

Engineering Mathematics and Computing

Task 1: Coursework Assessment

Student Name: Sakariye Abiikar
KID: 2371673

Last Updated: November 6, 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, for $y = \frac{\ln 3x}{2x}$:

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

First, find the derivatives:

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

Let:

$$\delta = 3x \text{ in so that now } u = \ln(\delta)$$

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

Using the chain rule, we can find:

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

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

General rule for composite functions
derived via chain rule is:

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

$$v = 2x$$

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

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 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.

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

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

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

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

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

The vector \vec{r} should be defined here as being perpendicular to both \mathbf{a} and \mathbf{b}

A vector perpendicular to two other vectors, \mathbf{a} and \mathbf{b} , in three-dimensional space can be obtained by calculating the cross product of those two vectors, $\mathbf{a} \times \mathbf{b}$. This is a well-known fact in so that its practically the cross product's justification and application.

The matrix determinant method is the fastest way to calculate cross products with vectors that are provided in Cartesian form.

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

Find the magnitude:

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

The unit vector is:

$$\hat{\mathbf{r}} = \frac{-38\mathbf{i} + 39\mathbf{j} + 7\mathbf{k}}{\sqrt{3014}} \approx -0.69\mathbf{i} + 0.71\mathbf{j} + 0.13\mathbf{k}$$

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:

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

$$x = 5, \quad x = -3$$

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

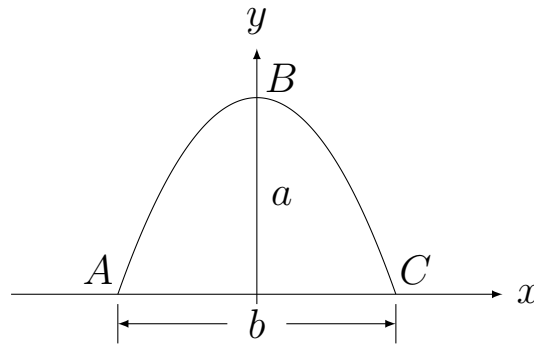
Notes:

- 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.
- 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 to calculate arc length L_ABC
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

% Display formula
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');
;

% Input validation loop for 'a'
while true
    a=input('Enter the value of a (height in cm): ');
    if a~=0
        % Input validation loop for 'b'
        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

% Calculate and display arc length
L_ABC=l_abc(a,b);
fprintf('The arc length L_ABC is: %.2f\n',L_ABC)
```

Workspace

Name	Value	Size	Class
a	11	1x1	double
b	9	1x1	double
L_ABC	24.5637	1x1	double

Command Window

>> q1
The formula for arc length L_ABC is:
$$L_ABC = (1/2) * \sqrt{b^2 + 16 * a^2} + (b^2 / (8 * a)) * \log((4 * a + \sqrt{b^2 + 16 * a^2}) / b)$$

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 .

B.1.2 General Formula for Arc Length

In general, the arc length L of a function $f(x)$ from x_1 to x_2 is given by the formula:

$$L = \int_{x_1}^{x_2} \sqrt{1 + \left(\frac{d}{dx}(f(x))\right)^2} dx$$

For our function, we can substitute the derived equation $y = -\left(\frac{2\sqrt{a}}{b}x\right)^2 + a$ into this formula and compute the arc length from $-\frac{b}{2}$ to $\frac{b}{2}$.

B.1.3 Simplification and Resulting Expression

After performing the integration, we arrive at the following expression for the arc length:

$$L = \int_{-\frac{b}{2}}^{\frac{b}{2}} \sqrt{1 + \left(\frac{\partial}{\partial x} \left(-\left(\frac{2\sqrt{a}}{b}x\right)^2 + a\right)\right)^2} dx = \frac{4a\sqrt{b^4 + 16a^2b^2} + \operatorname{arsinh}\left(\frac{4a}{b}\right)b^3}{8ab}$$

This is the arc length of the parabola from $x = -\frac{b}{2}$ to $x = \frac{b}{2}$ for our derived expression.

B.1.4 Comparison of the Two Expressions

We now compare this arc length expression with the expression in the question for the length L_{ABC} :

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

$$L = \frac{4a\sqrt{b^4 + 16a^2b^2} + \operatorname{arsinh}\left(\frac{4a}{b}\right) b^3}{8ab}$$

Both expressions aim to approximate the arc length of the parabola, they involve different methods of formulation since i didnt get the exact match of an expression. But i aim to make sure my derived expression tries to matches L_{ABC} .

B.1.5 matlab code

The following matlab code was used to perform the numerical evaluation:

matlab scripts/expression.m

```
% Define the expressions for E1 and E2
E1 = @(a, b) (4 * a * sqrt(b^4 + 16 * a^2 * b^2) + asinh(4 * a / b) * b^3) / (8 * a * b);
E2 = @(a, b) (1/2) * sqrt(b^2 + 16 * a^2) + (b^2 / (8 * a)) * log((4 * a + sqrt(b^2 + 16 * a^2)) / b);

% Initialize a cell array to store the results
results = {};

% Loop over the range of values for a and b
for a = 1:5
    for b = 1:5
        % Calculate E1 and E2
        e1 = E1(a, b);
        e2 = E2(a, b);
        % Calculate the difference
        difference = e1 - e2;
        % Set a threshold for small differences and treat them as 0
        if abs(difference) < 1e-10
            difference = 0.0;
        end
        % Append the results to the cell array
        results = [results; {a, b, e1, e2, difference}];
    end
end

% Convert the results into a table
T = cell2table(results, 'VariableNames', {'a', 'b', 'L', 'L_ABC', 'Difference'});

