if the `if` condition is `False`.
- `elif` (else if) allows checking multiple conditions.

Example:

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

4. Loops

Loops allow you to execute a block of code repeatedly.

Types of Loops:

- **For Loop:** Repeats a block of code for a specified number of times or iterates over a sequence (list, string).
- **While Loop:** Repeats a block of code as long as a condition is `True`.

Example of a For Loop:

```
for i in range(5):
    print(i)
```

Example of a While Loop:

```
x = 0
while x < 5:
    print(x)
    x += 1
```

5. Functions

A function is a block of reusable code that performs a specific task.

Defining a Function:

```
def my_function():
    print("Hello, World!")
```

Calling a Function:

```
my_function()
```

Parameters and Arguments:

```
def greet(name):
    print(f"Hello, {name}!")

greet("Alice")
```

Returning Values:

```
def add(a, b):
    return a + b

result = add(2, 3)
print(result)
```

Lambda Functions: A lambda function is a small anonymous function defined using the `lambda` keyword.

```
multiply = lambda x, y: x * y
print(multiply(5, 6))
```

LAB-02

DDA Line Algorithm

The Digital Differential Analyzer (DDA) algorithm is used for line generation in computer graphics.

Implementation:

```
import matplotlib.pyplot as plt

x1 = int(input("Enter the value of X1: "))
y1 = int(input("Enter the value of Y1: "))
x2 = int(input("Enter the value of X2: "))
y2 = int(input("Enter the value of Y2: "))

dy = y2 - y1
dx = x2 - x1
m = dy / dx

steps = max(abs(dx), abs(dy))

xcor = []
ycor = []

for i in range(int(steps)):
    if m < 1:
        x1 += 1
        y1 += m
    else:
        x1 += 1 / m
        y1 += 1

    xcor.append(round(x1))
    ycor.append(round(y1))
    print("X1:", round(x1), "Y1:", round(y1))

plt.plot(xcor, ycor, marker="o", color="red")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.title("DDA Algorithm")
plt.show()
```

Input:

```
Enter the value of X1: 32
Enter the value of Y1: 35
Enter the value of X2: 41
```

```
Enter the value of Y2: 41
X1: 33 Y1: 36
X1: 34 Y1: 36
X1: 35 Y1: 37
X1: 36 Y1: 38
X1: 37 Y1: 38
X1: 38 Y1: 39
X1: 39 Y1: 40
X1: 40 Y1: 40
X1: 41 Y1: 41
```

Output:

