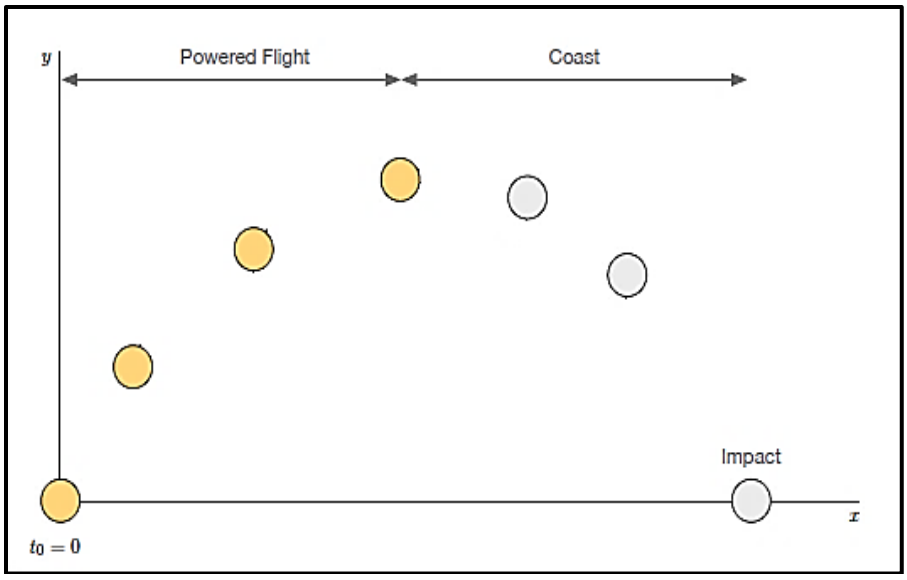
MEEM 4730 Project 2 Report

Simulation & verification of point mass rocket trajectory by mathematical modeling in Simulink & Physical modeling in Simscape



Created & Submitted by: Vrushaketu Mali

23-Feb-2020

Table of Contents

List of figures

Chapter 1. Introduction

The first step in the dynamic system simulation is to develop appropriate mathematical models of the system to be simulated. These models can either be constructed from the physics equations or experimental data. Apart from writing bunch of syntax in the MatLab code, there are two other ways for simulation. First is mathematical modeling in Simulink and second is physical modeling in Simscape.

Simulink is a model-based design environment integrated with MatLab used for modeling, simulating and analyzing multi-domain dynamic systems. It offers a way to solve the dynamic system equations numerically rather than requiring a code. Models contain blocks which are mathematical functions and signals which are lines connecting different block serving as carrying input and output values.

Simscape is an extension in the Simulink library which contains actual physical components, therefore complex multi-domain systems can be modeled without the need to build the mathematical equations.

In this project, we will be simulating the point mass rocket trajectory in Simulink by constructing the signal-based math model as well as in Simscape by constructing the component-based model. In order to ensure that both models are built correctly, we are going to compare the results of these simulation with closed form solution of given problem statement which is calculated through MatLab code.

Chapter 2. Summary

This project aims at creating a signal-based math-model in Simulink and component-based physical model in Simscape for simulating the point mass rocket trajectory and verifying the results of these simulations with different methods. The verification results, MatLab code description is discussed in detail in this project.

Accomplishment:

1. In this project, we have simulated the point mass rocket trajectory for two scenarios and verified the results of both scenarios.
2. MatLab script is written for setting up the simulation parameters as well as generating the closed form solution of the system.
3. Signal-based math model & component based physical model is built in Simulink & Simscape respectively for simulating the system.
4. For creating the models, we have used the buses and variant sub-systems in Simulink.
5. Results of scenario 1 of Simulink model are compared with the results of closed form solution and the conclusion is drawn from the results.
6. Results of scenario 2 of signal-based model are compared with the results of components-based model and thus appropriate conclusion is drawn.
7. Finally, we have verified the simulation by checking the total energy from burnout to impact time for both scenarios.

