# Engineering Mathematics and Computing

Task 1: Coursework Assessment

Student Name: Sakariye Abiikar
KID: 2371673

Last Updated: October 19, 2024
Submission Deadline: November 7, 2024, 5pm

Git Repo : https://github.com/sakx7/mathcompuni

# Part A

# Mathematics

# A.1 Q1

Use the quotient rule to differentiate the function $y = \frac{\ln 3x}{2x}$

The quotient rule states that if

$$y = \frac{u}{v} \quad \text{then} \quad y' = \frac{u'v - uv'}{v^2}$$

Here:

$$u = \ln 3x \qquad v = 2x$$

First, find the derivatives:

$$u = \ln(3x)$$

Let:

$$\delta = 3x \qquad u = \ln(\delta)$$

$$\frac{d\delta}{dx} = 3 \qquad \frac{du}{d\delta} = \frac{1}{\delta}$$

Using the chain rule, we can find:

$$\frac{du}{dx} = \frac{du}{d\delta} \cdot \frac{d\delta}{dx}$$

$$\frac{du}{dx} = \frac{3}{\delta} = \frac{3}{3x} = \frac{1}{x}$$

General rule for composite functions is solved via chain rule it's:

$$\frac{d}{dx}(f(g(x))) = f'(g(x)) \cdot g'(x)$$

$$v = 2x$$

$$v' = \frac{d}{dx}(2x) = 2$$

Now we can apply the quotient rule:

$$y' = \frac{\left(\frac{1}{x}\right)(2x) - (\ln(3x))(2)}{(2x)^2} = \frac{2 - 2\ln(3x)}{4x^2}$$

$$\boxed{y' = \frac{1 - \ln(3x)}{2x^2}}$$

## A.2 Q2

Find the angle between the vectors $2i - 11j - 10k$ and $5i + 8j + 7k$

The angle $\theta$ between two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by:

$$\cos\theta = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}$$

Calculate the dot product:

$$\mathbf{a}\cdot\mathbf{b} = \underbrace{(2)(5)}_{i} + \underbrace{(-11)(8)}_{j} + \underbrace{(-10)(7)}_{k} = -148$$

Calculate the magnitudes:

$$\|\mathbf{a}\| = \sqrt{2^2 + (-11)^2 + (-10)^2} = \sqrt{225} = 15$$

$$\|\mathbf{b}\| = \sqrt{5^2 + 8^2 + 7^2} = \sqrt{138}$$

Find $\cos\theta$:

$$\cos\theta = \frac{-148}{15 \times \sqrt{138}}$$

Thus, the angle $\theta$ is:

$$\boxed{\theta = \cos^{-1}\left(\frac{-148}{15 \times \sqrt{138}}\right) \approx 147.1^\circ}$$

## A.3 Q3

Find the rate of change of $y = \ln(16t^2 + 19)$ at the specified point $t = 9$

Differentiate $y$ with respect to $t$:

Note prior (q1) that the general rule for composite/nested functions is:

$$\frac{d}{dx}(f(g(x))) = f'(g(x)) \cdot g'(x)$$

Here for $y = \ln(16t^2 + 19)$

$$\frac{dy}{dt} = \frac{1}{16t^2 + 19} \cdot \frac{d}{dt}(16t^2 + 19)$$

$$\frac{d}{dt}(16t^2 + 19) = 32t$$

$$\frac{dy}{dt} = \frac{1}{16t^2 + 19} \cdot 32t = \frac{32t}{16t^2 + 19}$$

Evaluate at $t = 9$:

$$\boxed{\left.\frac{dy}{dt}\right|_{t=9} = \frac{32(9)}{16(9)^2 + 19} = \frac{288}{1315} \approx 0.219}$$

## A.4 Q4

Express $\cos t - 8\sin t$ in the form $A\cos(\omega t + \alpha)$, where $\alpha \geq 0$

I think the question isn't really wrong since it's trying to get us to express a function in a certain way. But introducing $\omega$ without clear context makes things confusing. Just because a variable is there doesn't mean it's actually important. For example, trying to express $\sin a$ in terms of $\sin(p + a)$ seems pointless if $a$ is just zero.

To me, it looks like $\omega$ is just 1 because we need the frequencies to match for the equation to work. This kind of assumption might trip some people up if they don't see it, honestly did for me a first. Honestly this might of just taken away the main aspect of this question.

Proceeding with knowing that $\omega = 1$ we start with the angle subtraction formula which is:

$$A\cos(t - \phi) = A\cos(\phi)\cos(t) + A\sin(\phi)\sin(t)$$

By comparing coefficients from both sides, we have:

$$a = A\cos(\phi)$$
$$b = A\sin(\phi)$$

To find $A$ we use $A = \sqrt{a^2 + b^2}$
This arises from squaring both equations $a = A\cos(\phi)$ and $b = A\sin(\phi)$:

$$a^2 + b^2 = (A\cos(\phi))^2 + (A\sin(\phi))^2 = A^2(\cos^2(\phi) + \sin^2(\phi)) = A^2$$

To find the phase shift $\phi$, we use $\tan\phi = \frac{b}{a}$
This comes from the definitions of sine and cosine:

$$\tan\phi = \frac{A\sin(\phi)}{A\cos(\phi)} = \frac{b}{a}$$

In so we derive and make use of:

$$A\cos(t - \phi) = a\cos(t) + b\sin(t)$$

Where $A = \sqrt{a^2 + b^2}$ and $\tan\phi = \frac{b}{a}$

for $b = -8$ and $a = 2$

$$A = \sqrt{1^2 + (-8)^2} = \sqrt{65}$$

$$\tan\phi = \frac{-8}{1} = -8 \qquad \phi = \arctan(-8) \approx -82.87°$$

In so plugging in gives

$$\begin{aligned}\cos t - 8\sin t &= A\cos(t - \phi)\\ &= \sqrt{65}\cos(t - \arctan(-8))\\ &= \sqrt{65}\cos(t - (-82.87°))\end{aligned}$$

$$\boxed{\cos t - 8\sin t \approx 8.06\cos(t + 82.87°)}$$

Were

$$A = \sqrt{65} \approx 8.06$$

$$\alpha = -\arctan(-8) \approx 82.87°$$

$$\omega = 1$$

Desmos Representaion

## A.5 Q5

Solve the following system of three linear equations using Cramer's rule

$$\begin{cases} 11v_1 - v_2 + v_3 = 31.4 \\ v_1 + \frac{v_2}{2} - v_3 = 1.9 \\ -9v_1 + 11v_3 = -12 \end{cases}$$

The system can be written in matrix form $A\mathbf{v} = \mathbf{b}$, specifically because of the variable distributions where:

$$A = \begin{bmatrix} 11 & -1 & 1 \\ 1 & \frac{1}{2} & -1 \\ -9 & 0 & 11 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 31.4 \\ 1.9 \\ -12 \end{bmatrix}$$

Calculate the determinant $\det(A)$.

$$\det(A) = 67$$

Now solve for each variable using Cramer's rule:

$$A_1 = \begin{bmatrix} 31.4 & -1 & 1 \\ 1.9 & \frac{1}{2} & -1 \\ -12 & 0 & 11 \end{bmatrix} \quad A_2 = \begin{bmatrix} 11 & 31.4 & 1 \\ 1 & 1.9 & -1 \\ -9 & -12 & 11 \end{bmatrix} \quad A_3 = \begin{bmatrix} 11 & -1 & 31.4 \\ 1 & \frac{1}{2} & 1.9 \\ -9 & 0 & -12 \end{bmatrix}$$

Calculate the determinants:

$$\det(A_1) \approx 187.6 \quad \det(A_2) \approx 40.2 \quad \det(A_3) \approx 80.4$$

$\mathbf{v}$ can be found as:

$$v_1 = \frac{\det(A_1)}{\det(A)}, \quad v_2 = \frac{\det(A_2)}{\det(A)}, \quad v_3 = \frac{\det(A_3)}{\det(A)}$$

$$\boxed{\begin{aligned} v_1 &= \tfrac{187.6}{67} = 2.8 \\ v_2 &= \tfrac{40.2}{67} = 0.6 \\ v_3 &= \tfrac{80.4}{67} = 1.2 \end{aligned}}$$

There are a few ways to calculate the determinants, but especially in this working out i chose not show the method i used since it's not the main focus of the question.

If you're curious about how I did it, the Sarrus method for simplicity, Laplace would have also sufficed.

## A.6 Q6

Transpose $z = d + a\sqrt{y}$ to make $y$ the subject.

Starting with:

$$z = d + a\sqrt{y}$$

$$z - d = a\sqrt{y}$$

$$\frac{z-d}{a} = \sqrt{y}$$

$$\left(\frac{z-d}{a}\right)^2 = y$$

Thus, the expression for $y$ is:

$$\boxed{y = \frac{(z-d)^2}{a^2}}$$

i generally consider this the most concise and general expression for $y$.
im aware you could expand $(z-d)^2$:

$$y = \frac{z^2 - 2zd + d^2}{a^2}$$

and additionally take into account of both possible factorisations:

$$y = \frac{z^2 + d(d-2z)}{a^2}$$

$$y = \frac{d^2 + z(z-2d)}{a^2}$$

## A.7 Q7

Find a vector that is perpendicular to both of the vectors

$$\boldsymbol{a} = 4\boldsymbol{i} + 3\boldsymbol{j} + 5\boldsymbol{k}$$

$$\boldsymbol{b} = 3\boldsymbol{i} + 4\boldsymbol{j} - 6\boldsymbol{k}$$

Hence find a unit vector that is perpendicular to both $\boldsymbol{a}$ and $\boldsymbol{b}$.