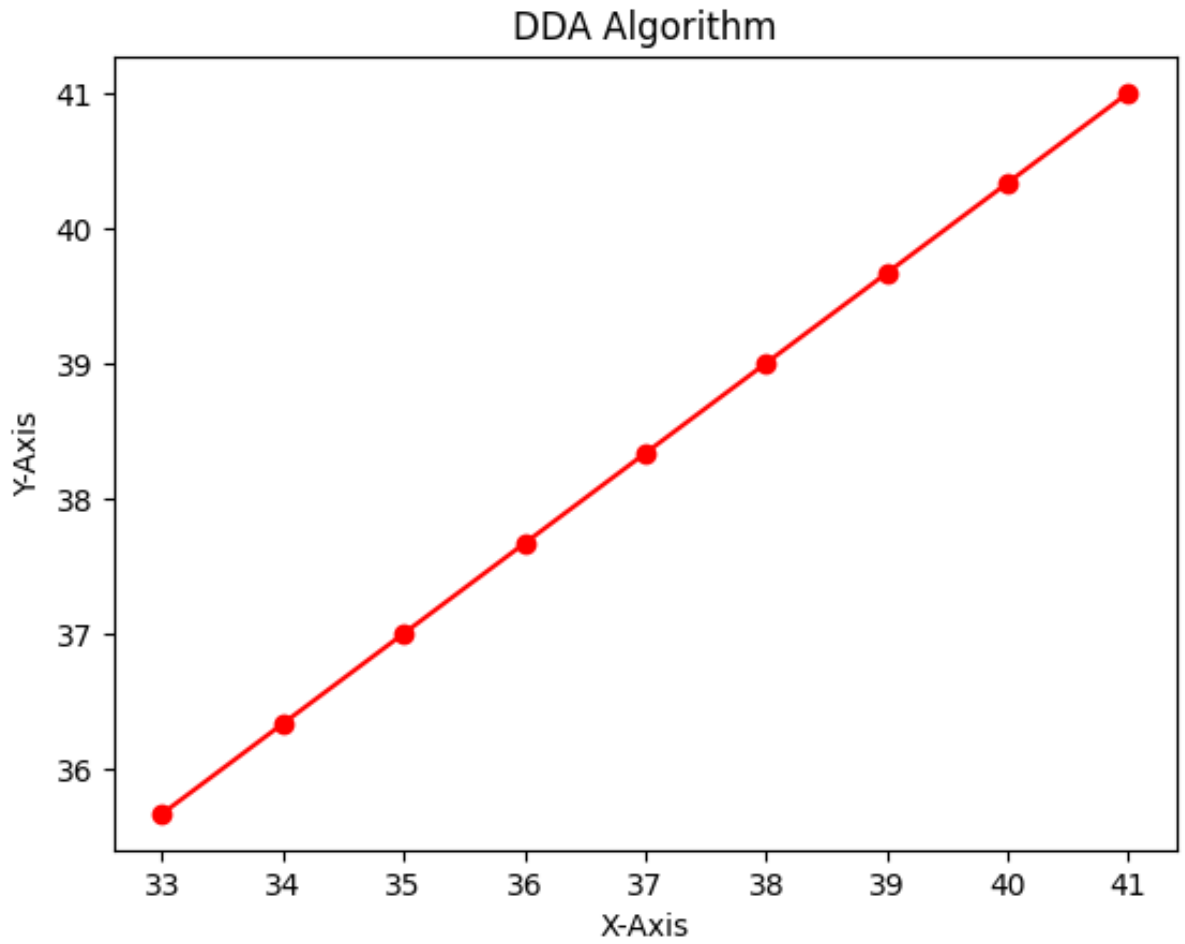

Figure 1: DDA Algorithm Output

LAB-03

Bresenham Line Algorithm

Bresenham's Line Algorithm is an efficient algorithm for drawing lines in computer graphics using only integer calculations.

Implementation:

```
import matplotlib.pyplot as plt
print("Enter the value of x1")
x1=int(input())
print("Enter the value of x2")
x2=int(input())
print("Enter the value of y1")
y1=int(input())
print("Enter the value of y2")
y2=int(input())
dx= x2-x1
dy=y2-y1
pk=2*dy-dx
if abs(dx) > abs(dy):
      steps = abs(dx)
else:
      steps = abs(dy)

xcoordinate = []
ycoordinate = []

i=0
while i<steps:
      i+=1
      if pk>=0:
       x1=x1+1
       y1=y1+1
       pk=pk+2*dy-2*dx
      elif pk<1:
       x1=x1+1
       y1=y1
       pk=pk+2*dy
      print("x1:",x1, "y1:", y1)
      xcoordinate.append(x1)
      ycoordinate.append(y1)
plt.plot(xcoordinate,ycoordinate, color='red', marker='o')
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("Bresenham Line Drawing Algorithm")
plt.show()
```

Input:

```
Enter the value of x1: 32
Enter the value of x2: 41
Enter the value of y1: 35
Enter the value of y2: 41
x1:  33 y1: 36
x1:  34 y1: 36
x1:  35 y1: 37
x1:  36 y1: 38
x1:  37 y1: 38
x1:  38 y1: 39
x1:  39 y1: 40
x1:  40 y1: 40
x1:  41 y1: 41
```

