5.9.3

Sai Krishna Bakki - EE25BTECH11049

Question

Two schools **P** and **Q** decided to award prizes to their students for two games of Hockey x per students and cricket y per student. School **P** decided to award a total of 9,500 for the two games to 5 and 4 students respectively; while school **Q** decided to award 7,370 for the two games to 4 and 3 students respectively. Based on the given information, answer the following questions :

1. Represent the following information algebraically (in terms of x and y).
2.
   1. What is the prize amount for hockey ?
   2. Prize amount on which game is more and by how much ?
3. What will be the total prize amount if there are 2 students each from two games ?

Theoretical Solution

Given

For Schools **P** and **Q**:

$$5x + 4y = 9500 \tag{1}$$

$$4x + 3y = 7370 \tag{2}$$

$$\implies \begin{pmatrix} 5 & 4 \\ 4 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 9500 \\ 7370 \end{pmatrix} \tag{3}$$

$$\left(\begin{array}{cc|c} 5 & 4 & 9500 \\ 4 & 3 & 7370 \end{array}\right) R_1 \to R_1 - R_2 \left(\begin{array}{cc|c} 1 & 1 & 2130 \\ 4 & 3 & 7370 \end{array}\right) \tag{4}$$

$$\left(\begin{array}{cc|c} 1 & 1 & 2130 \\ 4 & 3 & 7370 \end{array}\right) R_2 \to R_2 - 4R_2 \left(\begin{array}{cc|c} 1 & 1 & 2130 \\ 0 & -1 & -1150 \end{array}\right) \tag{5}$$

$$\left(\begin{array}{cc|c} 1 & 1 & 2130 \\ 0 & -1 & -1150 \end{array}\right) R_1 \to R_1 + R_2 \left(\begin{array}{cc|c} 1 & 0 & 980 \\ 0 & -1 & -1150 \end{array}\right) \tag{6}$$

Theoretical Solution

$$\left(\begin{array}{cc|c} 1 & 0 & 980 \\ 0 & -1 & -1150 \end{array}\right) R_2 \rightarrow -R_2 \left(\begin{array}{cc|c} 1 & 0 & 980 \\ 0 & 1 & 1150 \end{array}\right) \tag{7}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 980 \\ 1150 \end{pmatrix} \tag{8}$$

$\therefore$ The prize amount for Hockey(x) and Cricket(y) respectively are 980 and 1150. The Prize amount of Cricket is more than Hockey by a difference of 170.

$$\text{Total amount} = \begin{pmatrix} 2 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{9}$$

$$\text{Total amount} = \begin{pmatrix} 2 & 2 \end{pmatrix} \begin{pmatrix} 980 \\ 1150 \end{pmatrix} \tag{10}$$

$$\begin{aligned} \text{Total amount} &= 1960 + 2300 \\ &= 4260 \end{aligned} \tag{11}$$

C Code

```c
#include <stdio.h>
#include <math.h> // Required for fabs()

// This function solves a system of two linear equations using an
    augmented matrix
// and Gaussian elimination.
// a*x + b*y = e
// c*x + d*y = f
void solve_system(double a, double b, double c, double d, double
    e, double f, double* x, double* y) {
    // Create the augmented matrix: [ a b | e ]
    // [ c d | f ]
    double aug_matrix[2][3] = {
        {a, b, e},
        {c, d, f}
    };
```

C Code

```c
// --- Forward Elimination to get Row-Echelon Form ---

// If the pivot (a) is zero, swap the rows to avoid division
    by zero.
if (fabs(aug_matrix[0][0]) < 1e-9) {
    for (int i = 0; i < 3; i++) {
        double temp = aug_matrix[0][i];
        aug_matrix[0][i] = aug_matrix[1][i];
        aug_matrix[1][i] = temp;
    }
}

// Check if the pivot is still zero, which means no unique
    solution exists.
if (fabs(aug_matrix[0][0]) < 1e-9) {
    *x = -1.0/0.0; // Represents NaN
    *y = -1.0/0.0; // Represents NaN
    return;
}
```