The unit vector is

$$\hat{\mathbf{r}} = \frac{\vec{r}}{\|\vec{r}\|}$$

Here we want to define $\vec{r}$ as a vector perpendicular to both $\mathbf{a}$ and $\mathbf{b}$, to do that we take there cross product $\mathbf{a} \times \mathbf{b}$

$$\vec{r} = \mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 4 & 3 & 5 \\ 3 & 4 & -6 \end{vmatrix} = -38\mathbf{i} + 39\mathbf{j} + 7\mathbf{k}$$

Find the magnitude:

$$\|\vec{r}\| = \|\mathbf{a} \times \mathbf{b}\| = \sqrt{(-38)^2 + 39^2 + 7^2} = \sqrt{3014}$$

The unit vector is:

$$\boxed{\frac{-38\mathbf{i} + 39\mathbf{j} + 7\mathbf{k}}{\sqrt{3014}}}$$

## A.8 Q8

If $M = \begin{pmatrix} 7 & 9 \\ 1 & -2 \end{pmatrix}$ and $N = \begin{pmatrix} 2 & 1 \\ -2 & 6 \end{pmatrix}$ find $MN$ and $NM$

Calculate $MN$:

$$MN = \begin{bmatrix} 7 & 9 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ -2 & 6 \end{bmatrix}$$

$$= \begin{bmatrix} 7 \cdot 2 + 9 \cdot (-2) & 7 \cdot 1 + 9 \cdot 6 \\ 1 \cdot 2 + (-2) \cdot (-2) & 1 \cdot 1 + (-2) \cdot 6 \end{bmatrix}$$

$$\boxed{= \begin{bmatrix} -4 & 61 \\ 6 & -11 \end{bmatrix}}$$

Calculate $NM$:

$$NM = \begin{bmatrix} 2 & 1 \\ -2 & 6 \end{bmatrix} \begin{bmatrix} 7 & 9 \\ 1 & -2 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \cdot 7 + 1 \cdot 1 & 2 \cdot 9 + 1 \cdot (-2) \\ -2 \cdot 7 + 6 \cdot 1 & -2 \cdot 9 + 6 \cdot (-2) \end{bmatrix}$$

$$\boxed{= \begin{bmatrix} 15 & 16 \\ -8 & -30 \end{bmatrix}}$$

## A.9 Q9

If $y = x^4 - 4x^3 - 90x^2$, find the values of $x$ for which $y'' = 0$

First, find the first derivative:

$$y' = \frac{d}{dx}(x^4 - 4x^3 - 90x^2) = 4x^3 - 12x^2 - 180x$$

Find the second derivative:

$$y'' = \frac{d}{dx}(4x^3 - 12x^2 - 180x) = 12x^2 - 24x - 180$$

Set $y'' = 0$:

$$12x^2 - 24x - 180 = 0$$

Divide by 12:

$$x^2 - 2x - 15 = 0$$

Factor:

$$(x - 5)(x + 3) = 0$$

$$\boxed{x = 5,\ x = -3}$$

## A.10 Q10

Transpose $b = g + t(a - 3)$ to make $a$ the subject

Starting with:

$$b = g + t(a - 3)$$

this is easily rearranged to isolate $a$:

$$b - g = t(a - 3)$$

$$\frac{b - g}{t} = a - 3$$

$$\boxed{a = \frac{b - g}{t} + 3}$$

# Part B

# Computing

**Basis**

Part B should include commented MATLAB code and screenshots of the command window and graphs. All graph formatting must be done by code.
All programmes should have user-friendly input and formatted output.

## Notes:

- All graphs and images within the question boxes are created using TikZ PGF by yours truly.
- Code implementations are original and written primarily in MATLAB to adhere to guidelines, with some sections in Python.
- To develop a robust, universal, and user-friendly code for this section, it is crucial to consider both usability and accessibility across a broad spectrum of potential users. Whether users are seasoned developers or novices, the approach taken should cater to their needs in an intuitive and efficient manner. Two primary strategies can be employed to achieve this:

  1. **Approach 1: Enhancing User Experience through Clear and Visual Representation of Mathematical Solutions** When solving complex mathematical problems using software such as MATLAB, presenting the results in a clear and visually engaging format is crucial for enhancing the user's comprehension and interaction with the solution. Whether dealing with calculus, linear algebra, vector analysis, geometry, trigonometry, or statistical data, effective output presentation is key. The following strategies can be used to achieve this:
     - *Graphical Representation of Mathematical Solutions*: Solutions involving differentiation, integration, matrix operations, or vector analysis can often be more intuitively understood when visualized. Plotting graphs of functions (e.g., derivatives, integrals), 3D surfaces, vector fields, and matrix visualizations helps users grasp complex concepts quickly. MATLAB's built-in plotting functions such as `plot`, `surf`, and `quiver` can be utilized to generate these visualizations.
     - *Interactive Graphical Interfaces for Mathematical Exploration*: Creating interactive UI elements, such as sliders for parameter adjustment, allows users to dynamically explore how changes in inputs affect the solution. For example, users can visualize the effects of altering coefficients in a system of differential equations or observe real-time changes in vector fields with interactive 3D plots.
     - *Formatted Output of Analytical Results*: For problems involving algebraic solutions (e.g., symbolic differentiation, integration, matrix factorizations), presenting the results in a neatly formatted output, using LaTeX-style expressions or equation editors, ensures clarity. MATLAB's `live scripts` and symbolic math toolbox can display these outputs in a way that is both readable and professional.

  - *Data Analysis and Statistical Visualization*: When evaluating statistical data, probability distributions, or regression models, it's essential to represent results in charts such as histograms, scatter plots, box plots, or pie charts. Visual tools for analyzing statistical variance, probability distributions, or hypothesis testing can make data-driven insights more accessible.

  This approach's disadvantage is that it fundamentally contradicts the second strategy and is difficult to reconcile with it.

2. **Approach 2: Providing an Interactive Programming Environment** For users more comfortable with programming, but still seeking a level of simplicity, an interactive coding environment can serve as an excellent middle ground. This environment would provide:
  - *Clear and Accessible Code Snippets*: Providing well-commented, modular code blocks that users can easily adapt and integrate into their own projects encourages experimentation without requiring deep understanding of the entire system.
  - *Interactive Command-Line Interface (CLI)*: An interactive CLI can guide users through the execution of the code by providing prompts, options, and feedback, making it easier to follow steps and understand what each part of the code accomplishes.
  - *Jupyter Notebooks or Similar Tools*: Environments like Jupyter Notebooks allow users to experiment with the code interactively, running small sections, visualizing results, and receiving immediate feedback, all while maintaining clarity and structure in the workflow.

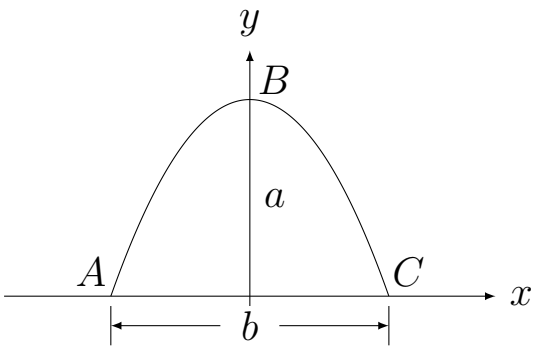
  In summary, these two approaches collectively cater to both ends of the user spectrum, ensuring that the code is accessible, user-friendly, and adaptable. For each query or task, I will provide at least one refined solution, allowing flexibility for users to select the one that best meets their individual preferences and expertise.

- Finally, please consider the answers as a whole.

# B.1 Q1

The arc length of a segment of a parabola $ABC$ of an ellipse with semi-minor axes $a$ and $b$ is given approximately by:

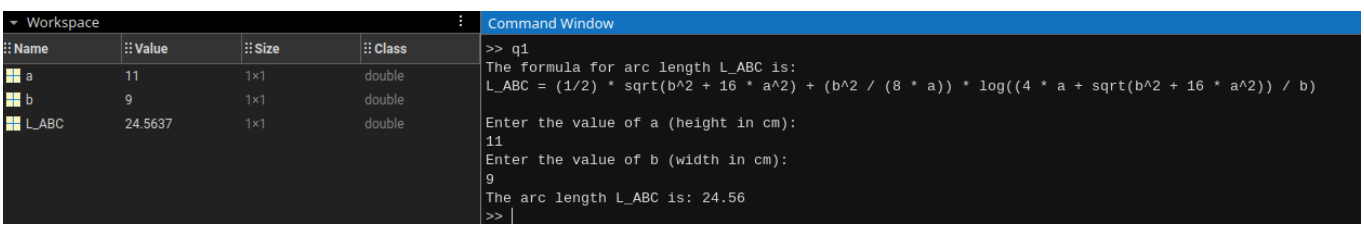
$$L_{ABC} = \frac{1}{2}\sqrt{b^2 + 16a^2} + \frac{b^2}{8a} \ln\left(\frac{4a + \sqrt{b^2 + 16a^2}}{b}\right)$$




Write a universal, user-friendly code, test your programme and determine $L_{ABC}$ if $a = 11$ cm and $b = 9$ cm.

