

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:

$$u = \ln 3x$$

$$v = 2x$$

First, find the derivatives:

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

Let:

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

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

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:

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

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

A.2 Q2

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

The angle θ 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 θ is:

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

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) general rule for composite functions is solved via chain rule 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 $t = 9$:

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

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 ω 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 ω 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 ϕ , 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 $a = 2$

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

$$\tan \phi = \frac{-8}{1} = -8 \quad \phi = \arctan(-8) \approx -82.87^\circ$$

In so plugging in gives

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

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

Were

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

$$\alpha = -\arctan(-8) \approx 82.87^\circ$$

$$\omega = 1$$

[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:

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

$$v_1 = \frac{187.6}{67} = 2.8$$

$$v_2 = \frac{40.2}{67} = 0.6$$

$$v_3 = \frac{80.4}{67} = 1.2$$

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:

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

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

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

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

Hence find a unit vector that is perpendicular to both \mathbf{a} and \mathbf{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:

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

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 :

$$\begin{aligned} MN &= \begin{bmatrix} 7 & 9 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ -2 & 6 \end{bmatrix} \\ &= \begin{bmatrix} 7 \cdot 2 + 9 \cdot (-2) & 7 \cdot 1 + 9 \cdot 6 \\ 1 \cdot 2 + (-2) \cdot (-2) & 1 \cdot 1 + (-2) \cdot 6 \end{bmatrix} \\ &= \boxed{\begin{bmatrix} -4 & 61 \\ 6 & -11 \end{bmatrix}} \end{aligned}$$

Calculate NM :

$$\begin{aligned} NM &= \begin{bmatrix} 2 & 1 \\ -2 & 6 \end{bmatrix} \begin{bmatrix} 7 & 9 \\ 1 & -2 \end{bmatrix} \\ &= \begin{bmatrix} 2 \cdot 7 + 1 \cdot 1 & 2 \cdot 9 + 1 \cdot (-2) \\ -2 \cdot 7 + 6 \cdot 1 & -2 \cdot 9 + 6 \cdot (-2) \end{bmatrix} \\ &= \boxed{\begin{bmatrix} 15 & 16 \\ -8 & -30 \end{bmatrix}} \end{aligned}$$

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:

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

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

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 :

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

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

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

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.

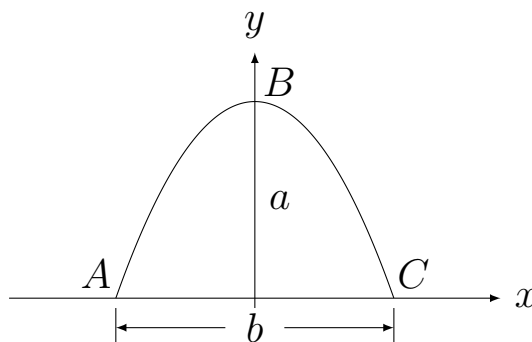
In summary, these two approaches collectively cater to both ends of the user spectrum, ensuring that the code is accessible, user-friendly, and adaptable. For each query or task, I will provide at least one refined solution, allowing flexibility for users to select the one that best meets their individual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

```

1 function L_ABC = l_abc(a, b)
2     L_ABC = ((1/2) * sqrt(b^2 + 16 * a^2) + (b^2 / (8 * a)) * log((4 * a + sqrt(b^2 +
3         16 * a^2)) / b));
4 end
5 fprintf('The formula for arc length L_ABC is:\n');
6 fprintf('L_ABC = (1/2) * sqrt(b^2 + 16 * a^2) + (b^2 / (8 * a)) * log((4 * a + sqrt(b
7     ^2 + 16 * a^2)) / b)\n\n');
8 while true
9     a = input('Enter the value of a (height in cm): ');
10    if a ~= 0
11        while true
12            b = input('Enter the value of b (width in cm): ');
13            if b > 0
14                break;
15            else
16                fprintf('b must be a greater than zero. Please try again.\n');
17            end
18        end
19    else
20        fprintf('a must be a non-zero value. Please try again.\n');
21    end
22 end
23 L_ABC = l_abc(a, b);
24 fprintf('The arc length L_ABC is: %.2f\n', L_ABC);

```

Workspace				Command Window
Name	Value	Size	Class	
a	11	1×1	double	>> q1
b	9	1×1	double	The formula for arc length L_ABC is:
L_ABC	24.5637	1×1	double	L_ABC = (1/2) * sqrt(b^2 + 16 * a^2) + (b^2 / (8 * a)) * log((4 * a + sqrt(b^2 + 16 * a^2)) / b)
				Enter the value of a (height in cm):
				11
				Enter the value of b (width in cm):
				9
				The arc length L_ABC is: 24.56
				>>

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:

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

To find β , we solve:

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

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

since we know γ :

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

Note: This requires $b \neq 0$ and $\sqrt{a} \neq 0$ (i.e., $a > 0$).

Substituting γ and β into our original equation:

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

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:

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

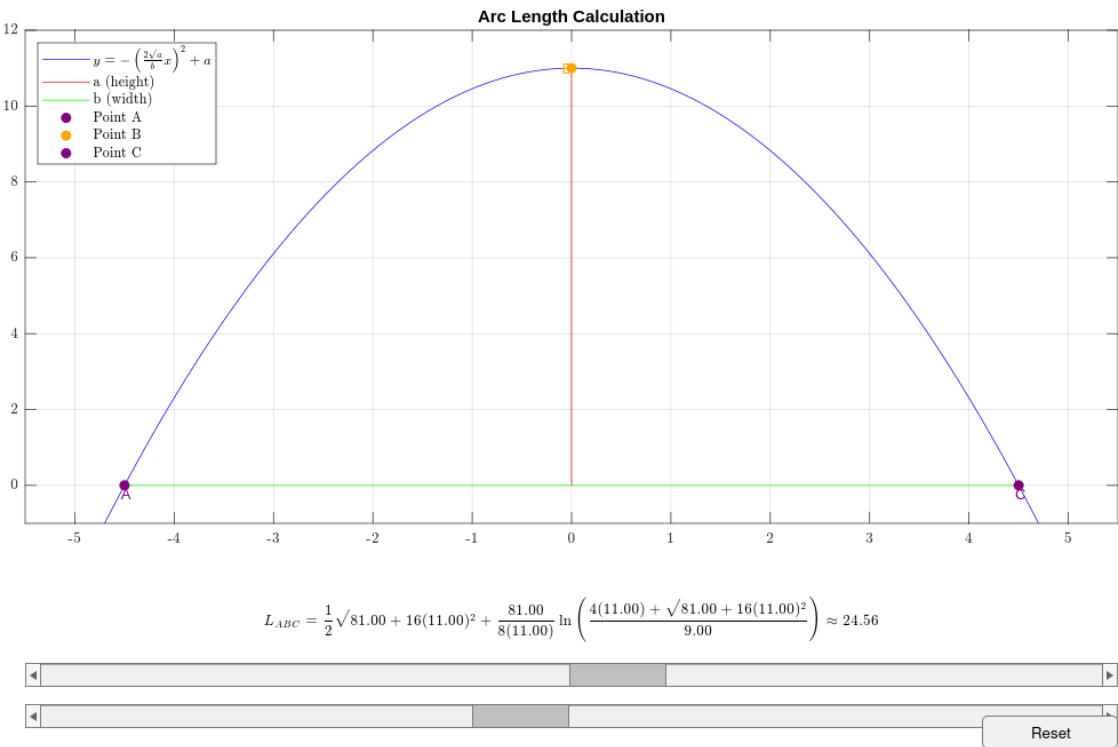
Side note: L_{ABC} is kinda close to evaluating the integral $\int \sqrt{1 + \left(\frac{\partial}{\partial x} \left(-\left(\frac{2\sqrt{a}}{b}x\right)^2 + a\right)\right)^2} dx$.

When implementing this in a program:

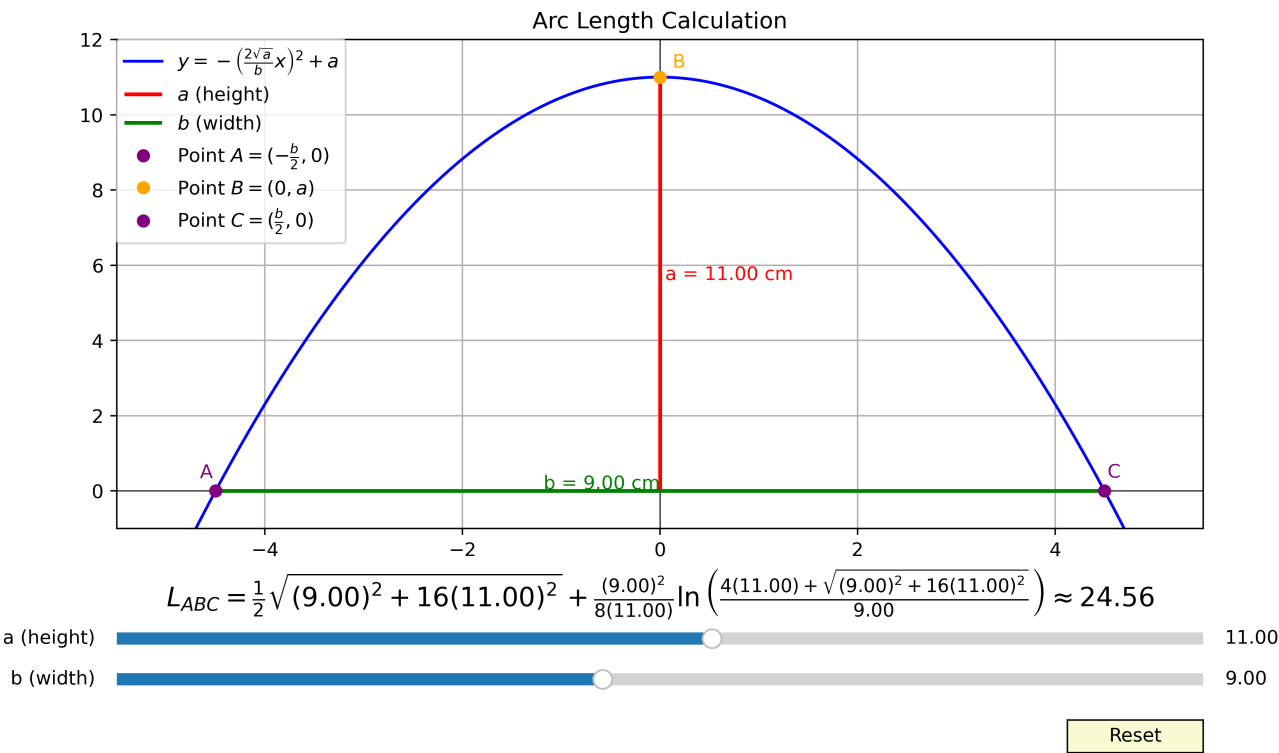
- Ensure that when there are no limit parameters for the inputs a and b , the conditions $a > 0$ and $b \neq 0$ are appropriately handled.
- Update the plot of $y = -\left(\frac{2\sqrt{a}}{b}x\right)^2 + a$ so that the sliders for a and b directly control the height and width, respectively.
- Calculate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