% Display the table
disp(T);
```

B.1.6 Numerical Evaluation

Using the code we evaluated both expressions for various values of a and b .
The results are summarized in Table

a	b	L	L_{ABC}	Difference
1	1	2.323392	2.323392	0.0
1	2	2.957886	2.957886	0.0
1	3	3.735939	3.735939	0.0
1	4	4.591174	4.591174	0.0
1	5	5.491150	5.491150	0.0
2	1	4.204658	4.204658	0.0
2	2	4.646784	4.646784	0.0
2	3	5.232421	5.232421	0.0
2	4	5.915771	5.915771	0.0
2	5	6.668527	6.668527	0.0
3	1	6.153288	6.153288	0.0
3	2	6.498059	6.498059	0.0
3	3	6.970176	6.970176	0.0
3	4	7.536853	7.536853	0.0
3	5	8.176498	8.176498	0.0
4	1	8.123944	8.123944	0.0
4	2	8.409317	8.409317	0.0
4	3	8.807604	8.807604	0.0
4	4	9.293568	9.293568	0.0
4	5	9.850171	9.850171	0.0
5	1	10.104730	10.104730	0.0
5	2	10.349698	10.349698	0.0
5	3	10.695939	10.695939	0.0
5	4	11.123014	11.123014	0.0
5	5	11.616959	11.616959	0.0

Table B.1: Comparison of L and L_{ABC} with differences

B.1.7 Conclusion

The primary objective of this analysis was to derive a function $f(x)$ whose arc length would graphically correspond to the expression for L_{ABC} as provided in the question. Initially, the aim was to find a suitable mathematical function that, when graphed, would exhibit the same arc length as that described by L_{ABC} .

In pursuit of this goal, we began with a simple parabolic shape and formulated the equation:

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

where the parameters a and b represent the height and width of the parabola, respectively. This function was chosen because its general form would allow us to manipulate the height and width as required by the problem.

To compute the arc length of this parabola, we substituted it into the general formula for arc length:

$$L = \int_{-\frac{b}{2}}^{\frac{b}{2}} \sqrt{1 + \left(\frac{d}{dx} \left(-\left(\frac{2\sqrt{a}}{b}x\right)^2 + a\right)\right)^2} dx$$

After performing the integration, we derived the following expression for the arc length:

$$L = \frac{4a\sqrt{b^4 + 16a^2b^2} + \operatorname{arsinh}\left(\frac{4a}{b}\right) b^3}{8ab}$$

This derived arc length expression was then compared to the given formula for L_{ABC} :

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

Upon performing numerical evaluations for various values of a and b , we observed that both arc length expressions provided identical results, with the differences being negligible. The numerical comparison confirmed that the arc length derived from our function matches the given formula for L_{ABC} , thus demonstrating that the function:

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

correctly models the desired parabola, and its arc length matches the expression for L_{ABC} derived from the problem statement.

In conclusion, we have successfully achieved the objective of finding a function whose arc length corresponds to the given expression for L_{ABC} . Through rigorous derivation and numerical verification, we have shown that both expressions for the arc length are equivalent. Therefore, we conclude that:

$$L \equiv L_{ABC}$$

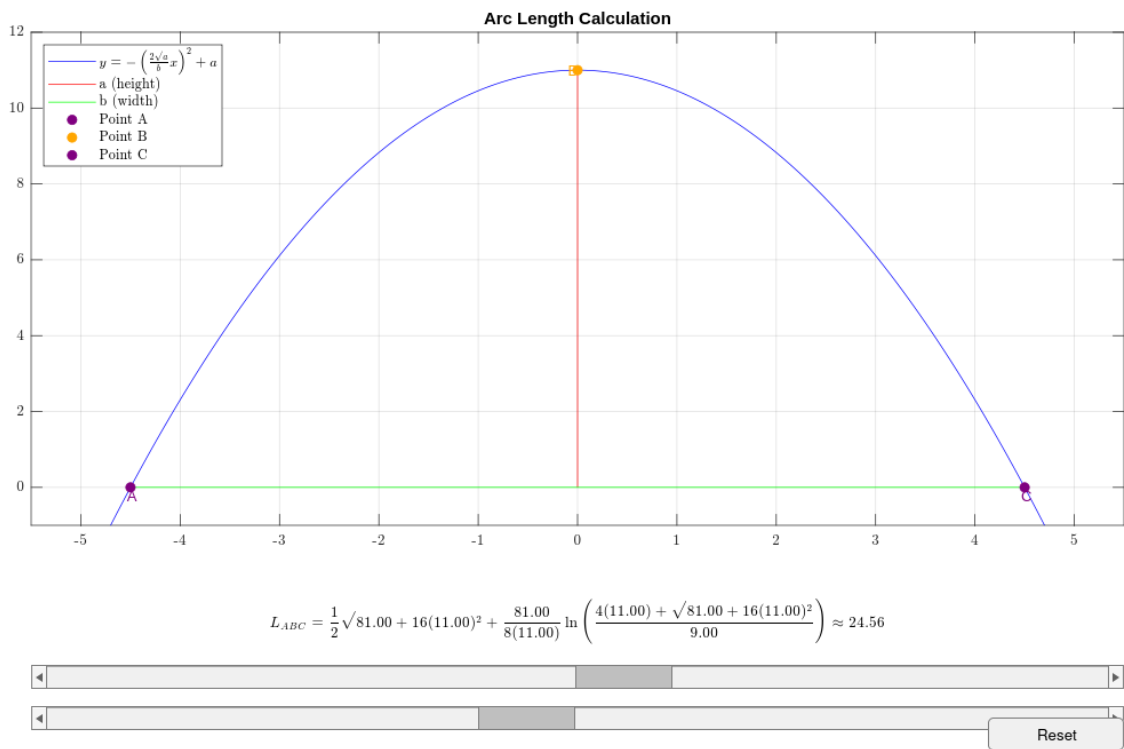
This result not only validates the correctness of the derived function but also demonstrates that different methods of obtaining the arc length lead to the same result, confirming the equivalence of the expressions. This completes the analysis and fulfills the original goal set out in the advanced method i aimed to grasp.

When implementing this in a program:

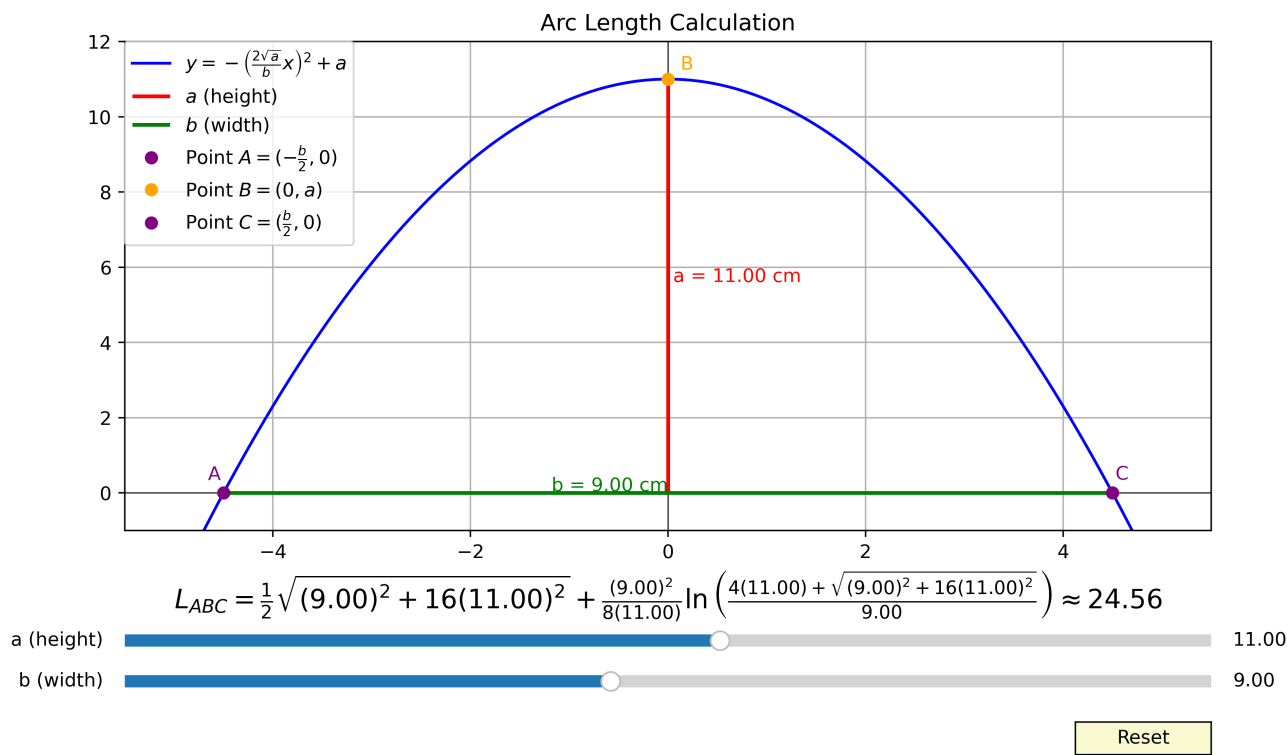
- Ensure that when there are no limit parameters for the inputs a and b , the conditions $a > 0$ and $b \neq 0$ are appropriately handled.
- Update the plot of $y = -\left(\frac{2\sqrt{a}}{b}x\right)^2 + a$ so that the sliders for a and b directly control the height and width, respectively.
- Calculate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

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

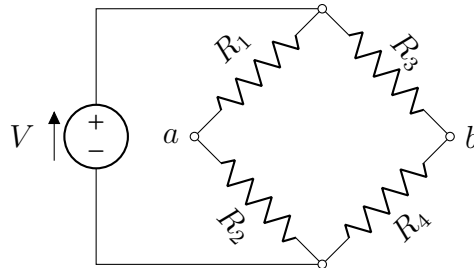
The voltage difference V_{ab} between points a and b in the Wheatstone bridge circuit is:

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

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