**matlab scripts/q1.m**

```
function L_ABC = l_abc(a, b)
    L_ABC = ((1/2) * sqrt(b^2 + 16 * a^2) + (b^2 / (8 * a)) * log((4 * a + sqrt(b^2 +
        16 * a^2)) / b));
end
fprintf('The formula for arc length L_ABC is:\n');
fprintf('L_ABC = (1/2) * sqrt(b^2 + 16 * a^2) + (b^2 / (8 * a)) * log((4 * a + sqrt(b
    ^2 + 16 * a^2)) / b)\n\n');
while true
    a = input('Enter the value of a (height in cm): ');
    if a ~= 0
        while true
            b = input('Enter the value of b (width in cm): ');
            if b > 0
                break;
            else
                fprintf('b must be a greater than zero. Please try again.\n');
            end
        end
        break;
    else
        fprintf('a must be a non-zero value. Please try again.\n');
    end
end
L_ABC = l_abc(a, b);
fprintf('The arc length L_ABC is: %.2f\n', L_ABC);
```



### B.1.1 Advanced version

We are tasked with analyzing a parabola where:

- $a$ represents the height of the parabola
- $b$ represents the width of the parabola

My goal is to plot the parabola, first by formulating an equation in terms of these factors. Initially, we begin with a simple parabola in the general shape ($x^2$) similar to shown in the question. Our goal is to select a reasonable function, with variables associated with the $x$ scale and $y$ location of the parabola:

$$y = -(\beta x)^2 + \gamma$$

Given our problem statement:

- $\gamma$ represents the height, so $\gamma = a$
- The width is related to the roots of the equation when $y = 0$

To find $\beta$, we solve:

$$0 = -(\beta x)^2 + \gamma$$

$$\beta = \frac{2\sqrt{\gamma}}{b}$$

since we know $\gamma$:

$$\beta = \frac{2\sqrt{a}}{b}$$

Note: This requires $b \neq 0$ and $\sqrt{a} \neq 0$ (i.e., $a > 0$).
Substituting $\gamma$ and $\beta$ into our original equation:

$$\boxed{y = -\left(\frac{2\sqrt{a}}{b}x\right)^2 + a}$$

This is our final parabola equation in terms of $a$ and $b$.
The approximation arc length of the parabola from $x = -\frac{b}{2}$ to $x = \frac{b}{2}$ is given by:

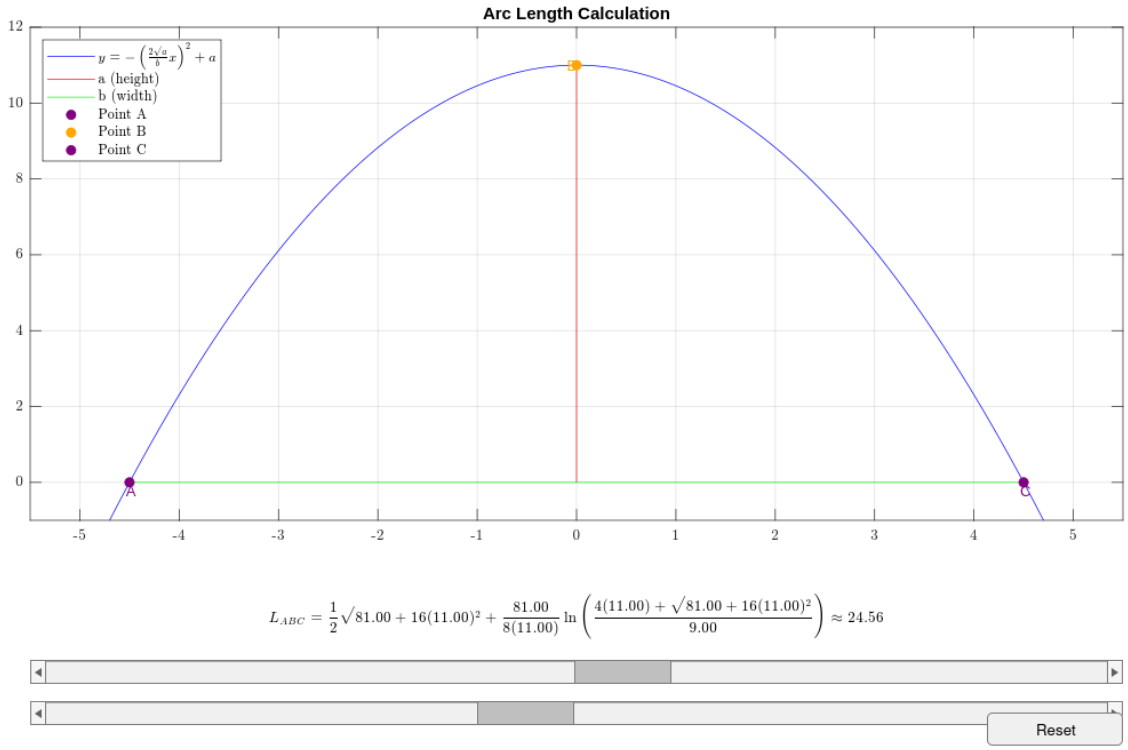
$$L_{ABC} = \frac{1}{2}\sqrt{b^2 + 16a^2} + \frac{b^2}{8a}\ln\left(\frac{4a + \sqrt{b^2 + 16a^2}}{b}\right)$$

Side note: $L_{ABC}$ is kinda close to evaluating the integral $\int \sqrt{1 + \left(\frac{\partial}{\partial x}\left(-\left(\frac{2\sqrt{a}}{b}x\right)^2 + a\right)\right)^2}\, dx$.
When implementing this in a program:

- Ensure that when there are no limit parameters for the inputs $a$ and $b$, the conditions $a > 0$ and $b \neq 0$ are appropriately handled.
- Update the plot of $y = -\left(\frac{2\sqrt{a}}{b}x\right)^2 + a$ so that the sliders for $a$ and $b$ directly control the height and width, respectively.
- Calculate the arc length of the parabola using the function given the question with inputs $a$ and $b$, and display the result in the plot.

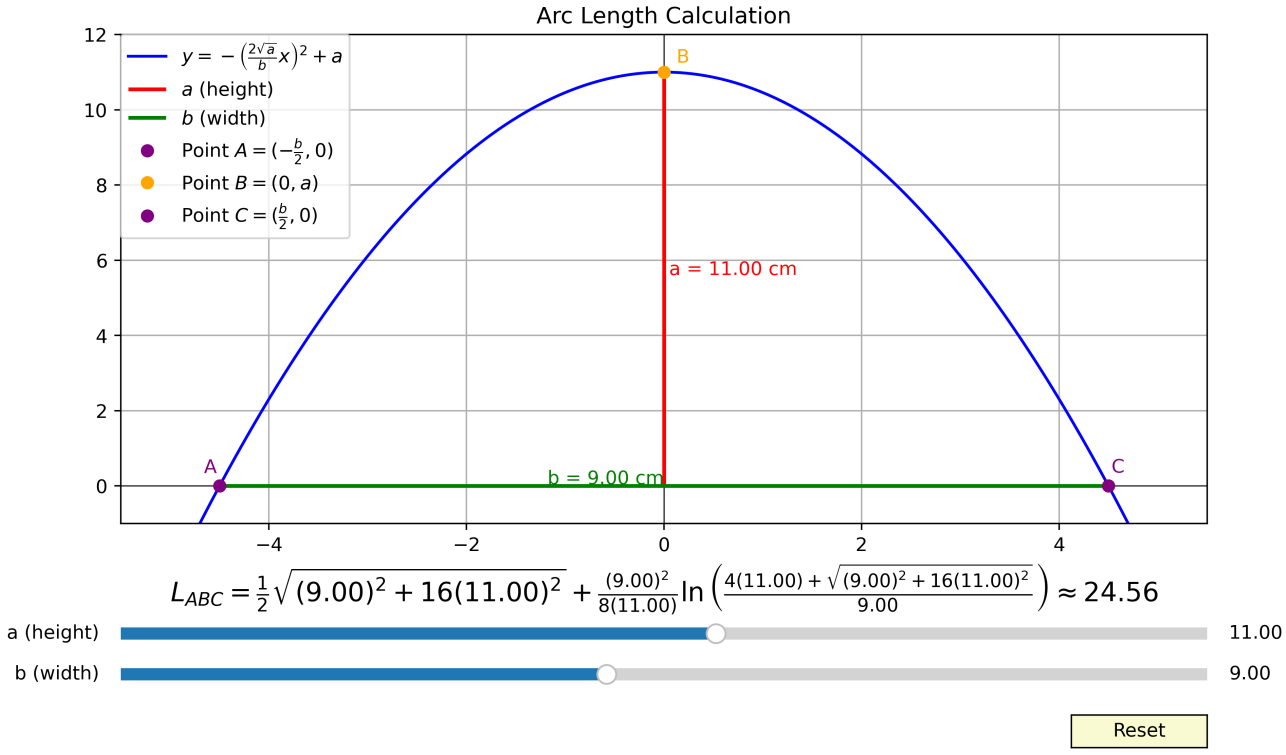
For the remaining portion, it just boils down to presentation and flavour. I tried using MATLAB to do this, but it seemed really awkward and unreliable. Although its fine `.m` version still could require development, but I find that the `.py` version functions better.

## matlab scripts/q1_advanced.m




---

## py scripts/q1_advanced.py



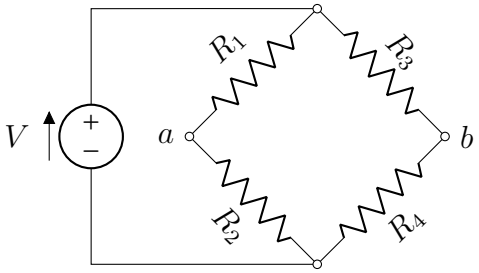
## B.2 Q2

The voltage difference $V_{ab}$ between points $a$ and $b$ in the Wheatstone bridge circuit is:

$$V_{ab} = V\left(\frac{R_1R_3 - R_2R_4}{(R_1 + R_2)(R_3 + R_4)}\right)$$

Write a universal, user-friendly program that calculates the voltage difference $V_{ab}$.
Test your program using the following values:

- $V$ = 14 volts
- $R_1$ = 120.6 Ω
- $R_2$ = 119.3 Ω
- $R_3$ = 121.2 Ω
- $R_4$ = 118.8 Ω

