```python
# QUESTION 1A

from sympy import diff, symbols,lambdify
import sympy as sp
import numpy as np

def gradient_descent(x0, alpha, n):
    # Define the variable
    x = symbols('x')

    # Define the function f(x)
    f = x**2 - 6*x + 9

    # Compute the derivative f'(x)
    f_prime =diff(f, x)

    # Convert symbolic expressions to numerical functions
    f_func = lambdify(x, f, 'numpy')
    f_prime_func =lambdify(x, f_prime, 'numpy')

    # Initialize lists to store x and f(x) values
    x_values = [x0]
    fx_values = [f_func(x0)]

    # Perform gradient descent iterations
    for i in range(n):
        x_current = x_values[-1]
        x_new = x_current - alpha * f_prime_func(x_current)
        x_values.append(x_new)
        fx_values.append(f_func(x_new))

    return x_values, fx_values

# QUESTION 1B
# Runing the function with x0 = 2, alpha = 0.1, and n = 10.
x_vals, f_vals = gradient_descent(x0=2, alpha=0.1, n=10)

# Print results as a table
print("{:<15} {:<15} {:<15}".format('Iteration', 'x-value', 'f(x)-value'))
print("-" * 30)
for i in range(len(x_vals)):
    print("{:<15} {:<15.5f} {:<15.5f}".format(i, x_vals[i], f_vals[i]))
```

```
Iteration       x-value         f(x)-value
------------------------------
0               2.00000         1.00000
1               2.20000         0.64000
2               2.36000         0.40960
3               2.48800         0.26214
4               2.59040         0.16777
5               2.67232         0.10737
6               2.73786         0.06872
7               2.79028         0.04398
8               2.83223         0.02815
9               2.86578         0.01801
10              2.89263         0.01153
```

```
# QUESTION 1C
# Runing the function with x0 = 2, alpha =1 and n = 10 to check the outputs
x_vals, f_vals = gradient_descent(x0=2, alpha=1, n=10)
print("{:<15} {:<15} {:<15}".format('Iteration', 'x-value', 'f(x)-value'))
print("-" * 30)
for i in range(len(x_vals)):
   print("{:<15} {:<15.5f} {:<15.5f}".format(i, x_vals[i], f_vals[i]))
```

```
Iteration       x-value         f(x)-value
------------------------------
0               2.00000         1.00000
1               4.00000         1.00000
2               2.00000         1.00000
3               4.00000         1.00000
4               2.00000         1.00000
5               4.00000         1.00000
6               2.00000         1.00000
7               4.00000         1.00000
8               2.00000         1.00000
9               4.00000         1.00000
10              2.00000         1.00000
```

EXPLAINING THE RESULTS WITH LEARNING RATE OF 0.1 AND LEARNING RATE OF 1

When the learning rate is 0.1: the updates to x are small and gradual,allowing the algorithm to slowly approach the minimum at x=3.The function f(x)=x**2-6*x+9 decreases steadily,and x moves closer to the minimum (x=3) while f(x) approaches 0.

When the learning rate is 1: The algorithm makes larger updates that overshoots the minimum.For instance,if x=2, the minimum jumps to 4 and if x=4 it jumps back to 2.Hence this makes the algorithm to oscillate between 2 and 4 instead of converging to 3, hence ineffective for minimizing the function.Therefore ,a learning rate of 1 can cause the algorithm to fail to converge and instead oscillate,leading to poor results.

```
# QUESTION 2A
# Solving a system of linear equations, 2I1 + 3I2 =8  AND 5I1 + 7I2 =19

import numpy as np
A=np.array([[2,3],[5,7]])
B=np.array([8,19])
I=np.linalg.solve(A,B)
print("I1 = {} Amperes\nI2 = {} Amperes".format(I[0], I[1]))
```

```
I1 = 1.0 Amperes
I2 = 2.0 Amperes
```

```
# QUESTION 2B
# Loading data from circuit.txt
data= np.loadtxt(r"G:\CS NOTES\2.2\SCIENTIFIC COMPUTING\Assignment\Scientic
computing cat\circuit.txt")

# Transpose the data matrix
C=data.T

# Extracting the coefficients matrix
```

```
A = C[0:2].T

# Extracting the costants matrix
B=C[2]

I=np.linalg.solve(A,B)

print("I1 = {} Amperes\nI2 = {} Amperes".format(I[0], I[1]))
```

```
I1 = 1.0 Amperes
I2 = 2.0 Amperes
```