```
% Function to calculate Wheatstone bridge voltage
function Vab=calculate_Wheatstone(V,R1,R2,R3,R4)
    Vab=V*(R1*R3-R2*R4)/((R1+R2)*(R3+R4));
end

% Display formula for context
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 to confirm user input
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

% Input voltage value
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;
    end
end
```

```

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
end

% Function to prompt for resistor values
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

% Get resistor values and calculate Wheatstone bridge voltage
[R1,R2,R3,R4]=promptForResistors();
fprintf('\nVoltage value : V=%.2f\n',V);
fprintf('Resistor values: R1=%.2f, R2=%.2f, R3=%.2f, R4=%.2f\n',R1,R2,R3,R4);
Vab=calculate_Wheatstone(V,R1,R2,R3,R4);

```

```
fprintf('The calculated voltage V_ab across the Wheatstone Bridge is: %.2f volts\n',
Vab)
```

Workspace

Name	Value	Size	Class
R1	120.6000	1x1	double
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

Command Window

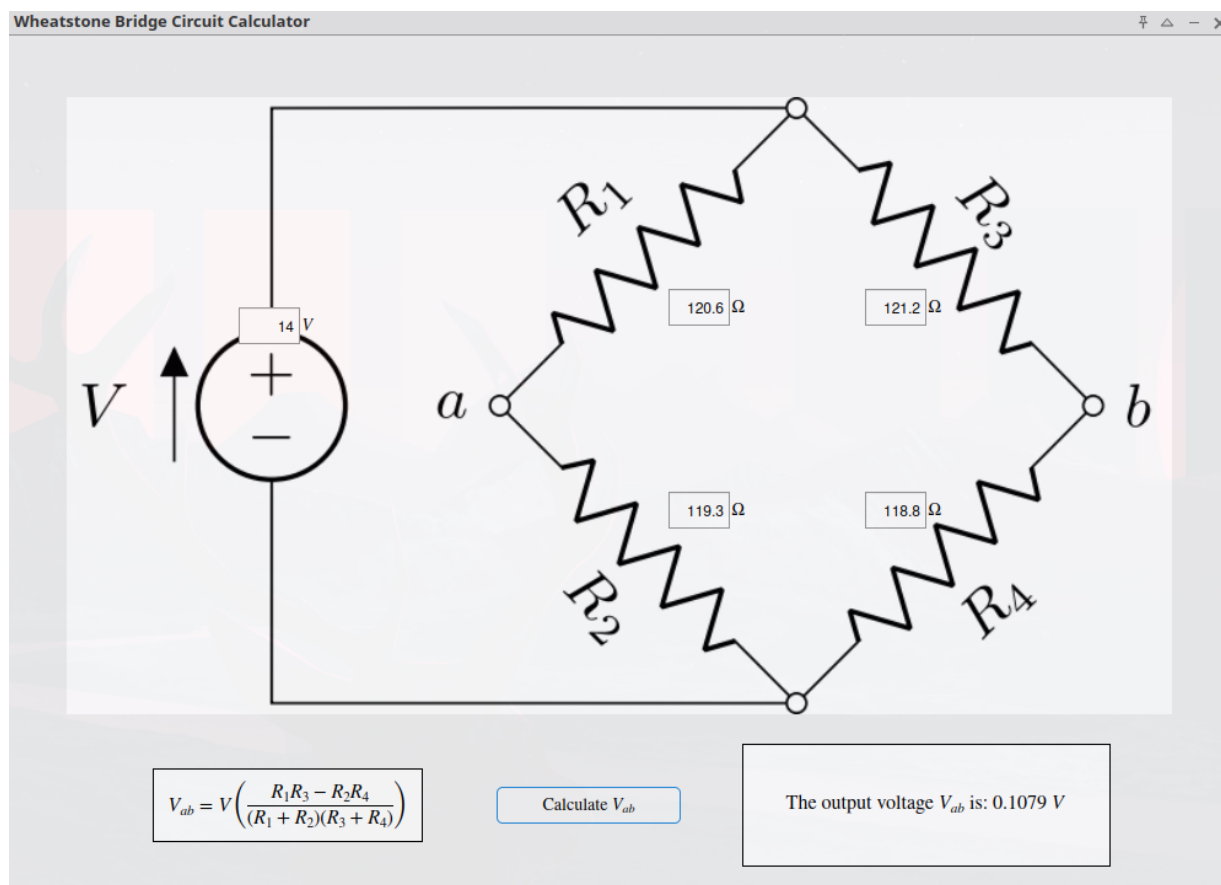
```
>> q2
The formula for voltage Vab in the Wheatstone Bridge is:
Vab = V * (R1 * R3 - R2 * R4) / ((R1 + R2) * (R3 + R4))
-----
Enter the value of V (voltage in volts):
14
-----
Enter the value of R1 (in ohms):
120.6
-----
Enter the value of R2 (in ohms):
119.3
-----
Enter the value of R3 (in ohms):
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
>> |
```

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

```

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

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

% User input section
T_normal=input('Enter the normal body temperature in degrees (usually 36.6 degrees): ');
T_s=input('Enter the surrounding temperature (T_s) in degrees (roomtemp is avg 20.5 degrees): ');
T_0=input('You have seen the body, enter the body temperature (T_0) in degrees: ');
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 degrees: ');
t1=input('Enter the time interval (in hours) between T_0 and T_1: ');

% Display formula and input values
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('-----');

% Calculate cooling constant and time since death
k=calculateCoolingConstant(T_1,T_s,T_0,t1);
t_death=calculateTimeSinceDeath(T_1,T_s,T_normal,k);
disp('-----');

% Calculate and display estimated time of death
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))

```

```

Enter the normal body temperature in degrees (usually 36.6 degrees): >>36.6
Enter the surrounding temperature (T_s) in degrees (roomtemp is avg 20.5 degrees): >>20.5
You have seen the body, enter the body temperature (T_0) in degrees: >>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 degrees: >>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

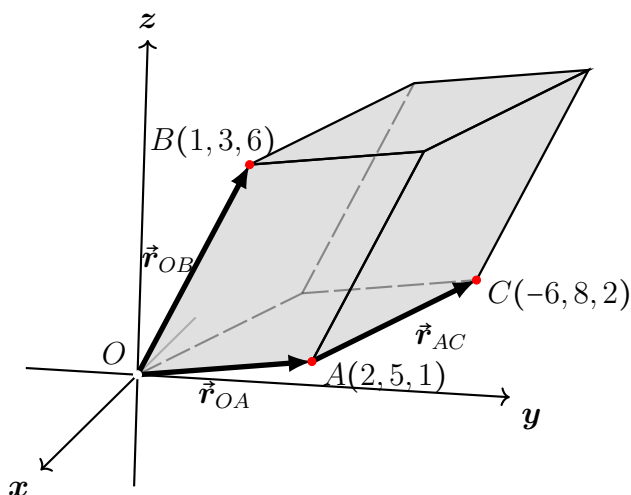
```

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

```
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)
```

```
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:

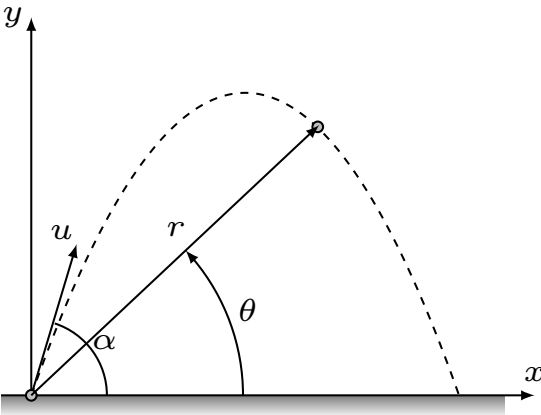
$$x(t) = v_0 \cos \alpha \cdot t \qquad y(t) = v_0 \sin \alpha \cdot t - \frac{1}{2}gt^2$$

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

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

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 $t = 1, 6, \dots, 31 \text{ sec}$.