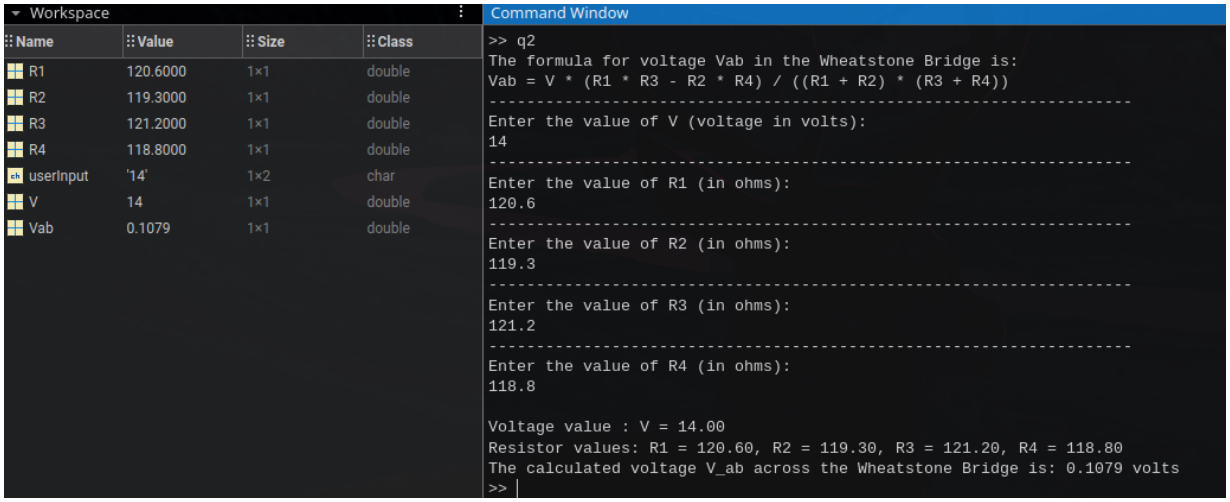



**matlab scripts/q2.m**

```
function Vab = calculate_Wheatstone(V, R1, R2, R3, R4)
    Vab = V * (R1 * R3 - R2 * R4) / ((R1 + R2) * (R3 + R4));
end



fprintf('The formula for voltage Vab in the Wheatstone Bridge is:\n');
fprintf('Vab = V * (R1 * R3 - R2 * R4) / ((R1 + R2) * (R3 + R4))\n');
disp('----------------------------------------------------------------------');


function confirm = confirmValue(type)
    while true
        sure = upper(input(sprintf('Are you sure a %s is what you want? (Y/N): ',
           type), 's'));
        if strcmp(sure, 'Y')
            confirm = true;
            return;
        elseif strcmp(sure, 'N')
            fprintf('Returning to previous input...\n');
            confirm = false;
            return;
        else
            fprintf('Invalid input. Please enter either Y or N.\n');
        end
    end
end


while true
    userInput = input('Enter the value of V (voltage in volts): ', 's');
    V = str2double(userInput);
    if isnan(V)
        fprintf('ERROR: Input must be a numeric value.\n');
        continue;
    end
    if V>0
        break;
    elseif V==0
```

```
        fprintf('ISSUE: V cannot equal 0\n')
    else
        fprintf('ISSUE: A negative voltage isn''t realistic.\n');
        if confirmValue('negative value')
            break;
        end
    end
end


function [R1, R2, R3, R4] = promptForResistors()
    function R = getResistor(name, comparisonValue)
        disp('----------------------------------------------------------------------');
        while true
            userInput = input(sprintf('Enter the value of %s (in ohms): ', name), 's'
                );

            R = str2double(userInput);
            if isnan(R)
                fprintf('ERROR: Input must be a numeric value.\n');
                continue;
            end

            issues = cell(0);

            if R < 0
                issues{end+1} = 'ISSUE: A negative resistor isn''t realistic.';
            end
            if R == 0
                issues{end+1} = 'ISSUE: A resistor of zero isn''t realistic.';
            end
            if nargin > 1 && R == -comparisonValue
                issues{end+1} = sprintf('ISSUE: %s cannot be equal and opposite to %s
                    .', name, inputname(2));
                if R < 0 || R==0
                    issues = issues(end);
                end
            end

            if ~isempty(issues)
                fprintf('%s\n', strjoin(issues, '\n'));
                if any(contains(issues, 'cannot be equal and opposite to'))
                    fprintf('Please enter a different value.\n');
                    continue;
                elseif any(contains(issues, 'zero'))
                    if ~confirmValue('value of zero')
                        continue;
                    end
                elseif ~confirmValue('negative value')
                   continue;
                end
            end
            break;

        end
    end
    R1 = getResistor('R1');
    R2 = getResistor('R2', R1);
    R3 = getResistor('R3');
    R4 = getResistor('R4', R3);
end

```

```
99  [R1, R2, R3, R4] = promptForResistors();
100 fprintf('\nVoltage value : V = %.2f\n', V);
101 fprintf('Resistor values: R1 = %.2f, R2 = %.2f, R3 = %.2f, R4 = %.2f\n', R1, R2, R3,
        R4);
102 Vab = calculate_Wheatstone(V, R1, R2, R3, R4);
103 fprintf('The calculated voltage V_ab across the Wheatstone Bridge is: %.2f volts\n',
        Vab);
```

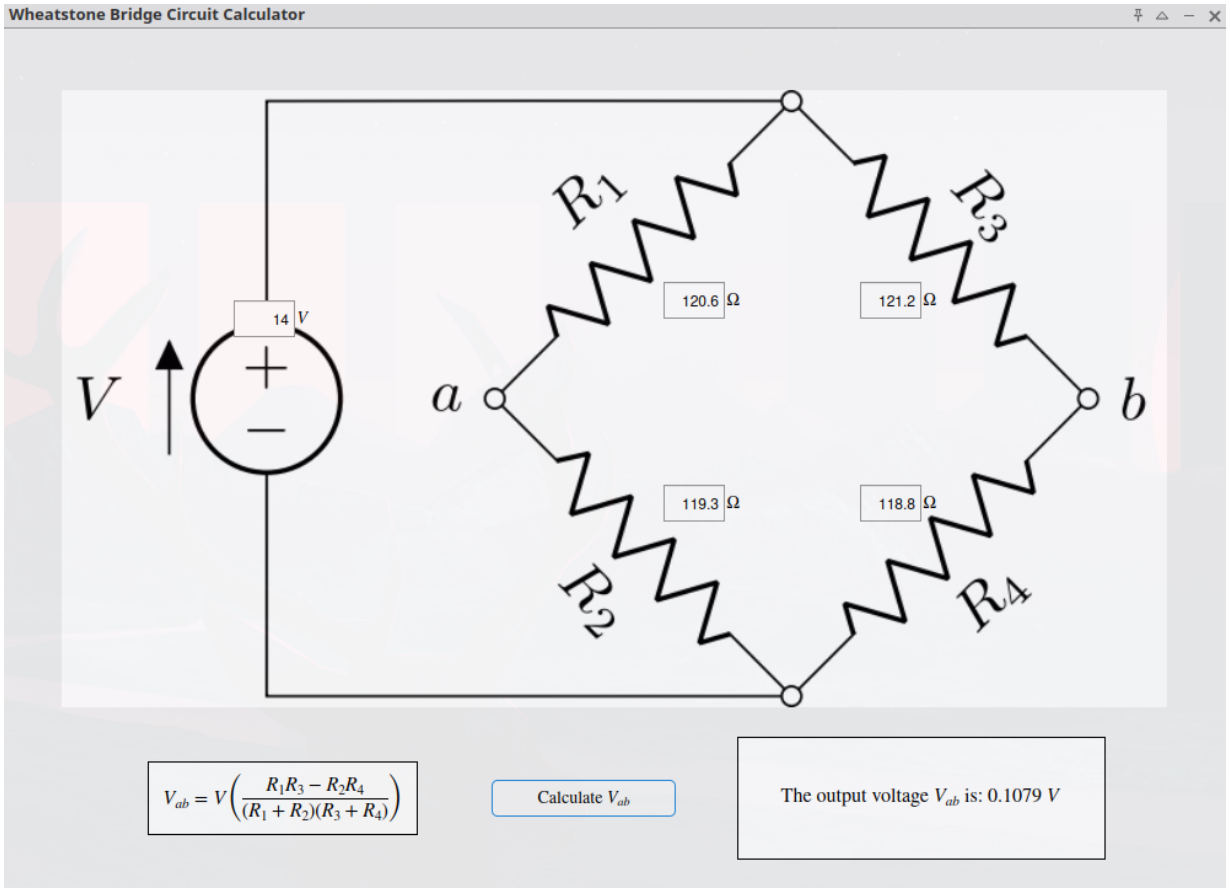


### B.2.1 Somewhat Advanced Version

I say "somewhat" because this isn't all that advanced. One way to improve this project is by incorporate real-time elements, such as allowing users to adjust the resistors while the voltage is still flowing, making it interactive. Ultimately, it's all about how creative you want to be with this.

For now, I have developed a nice UI interface in MATLAB, which should suffice. If I have enough time and motivation, I may revisit this and create some animations.

**matlab scripts/q2_advanced.m**



## B.3 Q3

Newton's law of cooling gives the temperature $T(t)$ of an object at time $t$ in terms of $T_0$, its temperature at $t = 0$, and $T_s$, the temperature of the surroundings:

$$T(t) = T_s + (T_0 - T_s)e^{-kt}$$

A police officer arrives at a crime scene in a hotel room at 9:18 PM, where he finds a dead body. He immediately measures the body's temperature and finds it to be $26.4°C$. Exactly one hour later, he measures the temperature again and finds it to be $25.5°C$.

Determine the time of death, assuming that the victim's body temperature was normal ($36.6°C$) prior to death, and that the room temperature was constant at $20.5°C$.

To solve this problem, I need to fully understand it first before actually importing computing aspects. First, I will use the formula given along with the given information to find the time of death.