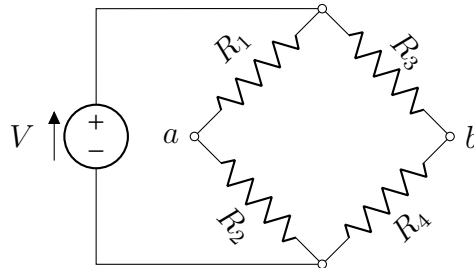
The voltage difference V_{ab} between points a and b in the Wheatstone bridge circuit is:

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

Write a universal, user-friendly program that calculates the voltage difference V_{ab} .

Test your program using the following values:

- $V = 14$ volts
- $R_1 = 120.6 \, \Omega$
- $R_2 = 119.3 \, \Omega$
- $R_3 = 121.2 \, \Omega$
- $R_4 = 118.8 \, \Omega$



matlab scripts/q2.m

```

1 function Vab = calculate_Wheatstone(V, R1, R2, R3, R4)
2     Vab = V * (R1 * R3 - R2 * R4) / ((R1 + R2) * (R3 + R4));
3 end
4
5
6
7 fprintf('The formula for voltage Vab in the Wheatstone Bridge is:\n');
8 fprintf('Vab = V * (R1 * R3 - R2 * R4) / ((R1 + R2) * (R3 + R4))\n');
9 disp('-----');
10
11
12 function confirm = confirmValue(type)
13     while true
14         sure = upper(input(sprintf('Are you sure a %s is what you want? (Y/N): ',
15                                     type), 's'));
16         if strcmp(sure, 'Y')
17             confirm = true;
18             return;
19         elseif strcmp(sure, 'N')
20             fprintf('Returning to previous input...\n');
21             confirm = false;
22             return;
23         else
24             fprintf('Invalid input. Please enter either Y or N.\n');
25         end
26     end
27 end
28
29 while true
30     userInput = input('Enter the value of V (voltage in volts): ', 's');
31     V = str2double(userInput);
32     if isnan(V)
33         fprintf('ERROR: Input must be a numeric value.\n');
34         continue;
35     end
36     if V>0
37         break;
38     elseif V==0

```

```

39     fprintf('ISSUE: V cannot equal 0\n')
40 else
41     fprintf('ISSUE: A negative voltage isn''t realistic.\n');
42     if confirmValue('negative value')
43         break;
44     end
45 end
46 end
47
48
49 function [R1, R2, R3, R4] = promptForResistors()
50     function R = getResistor(name, comparisonValue)
51         disp('-----');
52         while true
53             userInput = input(sprintf('Enter the value of %s (in ohms): ', name), 's')
54                 ;
55             R = str2double(userInput);
56             if isnan(R)
57                 fprintf('ERROR: Input must be a numeric value.\n');
58                 continue;
59             end
60
61             issues = cell(0);
62
63             if R < 0
64                 issues{end+1} = 'ISSUE: A negative resistor isn''t realistic.';
65             end
66             if R == 0
67                 issues{end+1} = 'ISSUE: A resistor of zero isn''t realistic.';
68             end
69             if nargin > 1 && R == -comparisonValue
70                 issues{end+1} = sprintf('ISSUE: %s cannot be equal and opposite to %s
71                                     .', name, inputname(2));
72                 if R < 0 || R==0
73                     issues = issues(end);
74                 end
75             end
76
77             if ~isempty(issues)
78                 fprintf('%s\n', strjoin(issues, '\n'));
79                 if any(contains(issues, 'cannot be equal and opposite to'))
80                     fprintf('Please enter a different value.\n');
81                     continue;
82                 elseif any(contains(issues, 'zero'))
83                     if ~confirmValue('value of zero')
84                         continue;
85                     end
86                 elseif ~confirmValue('negative value')
87                     continue;
88                 end
89             end
90             break;
91         end
92     end
93     R1 = getResistor('R1');
94     R2 = getResistor('R2', R1);
95     R3 = getResistor('R3');
96     R4 = getResistor('R4', R3);
97 end
98

```

```

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);

```

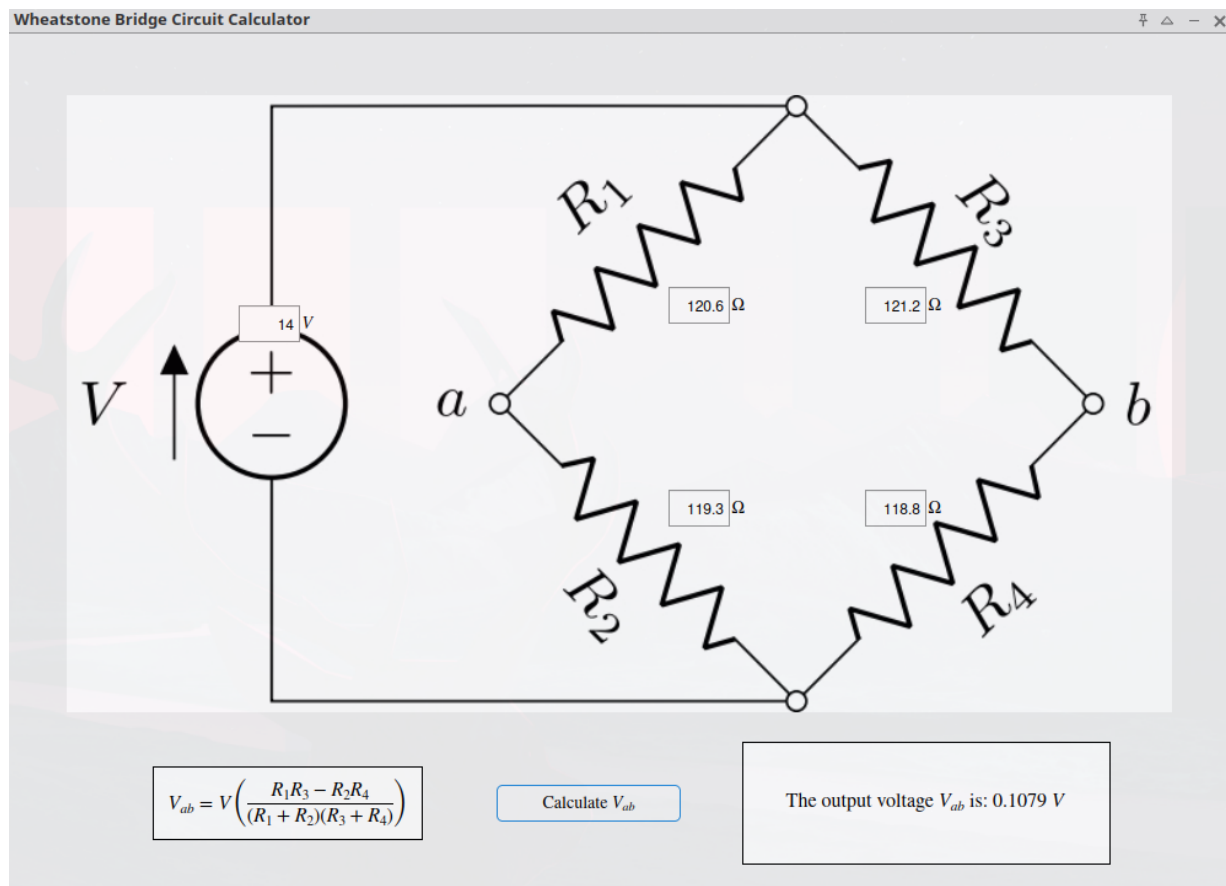
Workspace				Command Window
Name	Value	Size	Class	
R1	120.6000	1x1	double	>> q2 The formula for voltage Vab in the Wheatstone Bridge is: $V_{ab} = V * (R1 * R3 - R2 * R4) / ((R1 + R2) * (R3 + R4))$ ----- Enter the value of V (voltage in volts): 14 ----- Enter the value of R1 (in ohms): 120.6 ----- Enter the value of R2 (in ohms): 119.3 ----- Enter the value of R3 (in ohms): 121.2 ----- Enter the value of R4 (in ohms): 118.8 ----- Voltage value : V = 14.00 Resistor values: R1 = 120.60, R2 = 119.30, R3 = 121.20, R4 = 118.80 The calculated voltage V_ab across the Wheatstone Bridge is: 0.1079 volts >>
R2	119.3000	1x1	double	
R3	121.2000	1x1	double	
R4	118.8000	1x1	double	
userInput	'14'	1x2	char	
V	14	1x1	double	
Vab	0.1079	1x1	double	