QUESTION 2C

If the matrix of coefficients is nearly singular, it means the two equations are almost the same or very close to being dependent. This causes problems when solving because small errors in the numbers (like rounding or small measurement mistakes) can make the solution change a lot. This makes the result unstable and not reliable.

To fix this in Python, we can use more stable methods like np.linalg.lstsq, which gives an approximate solution that is less sensitive to small errors. We can also check how close the matrix is to being singular using np.linalg.cond(A). A high condition number means the system is not stable and needs careful handling.

```
# QUESTION 3A

import pandas as pd
# Reading data using pandas
data = pd.read_csv(r"G:\CS NOTES\2.2\SCIENTIFIC COMPUTING\Assignment\Scientic
computing cat\Temp_dataset.csv", encoding='latin1')
data
```

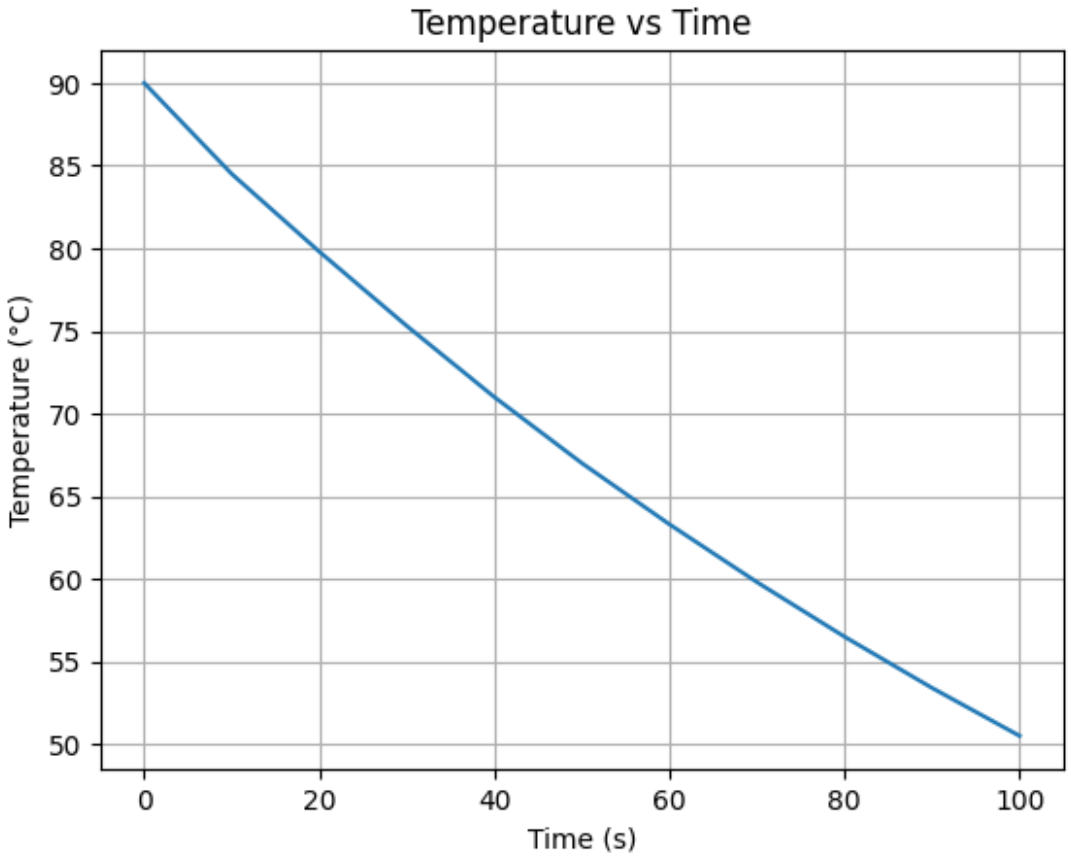
	Time(s)	Temperature(°C)
0	0	90.0
1	10	84.5
2	20	79.8
3	30	75.3
4	40	71.0
5	50	67.0
6	60	63.3
7	70	59.8
8	80	56.5
9	90	53.4
10	100	50.5

```
import matplotlib.pyplot as plt
plt.plot(data.iloc[:, 0], data.iloc[:, 1])

plt.xlabel("Time (s)")
plt.ylabel("Temperature (°C)")

plt.title("Temperature vs Time")

plt.grid()
plt.show()
```