Here, I present my past knowledge and comprehension for the equations for projectile motion.

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 / $s(t)$	$s_x(t) = u_x t$	$s_y(t) = u_y t - \frac{1}{2}gt^2 + h$

Time of Flight / T / $s_y(t) = 0$

$\frac{1}{g} \left(u_y + \sqrt{u_y^2 + 2gh} \right)$

Parameter	x	y
Velocity at Separation / $v(T)$	$v_x(T) = u_x$	$v_y(T) = \sqrt{u_y^2 + 2gh}$
Range / R / $s_x(T)$	$\frac{u_x}{g} \left(u_y + \sqrt{u_y^2 + 2gh} \right)$	$-h$
Maximum Height / H / $s(v_y(t) = 0)$	$\frac{u_y u_x}{g}$	$\frac{u_y^2}{2g} + h$

Table B.2: Derivations of Projectile Motion Parameters and Their Component Forms

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

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: Energy Components in Projectile Motion (Non-Drag)

I will not be implementing drag components (air-resistance), as they are complex and at this point i know i am doing more than what the question asks. In so I will focus solely on the motion parameters rather than energy quantities in the script.

matlab scripts/q5.m

```
% Constant(s)
g=9.81;

% 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: ');
fprintf('\n----- Values of r(t) and theta(t) for t = 1,2,3,...31
-----\n\n')

% initial calculations
theta_rad=deg2rad(theta);
u_x=u*cos(theta_rad);
u_y=u*sin(theta_rad);

% Compute flight characteristics
T=(u_y+sqrt(u_y^2+2*g*h))/g;
R=(u_x/g)*(u_y+sqrt(u_y^2+2*g*h));
H=(u_y^2/(2*g))+h;

% Generate time vector and calculate displacements
t=linspace(0,T,100); % since i named the time of flight its always gonna plot until
it hits the ground
tp = 1:1:31;
s_x=u_x*t;
s_y=(u_y*t)-(0.5*g*t.^2)+h;
```

```

% Calculate radial distance and angle
r=sqrt(s_x.^2+s_y.^2);
theta_t=atan2(s_y,s_x);

x_tp=zeros(size(tp));
y_tp=zeros(size(tp));
r_tp=zeros(size(tp));
theta_tp=zeros(size(tp));

for i = 1:length(tp)
    tc=tp(i);
    x_tp(i)=u_x*tc;
    y_tp(i)=(u_y*tc)-(0.5*g*tc.^2)+h;
    r_tp(i) = sqrt(x_tp(i)^2 + y_tp(i)^2);
    theta_tp(i) = rad2deg(atan2(y_tp(i), x_tp(i)));
    fprintf('r(%d) = %.2f, theta(%d) = %.2f\n',tc,r_tp(i),tc,theta_tp(i));
end

fprintf('\n-----Simulation of full time of full flight
-----\n\n')
% Find maximum values for radial distance and angle
[max_r, max_r_index]=max(r);
max_time_r=t(max_r_index);
[max_theta,max_theta_index]=max(theta_t);
max_time_theta=t(max_theta_index);
max_theta_deg=rad2deg(max_theta);

% Calculate final velocity components and magnitude
v_x_T=u_x;
v_y_T=sqrt(u_y^2+2*g*h);
v_T=sqrt(v_x_T^2+v_y_T^2);

% Display calculated results
fprintf('Time of Flight (T): %.2f seconds\n',T);
fprintf('Range (R): %.2f meters\n',R);
fprintf('Max Height (H): %.2f meters\n',H);
fprintf('Velocity at separation (v(T)): %.2f m/s\n',v_T);
fprintf('Maximum radial distance (r(t)): %.2f meters at t = %.2f seconds\n',max_r,
    max_time_r);
fprintf('Maximum angle (theta(t)): %.2f degrees at t = %.2f seconds\n',max_theta_deg,
    max_time_theta);

% Set up the figure
figure('Position',[100,100,800,600]);

% gudlines
function applyAxesFormatting(ax)
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    ax.XMinorGrid = 'on';
    ax.YMinorGrid = 'on';
    ax.XMinorTick = 'on';
    ax.YMinorTick = 'on';
    ax.TickDir = 'out';
    ax.FontName = 'Calibri';
    ax.FontSize = 12;
end

% Create figure with white background
fig = figure;
fig.Color = 'w';