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 ($t = 0$): $T_0 = 26.4^\circ\text{C}$
- At 10:18 PM ($t = 1$ hour): $T(1) = 25.5^\circ\text{C}$
- 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 $T_0 = 36.6^\circ\text{C}$:

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

```

1 function k = calculateCoolingConstant(T_1, T_s, T_0, t1)
2     ratio = (T_1 - T_s) / (T_0 - T_s);
3     k = -log(ratio) / t1;
4     fprintf('Cooling constant k = %.4f per hour\n', k);
5 end
6
7 function t_death = calculateTimeSinceDeath(T_1, T_s, T_normal, k)
8     ratio = (T_1 - T_s) / (T_normal - T_s);
9     t_death = -log(ratio) / k;
10    fprintf('Time since death = %.2f hours\n', t_death);
11 end
12
13 T_normal = input('Enter the normal body temperature in C (usually 36.6C): ');
14 T_s = input('Enter the surrounding temperature (T_s) in C (roomtemp is avg 20.5C): ');
15 T_0 = input('You have seen the body, enter the body temperature (T_0) in C: ');
16 time_arrival = input('Enter the time of body discovery (in decimal hours, e.g., 21.3
    for 9:18 PM): ');
17 T_1 = input('Enter the temperature some set time after the discovery (T_1) in C: ');
18 t1 = input('Enter the time interval (in hours) between T_0 and T_1: ');
19 disp('-----');
20 disp('Newton''s Law of Cooling formula:');
21 disp('T(t) = T_s + (T_0 - T_s) * exp(-k * t)');
22 disp('-----');
23 fprintf('Initial body temperature (T_0): %.1fC\n', T_0);
24 fprintf('Observed temperature (T_1) after %.1f hour(s): %.1fC\n', t1, T_1);
25 fprintf('Surrounding temperature (T_s): %.1fC\n', T_s);
26 disp('-----');
27 k = calculateCoolingConstant(T_1, T_s, T_0, t1);
28 t_death = calculateTimeSinceDeath(T_1, T_s, T_normal, k);
29 disp('-----');
30 time_of_death = time_arrival - t_death;
31 hours = floor(time_of_death);
32 minutes = (time_of_death - hours) * 60;
33 fprintf('Estimated time of death: %02d:%02d\n', mod(hours, 24), round(minutes));

```

```

Enter the normal body temperature in C (usually 36.6C): >>36.6
Enter the surrounding temperature (T_s) in C (roomtemp is avg 20.5C): >>20.5
You have seen the body, enter the body temperature (T_0) in C: >>26.4
Enter the time of body discovery (in decimal hours, e.g., 21.3 for 9:18 PM): >>21.3
Enter the temperature some set time after the discovery (T_1) in C: >>25.5
Enter the time interval (in hours) between T_0 and T_1: >>1
-----
Newton's Law of Cooling formula:
T(t) = T_s + (T_0 - T_s) * exp(-k * t)
-----
Initial body temperature (T_0): 26.4C
Observed temperature (T_1) after 1.0 hour(s): 25.5C
Surrounding temperature (T_s): 20.5C
-----
Cooling constant k = 0.1655 per hour
Time since death = 7.07 hours
-----
Estimated time of death: 14:14

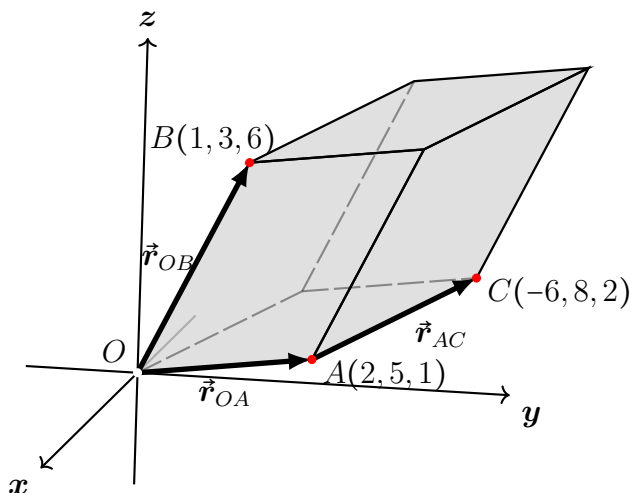
```

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

```

1 fprintf('V = OB * (OA x AC)\n')
2 disp('-----')
3
4 % Prompt user for inputs
5 A = input('Enter point A (in Cartesian coordinates, e.g., [1 2 3]): ');
6 B = input('Enter point B (in Cartesian coordinates, e.g., [4 5 6]): ');
7 C = input('Enter point C (in Cartesian coordinates, e.g., [7 8 9]): ');
8
9 % Calculate vectors OA, OB, and AC
10 OA = A;
11 OB = B;
12 AC = C-A;
13
14 % Calculate the cross product and dot product
15 OAcrossAC = cross(OA, AC);
16 OBdotOAcrossAC = dot(OB, OAcrossAC);
17
18 % Display intermediate results and final value
19 disp('-----')
20 fprintf('Cross product (OA x AC) = [%d %d %d]\n', OAcrossAC);
21 fprintf('Dot product OB * (OA x AC) = %d\n', OBdotOAcrossAC);
22 fprintf('V = %d\n', OBdotOAcrossAC);

```

```

V = OB * (OA x AC)
-----
Enter point A (in Cartesian coordinates, e.g., [1 2 3]): >>[2 5 1]
Enter point B (in Cartesian coordinates, e.g., [4 5 6]): >>[1 3 6]
Enter point C (in Cartesian coordinates, e.g., [7 8 9]): >>[-6 8 2]
-----
Cross product (OA x AC) = [2 -10 46]
Dot product OB * (OA x AC) = 248
V = 248

```


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 α is given by:

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

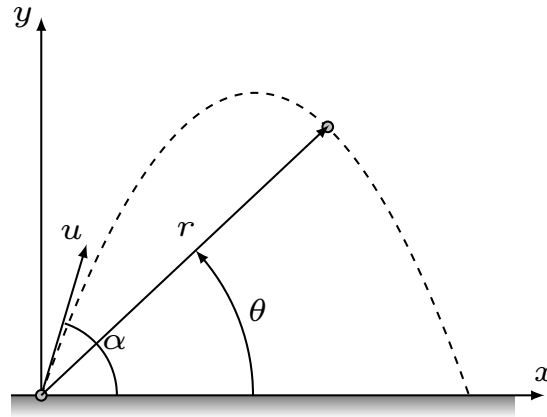
where $g = 9.81 \text{ m/s}^2$ is the acceleration due to gravity.

The polar coordinates of the projectile at time t are $(r(t), \theta(t))$, where

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

Write a universal, user-friendly code. Test case: $v_0 = 162 \text{ m/sec}$ and $\alpha = 70^\circ$.

Determine $r(t)$ and $\theta(t)$ for $t = 1, 6, \dots, 31 \text{ sec}$.



Parameter	x	y
Acceleration / a	$a_x = 0$	$a_y = g = 9.81 \text{ m/s}^2$
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	$\frac{1}{g} (u_y + \sqrt{u_y^2 + 2gh})$
Range / R / $s_x(T)$	$\frac{u_x}{g} (u_y + \sqrt{u_y^2 + 2gh})$
Maximum Height / H	$\frac{1}{2g} (u_y^2) + h$

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 / \mathcal{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 / \mathcal{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

Im not gonna implement all of these just some

matlab scripts/q5.m

```

1 % Constants, user inputs, and initial calculations
2 % Define gravity and prompt user for initial velocity, launch angle, and height.
3 % Convert launch angle to radians and compute initial x and y velocity components.
4 g = 9.81;
5 u = input('Enter the initial velocity (u) in m/s: ');
6 theta = input('Enter the launch angle (theta) in degrees: ');
7 h = input('Enter the initial height (h) in meters: ');
8 theta_rad = deg2rad(theta);
9 u_x = u * cos(theta_rad);
10 u_y = u * sin(theta_rad);
11
12 % Compute flight characteristics: time of flight, range, and maximum height
13 % Use motion equations to determine these parameters based on initial conditions.
14 T = (u_y + sqrt(u_y^2 + 2 * g * h)) / g;
15 R = (u_x / g) * (u_y + sqrt(u_y^2 + 2 * g * h));
16 H = (u_y^2 / (2 * g)) + h;