QUESTION 3B

Observations from graph:

1. The temperature decreases over time: The plot clearly shows that as time progresses (along the x-axis), the temperature (along the y-axis) consistently goes down. This therefore indicates a cooling process is taking place.

2. The rate of cooling is not constant: Since the line on the graph is curved, specifically concave up.This tells us that the temperature is dropping more rapidly at the beginning of the process and then the rate of cooling slows down as time goes on. In other words, the temperature changes more drastically in the initial few seconds compared to later on.

QUESTION 3C

A situation in engineering where such visualization can be used to influence decision-making:

In Embedded Systems & Electronics, temperature vs. time graphs are used to monitor the thermal behavior of components like CPUs or microcontrollers. This helps engineers detect overheating, evaluate cooling efficiency, and make informed decisions—such as adding heat sinks, improving airflow, or adjusting processor speed through firmware. The goal is to prevent thermal damage, maintain performance, and ensure system reliability.

QUESTION 4A

Euler's method formula:

```
Yn+1 =Yn +h*f'(Tn,Yn)

Tn+1=Tn+h
```

First order Differential equation:

```
dy/dt=0.5y
```

Intitial condtions:

```
t0=0

Y0=100
```

Step size:

```
h=0.5
```

When n=1:

```
t1=t0+h=0+0.5=0.5

Y1=Y0 +h*f'(Y0)=100+0.5*0.5*100=125
```

When n=2:

```
t2=t1+h=0.5+0.5=1

Y2=Y1 +0.5*0.5*125=156.25
```

When n=3:

```
t3=t2+h=1+0.5=1.5

Y3=156.25+0.5*0.5*156.25=195.3125
```

When n=4:

```
t4=t3+h=1.5+0.5=2

Y4=195.3125+0.5*0.5*195.3125=244.140625
```

Hence the approximate population size using Euler's method is: **244.140625**

```
# QUESTION 4B
def euler_method(f, y0, t0, h, n):
    """
    f=first order Differential equation
    y0,t0=intitial condtions
    h=step size
    n=number of steps
    """
    t_values = [t0]
    y_values = [y0]

    t_current = t0
```

```
    y_current = y0

    for _ in range(n):
        y_current = y_current + h * f(t_current, y_current)
        t_current += h

        t_values.append(t_current)
        y_values.append(y_current)

    return t_values, y_values
```

QUESTION 4 C

LIMITATIONS OF EULER'S METHOD

1. Stability Issues In some problems especially those with stiff differential equations,that are common in engineering.Euler's method can become unstable.This means the solution may grow widly or give wrong results unless the step size is made very small.Using small step sizes increases the amount of time and computing power needed.Moreover,if the step size is not chosen carefully, Euler's method might still fail to.

2. Low Accuracy Euler's Method is not very accurate.To get good results, you require very small step sizes, which means doing alot of calculations.This makes the Euler's method slow and not very efficient, especially when high precision is needed.

ALTERNATIVE FOR EULER'S METHOD

Runge-Kutta Methods:

Especially the RK4 method is a better option than Euler's method for solving real world problems.It gives more accurate results and is more stable, even when the step size is not very small.Unlike Euler's method, which can easily become inaccurate or unstable. RK4 takes more careful steps by averaging several estimates at each stage.This makes i more efficient and precise.Because of this RK4 is often used in science and engineering to model systems more reliably and safely.