Output:

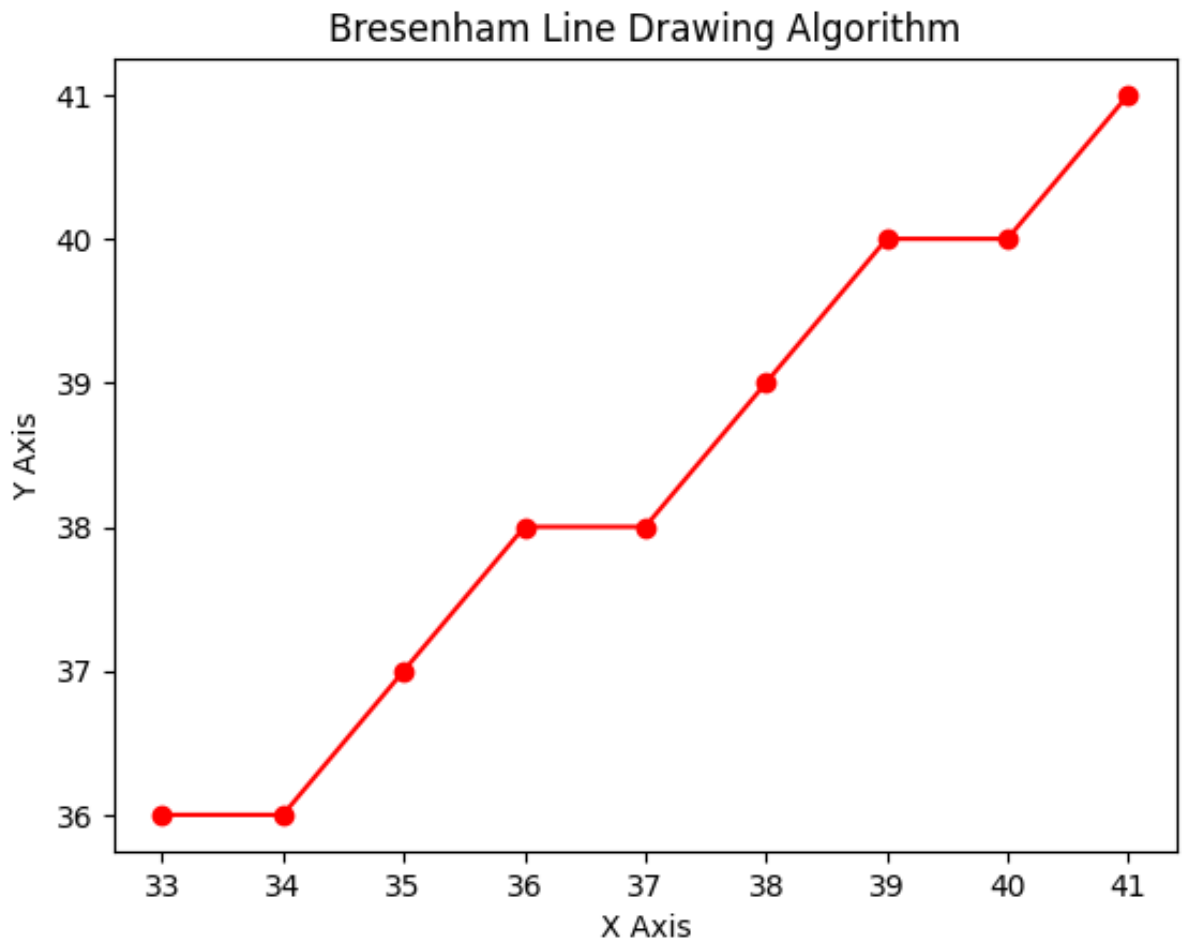

Figure 2: Bresenham Algorithm Output

LAB-04

Midpoint Line Algorithm

The Midpoint Line Algorithm is an efficient method for drawing lines in computer graphics by selecting pixels closest to the line using integer calculations.ions.