```

```

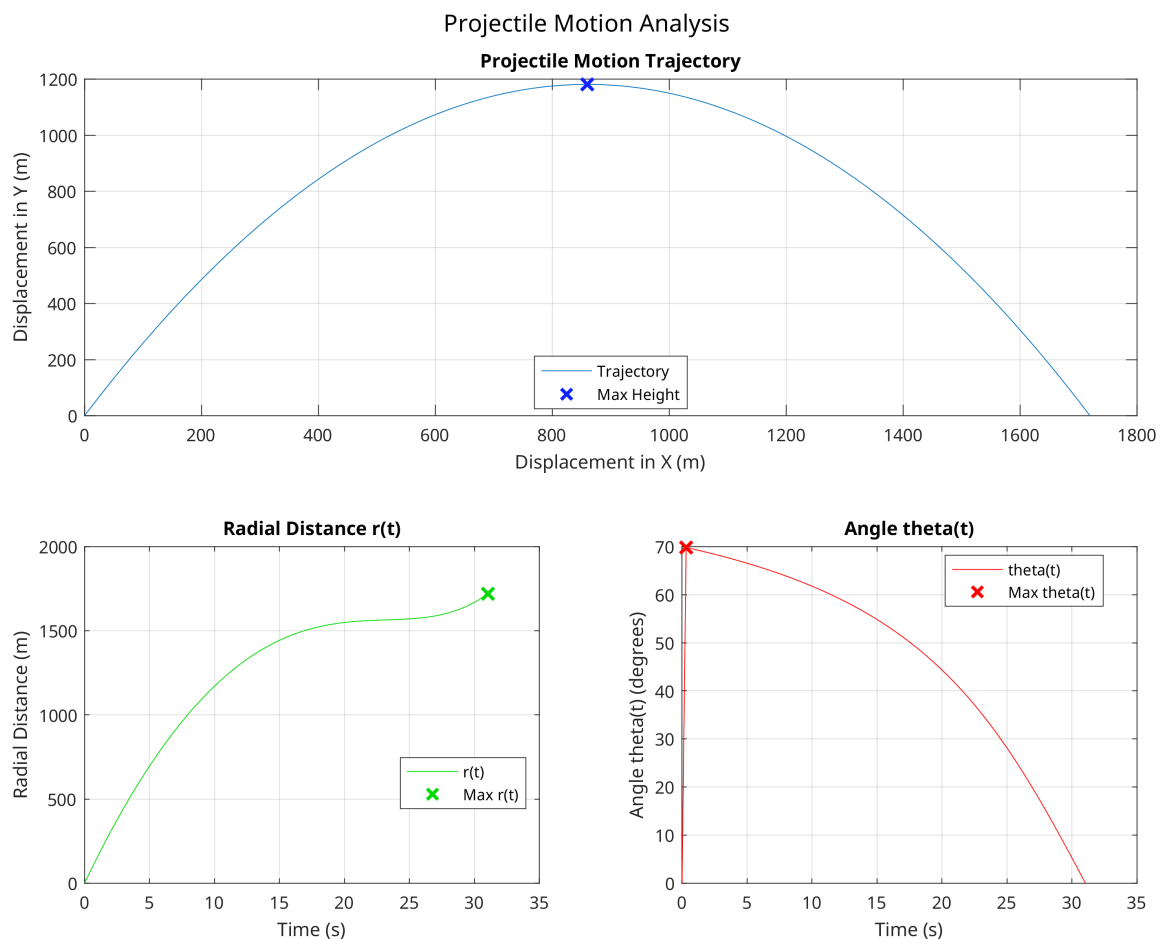
17
18 % Generate time vector and calculate displacements in x and y over time
19 % Determine x and y positions at each time step to represent projectile motion.
20 t = linspace(0, T, 100);
21 s_x = u_x * t;
22 s_y = (u_y * t) - (0.5 * g * t.^2) + h;
23
24 % Calculate radial distance and angle as functions of time
25 % Determine radial distance r(t) and angle theta(t) to analyze the trajectory.
26 r = sqrt(s_x.^2 + s_y.^2);
27 theta_t = atan2(s_y, s_x);
28
29 % Find maximum values for radial distance and angle, along with their time points
30 % Identify when r(t) and theta(t) reach their maximum values during flight.
31 max_r = max(r);
32 max_r_index = find(r == max_r, 1);
33 max_time_r = t(max_r_index);
34 max_theta = max(theta_t);
35 max_theta_index = find(theta_t == max_theta, 1);
36 max_time_theta = t(max_theta_index);
37
38 % Calculate final velocity components and magnitude at time of flight
39 % Use final vertical and constant horizontal velocities to compute total velocity at
    T.
40 v_x_T = u_x;
41 v_y_T = sqrt(u_y^2 + 2 * g * h);
42 v_T = sqrt(v_x_T^2 + v_y_T^2);
43
44 % Display calculated results for projectile motion analysis
45 fprintf('Time of Flight (T): %.2f seconds (time until projectile hits the ground)\n',
    T);
46 fprintf('Range (R): %.2f meters (horizontal distance covered)\n', R);
47 fprintf('Max Height (H): %.2f meters (peak vertical height above the ground)\n', H);
48 fprintf('Velocity at separation (v(T)): %.2f m/s (Note: This simulation assumes
    negligible air resistance.)\n', v_T);
49 fprintf('Maximum radial distance (r(t)): %.2f meters at t = %.2f seconds\n', max_r,
    max_time_r);
50
51 % Display maximum angle in degrees for clarity
52 max_theta_deg = max_theta * 180 / pi;
53 fprintf('Maximum angle (theta(t)): %.2f degrees at t = %.2f seconds\n', max_theta_deg
    , max_time_theta);
54
55 % Plot trajectory and analysis results in a single figure with three subplots
56 figure('Position', [100, 100, 800, 600]);
57
58 % Plot trajectory in Cartesian coordinates
59 subplot(2,1,1);
60 plot(s_x, s_y);
61 title('Projectile Motion Trajectory');
62 xlabel('Displacement in X (m)');
63 ylabel('Displacement in Y (m)');
64 grid on;
65 hold on;
66 plot(u_x*u_y/g, H, 'x', 'MarkerSize', 10, 'LineWidth', 2, 'Color', [0.059, 0.125, 1],
    'DisplayName', 'Max Height');
67 legend('Trajectory', 'Max Height', 'Location', 'best');
68 hold off;
69
70 % Plot radial distance over time
71 subplot(2,2,3);
72 plot(t, r, 'Color', [0, 0.85, 0], 'DisplayName', 'r(t)');

```

```

73 hold on;
74 plot(max_time_r, max_r, 'x', 'MarkerSize', 10, 'LineWidth', 2, 'Color', [0, 0.85, 0],
      'DisplayName', 'Max r(t)');
75 xlabel('Time (s)');
76 ylabel('Radial Distance (m)');
77 title('Radial Distance r(t)');
78 grid on;
79 legend('Location', 'best');
80 hold off;
81
82 % Plot angle theta(t) over time
83 subplot(2,2,4);
84 plot(t, rad2deg(theta_t), 'r-', 'DisplayName', 'theta(t)');
85 hold on;
86 plot(max_time_theta, rad2deg(max_theta), 'x', 'MarkerSize', 10, 'LineWidth', 2, '
      Color', 'r', 'DisplayName', 'Max theta(t)');
87 xlabel('Time (s)');
88 ylabel('Angle theta(t) (degrees)');
89 title('Angle theta(t)');
90 grid on;
91 legend('Location', 'best');
92 hold off;
93
94 % Adjust overall figure layout and add title
95 sgtitle('Projectile Motion Analysis');
96
97 print('-dpng', '-r300', 'Projectile_Motion.png')

```



```
Enter the initial velocity (u) in m/s: >>162
Enter the launch angle (theta) in degrees: >>70
Enter the initial height (h) in meters: >>0
Time of Flight (T): 31.04 seconds (time until projectile hits the ground)
Range (R): 1719.60 meters (horizontal distance covered)
Max Height (H): 1181.14 meters (peak vertical height above the ground)
Velocity at separation (v(T)): 162.00 m/s (Note: This simulation assumes negligible air resistance.)
Maximum radial distance (r(t)): 1719.60 meters at t = 31.04 seconds
Maximum angle (theta(t)): 69.81 degrees at t = 0.31 seconds
```

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, $R = 0.08206$ (L atm)/(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

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

where a and b are material constants. Consider 1 mole ($n = 1$) of nitrogen gas at $T = 300$ K. (For nitrogen gas $a = 1.39$ (L² atm)/mol² and $b = 0.0391$ L/mol.) Create a vector with values of V for $0.1 \leq V \leq 1$ L, using increments of 0.02 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

```

1 % Function to calculate pressure using the Ideal Gas Law
2 function P1 = idealgas(n, R, T, V)
3     P1 = n * R * T / V;
4 end
5
6 % Function to calculate pressure using the Van der Waals equation
7 function P2 = van(n, R, T, V, a, b)
8     P2 = n * R * T / (V - n * b) - n^2 * a / V^2;
9 end
10
11 % Function to calculate the percentage error between two pressures
12 function error_percent = percent_error(P1, P2)
13     error_percent = abs((P2 - P1) / P2) * 100;
14 end
15
16 % Nitrogen properties
17 a = 1.39;
18 b = 0.0391;
19 n = 1;
20 T = 300;
21 R = 0.08206;
22
23 % Volume range
24 Vol = 0.1:0.02:1;
25 P1 = zeros(size(Vol));
26 P2 = zeros(size(Vol));
27 error_vals = zeros(size(Vol));
28
29 % Calculate pressures and errors for each volume
30 for i = 1:length(Vol)