```

```

% Plot trajectory in Cartesian coordinates
subplot(2,1,1);
plot(s_x, s_y, 'b-', 'LineWidth', 2);
title('Projectile Motion Trajectory');
xlabel('Displacement in X (m)');
ylabel('Displacement in Y (m)');
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;
applyAxesFormatting(gca);

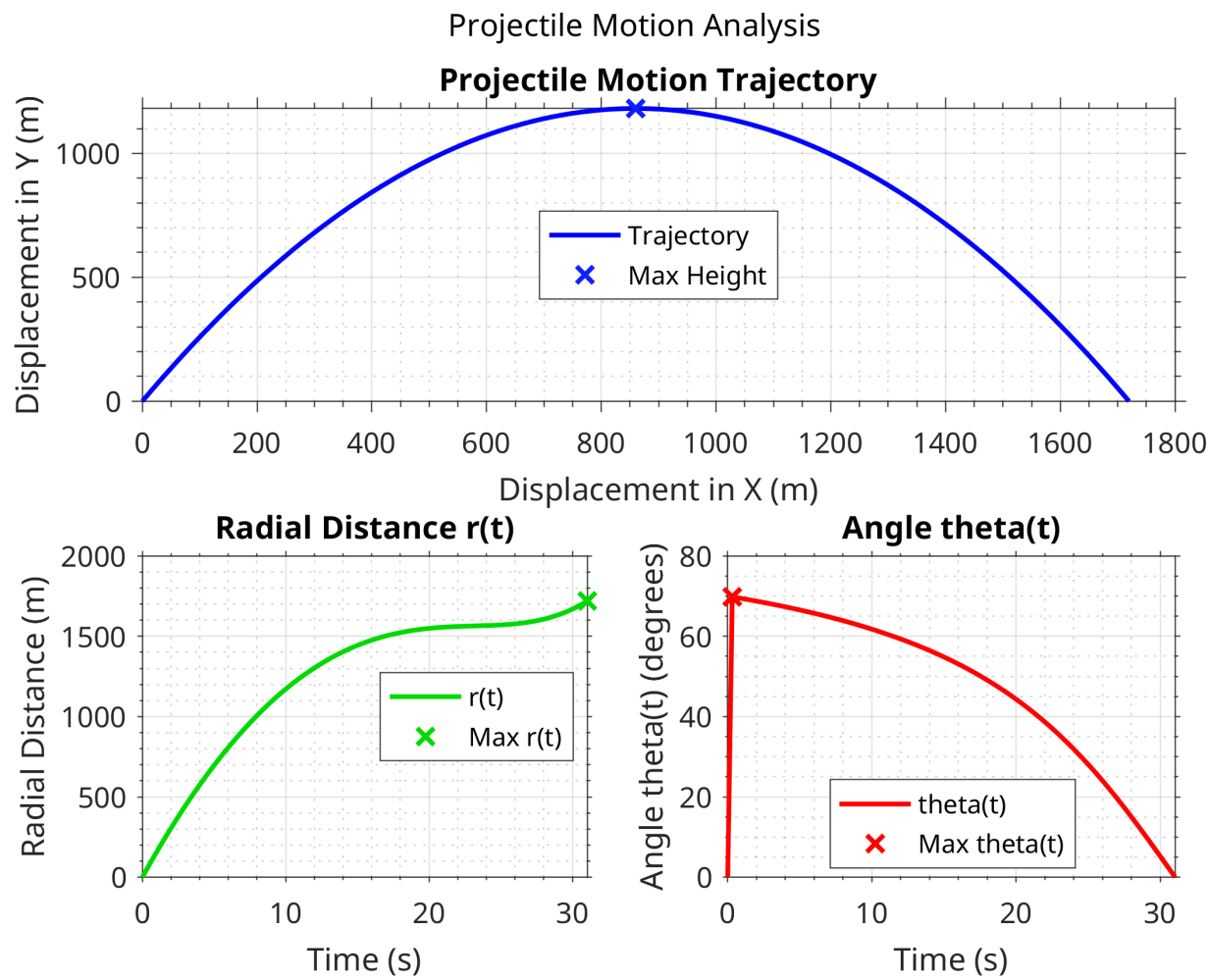
% Plot radial distance over time
subplot(2,2,3);
plot(t, r, 'Color', [0, 0.85, 0], 'LineWidth', 2, 'DisplayName', 'r(t)');
hold on;
plot(max_time_r, max_r, 'x', 'MarkerSize', 10, 'LineWidth', 2, 'Color', [0, 0.85, 0],
     'DisplayName', 'Max r(t)');
xlabel('Time (s)');
ylabel('Radial Distance (m)');
title('Radial Distance r(t)');
legend('Location', 'best');
hold off;
applyAxesFormatting(gca);

% Plot angle theta(t) over time
subplot(2,2,4);
plot(t, rad2deg(theta_t), 'r-', 'LineWidth', 2, '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)');
legend('Location', 'best');
hold off;
applyAxesFormatting(gca);

% Set overall title for the figure
sgtitle('Projectile Motion Analysis');

% Save figure as PNG with high resolution
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

----- Values of r(t) and theta(t) for t = 1,2,3,...31 -----

r(1) = 157.40, theta(1) = 69.39
r(2) = 305.64, theta(2) = 68.74
r(3) = 444.77, theta(3) = 68.05
r(4) = 574.88, theta(4) = 67.32
r(5) = 696.03, theta(5) = 66.55
r(6) = 808.33, theta(6) = 65.72
r(7) = 911.86, theta(7) = 64.83
r(8) = 1006.75, theta(8) = 63.88
r(9) = 1093.13, theta(9) = 62.86
r(10) = 1171.16, theta(10) = 61.76
r(11) = 1241.00, theta(11) = 60.59
r(12) = 1302.87, theta(12) = 59.31
r(13) = 1356.99, theta(13) = 57.94
r(14) = 1403.65, theta(14) = 56.45
r(15) = 1443.17, theta(15) = 54.84
r(16) = 1475.91, theta(16) = 53.08
r(17) = 1502.33, theta(17) = 51.17
r(18) = 1522.92, theta(18) = 49.09
r(19) = 1538.31, theta(19) = 46.82
r(20) = 1549.20, theta(20) = 44.33
r(21) = 1556.42, theta(21) = 41.62
r(22) = 1560.95, theta(22) = 38.66
r(23) = 1563.92, theta(23) = 35.43
r(24) = 1566.62, theta(24) = 31.92
r(25) = 1570.52, theta(25) = 28.12
r(26) = 1577.25, theta(26) = 24.03
r(27) = 1588.60, theta(27) = 19.66
r(28) = 1606.45, theta(28) = 15.04
r(29) = 1632.69, theta(29) = 10.22
r(30) = 1669.19, theta(30) = 5.24
r(31) = 1717.63, theta(31) = 0.18

-----Simulation of full time of full flight-----

Time of Flight (T): 31.04 seconds
Range (R): 1719.60 meters
Max Height (H): 1181.14 meters
Velocity at separation (v(T)): 162.00 m/s
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, $R = 0.08206 \text{ (L atm)/(mol K)}$ 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 $T = 300 \text{ K}$. (For nitrogen gas $a = 1.39 \text{ (L}^2 \text{ atm)/mol}^2$ and $b = 0.0391 \text{ L/mol}$.) Create a vector with values of V for $0.1 \leq V \leq 1 \text{ L}$, 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

```
% Function to calculate pressure using the Ideal Gas Law
function P1 = idealgas(n, R, T, V)
    P1 = n * R * T / V;
end

% Function to calculate pressure using the Van der Waals equation
function P2 = van(n, R, T, V, a, b)
    P2 = n * R * T / (V - n * b) - n^2 * a / V^2;
end

% Function to calculate the percentage error between pressures
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;

% Define volume range and initialize arrays
Vol = 0.1:0.02:1;
P1 = zeros(size(Vol));
P2 = zeros(size(Vol));
error_vals = zeros(size(Vol));

% Calculate pressures and errors for each volume
for i = 1:length(Vol)
    P1(i) = idealgas(n, R, T, Vol(i));
    P2(i) = van(n, R, T, Vol(i), a, b);
    error_vals(i) = percent_error(P1(i), P2(i));
end
```

```

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));
fprintf('Vol: %.2f [P1: %.4f, P2: %.4f, Error: %.2f%%]\n', Vol(i), P1(i), P2(i),
    error_vals(i));
end

% Find maximum error and corresponding volume
[max_error, idx] = max(error_vals);
V_max_error = Vol(idx);
fprintf('Maximum Error: %.2f%% at Volume: %.2f\n', max_error, V_max_error);

% Plot pressures vs volume
figure;
plot(Vol, P1, 'b-', 'LineWidth', 2, 'DisplayName', 'Ideal Gas Law');
hold on;
plot(Vol, P2, 'r-', 'LineWidth', 2, 'DisplayName', 'Van der Waals');
plot(Vol, P1, 'bx', 'LineWidth', 2, 'Markersize', 4);
plot(Vol, P2, 'rx', 'LineWidth', 2, 'Markersize', 4);