Let's break it down step by step:

1. **Finding the cooling constant** $k$:

- At 9:18 PM ($t = 0$): $T_0 = 26.4°C$
- At 10:18 PM ($t = 1$ hour): $T(1) = 25.5°C$
- Room temperature ($T_s$) $= 20.5°C$

Using the equation:

$$T(t) = T_s + (T_0 - T_s)e^{-kt}$$

Rearrange for $k$:

$$k = -\frac{1}{t}\ln\left(\frac{T(t) - T_s}{T_0 - T_s}\right)$$

Plugging in the values:

$$k = -\frac{1}{1}\ln\left(\frac{25.5 - 20.5}{26.4 - 20.5}\right) \approx 0.1656$$

2. **Finding the time when** $T_0 = 36.6°C$:
Now that we have $k$, we can use the original equation to find $t$, rearrange the equation for $t$:

$$t = -\frac{1}{k}\ln\left(\frac{T(t) - T_s}{T_0 - T_s}\right)$$

Plugging in values and solving for $t$:

$$t = -\frac{1}{0.1656}\ln\left(\frac{25.5 - 20.5}{36.6 - 20.5}\right) \approx 7.06513$$

3. **Calculating the time of death**:
This means the body had been cooling for about 7.07 hours when the officer arrived at $9:18$ PM. Converting decimal hours into hours and minutes:

$$7 \text{ hr} + 0.07 \times 60 \text{ min} = 7 \text{ hr} + 4.2 \text{ min} \approx 7 \text{ hr} + 4 \text{ min}$$

To find the time of death, we subtract 7 hr and 4 min from $9:18$ PM:

$$9:18 \text{ PM} - 7 \text{ hr}4 \text{ min} \approx 2:14 \text{ PM}$$

Therefore, the estimated time of death is around **2:14 PM** on the same day.

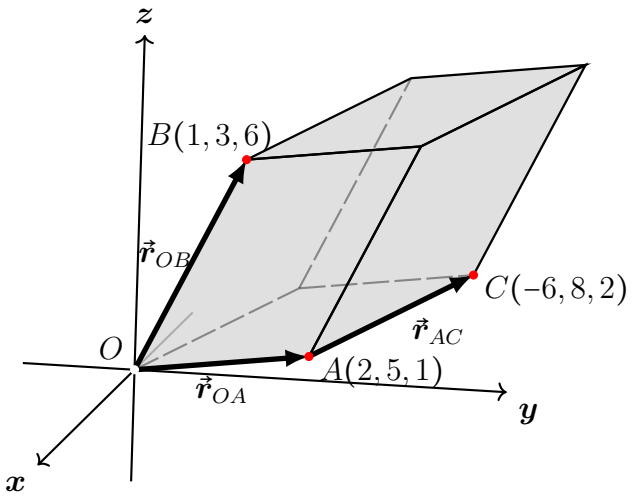
**matlab scripts/q3.m**

```
function k = calculateCoolingConstant(T_1, T_s, T_0, t1)
    ratio = (T_1 - T_s) / (T_0 - T_s);
    k = -log(ratio) / t1;
    fprintf('Cooling constant k = %.4f per hour\n', k);
end

function t_death = calculateTimeSinceDeath(T_1,T_s,T_normal,k)
    ratio = (T_1 - T_s) / (T_normal - T_s);
    t_death = -log(ratio) / k;
    fprintf('Time since death = %.2f hours\n', t_death);
end

T_normal = input('Enter the normal body temperature in C (usually 36.6C): ');
T_s = input('Enter the surrounding temperature (T_s) in C (roomytemp is avg 20.5C): '
    );
T_0 = input('You have seen the body, enter the body temperature (T_0) in C: ');
time_arrival = input('Enter the time of body discovery (in decimal hours, e.g., 21.3
    for 9:18 PM): ');
T_1 = input('Enter the temperature some set time after the discovery (T_1) in C: ');
t1 = input('Enter the time interval (in hours) between T_0 and T_1: ');
disp('------------------------------------------------------------');
disp('Newton''s Law of Cooling formula:');
disp('T(t) = T_s + (T_0 - T_s) * exp(-k * t)');
disp('------------------------------------------------------------');
fprintf('Initial body temperature (T_0): %.1fC\n', T_0);
fprintf('Observed temperature (T_1) after %.1f hour(s): %.1fC\n', t1, T_1);
fprintf('Surrounding temperature (T_s): %.1fC\n', T_s);
disp('------------------------------------------------------------');
k = calculateCoolingConstant(T_1, T_s, T_0, t1);
t_death = calculateTimeSinceDeath(T_1,T_s,T_normal, k);
disp('------------------------------------------------------------');
time_of_death = time_arrival - t_death;
hours = floor(time_of_death);
minutes = (time_of_death - hours) * 60;
fprintf('Estimated time of death: %02d:%02d\n', mod(hours, 24), round(minutes));
```

## B.4 Q4

The volume of the parallelepiped shown can be calculated by $\vec{r}_{OB} \cdot (\vec{r}_{OA} \times \vec{r}_{AC})$.
Use the following steps in a script file to calculate the area:

1. Define the vectors $\vec{r}_{OA}$, $\vec{r}_{OB}$, and $\vec{r}_{AC}$ from inputted positions of points $A$, $B$, and $C$.
2. Determine the volume by using MATLAB's built-in functions `dot` and `cross`.




**matlab scripts/q4.m**

```
fprintf('V = OB * (OA x AC)\n')
disp('------------------------------')

% Prompt user for inputs
A = input('Enter point A (in Cartesian coordinates, e.g., [1 2 3]): ');
B = input('Enter point B (in Cartesian coordinates, e.g., [4 5 6]): ');
C = input('Enter point C (in Cartesian coordinates, e.g., [7 8 9]): ');

% Calculate vectors OA, OB, and AC
OA = A;
OB = B;
AC = C-A;

% Calculate the cross product and dot product
OAcrossAC = cross(OA, AC);
OBdotOAcrossAC = dot(OB, OAcrossAC);

% Display intermediate results and final value
disp('------------------------------')
fprintf('Cross product (OA x AC) = [%d %d %d]\n', OAcrossAC);
fprintf('Dot product OB * (OA x AC) = %d\n', OBdotOAcrossAC);
fprintf('V = %d\n', OBdotOAcrossAC);
```

## B.5 Q5

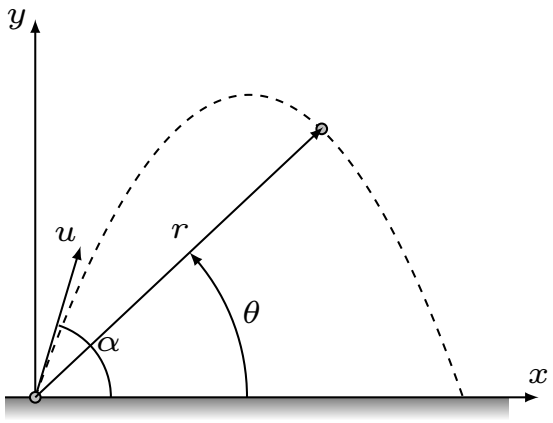
The position as a function of time $(x(t), y(t))$ of a projectile fired with an initial speed $v_0$ at an angle $\alpha$ is given by:

$$x(t) = v_0 \cos\alpha \cdot t \qquad y(t) = v_0 \sin\alpha \cdot t - \frac{1}{2}gt^2$$

where $g = 9.81\,\mathrm{m/s^2}$ is the acceleration due to gravity.
The polar coordinates of the projectile at time $t$ are $(r(t), \theta(t))$, where

$$r(t) = \sqrt{x(t)^2 + y(t)^2} \qquad \tan\theta(t) = \frac{y(t)}{x(t)}$$

Write a universal, user-friendly code. Test case: $v_0 = 162$ m/sec and $\alpha = 70°$.
Determine $r(t)$ and $\theta(t)$ for $t = 1, 6, \ldots, 31$ sec.




| Parameter | $x$ | $y$ |
|---|---|---|
| Acceleration / $a$ | $a_x = 0$ | $a_y = g = 9.81\,\mathrm{m/s^2}$ |
| Initial Velocity / $u$ / $v(0)$ | $u_x = u\cos(\theta)$ | $u_y = u\sin(\theta)$ |
| Velocity / $v(t)$ | $v_x(t) = u_x$ | $v_y(t) = u_y - gt$ |
| Displacement / $s(t)$ | $s_x(t) = u_x t$ | $s_y(t) = u_y t - \frac{1}{2}gt^2 + h$ |
| Velocity at Separation / $v(T)$ | $v_x(T) = u_x$ | $v_y(T) = \sqrt{u_y^2 + 2gh}$ |

Table B.1: Typical Projectile Motion stuff

| | |
|---|---|
| Time of Flight / $T$ | $\frac{1}{g}\left(u_y + \sqrt{u_y^2 + 2gh}\right)$ |
| Range / $R$ / $s_x(T)$ | $\frac{u_x}{g}\left(u_y + \sqrt{u_y^2 + 2gh}\right)$ |
| Maximum Height / $H$ | $\frac{1}{2g}\left(u_y^2\right) + h$ |

Table B.2: Other Derivations

| | |
|---|---|
| Radial Position / $r(t)$ | $\sqrt{(s_x(t))^2 + (s_y(t))^2}$ |
| Angular Position / $\theta(t)$ | $\tan^{-1}\left(\frac{s_y(t)}{s_x(t)}\right)$ |
| Angular Velocity / $\omega(t)$ | $\frac{d}{dt}(\theta(t))$ |
| Angular Acceleration / $\alpha(t)$ | $\frac{d}{dt}(\omega(t))$ |

Table B.3: Circular Motion Parameters

| Energy Component | $x$ | $y$ |
|---|---|---|
| Kinetic Energy (KE) | $\frac{1}{2}mu_x^2$ | $\frac{1}{2}mu_y^2$ |
| Potential Energy (PE) | 0 | $mgh$ |
| Total Energy (TE) | $\frac{1}{2}mu_x^2$ | $\frac{1}{2}mu_y^2 + mgh$ |
| Lagrangian / $\mathscr{L}$ | $\frac{1}{2}mu_x^2$ | $\frac{1}{2}mu_y^2 - mgh$ |
| Power (P) | $ma_xu_x = 0$ | $ma_yu_y = mgu_y$ |
| Work Done (W) | 0 | $mgh$ |