C Code

```
// Perform the row operation: R2 -> R2 - (c/a) * R1
double factor = aug_matrix[1][0] / aug_matrix[0][0];
aug_matrix[1][0] = 0.0; // This is the goal
aug_matrix[1][1] -= factor * aug_matrix[0][1];
aug_matrix[1][2] -= factor * aug_matrix[0][2];

// --- Back Substitution ---

// Check if the second pivot element is zero. If so, there's
    no unique solution.
if (fabs(aug_matrix[1][1]) < 1e-9) {
    *x = -1.0/0.0; // NaN
    *y = -1.0/0.0; // NaN
    return;
}
```

C Code

```c
    // Solve for y from the second row: (d')*y = f'
    *y = aug_matrix[1][2] / aug_matrix[1][1];

    // Solve for x from the first row: a*x + b*y = e => x = (e -
        b*y) / a
    *x = (aug_matrix[0][2] - aug_matrix[0][1] * (*y)) /
        aug_matrix[0][0];
}
```

Python Through Shared Output

```
import ctypes
import numpy as np
import matplotlib.pyplot as plt
from libs.funcs import line_dir_pt, param_norm

# --- Ctypes setup to call the C function ---

# Load the shared library.
# NOTE: You must compile the corresponding C file into a shared
    library first.
# On Linux/macOS, use: gcc -shared -o intersection.so -fPIC
    your_c_file.c
try:
    # The script was named intersection.py, so we assume the
        shared library
    # might be named intersection.so
    solver_lib = ctypes.CDLL('./line.so')
except OSError:
```

```
print(Error: Could not find 'line.so'.)
print(Please ensure the C code is compiled into a shared library
    named 'line.so'.)
exit()

# Define the function signature from the C code.
solve_system_c = solver_lib.solve_system
solve_system_c.argtypes = [ctypes.c_double, ctypes.c_double,
    ctypes.c_double, ctypes.c_double, ctypes.c_double, ctypes.
    c_double, ctypes.POINTER(ctypes.c_double), ctypes.POINTER(
    ctypes.c_double)]
solve_system_c.restype = None
# --- Solving the system of equations ---

# The first equation is 5x + 4y = 9500
a, b, e = 5.0, 4.0, 9500.0

# The second equation is 4x + 3y = 7370
c, d, f = 4.0, 3.0, 7370.0
```

Python Through Shared Output

```
# Create pointers for the output variables x and y
x = ctypes.c_double()
y = ctypes.c_double()

# Call the C function to solve the system
solve_system_c(a, b, c, d, e, f, ctypes.byref(x), ctypes.byref(y)
    )

# Get the Python values from the ctypes objects
x_sol, y_sol = x.value, y.value

print(fThe solution from the C library is:)
print(fx = {x_sol})
print(fy = {y_sol})

# --- Verification and Plotting ---
print(\nVerification:)
# Note: Due to floating-point precision, the result might be
    extremely close but not exactly the integer value.
```

```
print(f5*({x_sol}) + 4*({y_sol}) = {5*x_sol + 4*y_sol})
print(f4*({x_sol}) + 3*({y_sol}) = {4*x_sol + 3*y_sol})

# Normal vectors for plotting
n1 = np.array([[a], [b]])
c1 = e
n2 = np.array([[c], [d]])
c2 = f

# Generate points for the first line (widened range for
    visibility)
m1, A1 = param_norm(n1, c1)
line1_pts = line_dir_pt(m1, A1, -5000, 5000)

# Generate points for the second line (widened range for
    visibility)
m2, A2 = param_norm(n2, c2)
line2_pts = line_dir_pt(m2, A2, -5000, 5000)
```