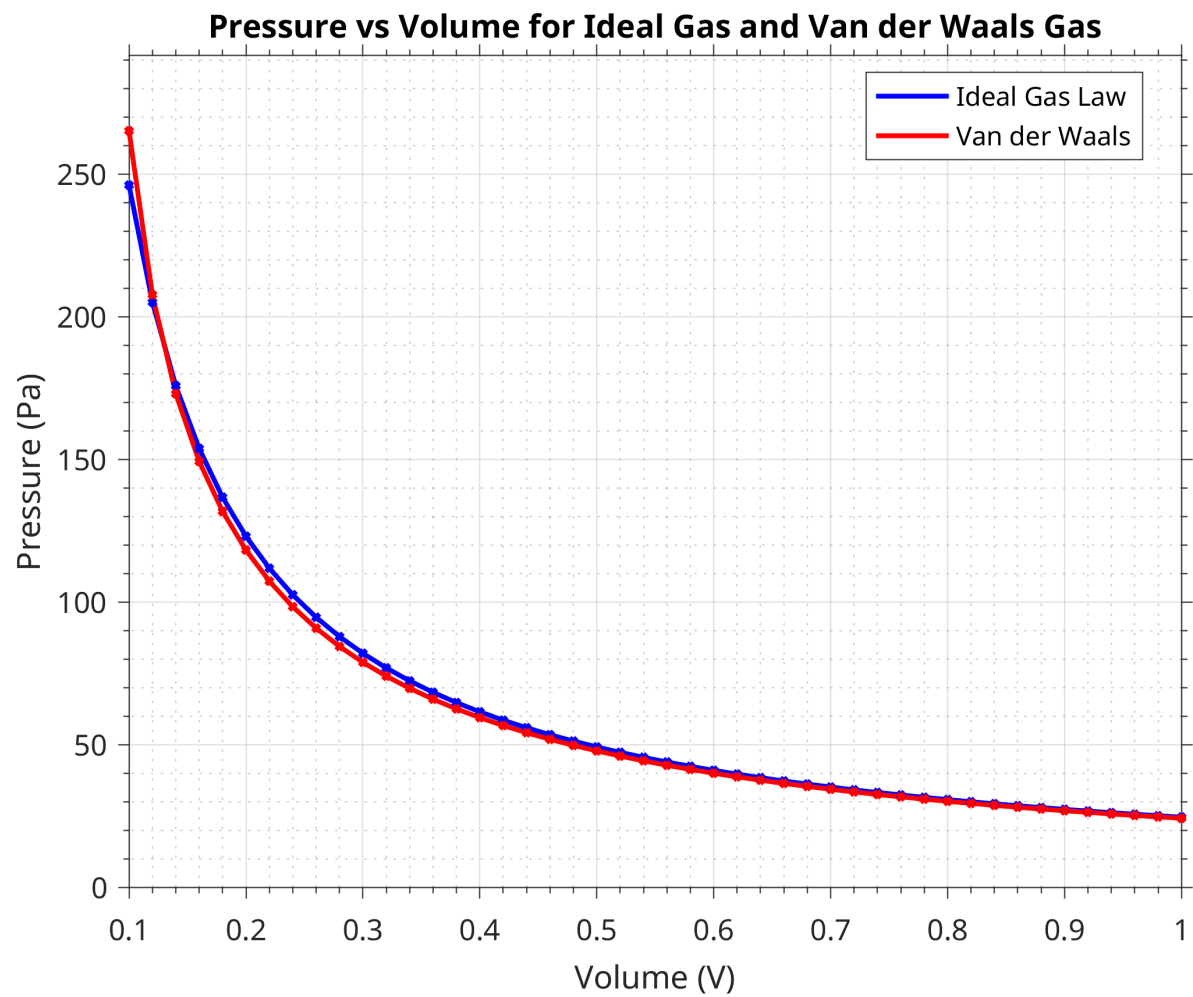
% Set labels, title, and legend
xlabel('Volume (V)');
ylabel('Pressure (Pa)');
title('Pressure vs Volume for Ideal Gas and Van der Waals Gas');
legend('Ideal Gas Law', 'Van der Waals', 'Location', 'best');
grid on;

% Apply consistent axis formatting
ax = gca;
ax.XGrid = 'on';
ax.YGrid = 'on';
ax.XMinorGrid = 'on';
ax.YMinorGrid = 'on';
ax.XMinorTick = 'on';
ax.YMinorTick = 'on';
ax.TickDir = 'out';
ax.FontName = 'Calibri';
ax.FontSize = 12;

% Set y-axis limits based on max pressure for scaling
ylim([0, max(max(P1), max(P2)) * 1.1]);

% Save the final plot as a PNG file
print('-dpng', '-r300', 'Pressurevolume.png');
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
```


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

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

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

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

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);
optimal_y=y_values(idx);

% Verify the result with Snell's Law
phi=atand(optimal_y/d_s);
alpha=atand((L-optimal_y)/d_s);
snell_ratio=sind(phi)/sind(alpha);
expected_ratio=v_run/v_swim;

% Display in console
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 essentially is satisfied.');
```

```
else
```

```
    disp('Snell's Law is not satisfied.');
```

```
end
```

```
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

```
% 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;
r=6.4/1000; % convert mm to m
A0=pi*r^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

fig = figure ;

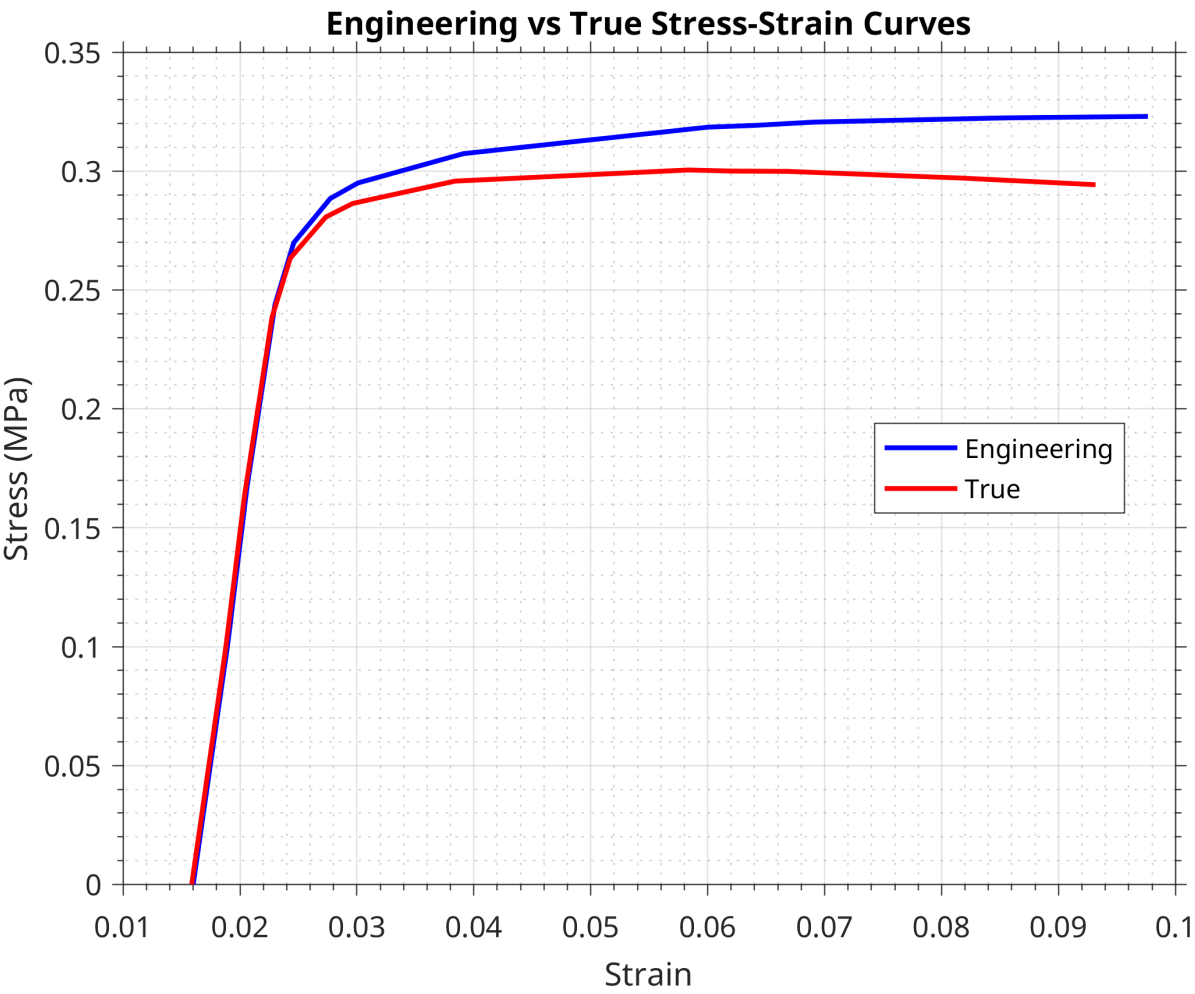
plot(eng_strain,eng_stress,'b-', 'Linewidth',2);
hold on;
plot(true_strain,true_stress,'r-', 'Linewidth',2);
xlabel('Strain');
ylabel('Stress (MPa)');
title('Engineering vs True Stress-Strain Curves');
legend('Engineering','True', 'Location', 'best');
grid on;
hold off;
ax = gca;

fig.Color='w';

ax.XGrid='on';
ax.YGrid='on';
ax.XMinorGrid='on';
ax.YMinorGrid='on';
ax.XMinorTick='on';
ax.YMinorTick='on';

ax.TickDir='out';
ax.FontName='Calibri';
ax.FontSize=12;