Table B.4: Typical Energy Components in Projectile Motion (Non-Drag)

| Energy Component | $x$ | $y$ |
|---|---|---|
| Drag Force / $F_d$ | $F_{d_x} = \frac{1}{2}\rho C_d A v_x^2$ | $F_{d_y} = \frac{1}{2}\rho C_d A v_y^2$ |
| Energy Loss due to Drag / $E_d$ | $\int F_{d_x}\,dx$ | $\int F_{d_y}\,dy$ |
| Total Energy with Drag / $TE_d$ | $\frac{1}{2}mu_x^2 - \int F_{d_x}\,dx$ | $\frac{1}{2}mu_y^2 + mgh - \int F_{d_y}\,dy$ |
| Total Lagrangian with Drag / $\mathscr{L}_d$ | $\frac{1}{2}mu_x^2 - \int F_{d_x}\,dx$ | $\frac{1}{2}mu_y^2 - mgh - \int F_{d_y}\,dy$ |

Table B.5: Energy Components with Drag in Projectile Motion

**matlab scripts/q5.m**

```
% Constants
g = 9.81; % Acceleration due to gravity (m/s^2)
% User Inputs
u = input('Enter the initial velocity (u) in m/s: ');
theta = input('Enter the launch angle (theta) in degrees: ');
h = input('Enter the initial height (h) in meters: ');
% Convert angle from degrees to radians
theta_rad = deg2rad(theta);
% Calculate initial velocities in x and y directions
u_x = u * cos(theta_rad);
u_y = u * sin(theta_rad);
% Calculate Time of Flight (T)
T = (u_y + sqrt(u_y^2 + 2 * g * h)) / g;
% Calculate Range (R)
R = (u_x / g) * (u_y + sqrt(u_y^2 + 2 * g * h));
% Calculate maximum height (H)
H = (u_y^2 / (2 * g)) + h;
```

```
% Time vector for plotting
t = linspace(0, T, 100);
% Calculate displacements in x and y directions
s_x = u_x * t; % Displacement in x
s_y = (u_y * t) - (0.5 * g * t.^2) + h; % Displacement in y
% Calculate radial distance r(t) and angle theta(t)
r = sqrt(s_x.^2 + s_y.^2);
theta_t = atan2(s_y, s_x); % Angle in radians
% Find the maximum r(t)
max_r = max(r);
max_r_index = find(r == max_r, 1);
max_time_r = t(max_r_index);
% Find the maximum theta(t)
max_theta = max(theta_t);
max_theta_index = find(theta_t == max_theta, 1);
max_time_theta = t(max_theta_index);
% Calculate velocity at time T
v_x_T = u_x; % Constant velocity in x
v_y_T = sqrt(u_y^2 + 2 * g * h); % Velocity in y at time T
v_T = sqrt(v_x_T^2 + v_y_T^2); % Total velocity at time T
% Display results
fprintf('Time of Flight (T): %.2f seconds (time until projectile hits the ground)\n',
    T);
fprintf('Range (R): %.2f meters (horizontal distance covered)\n', R);
fprintf('Max Height (H): %.2f meters (peak vertical height above the ground)\n', H);
fprintf('Velocity at separation (v(T)): %.2f m/s (Note: This simulation assumes
   negligible air resistance.)\n', v_T);
fprintf('Maximum radial distance (r(t)): %.2f meters at t = %.2f seconds\n', max_r,
   max_time_r);

max_theta_deg = max_theta * 180 / pi;  % Convert to degrees for clarity
fprintf('Maximum angle (theta(t)): %.2f degrees at t = %.2f seconds\n', max_theta_deg
   , max_time_theta);

% Create a single figure with three subplots
figure('Position', [100, 100, 800, 600]);
% Plot the trajectory in Cartesian coordinates (top subplot)
subplot(2,1,1);
plot(s_x, s_y);
title('Projectile Motion Trajectory');
xlabel('Displacement in X (m)');
ylabel('Displacement in Y (m)');
grid on;
hold on;
plot(u_x*u_y/g, H, 'x', 'MarkerSize', 10, 'LineWidth', 2,'Color', [0.059, 0.125, 1],
   'DisplayName', 'Max Height');
legend('Trajectory', 'Max Height', 'Location', 'best');
hold off;
% Plot r(t) (bottom-left subplot)
subplot(2,2,3);
plot(t, r, 'Color', [0, 0.85, 0], 'DisplayName', 'r(t)'); % Darker green
hold on;
plot(max_time_r, max_r, 'x', 'MarkerSize', 10, 'LineWidth', 2, 'Color', [0, 0.85, 0],
    'DisplayName', 'Max r(t)');
xlabel('Time (s)');
ylabel('Radial Distance (m)');
title('Radial Distance r(t)');
grid on;
legend('Location', 'best');
hold off;
% Plot theta(t) (bottom-right subplot)
subplot(2,2,4);
```

```
plot(t, rad2deg(theta_t), 'r-', 'DisplayName', 'theta(t)');
hold on;
plot(max_time_theta, rad2deg(max_theta), 'x', 'MarkerSize', 10, 'LineWidth', 2,'Color
    ', 'r', 'DisplayName', 'Max theta(t)');
xlabel('Time (s)');
ylabel('Angle theta(t) (degrees)');
title('Angle theta(t)');
grid on;
legend('Location', 'best');
hold off;
% Adjust the layout
sgtitle('Projectile Motion Analysis');
```

## B.6 Q6

The ideal gas equation states that

$$P = \frac{nRT}{V}$$

where $P$ is the pressure, $V$ is the volume, $T$ is the temperature, $R = 0.08206\,(\text{L atm})/(\text{mol K})$ is the gas constant, and $n$ is the number of moles. Real gases, especially at high pressure, deviate from this behavior. Their response can be modeled with the van der Waals equation

$$P = \frac{nRT}{V - nb} - \frac{n^2 a}{V^2}$$

where $a$ and $b$ are material constants. Consider 1 mole ($n = 1$) of nitrogen gas at $T = 300\,\text{K}$. (For nitrogen gas $a = 1.39\,(\text{L}^2\ \text{atm})/\text{mol}^2$ and $b = 0.0391\,\text{L/mol}$.) Create a vector with valu3es of $V$ for $0.1 \le V \le 1\,\text{L}$, using increments of $0.02\,\text{L}$. Using this vector, calculate $P$ twice for each value of $V$: once using the ideal gas equation and once with the van der Waals equation. Using the two sets of values for $P$, calculate the percent error

$$\left(\frac{P_{\text{ideal}} - P_{\text{waals}}}{P_{\text{waals}}}\right) \times 100$$

for each value of $V$. Finally, by using MATLAB's built-in function `max`, determine the maximum error and the corresponding volume.

**matlab scripts/q6.m**

```
function P1 = idealgas(n, R, T, V)
    P1 = n * R * T / V;
end

function P2 = van(n, R, T, V, a, b)
    P2 = n * R * T / (V - n * b) - n^2 * a / V^2;
end

function error_percent = percent_error(P1, P2)
    error_percent = abs((P2 - P1) / P2) * 100;
end

% Nitrogen properties
a = 1.39;
b = 0.0391;
n = 1;
T = 300;
R = 0.08206;

% Volume range
Vol = 0.1:0.02:1;
P1 = zeros(size(Vol));
P2 = zeros(size(Vol));
error_vals = zeros(size(Vol));

for i = 1:length(Vol)
    V = Vol(i);
    P1(i) = idealgas(n, R, T, V);
    P2(i) = van(n, R, T, V, a, b);
    error_vals(i) = percent_error(P1(i), P2(i));
```

```
31
32      fprintf('Vol: %.2f [P1: %.4f, P2: %.4f, Error: %.2f%%]\n', Vol(i), P1(i), P2(i),
            error_vals(i));
33  end
34
35  [max_error, idx] = max(error_vals);
36  V_max_error = Vol(idx);
37  max_P1 = max(P1);
38  max_P2 = max(P2);
39  max_P = max(max_P1, max_P2);
40  min_V = min(V);
41
42  fprintf('Maximum Error: %.2f%% at Volume: %.2f\n', max_error, V_max_error);
43
44  figure;
45  for i = 1:length(Vol)
46      ax = gca;
47       if min_V < Vol(i)
48          xlim([min_V, Vol(i)]);
49      end
50      current_max_pressure = max(max(P1(1:i)), max(P2(1:i)));
51      ylim([0, current_max_pressure * 1.1]);
52
53      plot(Vol(1:i), P1(1:i),'color','b', 'DisplayName', 'Ideal Gas Law');
54      plot(Vol(1:i), P2(1:i),'color','r', 'DisplayName', 'Van der Waals');
55      hold on
56      plot(Vol(1:i), P1(1:i), 'x','color','b', 'Markersize', 2, 'LineWidth',1);
57      plot(Vol(1:i), P2(1:i), 'x','color','r','Markersize', 2,'LineWidth',1);
58
59      x_lim = xlim;
60      y_lim = ylim;
61
62      x_norm = (Vol(i) - x_lim(1)) / (x_lim(2) - x_lim(1));
63      y_norm_P1 = (P1(i) - y_lim(1)) / (y_lim(2) - y_lim(1));
64      y_norm_P2 = (P2(i) - y_lim(1)) / (y_lim(2) - y_lim(1));
65
66      annotation('arrow', ax.Position([1 3]), ax.Position([2 4]), ...
67          'Position', [ax.Position(1) ax.Position(2) x_norm*ax.Position(3) y_norm_P1*ax
                .Position(4)], ...
68          'Color', 'b', 'LineWidth', 1.5, 'HeadLength', 10, 'HeadWidth', 5);
69
70      annotation('arrow', ax.Position([1 3]), ax.Position([2 4]), ...
71          'Position', [ax.Position(1) ax.Position(2) x_norm*ax.Position(3) y_norm_P2*ax
                .Position(4)], ...
72          'Color', 'r', 'LineWidth', 1.5, 'HeadLength', 10, 'HeadWidth', 5);
73
74      hold on;
75
76      xlabel('Volume (V)');
77      ylabel('Pressure (Pa)');
78      title('Pressure vs Volume for Ideal Gas and Van der Waals Gas');
79      grid on;
80
81      legend('Ideal Gas Law', 'Van der Waals');
82
83
84      if i > 1
85          delta_P1 = abs(P1(i) - P1(i-1));
86          delta_P2 = abs(P2(i) - P2(i-1));
87          delta_P = max(delta_P1, delta_P2);
88          pause_duration = delta_P/30;
89      else
```

```
        pause_duration = 0.25;
    end
    pause(pause_duration);

    if i < length(Vol)
        clf;
        xlabel('Volume (V)');
        ylabel('Pressure (Pa)');
        title('Pressure vs Volume for Ideal Gas and Van der Waals Gas');
        grid on;
        hold on;
    end
end

hold off;
```