Python Through Shared Output

```
# Plot the lines
plt.plot(line1_pts[0,:], line1_pts[1,:], label='5x + 4y = 9500')
plt.plot(line2_pts[0,:], line2_pts[1,:], label='4x + 3y = 7370')
# Plot the intersection point
plt.plot(x_sol, y_sol, 'o', markersize=8, label=f'Intersection ({
    x_sol:.2f}, {y_sol:.2f})')
# Draw x and y axes
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)

# Add labels and title for clarity
plt.xlabel(x-axis)
plt.ylabel(y-axis)
plt.title(Intersection of Two Lines (Solved with C))
plt.grid(True)
plt.legend()
plt.axis('equal')
plt.show()
```

Python Code

```
import numpy as np
from libs.funcs import line_isect, line_dir_pt, param_norm
import matplotlib.pyplot as plt

# The first equation is 5x + 4y = 9500
# The normal vector n1 is the coefficients of x and y
n1 = np.array([[5], [4]])
# The constant c1 is 9500
c1 = 9500

# The second equation is 4x + 3y = 7370
# The normal vector n2 is the coefficients of x and y
n2 = np.array([[4], [3]])
# The constant c2 is 7370
c2 = 7370

# The line_intersect function from funcs.py solves the system of
    equations.
```

Python Code

```
# It takes the two normal vectors and two constants as input.
# The system can be represented as:
# [5 4] [x] = [9500]
# [4 3] [y] [7370]
solution = line_isect(n1, c1, n2, c2)

print(fThe solution to the system of equations is:)
print(fx = {solution[0][0]})
print(fy = {solution[1][0]})

# Verification
# Let's plug the values back into the equations to check
x = solution[0][0]
y = solution[1][0]

print(\nVerification:)
# Note: Due to floating-point precision, the result might be
    extremely close but not exactly the integer value.
```

Python Code

```
print(f5*({x}) + 4*({y}) = {5*x + 4*y})
print(f4*({x}) + 3*({y}) = {4*x + 3*y})

# Plotting the lines and the intersection point
# Generate points for the first line (widened range for visibility)
m1, A1 = param_norm(n1, c1)
line1_pts = line_dir_pt(m1, A1, -5000, 5000)

# Generate points for the second line (widened range for visibility)
m2, A2 = param_norm(n2, c2)
line2_pts = line_dir_pt(m2, A2, -5000, 5000)

# Plot the lines
plt.plot(line1_pts[0,:], line1_pts[1,:], label='5x + 4y = 9500')
plt.plot(line2_pts[0,:], line2_pts[1,:], label='4x + 3y = 7370')
```

Python Code

```
# Plot the intersection point
plt.plot(solution[0], solution[1], 'o', markersize=8, label=f'
    Intersection ({x:.2f}, {y:.2f})')

# Draw x and y axes
plt.axhline(0, color='black', linewidth=0.9)
plt.axvline(0, color='black', linewidth=0.9)

# Add labels and title for clarity
plt.xlabel(x-axis)
plt.ylabel(y-axis)
plt.title(Intersection of Two Lines)
plt.grid(True)
plt.legend()
plt.axis('equal') # Ensures the axes are scaled equally
plt.show()
```

Plot By C code and Python Code

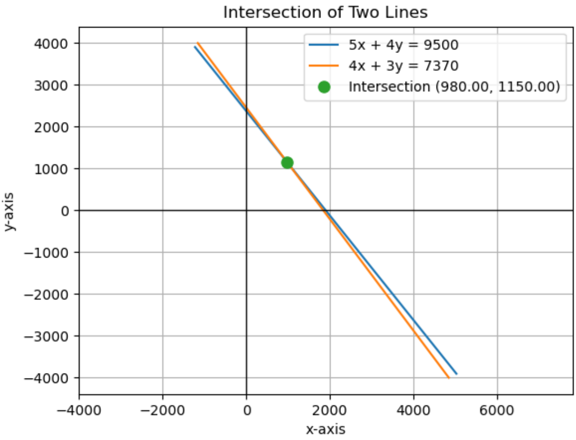

Figure: 1