Chapter 3. Project Statement/Problem Statement

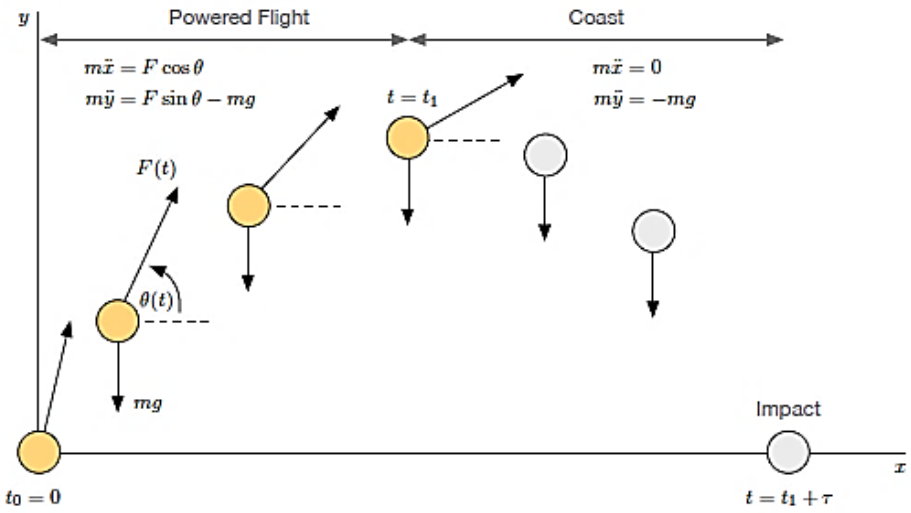


Figure 1: Project/Problem Statement

As shown in figure, Consider the rocket with mass m = 1000 kg that is launched at t = 0 seconds with a thruster that has controllable magnitude, F(t), and angle, θ(t). Its flight has two phases (1) Powered Flight and (2) Coast.

1. Create a simulation that can be used to evaluate how the rocket behaves for different time histories of θ(t) and F(t).
2. Using a free body diagram and Newton's 2nd Law, show the equations of motion of the system
3. In a single Simulink model, named rocketSim.slx implement this model in two ways (1) using a MATLAB Function block with a single integrator and (2) using Simscape Multibody.
4. Verify the simulation in following three ways –
   - 4.1 **Task 2a**: Set F = 15 m/s2 and θ = 56◦. Solve the equations above in closed form where the burn time is t1 = 5 seconds and the impact time is t2. Show that your closed form solution is nearly identical to the two integrated solutions, both the component-based and signal-based models. This means you will have selected an appropriate integration method and time step. Select these so that the error between your simulated and closed form state at burnout is less than 1 × 10−5 for all four states. Show a comparison between simulated and closed form x and y trajectories for both simulations and your justification for your integration method and time step.
   - 4.2 **Task 2b:** Set F = 15 m/s2 and θ = (56+20cos(2πt) ◦ during the powered phase, 0 ≤ t ≤ t1 where t1 = 5 seconds. Let the simulation run until impact, t = t2 seconds, or just a little beyond impact. Show that the rocket's x and y trajectories are identical for both the signal-based and component-based simulations. This shows that you've likely implemented both models correctly.
   - 4.3 **Task 2c:** The total energy of the rocket is E = T +V and is conserved after burnout. where T is its kinetic energy and V its potential energy. Harvesting the data from your simulation, show that energy is conserved between burnout and impact by showing that the total energy changes only a small amount.

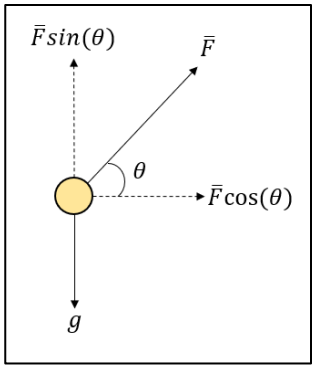
Chapter 4. Equations of motion of system

The first step in deriving the mathematical equations that govern a physical system is to draw the free-body diagram representing the system. This is done below for our point mass rocket system.