```

```

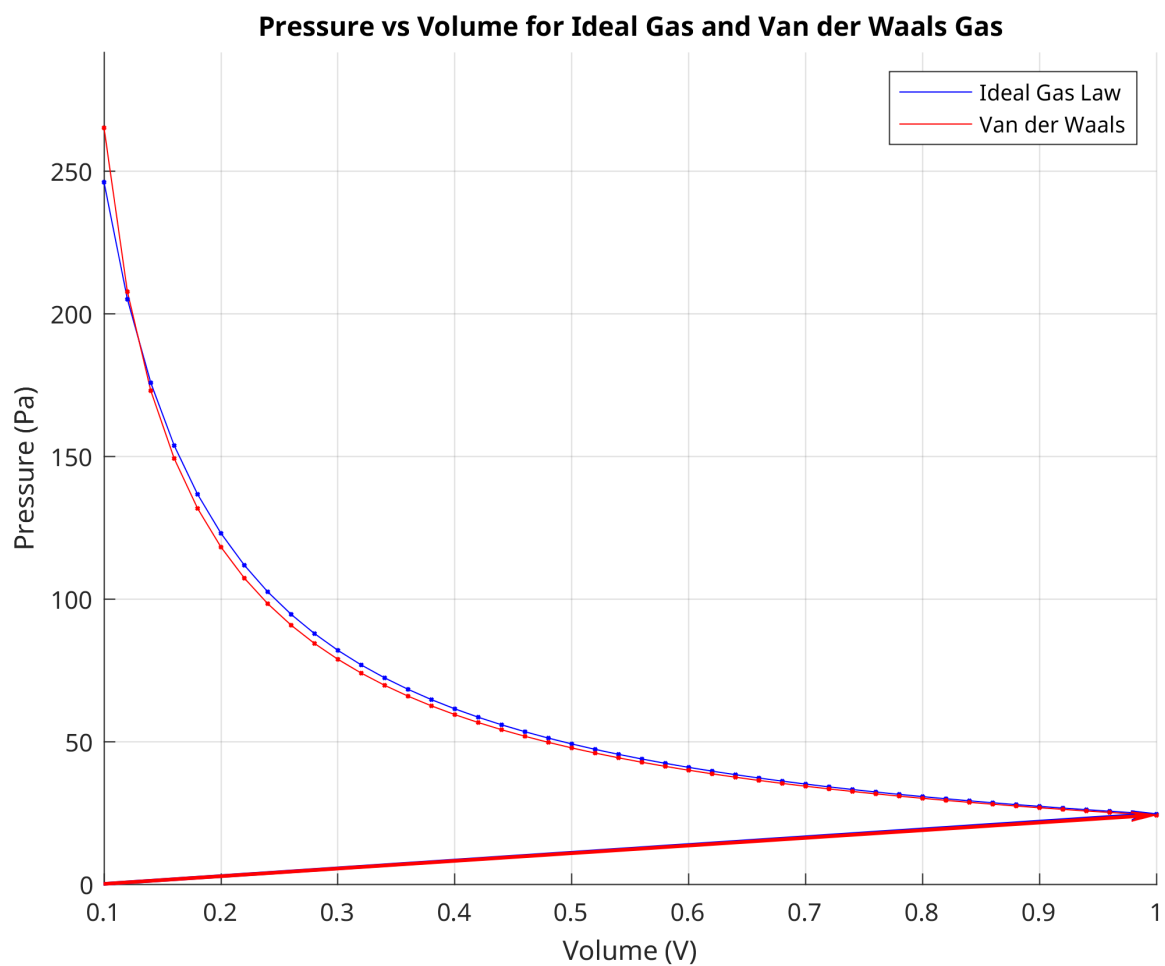
31     V = Vol(i);
32     P1(i) = idealgas(n, R, T, V);
33     P2(i) = van(n, R, T, V, a, b);
34     error_vals(i) = percent_error(P1(i), P2(i));
35     fprintf('Vol: %.2f [P1: %.4f, P2: %.4f, Error: %.2f%%]\n', Vol(i), P1(i), P2(i),
36           error_vals(i));
37
38 end
39
40 % Find maximum error and corresponding volume
41 [max_error, idx] = max(error_vals);
42 V_max_error = Vol(idx);
43 max_P1 = max(P1);
44 max_P2 = max(P2);
45 max_P = max(max_P1, max_P2);
46 min_V = min(Vol);
47
48 fprintf('Maximum Error: %.2f%% at Volume: %.2f\n', max_error, V_max_error);
49
50 % Plotting pressures vs volume
51 figure;
52 for i = 1:length(Vol)
53     ax = gca;
54     if min_V < Vol(i)
55         xlim([min_V, Vol(i)]);
56     end
57     current_max_pressure = max(max(P1(1:i)), max(P2(1:i)));
58     ylim([0, current_max_pressure * 1.1]);
59
60     plot(Vol(1:i), P1(1:i), 'color', 'b', 'DisplayName', 'Ideal Gas Law');
61     plot(Vol(1:i), P2(1:i), 'color', 'r', 'DisplayName', 'Van der Waals');
62     hold on
63     plot(Vol(1:i), P1(1:i), 'x', 'color', 'b', 'Markersize', 2, 'LineWidth', 1);
64     plot(Vol(1:i), P2(1:i), 'x', 'color', 'r', 'Markersize', 2, 'LineWidth', 1);
65
66     x_lim = xlim;
67     y_lim = ylim;
68
69     x_norm = (Vol(i) - x_lim(1)) / (x_lim(2) - x_lim(1));
70     y_norm_P1 = (P1(i) - y_lim(1)) / (y_lim(2) - y_lim(1));
71     y_norm_P2 = (P2(i) - y_lim(1)) / (y_lim(2) - y_lim(1));
72
73     annotation('arrow', ax.Position([1 3]), ax.Position([2 4]), ...
74         'Position', [ax.Position(1) ax.Position(2) x_norm*ax.Position(3) y_norm_P1*ax.
75             Position(4)], ...
76         'Color', 'b', 'LineWidth', 1.5, 'HeadLength', 10, 'HeadWidth', 5);
77
78     annotation('arrow', ax.Position([1 3]), ax.Position([2 4]), ...
79         'Position', [ax.Position(1) ax.Position(2) x_norm*ax.Position(3) y_norm_P2*ax.
80             Position(4)], ...
81         'Color', 'r', 'LineWidth', 1.5, 'HeadLength', 10, 'HeadWidth', 5);
82
83 hold on;
84
85 xlabel('Volume (V)');
86 ylabel('Pressure (Pa)');
87 title('Pressure vs Volume for Ideal Gas and Van der Waals Gas');
88 grid on;
89
90 legend('Ideal Gas Law', 'Van der Waals');
91
92 if i > 1
93     delta_P1 = abs(P1(i) - P1(i-1));

```

```

90     delta_P2 = abs(P2(i) - P2(i-1));
91     delta_P = max(delta_P1, delta_P2);
92     pause_duration = delta_P/30;
93 else
94     pause_duration = 0.25;
95 end
96 pause(pause_duration);
97
98 if i < length(Vol)
99     clf;
100    xlabel('Volume (V)');
101    ylabel('Pressure (Pa)');
102    title('Pressure vs Volume for Ideal Gas and Van der Waals Gas');
103    grid on;
104    hold on;
105 end
106 end
107
108 print('-dpng', '-r300', 'Pressurevolume.png')
109 hold off;

```




```
Vol: 0.10 [P1: 246.1800, P2: 265.2365, Error: 7.18%]  
Vol: 0.12 [P1: 205.1500, P2: 207.7738, Error: 1.26%]  
Vol: 0.14 [P1: 175.8429, P2: 173.0658, Error: 1.60%]  
Vol: 0.16 [P1: 153.8625, P2: 149.3260, Error: 3.04%]  
Vol: 0.18 [P1: 136.7667, P2: 131.8184, Error: 3.75%]  
Vol: 0.20 [P1: 123.0900, P2: 118.2519, Error: 4.09%]  
Vol: 0.22 [P1: 111.9000, P2: 107.3672, Error: 4.22%]  
Vol: 0.24 [P1: 102.5750, P2: 98.4066, Error: 4.24%]  
Vol: 0.26 [P1: 94.6846, P2: 90.8820, Error: 4.18%]  
Vol: 0.28 [P1: 87.9214, P2: 84.4622, Error: 4.10%]  
Vol: 0.30 [P1: 82.0600, P2: 78.9135, Error: 3.99%]  
Vol: 0.32 [P1: 76.9312, P2: 74.0655, Error: 3.87%]  
Vol: 0.34 [P1: 72.4059, P2: 69.7903, Error: 3.75%]  
Vol: 0.36 [P1: 68.3833, P2: 65.9902, Error: 3.63%]  
Vol: 0.38 [P1: 64.7842, P2: 62.5887, Error: 3.51%]  
Vol: 0.40 [P1: 61.5450, P2: 59.5253, Error: 3.39%]  
Vol: 0.42 [P1: 58.6143, P2: 56.7513, Error: 3.28%]  
Vol: 0.44 [P1: 55.9500, P2: 54.2271, Error: 3.18%]  
Vol: 0.46 [P1: 53.5174, P2: 51.9200, Error: 3.08%]  
Vol: 0.48 [P1: 51.2875, P2: 49.8028, Error: 2.98%]  
Vol: 0.50 [P1: 49.2360, P2: 47.8529, Error: 2.89%]  
Vol: 0.52 [P1: 47.3423, P2: 46.0510, Error: 2.80%]  
Vol: 0.54 [P1: 45.5889, P2: 44.3807, Error: 2.72%]  
Vol: 0.56 [P1: 43.9607, P2: 42.8281, Error: 2.64%]  
Vol: 0.58 [P1: 42.4448, P2: 41.3810, Error: 2.57%]  
Vol: 0.60 [P1: 41.0300, P2: 40.0291, Error: 2.50%]  
Vol: 0.62 [P1: 39.7065, P2: 38.7630, Error: 2.43%]  
Vol: 0.64 [P1: 38.4656, P2: 37.5750, Error: 2.37%]  
Vol: 0.66 [P1: 37.3000, P2: 36.4579, Error: 2.31%]  
Vol: 0.68 [P1: 36.2029, P2: 35.4056, Error: 2.25%]  
Vol: 0.70 [P1: 35.1686, P2: 34.4125, Error: 2.20%]  
Vol: 0.72 [P1: 34.1917, P2: 33.4738, Error: 2.14%]  
Vol: 0.74 [P1: 33.2676, P2: 32.5851, Error: 2.09%]  
Vol: 0.76 [P1: 32.3921, P2: 31.7425, Error: 2.05%]  
Vol: 0.78 [P1: 31.5615, P2: 30.9425, Error: 2.00%]  
Vol: 0.80 [P1: 30.7725, P2: 30.1819, Error: 1.96%]  
Vol: 0.82 [P1: 30.0220, P2: 29.4579, Error: 1.91%]  
Vol: 0.84 [P1: 29.3071, P2: 28.7680, Error: 1.87%]  
Vol: 0.86 [P1: 28.6256, P2: 28.1096, Error: 1.84%]  
Vol: 0.88 [P1: 27.9750, P2: 27.4808, Error: 1.80%]  
Vol: 0.90 [P1: 27.3533, P2: 26.8796, Error: 1.76%]  
Vol: 0.92 [P1: 26.7587, P2: 26.3042, Error: 1.73%]  
Vol: 0.94 [P1: 26.1894, P2: 25.7529, Error: 1.69%]  
Vol: 0.96 [P1: 25.6438, P2: 25.2243, Error: 1.66%]  
Vol: 0.98 [P1: 25.1204, P2: 24.7170, Error: 1.63%]  
Vol: 1.00 [P1: 24.6180, P2: 24.2297, Error: 1.60%]  
Maximum Error: 7.18% at Volume: 0.10
```