## B.7 Q7

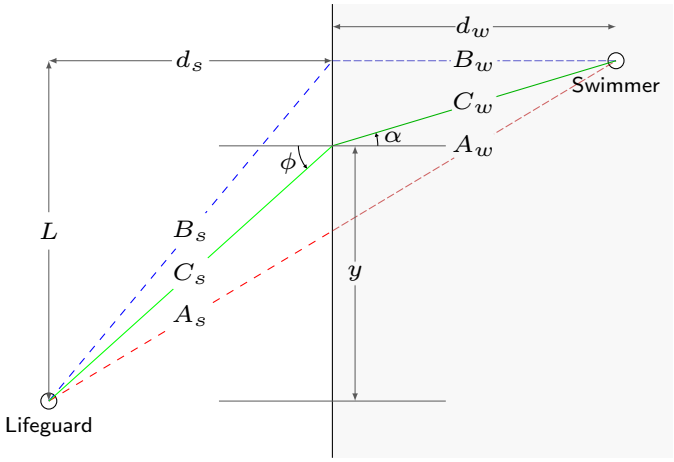
A student has a summer job as a lifeguard at the beach.

After spotting a swimmer in trouble, they try to determine the path that will allow them to reach the swimmer in the shortest time.

With the lifeguard being able to run faster than they can swim, the path of shortest distance, **Path A**, is obviously not ideal because it maximises the time spent swimming $A_w$.

Although **Path B** reduces the amount of time spent swimming $B_w$, it is probably not the best option because it is the longest feasible path.

Therefore, somewhere in between Paths A and B is the best course of action.




Consider the intermediate **path C** consisting of $C_s$ and $C_w$ and determine the time required to reach the swimmer in terms of the following:

- Running speed: $v_{\text{run}} = 3\,\text{m/s}$
- Swimming speed: $v_{\text{swim}} = 1\,\text{m/s}$
- Distance $L = 48\,\text{m}$
- Distance $d_s = 30\,\text{m}$
- Distance $d_w = 42\,\text{m}$
- Lateral distance $y$ at which the lifeguard enters the water.

Create a vector $y$ that ranges between path A and path B ($y = 20, 21, 22, \ldots, 48\,\text{m}$) and compute a time $t$ for each $y$.

Use MATLAB's built-in function `min` to find the minimum time and the entry point $y$ for which it occurs.

Determine the angles that correspond to the calculated value of $y$ and investigate whether your result satisfies Snell's law of refraction:

$$\frac{\sin\phi}{\sin\alpha} = \frac{v_{\text{run}}}{v_{\text{swim}}}$$

The total time $t$ to reach the swimmer consists of two components: the time spent running along the shore and the time spent swimming to the swimmer.

**1. Running Time** The running distance $C_s$ in the ideal path $C$ is determined by the vertical distance $y$ and the horizontal distance $d_s$. Thus, the distance the lifeguard runs to the shore is given by:

$$C_s = \sqrt{y^2 + d_s^2}$$

Knowing the running distance $C_s$, we can express the running time in terms of velocity and time:

$$t_{\text{run}} = \frac{C_s}{v_{\text{run}}} = \frac{\sqrt{y^2 + d_s^2}}{v_{\text{run}}}$$

**2. Swimming Time** After reaching point $y$, the lifeguard swims directly to the swimmer. The swimming distance $C_w$ is determined by the horizontal distance $L - y$ and the vertical distance $d_s$.

Thus, the distance the lifeguard swims is given by:

$$C_w = \sqrt{d_s^2 + (L - y)^2}$$

The swimming time $t_{\text{swim}}$ is then given by:

$$t_{\text{swim}} = \frac{C_w}{v_{\text{swim}}} = \frac{\sqrt{d_s^2 + (L - y)^2}}{v_{\text{swim}}}$$

As an alternative, we may correlate $C_w$ with another triangle that involves $d_w$ the swimmer's horizontal distance from the shore

**3. Total Time** Combining both components, the total time $t$ required to reach the swimmer is:

$$t = t_{\text{run}} + t_{\text{swim}} = \frac{\sqrt{y^2 + d_s^2}}{v_{\text{run}}} + \frac{\sqrt{d_s^2 + (L - y)^2}}{v_{\text{swim}}}$$

Substituting the known values, we have:

$$t = \frac{\sqrt{y^2 + 30^2}}{3} + \frac{\sqrt{30^2 + (48 - y)^2}}{1}$$

**Angles for verification with Snells law**

$$\frac{\sin\phi}{\sin\alpha} = \frac{v_{\text{run}}}{v_{\text{swim}}}$$

were:

$$\phi = \arctan\left(\frac{y}{30}\right)$$

$$\alpha = \arctan\left(\frac{48 - y}{30}\right)$$

Worked out through observation of triangles formed

**Finding the Optimal Path**

To find the optimal entry point $y$ that minimizes $t$, we compute the time $t$ for each possible $y$ within the range of $20 \le y \le 48$ m. The entry point $y$ that yields the minimum time will be our optimal path.

To solve this problem in code, we:

1. Define the running and swimming speeds, as well as distances along and perpendicular to the shore.
2. Use the distance $y$ as a variable representing where the lifeguard enters the water.
3. Calculate the total time $t$ required to reach the swimmer for each $y$ value.
4. Use MATLAB's `min` function to find the optimal $y$ with minimum time.
5. Verify the result with Snell's law.

**matlab scripts/q7.m**

```
% Define the running and swimming speeds, and distances
v_run = 3; % Running speed in m/s
v_swim = 1; % Swimming speed in m/s
L = 48; % Total distance along the shore in m
d_s = 30; % Distance perpendicular to the shore in m
```

```

% Define the range of y values
y_values = 0:0.0001:L; % Range of y values

% Calculate the total time for each y value
total_time = zeros(size(y_values)); % Initialize the total time array

for i = 1:length(y_values)
    y = y_values(i);
    C_s = sqrt(y^2+d_s^2);
    C_w = sqrt(d_s^2+(L-y)^2);
    t_run = C_s/v_run;
    t_swim = C_w/v_swim;
    total_time(i) = t_run+t_swim;
end

% Find the optimal y with minimum time
[min_time, idx] = min(total_time); % Find the minimum time and its index
optimal_y = y_values(idx); % Optimal y value

% Verify the result with Snell's Law
phi = 90 - atand(d_s/optimal_y); % Angle phi in degrees
alpha = asind((L-optimal_y)/sqrt(d_s^2+(L-optimal_y)^2)); % Angle alpha in degrees

% Check Snell's Law
snell_ratio = sind(phi) / sind(alpha);
expected_ratio = v_run / v_swim;

fprintf('Optimal y value: %.2f m\n', optimal_y);
fprintf('Minimum time: %.2f s\n', min_time);
fprintf('Phi: %.2f degrees\n', phi);
fprintf('Alpha: %.2f degrees\n', alpha);
fprintf('Snell ratio: %.6f\n', snell_ratio);
fprintf('Expected ratio: %.6f\n', expected_ratio);

if abs(snell_ratio-expected_ratio)<1e-4
    disp('Snell''s Law is satisfied.');
else
    disp('Snell''s Law is not satisfied.');
end
```

## B.8 Q8

In a typical tension test, a dog bone-shaped specimen is pulled in a machine. During the test, the force needed to pull the specimen and the length $L$ of a gauge section are measured. This data is used for plotting a stress-strain diagram of the material.

Two definitions, engineering and true, exist for stress and strain. The engineering stress $\sigma_c$ and strain $\varepsilon_c$ are defined by

$$\sigma_c = \frac{F}{A_0}, \quad \varepsilon_c = \frac{L - L_0}{L_0},$$

where $L_0$ and $A_0$ are the initial gauge length and the initial cross-sectional area of the specimen, respectively.

The true stress $\sigma_t$ and strain $\varepsilon_t$ are defined by

$$\sigma_t = \frac{FL_0}{A_0 L}, \quad \varepsilon_t = \ln\left(\frac{L}{L_0}\right).$$

The following are measurements of force and gauge length from a tension test with an aluminium specimen. The specimen has a round cross-section with a radius of 6.4 mm (before the test). The initial gauge length is $L_0 = 25$ mm.

Use the data to calculate and generate the engineering and true stress-strain curves, both on the same plot. Label the axes and use a legend to identify the curves.