% Save as png to print
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

```
% Define time range
t = linspace(0, 4, 1000);

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

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

% guidelines
function applyAxesFormatting(ax)
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    ax.XMinorGrid = 'on';
    ax.YMinorGrid = 'on';
    ax.XMinorTick = 'on';
    ax.YMinorTick = 'on';
    ax.TickDir = 'out';
    ax.FontName = 'Calibri';
    ax.FontSize = 12;
end

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

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

```

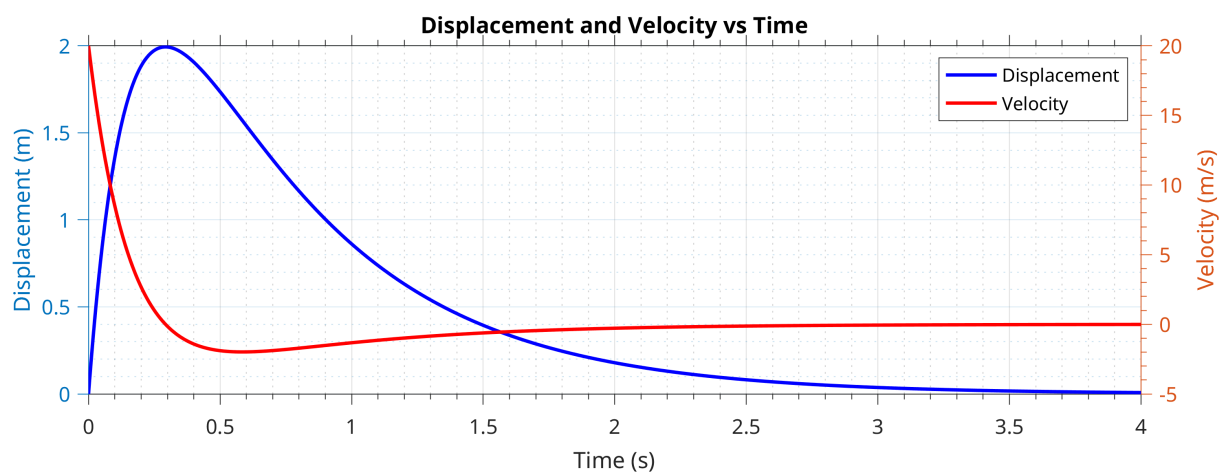
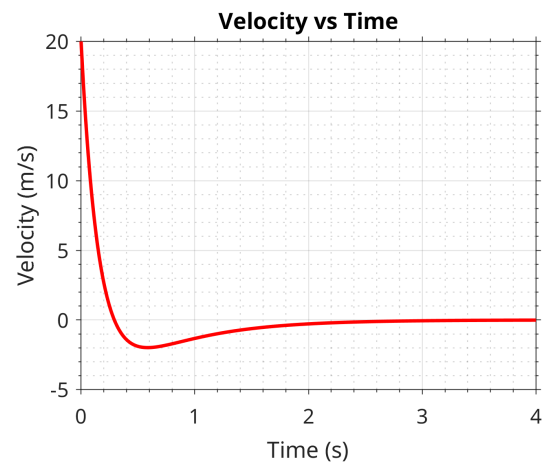
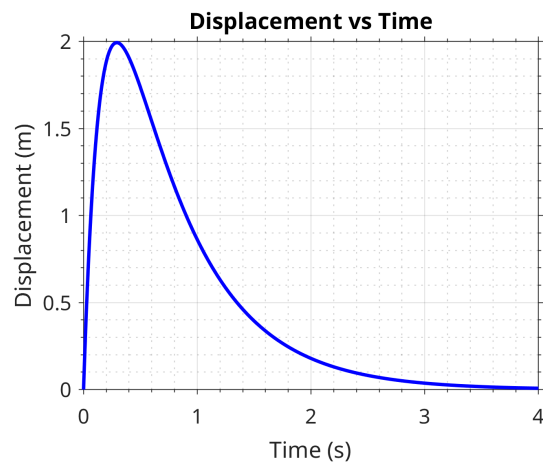
applyAxesFormatting(gca);

% Plot both displacement and velocity
subplot(2, 1, 2);

yyaxis left;
plot(t, x(t), 'b', 'LineWidth', 2, 'DisplayName', 'Displacement');
ylabel('Displacement (m)');
yyaxis right;
plot(t, v(t), 'r', 'LineWidth', 2, 'DisplayName', 'Velocity');
ylabel('Velocity (m/s)');
title('Displacement and Velocity vs Time');
xlabel('Time (s)');
grid on;
legend;
applyAxesFormatting(gca);

% Save as a PNG image
print('-dpng', '-r300', 'railroad_bumper_plots.png');

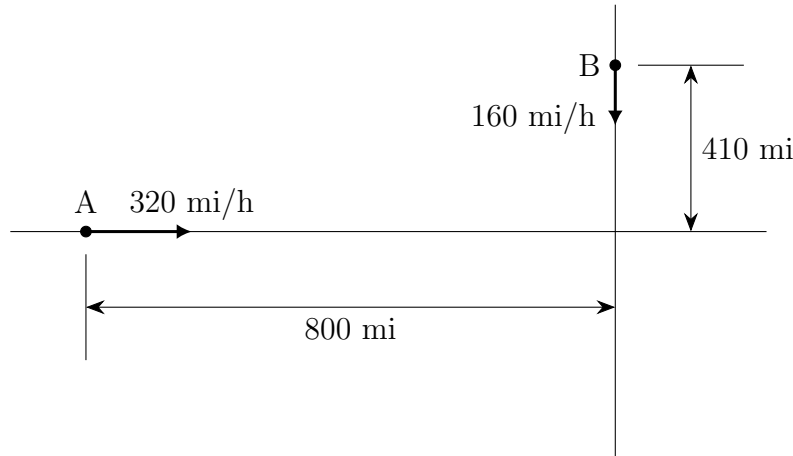
```



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

$$D(t) = \sqrt{(x_0 - V_A t)^2 + (y_0 - V_B t)^2}$$

- **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. we need in form of quadratic for `roots` function:

$$0 = (V_A + V_B)t^2 - 2(V_A x_0 + V_B y_0)t + (x_0^2 + y_0^2 - 30^2)$$

this yields the roots; using `min(root(coeff))` will give the first time the distance reaches 30 mi.

matlab scripts/q10.m

```
% Define constants
v_A=320;
v_B=160;
x0=800;
y0=410;

% Define time range (in hours)
t=0:0.01:3;
D=sqrt((x0-v_A*t).^2+(y0-v_B*t).^2);

% Plot D versus time
```

```

figure('Position',[100,100,800,600])
plot(t,D,'LineWidth',2)
hold on

% Mark minimum distance point
[min_D,min_index]=min(D);
t_min=t(min_index);
plot(t_min,min_D,'ro','MarkerSize',6,'MarkerFaceColor','r')
text(t_min+0.05,min_D,sprintf('Minimum: (%.2f hr, %.2f mi)',t_min,min_D),...
     'FontSize',12,'VerticalAlignment','top','Color','r')

% Add horizontal line showing 30 miles
yline(30,'--','Color','#7E2F8E','LineWidth',1.5);
text(0,30,'30 mi threshold','FontSize',12,'VerticalAlignment','bottom','
     HorizontalAlignment','left','Color','#7E2F8E')