B.7 Q7

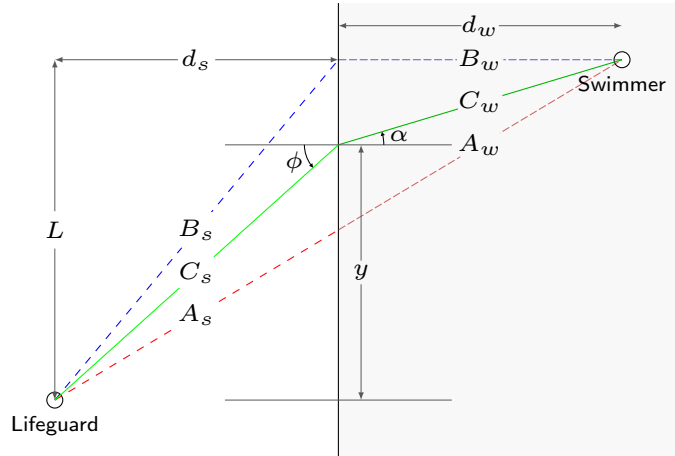
A student has a summer job as a lifeguard at the beach.

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 ($y = 20, 21, 22, \dots, 48 \text{ m}$) 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_{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:

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

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

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 $20 \leq y \leq 48$ 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

```

1 % Define the running and swimming speeds, and distances
2 v_run = 3;
3 v_swim = 1;
4 L = 48;
5 d_s = 30;

```

```
6
7 % Define the range of y values
8 y_values = 0:0.0001:L;
9
10 % Calculate the total time for each y value
11 total_time = zeros(size(y_values));
12
13 for i = 1:length(y_values)
14     y = y_values(i);
15     C_s = sqrt(y^2+d_s^2);
16     C_w = sqrt(d_s^2+(L-y)^2);
17     t_run = C_s/v_run;
18     t_swim = C_w/v_swim;
19     total_time(i) = t_run+t_swim;
20 end
21
22 % Find the optimal y with minimum time
23 [min_time, idx] = min(total_time); % Find the minimum time and its index
24 optimal_y = y_values(idx); % Optimal y value
25
26 % Verify the result with Snell's Law
27 phi = atand(optimal_y/d_s);
28 alpha = atand((L-optimal_y)/d_s);
29
30 % Check Snell's Law
31 snell_ratio = sind(phi)/sind(alpha);
32 expected_ratio = v_run/v_swim;
33
34 fprintf('Optimal y value: %.2f m\n', optimal_y);
35 fprintf('Minimum time: %.2f s\n', min_time);
36 fprintf('Phi: %.2f degrees\n', phi);
37 fprintf('Alpha: %.2f degrees\n', alpha);
38 fprintf('Snell ratio: %.6f\n', snell_ratio);
39 fprintf('Expected ratio: %.6f\n', expected_ratio);
40
41 if abs(snell_ratio-expected_ratio)<1e-4
42     disp('Snell's Law essentially is satisfied.');
```

```
Optimal y value: 39.72 m
Minimum time: 47.71 s
Phi: 52.94 degrees
Alpha: 15.43 degrees
Snell ratio: 2.999988
Expected ratio: 3.000000
Snell's Law essentially is satisfied.
```

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 σ_c and strain ε_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 σ_t and strain ε_t are defined by

$$\sigma_t = \frac{FL_0}{A_0L} \quad \varepsilon_t = \ln\left(\frac{L}{L_0}\right)$$

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

```

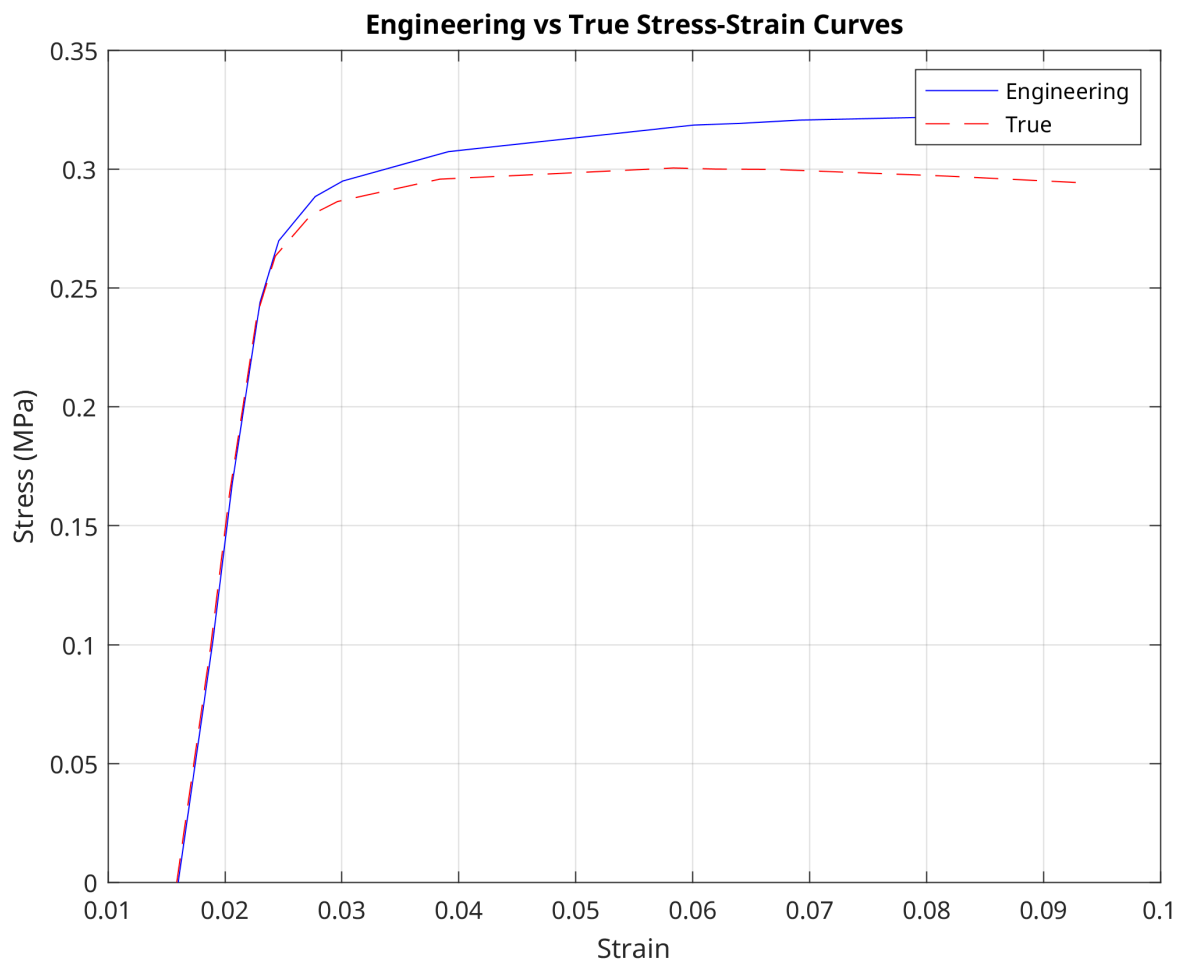
1 % Input data
2 F = [0, 13.031, 21.485, 31.3963, 34.727, 37.119, 37.960, 39.550, 40.758, 40.986, 41.0
   76, 41.255, 41.481, 41.564];
3 L = [25.400, 25.474, 25.515, 25.575, 25.615, 25.693, 25.752, 25.978, 26.419, 26.502,
   26.600, 26.728, 27.130, 27.441];
4
5 % Initial parameters
6 L0 = 25; % mm
7 r = 6.4/1000; % m (convert mm to m)
8 A0 = pi*r^2; % m^2
9
10 % Calculate engineering stress and strain
11 eng_strain = (L-L0)/L0;
12 eng_stress = F/A0 /1e6; % Convert N/m^2 to MPa
13