**Units**: When the force is measured in newtons (N) and the area is calculated in $m^2$, the unit of the stress is pascals (Pa).

| F, N | L, mm |
|---|---|
| 0 | 25.400 |
| 13.031 | 25.474 |
| 21.485 | 25.515 |
| 31.3963 | 25.575 |
| 34.727 | 25.615 |
| 37.119 | 25.693 |
| 37.960 | 25.752 |
| 39.550 | 25.978 |
| 40.758 | 26.419 |
| 40.986 | 26.502 |
| 41.076 | 26.600 |
| 41.255 | 26.728 |
| 41.481 | 27.130 |
| 41.564 | 27.441 |

**matlab scripts/q8.m**

```
% Input data
F = [0, 13.031, 21.485, 31.3963, 34.727, 37.119, 37.960, 39.550, 40.758, 40.986, 41.076, 41.255, 41.481, 41.564];
L = [25.400, 25.474, 25.515, 25.575, 25.615, 25.693, 25.752, 25.978, 26.419, 26.502, 26.600, 26.728, 27.130, 27.441];
% Initial parameters
L0 = 25; % mm
r = 6.4/1000; % m (convert mm to m)
A0 = pi*r^2; % m^2
% Calculate engineering stress and strain
eng_strain = (L-L0)/L0;
eng_stress = F/A0 /1e6; % Convert N/m^2 to MPa
% Calculate true stress and strain
true_strain = log(L/L0);
true_stress = F.*L0./(A0*L)/1e6; % Convert N/m^2 to MPa
% Plot the curves
figure;
plot(eng_strain, eng_stress, 'b-');
hold on;
plot(true_strain, true_stress, 'r--');
xlabel('Strain');
ylabel('Stress (MPa)');
title('Engineering vs True Stress-Strain Curves');
legend('Engineering', 'True');
grid on;
hold off;
```

## B.9 Q9

A railroad bumper is designed to slow down a rapidly moving railroad car. After a 20,000 kg railroad car traveling at 20 m/s engages the bumper, its displacement $x$ (in meters) and velocity $v$ (in m/s) as a function of time $t$ (in seconds) is given by:

$$x(t) = 4.219(e^{-1.58t} - e^{-6.32t})$$

and

$$v(t) = 26.67e^{-6.32t} - 6.67e^{-1.58t}$$

Plot the displacement and the velocity as a function of time for $0 \le t \le 4$ seconds. Fit two plots at the top of the window and a plot of both with two vertical axes underneath them. All plots must be of printing quality.

**matlab scripts/q9.m**

```
% Define time range
t = linspace(0, 4, 1000);  % Time from 0 to 4 seconds

% Define displacement and velocity functions
x = @(t) 4.219 * (exp(-1.58*t) - exp(-6.32*t));  % Displacement function
v = @(t) 26.67 * exp(-6.32*t) - 6.67 * exp(-1.58*t);  % Velocity function

% Create figure
figure('Position', [100, 100, 800, 600]);

% Plot displacement
subplot(2,2,1)
plot(t, x(t), 'LineWidth', 2)  % Displacement plot
title('Displacement vs Time')
xlabel('Time (s)')
ylabel('Displacement (m)')
grid on

% Plot velocity
subplot(2,2,2)
plot(t, v(t), 'LineWidth', 2)  % Velocity plot
title('Velocity vs Time')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
grid on

% Plot both displacement and velocity
subplot(2,1,2)
yyaxis left
plot(t, x(t), 'LineWidth', 2)  % Displacement on left y-axis
ylabel('Displacement (m)')
yyaxis right
plot(t, v(t), 'LineWidth', 2)  % Velocity on right y-axis
ylabel('Velocity (m/s)')
title('Displacement and Velocity vs Time')
xlabel('Time (s)')
grid on

% Adjust layout
set(gcf, 'Color', 'w');  % Set background color
set(findall(gcf,'-property','FontSize'),'FontSize', 12)  % Set font size
```

```
42
43 % Save figure as high-quality image
44 print('-dpng', '-r300', 'railroad_bumper_plots.png')  % Save the figure
```

## B.10 Q10

Aircraft A is flying east at 320 mi/hr, while aircraft B is flying south at 160 mi/hr. At 1:00 p.m., the aircraft are located as shown.

Obtain the expression for the distance $D$ between the aircraft as a function of time $t$. Plot $D$ versus time until $D$ reaches its minimum value. The plot must be of printing quality. Use the `roots` function to compute the time when the aircraft are first within 30 mi of each other.

**matlab scripts/q10.m**

```
% Define constants
v_A = 320; % speed of aircraft A in mi/hr
v_B = 160; % speed of aircraft B in mi/hr
x0 = 400; % initial x-distance between aircraft in mi
y0 = 300; % initial y-distance between aircraft in mi

% Define time range (in hours)
t = 0:0.01:2;

% Calculate distance as a function of time
D = sqrt((x0 - v_A*t).^2 + (y0 - v_B*t).^2);

% Find minimum distance and time of minimum
[min_D, min_index] = min(D);
t_min = t(min_index);

% Plot D versus time
figure('Position', [100, 100, 800, 600])
plot(t, D, 'LineWidth', 2)
hold on
plot(t_min, min_D, 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r')
xlabel('Time (hours)', 'FontSize', 14)
ylabel('Distance between aircraft (mi)', 'FontSize', 14)
title('Distance between Aircraft vs Time', 'FontSize', 16)
grid on
axis([0 t_min+0.5 0 max(D)+50])

% Add annotation for minimum point
text(t_min+0.05, min_D, sprintf('Minimum: (%.2f, %.2f)', t_min, min_D), ...
    'FontSize', 12, 'VerticalAlignment', 'bottom')

% Enhance plot appearance
set(gca, 'FontSize', 12)
box on

% Find time when aircraft are first within 30 mi of each other
coeff = [v_A^2 + v_B^2, -2*(v_A*x0 + v_B*y0), x0^2 + y0^2 - 30^2];
t_30 = min(roots(coeff));

% Add this information to the plot
plot(t_30, sqrt((x0 - v_A*t_30)^2 + (y0 - v_B*t_30)^2), 'gs', 'MarkerSize', 10, '
    MarkerFaceColor', 'g')
text(t_30+0.05, sqrt((x0 - v_A*t_30)^2 + (y0 - v_B*t_30)^2), ...
    sprintf('First within 30 mi: (%.2f, %.2f)', t_30, 30), ...
    'FontSize', 12, 'VerticalAlignment', 'bottom')

% Display results
fprintf('Minimum distance: %.2f mi at t = %.2f hours\n', min_D, t_min)
fprintf('Aircraft are first within 30 mi of each other at t = %.2f hours\n', t_30)
```

```
49
50 % Save the figure as a high-quality image
51 print('-dpng', '-r300', 'aircraft_distance_plot.png')
```

## B.11 Q11

A two-dimensional state of stress at a point in a loaded material in the direction defined by the $x-y$ coordinate system is defined by three components of stress $\sigma_{xx}$, $\sigma_{yy}$, and $\tau_{xy}$. The stresses at the point in the direction defined by the $x'-y'$ coordinate system are calculated by the stress transformation equations:

$$\sigma_{x'x'} = \frac{\sigma_{xx}+\sigma_{yy}}{2} + \frac{\sigma_{xx}-\sigma_{yy}}{2}\cos(2\theta) + \tau_{xy}\sin(2\theta)$$

$$\sigma_{y'y'} = \sigma_{xx}+\sigma_{yy}-\sigma_{x'x'} = \frac{\sigma_{xx}+\sigma_{yy}}{2} - \frac{\sigma_{xx}-\sigma_{yy}}{2}\cos(2\theta) - \tau_{xy}\sin(2\theta)$$

$$\tau_{x'y'} = -\frac{\sigma_{xx}-\sigma_{yy}}{2}\sin(2\theta) + \tau_{xy}\cos(2\theta)$$

where $\theta$ is the angle shown in the figure.

Write a user-defined MATLAB function that determines the stresses $\sigma_{x'x'}$, $\sigma_{y'y'}$, and $\tau_{x'y'}$ given the stresses $\sigma_{xx}$, $\sigma_{yy}$, and $\tau_{xy}$, and the angle $\theta$. For the function name and arguments, use

$$[\text{Strain}] = \text{StressTrans}(S, \theta)$$

The input argument $S$ is a vector with the values of the three stress components $\sigma_{xx}$, $\sigma_{yy}$, and $\tau_{xy}$, and the input argument $\theta$ is a scalar with the value of $\theta$. The output argument Strain is a vector with the values of the three stress components $\sigma_{x'x'}$, $\sigma_{y'y'}$, and $\tau_{x'y'}$.

**matlab scripts/q11.m**

```
function [Strain] = StressTrans(S, theta)
    sigma_xx = S(1);
    sigma_yy = S(2);
    tau_xy = S(3);

    sigma_xx_prime = (sigma_xx + sigma_yy)/2 + (sigma_xx - sigma_yy)/2 * cos(2*theta)
        + tau_xy * sin(2*theta);
    sigma_yy_prime = (sigma_xx + sigma_yy)/2 - (sigma_xx - sigma_yy)/2 * cos(2*theta)
        - tau_xy * sin(2*theta);
    tau_xy_prime = -(sigma_xx - sigma_yy)/2 * sin(2*theta) + tau_xy * cos(2*theta);

    Strain = [sigma_xx_prime, sigma_yy_prime, tau_xy_prime];
end

sigma_xx = input('Enter the stress component sigma_xx (in Pa): ');
sigma_yy = input('Enter the stress component sigma_xx_yy (in Pa): ');
tau_xy = input('Enter the shear stress component tau_xy (in Pa): ');

S = [sigma_xx, sigma_yy, tau_xy];

theta_degrees = input('Enter the angle (in degrees): ');
theta = deg2rad(theta_degrees); % Convert degrees to radians

Strain = StressTrans(S, theta);
fprintf('Strain = [%.2f, %.2f, %.2f]\n', Strain(1),Strain(2),Strain(3));
fprintf('Transformed stresses:\n');
fprintf('sigma_x''x'' = %.2f Pa\n', Strain(1));
fprintf('sigma_y''y'' = %.2f Pa\n', Strain(2));
fprintf('tau_x''y'' = %.2f Pa\n', Strain(3));
```