% Find times when the distance is exactly 30 miles
coeff=[v_A^2+v_B^2,-2*(v_A*x0+v_B*y0),x0^2+y0^2-30^2];
t_30_all=roots(coeff);
t_30_all=t_30_all(t_30_all>0); % used as negative time values would not make sense
t_30_first=min(t_30_all);
t_30_second=max(t_30_all);

function applyAxesFormatting(ax)
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    ax.XMinorGrid = 'on';
    ax.YMinorGrid = 'on';
    ax.XMinorTick = 'on';
    ax.YMinorTick = 'on';
    ax.TickDir = 'out';
    ax.FontName = 'Calibri';
    ax.FontSize = 12;
end

% Mark the points where distance is exactly 30 miles
plot([t_30_first,t_30_second],[30,30],'o','MarkerSize',6,'MarkerFaceColor','#7E2F8E')
text(t_30_first,30,sprintf('%.2f hr',t_30_first),'FontSize',14,'VerticalAlignment','
     bottom','HorizontalAlignment','right','Color','#7E2F8E')
text(t_30_second,30,sprintf('%.2f hr',t_30_second),'FontSize',14,'VerticalAlignment','
     bottom','Color','#7E2F8E')

% Set up plot labels and title
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(end) 0 max(D)+50])
set(gca,'FontSize',12)
box on
applyAxesFormatting(gca);

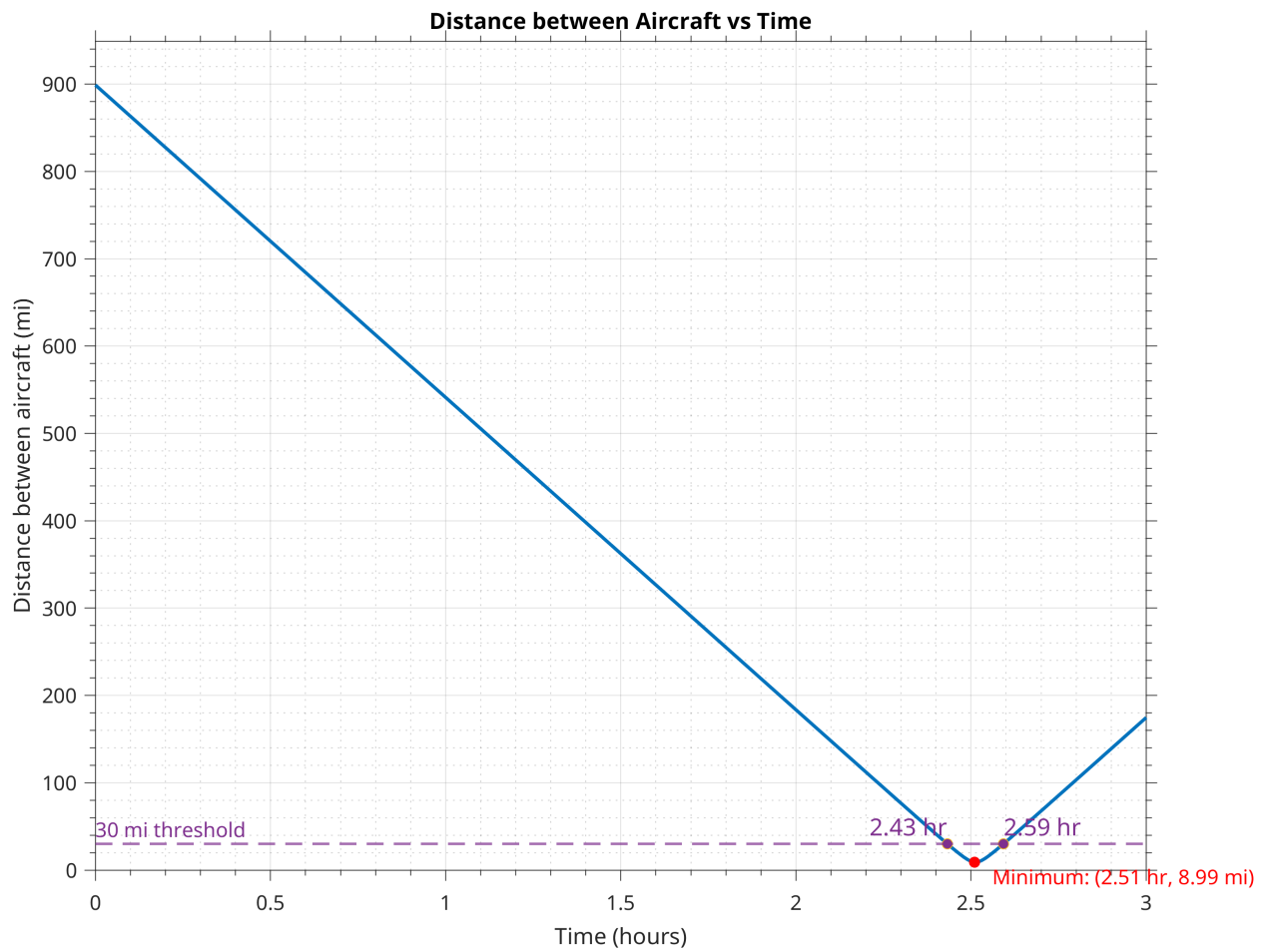
% Save the figure
print('-dpng','-r300','aircraft_distance_plot.png');

% Displaying results in console
initial_time=datetime('21:18:00','InputFormat','HH:mm:ss');
min_time=initial_time+hours(t_min);
first_30mi_time=initial_time+hours(t_30_first);
fprintf('Minimum distance: %.2f mi apart at %s (%.2f hours after 9:18 PM)\n',min_D,
        datestr(min_time,'HH:MM:SS PM'),t_min);

```



```
fprintf('Aircraft are first within 30 mi of each other at %s (%.2f hours after 9:18 PM)\n', datestr(first_30mi_time, 'HH:MM:SS PM'), t_30_first);
```



```
Minimum distance: 8.99 mi apart at 11:48:36 PM (2.51 hours after 9:18 PM)  
Aircraft are first within 30 mi of each other at 11:43:56 PM (2.43 hours after 9:18 PM)
```

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:

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

$$\sigma_{y'y'} = \sigma_{xx} + \sigma_{yy} - \sigma_{x'x'}$$

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

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

```
function [Strain]=StressTrans(S,theta)
    % split the vector into accessible parts for the equations
    sigma_xx=S(1);
    sigma_yy=S(2);
    tau_xy=S(3);

    % Calculate the equations
    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-sigma_xx_prime;
    tau_xy_prime=-(sigma_xx-sigma_yy)/2*sin(2*theta)+tau_xy*cos(2*theta);

    % Define Strain to return
    Strain=[sigma_xx_prime,sigma_yy_prime,tau_xy_prime];
end

% Get inputs
sigma_xx=input('Enter the stress component sigma_xx (in Pa): ');
sigma_yy=input('Enter the stress component sigma_yy (in Pa): ');
tau_xy=input('Enter the shear stress component tau_xy (in Pa): ');
theta_degrees=input('Enter the angle (in degrees): ');

% Convert into what will/can be put into the function
theta=deg2rad(theta_degrees);
S=[sigma_xx,sigma_yy,tau_xy];

% Call it and get the Strain out
[Strain]=StressTrans(S,theta);

% Display in console
fprintf('\n-----\n');
```

```
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));
disp('-----');
fprintf('Strain = [%.2f, %.2f, %.2f]\n\n',Strain(1),Strain(2),Strain(3));
```

```
Enter the stress component sigma_xx (in Pa): >>23
Enter the stress component sigma_yy (in Pa): >>56
Enter the shear stress component tau_xy (in Pa): >>34
Enter the angle (in degrees): >>93
```

```
-----
Transformed stresses:
sigma_x'x' = 52.36 Pa
sigma_y'y' = 26.64 Pa
tau_x'y' = -35.54 Pa
-----
Strain = [52.36, 26.64, -35.54]
```