```

```

14 % Calculate true stress and strain
15 true_strain = log(L/L0);
16 true_stress = F.*L0./(A0*L)/1e6; % Convert N/m^2 to MPa
17
18 % Plot the curves
19 figure;
20 plot(eng_strain, eng_stress, 'b-');
21 hold on;
22 plot(true_strain, true_stress, 'r--');
23 xlabel('Strain');
24 ylabel('Stress (MPa)');
25 title('Engineering vs True Stress-Strain Curves');
26 legend('Engineering', 'True');
27 grid on;
28 hold off;
29
30 print('-dpng', '-r300', 'stressvstran.png')

```



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

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

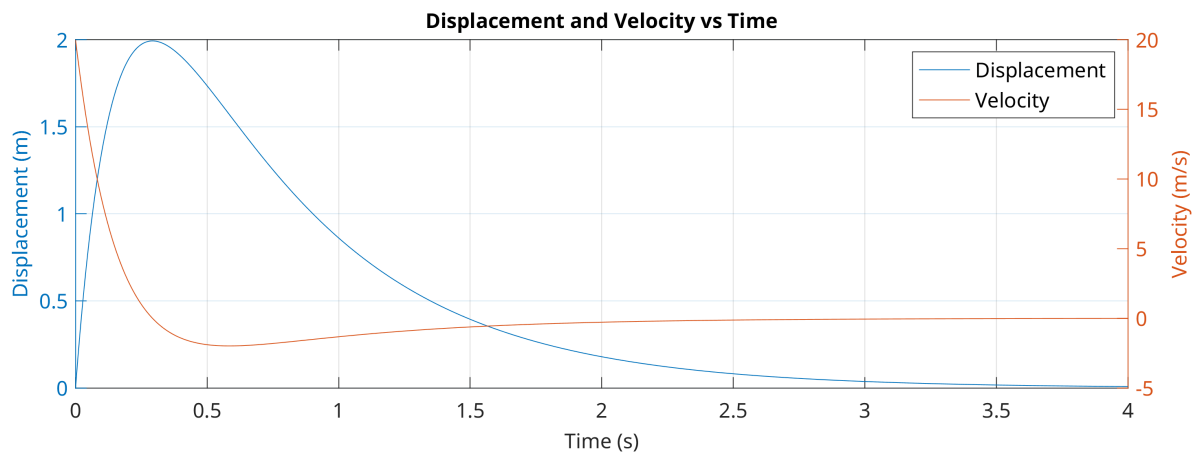
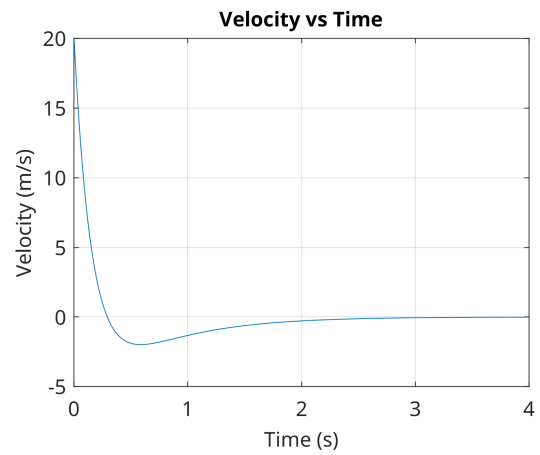
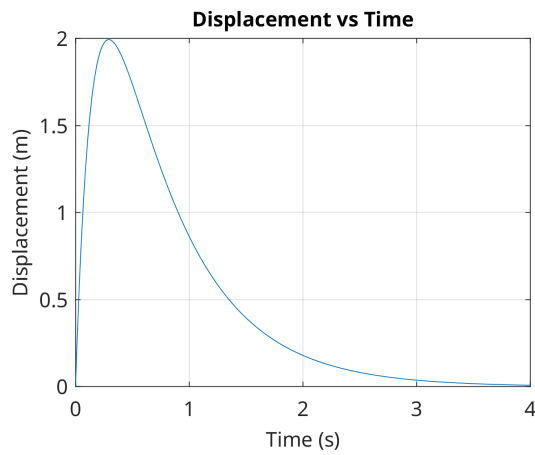
Plot the displacement and the velocity as a function of time for $0 \leq t \leq 4$ seconds. Fit two plots at the top of the window and a plot of both with two vertical axes underneath them. All plots must be of printing quality.

matlab scripts/q9.m

```

1 % Define time range
2 t = linspace(0, 4, 1000);
3
4 % Define displacement and velocity functions
5 x = @(t) 4.219 * (exp(-1.58*t) - exp(-6.32*t));
6 v = @(t) 26.67 * exp(-6.32*t) - 6.67 * exp(-1.58*t);
7
8 % Create figure
9 figure('Position', [100, 100, 800, 600]);
10
11 % Plot displacement
12 subplot(2,2,1)
13 plot(t, x(t))
14 title('Displacement vs Time')
15 xlabel('Time (s)')
16 ylabel('Displacement (m)')
17 grid on
18
19 % Plot velocity
20 subplot(2,2,2)
21 plot(t, v(t))
22 title('Velocity vs Time')
23 xlabel('Time (s)')
24 ylabel('Velocity (m/s)')
25 grid on
26
27 % Plot both displacement and velocity
28 subplot(2,1,2)
29 yyaxis left
30 plot(t, x(t), 'DisplayName', 'Displacement')
31 ylabel('Displacement (m)')
32 yyaxis right
33 plot(t, v(t), 'DisplayName', 'Velocity')
34 ylabel('Velocity (m/s)')
35 title('Displacement and Velocity vs Time')
36 xlabel('Time (s)')
37 grid on
38 legend;
39
40 % Adjust layout
41 set(gcf, 'Color', 'w');
```

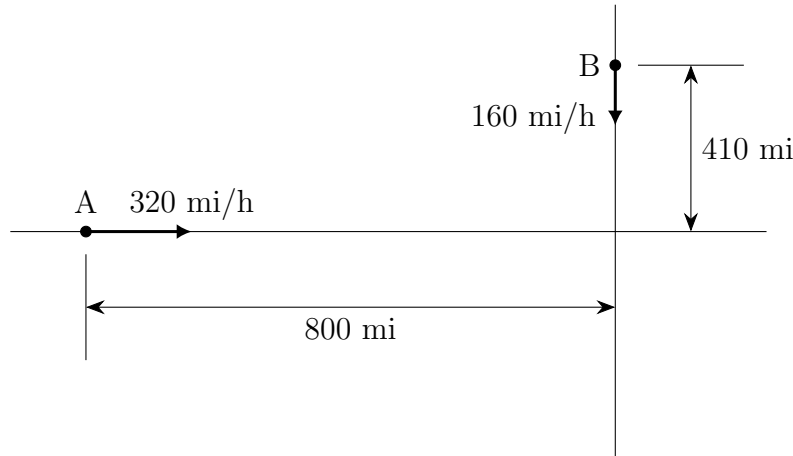
```
42 set(findall(gcf, '-property', 'FontSize'), 'FontSize', 12)
43 print('-dpng', '-r300', 'railroad_bumper_plots.png')
```



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.

Step 1: Setting Up the Position Coordinates

At 1:00 p.m., the initial positions of the two aircraft are as follows:

Aircraft A is located 800 miles west of the vertical line where Aircraft B is positioned, and Aircraft B is 410 miles north of the horizontal line where Aircraft A is located.

We assume that Aircraft A is flying east at a speed of 320 mi/hr, while Aircraft B is moving south at 160 mi/hr.

Let $t = 0$ represent 1:00 p.m., the starting time. We denote the eastward position of Aircraft A at time t as $x_A(t)$, and the southward position of Aircraft B at time $y_B(t)$.

Step 2: Expressing Positions as Functions of Time

Aircraft A's Position: Since Aircraft A is moving eastward at 320 mi/hr, given ($d = vt$) its position $x_A(t)$ relative to its starting point can be expressed as:

$$x_A(t) = 320t$$

Thus, the total horizontal distance from **Aircraft B** after t hours is:

$$x(t) = 800 - x_A(t) = 800 - 320t$$

Aircraft B's Position: Since Aircraft B is moving southward at 160 mi/hr, given ($d = vt$) its position $y_B(t)$ relative to its starting point can be expressed as:

$$y_B(t) = 160t$$

Thus, the total vertical distance from **Aircraft A** after t hours is:

$$y(t) = 410 - y_B(t) = 410 - 160t$$

Step 3: Deriving the Distance $D(t)$ Between the Aircraft

Using the Pythagorean theorem, the distance $D(t)$ between the aircraft is:

$$D(t) = \sqrt{x(t)^2 + y(t)^2}$$

Substituting for $x(t)$ and $y(t)$:

$$D(t) = \sqrt{(800 - 320t)^2 + (410 - 160t)^2}$$

Computing the Time When Aircraft Are Within 30 Miles

To find when the aircraft are first within 30 mi of each other, set $D(t) = 30$ and solve for t :

$$30 = \sqrt{(800 - 320t)^2 + (410 - 160t)^2}$$

Squaring both sides:

$$900 = (800 - 320t)^2 + (410 - 160t)^2$$

This equation can be simplified and solved using a computational tool such as MATLAB's `roots` function.

Determining the Minimum Distance

To find the time t at which the distance $D(t)$ reaches its minimum, we can take the derivative of $D(t)$ with respect to t , set it to zero, and solve for t .