Implementation:

```
import matplotlib.pyplot as plt
print("Enter the value of x1")
x1=int(input())
print("Enter the value of x2")
x2=int(input())
print("Enter the value of y1")
y1=int(input())
print("Enter the value of y2")
y2=int(input())
dx= x2-x1
dy=y2-y1
Dk=2*dy-dx
D=2*(dy-dx)
if abs(dx) > abs(dy):
      steps = abs(dx)
else:
      steps = abs(dy)

xcoordinate = []
ycoordinate = []
i=0
while i<steps:
      i+=1
      if Dk>=0:
       x1=x1+1
       y1=y1+1
       Dk=Dk+D
      elif Dk<1:
       x1=x1+1
       y1=y1
       Dk=Dk+2*dy
      print("x1:",x1, "y1:", y1)
      xcoordinate.append(x1)
      ycoordinate.append(y1)
plt.plot(xcoordinate,ycoordinate, color='red', marker='o')
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("Mid-point Line Drawing Algorithm")
plt.show()
```

Input:

```
Enter the value of x1: 10
Enter the value of x2: 20
Enter the value of y1: 15
Enter the value of y2: 20
x1:  11 y1: 16
x1:  12 y1: 16
x1:  13 y1: 17
x1:  14 y1: 17
x1:  15 y1: 18
x1:  16 y1: 18
x1:  17 y1: 19
x1:  18 y1: 19
x1:  19 y1: 20
x1:  20 y1: 20
```

Output:

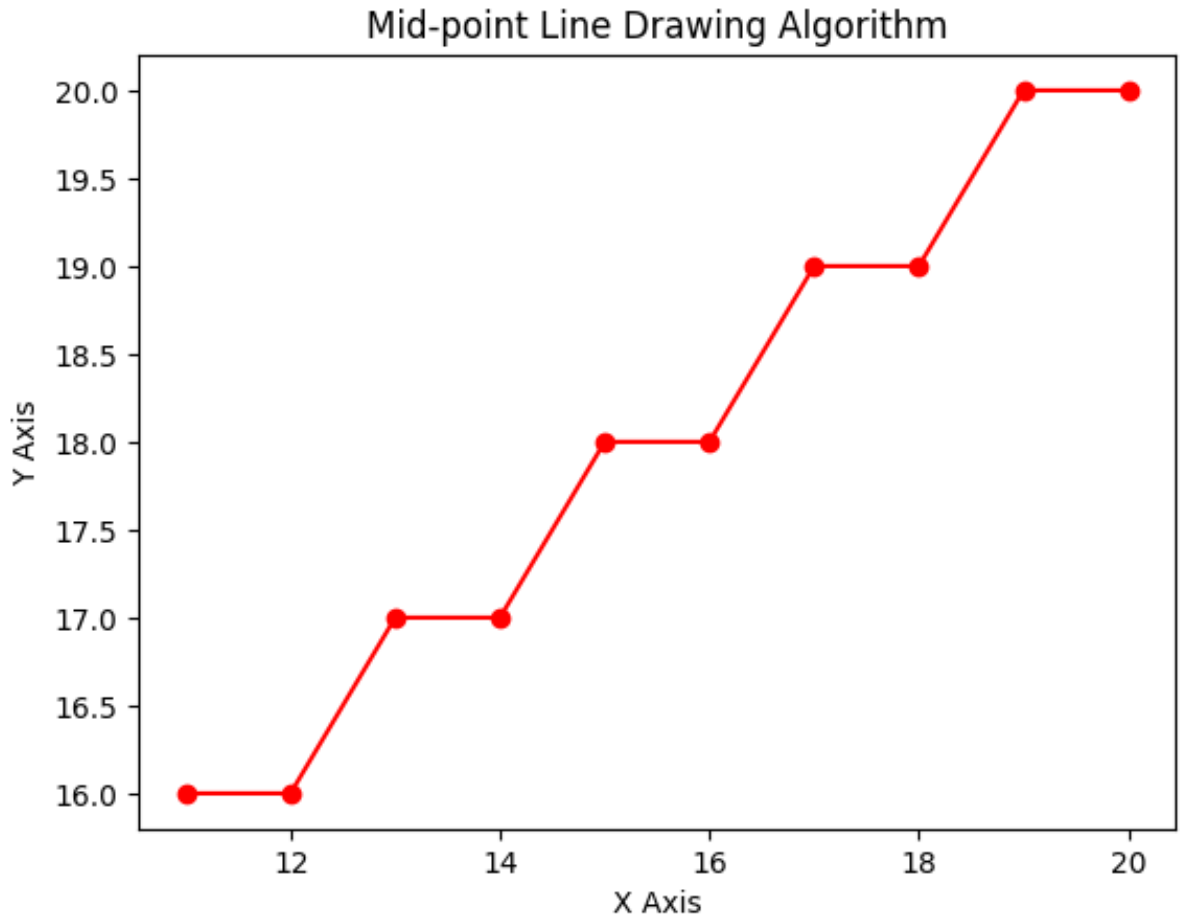

Figure 3: Midpoint Line Algorithm Output

LAB-05

Midpoint Circle Algorithm

The Midpoint Circle Algorithm is an efficient algorithm in computer graphics for drawing circles using only integer calculations to determine the nearest pixel positions.

Implementation:

```
import matplotlib.pyplot as plt
r = int(input("Enter the radius of the circle: "))
x = 0
y = r
p = 1 - r
xcor=[]
ycor=[]
while x <= y:

    if p < 0:
        x=x+1
        y=y
        p = p +  2 * x + 1
    else:
        x=x+1
        y=y-1
        p = p - 2*y + 2*x + 1

    xcor.append(x)
    ycor.append(y)
    xcor.append(y)
    ycor.append(x)
    xcor.append(x)
    ycor.append(-y)
    xcor.append(y)
    ycor.append(-x)
    xcor.append(-x)
    ycor.append(-y)
    xcor.append(-y)
    ycor.append(-x)
    xcor.append(-x)
    ycor.append(y)
    xcor.append(-y)
    ycor.append(x)

plt.scatter(xcor,ycor)
plt.gca().set_aspect('equal')
plt.xlabel("x-axis")
plt.ylabel("y-axis")
```

```
plt.title("MID-POINT-CIRCLE-ALGO")
plt.show()
```

Input:

```
Enter the radius of the circle: 200
```

Output:

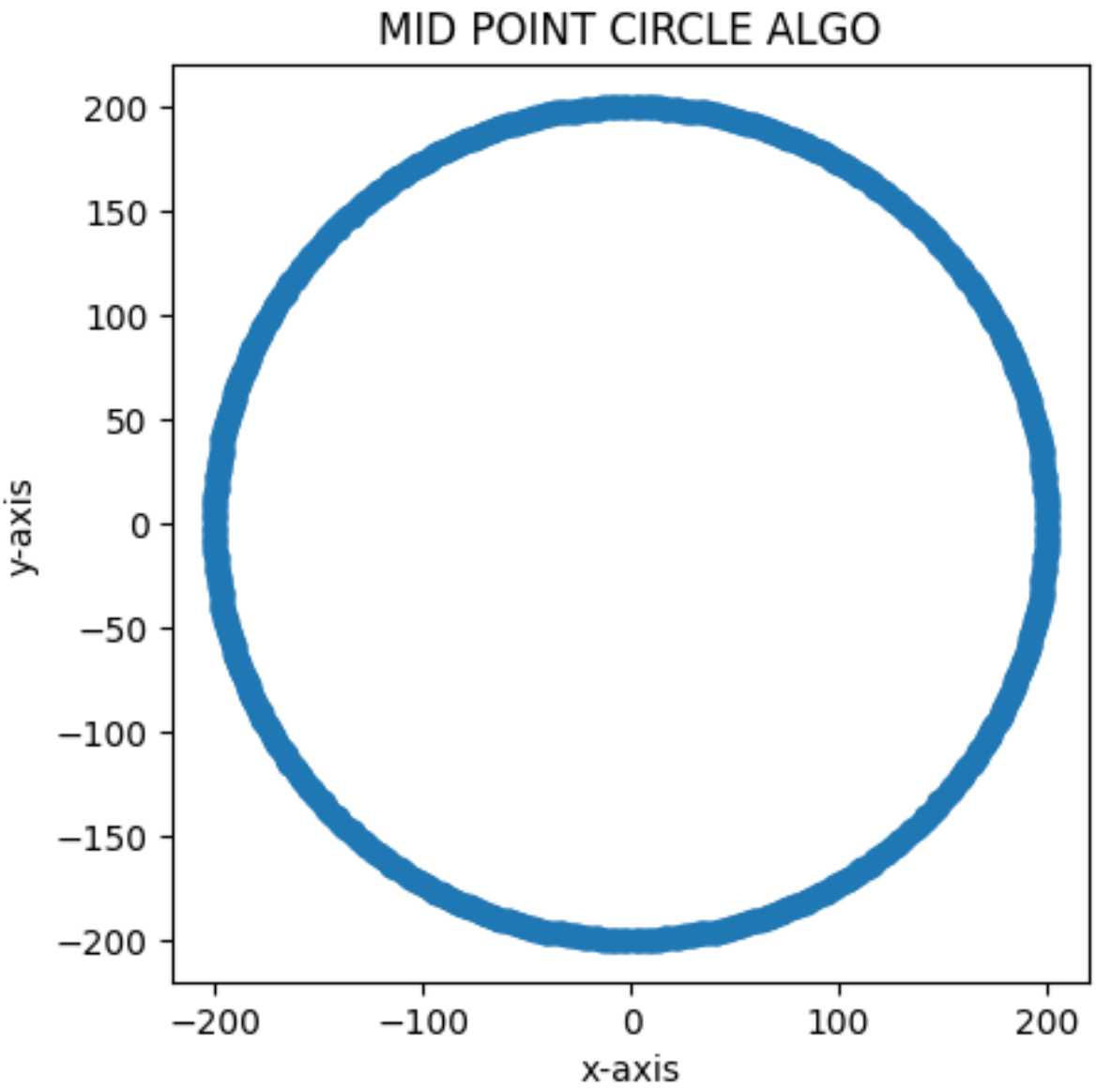

Figure 4: Midpoint Circle Algorithm Output

LAB-06

Bresenham Circle Algorithm

Bresenham's Circle Algorithm is an efficient method for drawing circles in computer graphics using only integer calculations and exploiting circle symmetry.

Implementation:

```
import matplotlib.pyplot as plt

r = int(input("Enter the radius of the circle: "))
x = 0
y = r
p = 1 - r

points = []

while x <= y:

    if p < 0:
        x=x+1
        y=y
        p=p+4*x+6
    else:
        x=x+1
        y=y-1
        p=p+4*(x-y)+10

    points.append((x, y))
    points.append((y, x))
    points.append((-x, y))
    points.append((-y, x))
    points.append((-x, -y))
    points.append((-y, -x))
    points.append((x, -y))
    points.append((y, -x))

x_vals, y_vals = zip(*points)

plt.scatter(x_vals, y_vals, color='black')
plt.gca().set_aspect('equal')
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("Bresenham Circle algorithm")
plt.show()
```

Input:

```
Enter the radius of the circle: 50
```

Output:

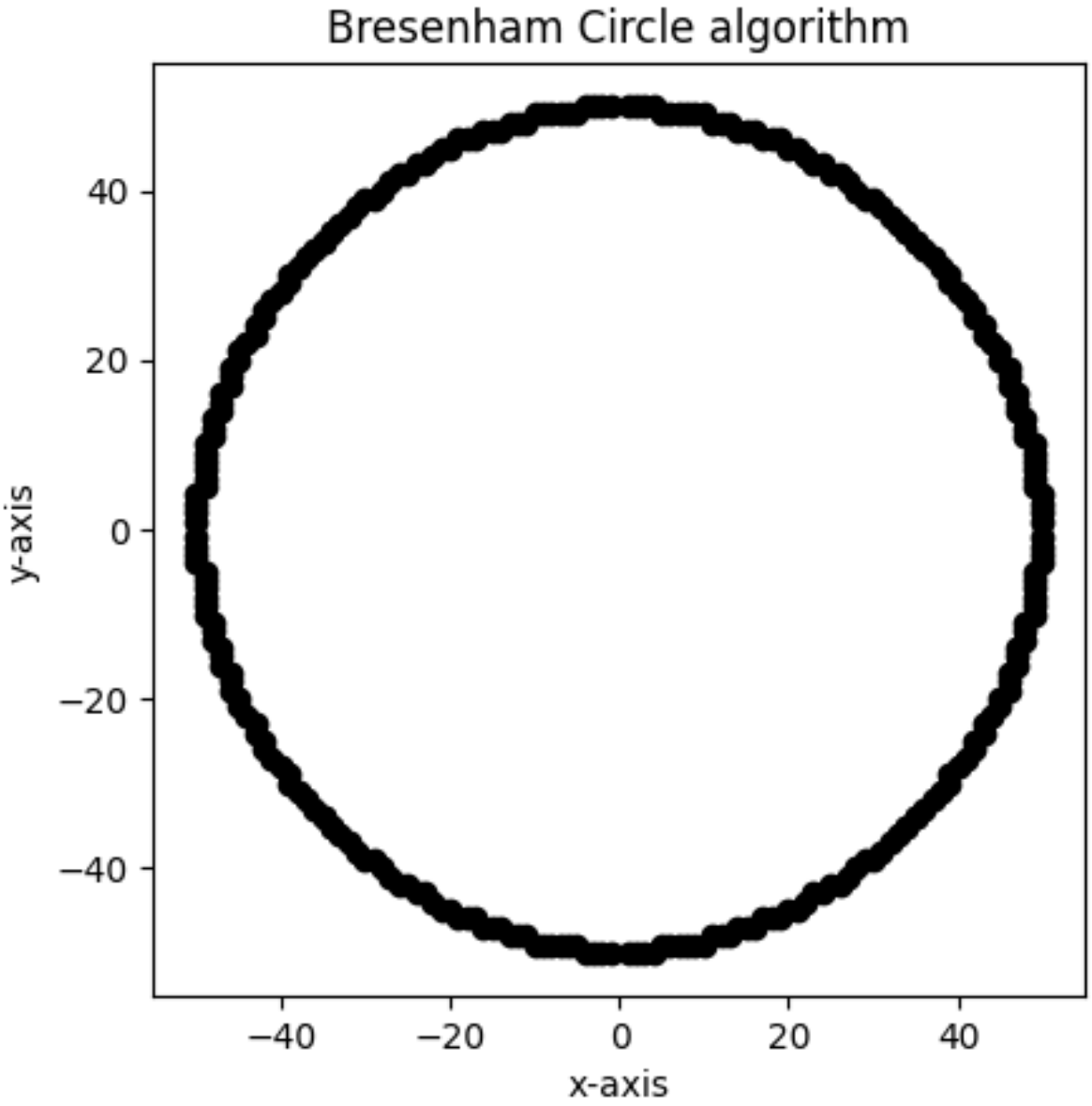

Figure 5: Bresenham Circle Algorithm Output