However, since this expression is not needed to differentiate manually, we can use numerical methods or computational tools to find this minimum.

Summary

- **Expression for $D(t)$:** The distance between the aircraft as a function of time is

$$D(t) = \sqrt{(800 - 320t)^2 + (410 - 160t)^2}$$

- **Finding Minimum Distance:** Use MATLAB's `min` function to find the minimum $D(t)$ and the corresponding time t .
- **Solving $D(t) = 30$:** Set $D(t) = 30$ and use MATLAB's `roots` function to find the time t when the aircraft are first within 30 mi of each other.

matlab scripts/q10.m

```

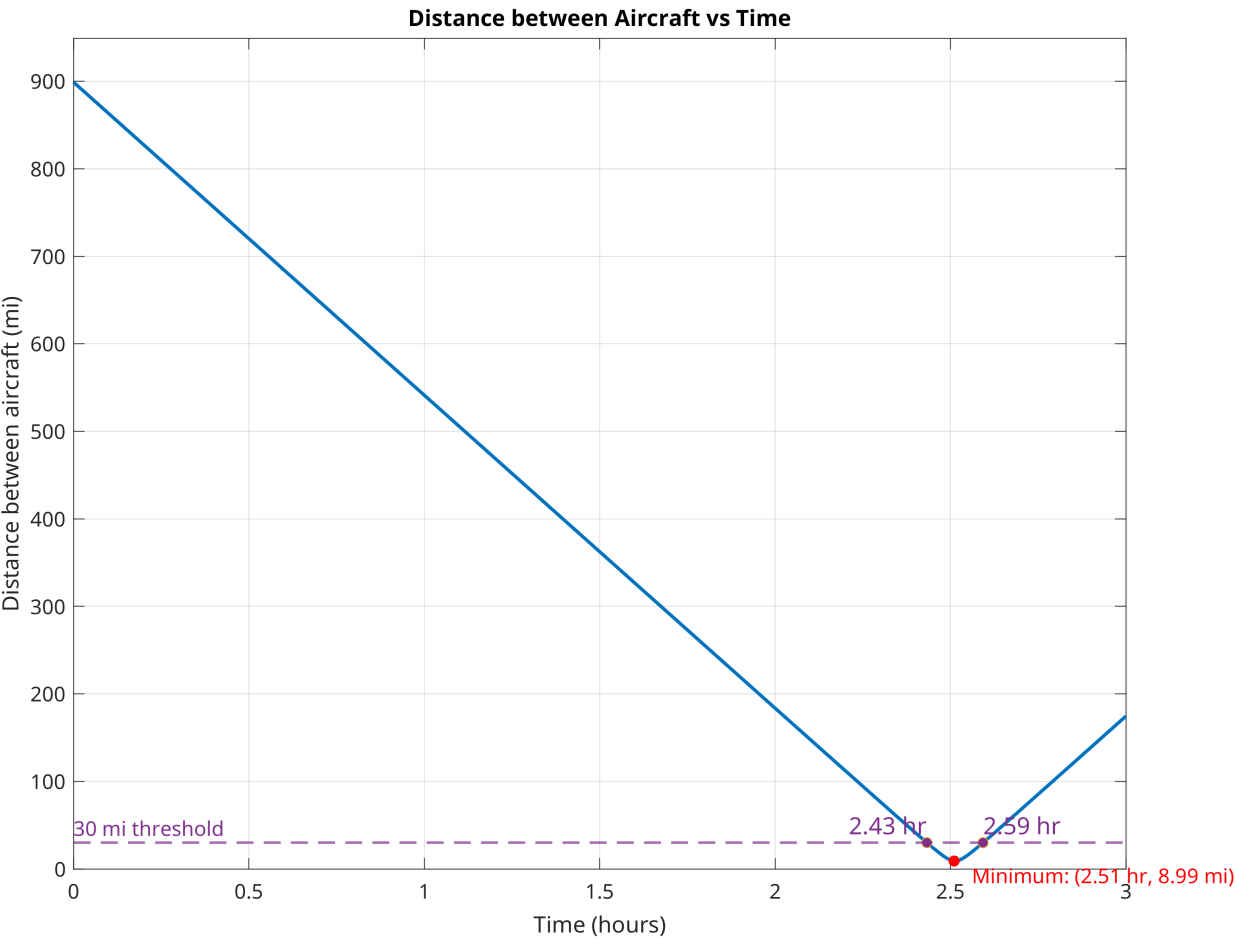
1 % Define constants
2 v_A = 320;
3 v_B = 160;
4 x0 = 800;
5 y0 = 410;
6
7 % Define time range (in hours)
8 t = 0:0.01:3;
9
10 D = sqrt((x0 - v_A * t).^2 + (y0 - v_B * t).^2);
11
12 [min_D, min_index] = min(D);
13 t_min = t(min_index);
14
15 % Plot D versus time
16 figure('Position', [100, 100, 800, 600])

```

```

17 plot(t, D, 'LineWidth', 2)
18 hold on
19
20 % Mark minimum distance point
21 plot(t_min, min_D, 'ro', 'MarkerSize', 6, 'MarkerFaceColor', 'r')
22 text(t_min + 0.05, min_D, sprintf('Minimum: (%.2f hr, %.2f mi)', t_min, min_D), ...
23      'FontSize', 12, 'VerticalAlignment', 'top', 'Color', 'r')
24
25 % Add horizontal line at 30 miles
26 yline(30, '--', 'Color', '#7E2F8E', 'LineWidth', 1.5);
27 text(0, 30, '30 mi threshold', 'FontSize', 12, 'VerticalAlignment', 'bottom', '
    HorizontalAlignment', 'left', 'Color', '#7E2F8E')
28
29 % Find times when the distance is exactly 30 miles
30 coeff = [v_A^2 + v_B^2, -2 * (v_A * x0 + v_B * y0), x0^2 + y0^2 - 30^2];
31
32 t_30_all = roots(coeff);
33 t_30_all = t_30_all(t_30_all > 0);
34 t_30_first = min(t_30_all);
35 t_30_second = max(t_30_all);
36
37 % Mark the points where distance is exactly 30 miles
38 plot([t_30_first, t_30_second], [30, 30], 'o', 'MarkerSize', 6, 'MarkerFaceColor', '#7
    E2F8E')
39 text(t_30_first, 30, sprintf('%.2f hr', t_30_first), 'FontSize', 14, '
    VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Color', '#7E2F8E')
40 text(t_30_second, 30, sprintf('%.2f hr', t_30_second), 'FontSize', 14, '
    VerticalAlignment', 'bottom', 'Color', '#7E2F8E')
41
42 % Set up plot labels and title
43 xlabel('Time (hours)', 'FontSize', 14)
44 ylabel('Distance between aircraft (mi)', 'FontSize', 14)
45 title('Distance between Aircraft vs Time', 'FontSize', 16)
46 grid on
47 axis([0 t(end) 0 max(D)+50])
48 set(gca, 'FontSize', 12)
49 box on
50
51 % Save the figure as a high-quality image
52 print('-dpng', '-r300', 'aircraft_distance_plot.png');
53
54 % Displaying results in console
55 fprintf('Minimum distance: %.2f mi at t = %.2f hours\n', min_D, t_min);
56 t=9+60/18
57 fprintf('Aircraft are first within 30 mi of each other at t = %.2f \n', t_30_first);

```



```
Minimum distance: 8.99 mi at t = 2.51 hours
Aircraft are first within 30 mi of each other at t = 2.43
```

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 σ_{xx} , σ_{yy} , and τ_{xy} . The stresses at the point in the direction defined by the $x' - y'$ coordinate system are calculated by the stress transformation equations:

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

where θ 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 σ_{xx} , σ_{yy} , and τ_{xy} , and the angle θ . 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 σ_{xx} , σ_{yy} , and τ_{xy} , and the input argument θ is a scalar with the value of θ . 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

```

1 function [Strain] = StressTrans(S, theta)
2     sigma_xx = S(1);
3     sigma_yy = S(2);
4     tau_xy = S(3);
5
6     sigma_xx_prime = (sigma_xx + sigma_yy)/2 + (sigma_xx - sigma_yy)/2 * cos(2*theta)
7         + tau_xy * sin(2*theta);
8     sigma_yy_prime = (sigma_xx + sigma_yy)/2 - (sigma_xx - sigma_yy)/2 * cos(2*theta)
9         - tau_xy * sin(2*theta);
10    tau_xy_prime = -(sigma_xx - sigma_yy)/2 * sin(2*theta) + tau_xy * cos(2*theta);
11
12    Strain = [sigma_xx_prime, sigma_yy_prime, tau_xy_prime];
13 end
14
15 sigma_xx = input('Enter the stress component sigma_xx (in Pa): ');
16 sigma_yy = input('Enter the stress component sigma_yy (in Pa): ');
17 tau_xy = input('Enter the shear stress component tau_xy (in Pa): ');
18
19 S = [sigma_xx, sigma_yy, tau_xy];
20
21 theta_degrees = input('Enter the angle (in degrees): ');
22 theta = deg2rad(theta_degrees); % Convert degrees to radians
23
24 Strain = StressTrans(S, theta);
25 fprintf('Strain = [%.2f, %.2f, %.2f]\n', Strain(1), Strain(2), Strain(3));
26 fprintf('Transformed stresses:\n');
27 fprintf('sigma_x''x'' = %.2f Pa\n', Strain(1));
28 fprintf('sigma_y''y'' = %.2f Pa\n', Strain(2));
29 fprintf('tau_x''y'' = %.2f Pa\n', Strain(3));

```