From Newton's second law, we know that the sum of the forces acting on a body is equal to the product of the mass of the body (m) and its acceleration (a).

$$\boldsymbol{F = ma}$$

In this case, the thrust acting on the point mass is $F(t)$ which is inclined with horizontal at an angle of $\theta(t)$.
The force in the horizontal direction is the resolved component of $F(t)$ in horizontal direction which is $F(t)\cos\theta(t)$.
The forces in the vertical direction is the resolved component of $F(t)$ in vertical direction which is $F(t)\sin\theta(t)$ in upward direction and the weight of point mass in downward direction.
Let us define the normalized thrust $\boldsymbol{\bar{F} = F / m}$ as thrust per unit mass which will have unit as that of acceleration.
Applying Newton's second law in both directions based on the below free-body diagram leads to the following governing equations for the system.



$$\sum F_x = m\ddot{x} = F\cos(\theta)$$
$$\ddot{x} = \bar{F}cos(\theta) \quad \text{---------- 1}$$

$$\sum F_y = m\ddot{y} = F\sin(\theta) - mg$$
$$\ddot{y} = \bar{F}sin(\theta) - g \quad \text{---------- 2}$$

Figure 2: Free body diagram

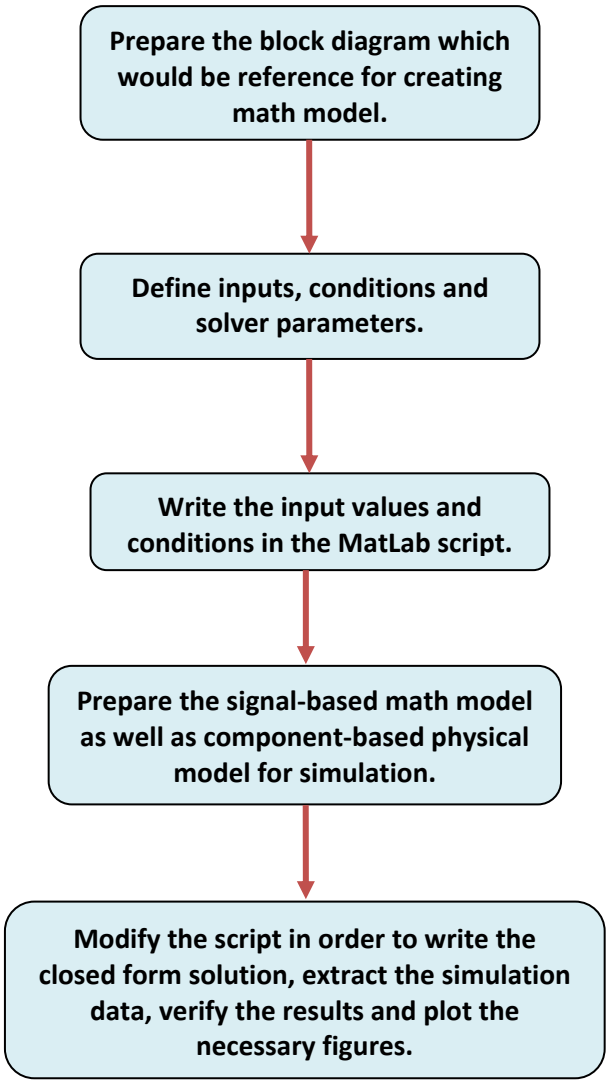
Let us represent this system in state-space representation and thus form a matrix of state-space variables. This is second order differential equation system with 2 different variables X & Y, therefore it will have total 4 state variables.

Say, $x_1 = x$ _____ Horizontal displacement of point mass rocket
$x_2 = y$ _____ Vertical displacement of point mass rocket
$x_3 = \dot{x_1} = \dot{x}$ _____ Horizontal velocity of point mass rocket
$x_4 = \dot{x_2} = \dot{y}$ _____ Vertical velocity of point mass rocket
$x_5 = \dot{x_3} = \ddot{x} = \bar{F}cos(\theta)$ _____ From equation 1 ___ Horizontal acceleration of point mass rocket
$x_6 = \dot{x_4} = \ddot{y} = \bar{F}sin(\theta) - g$_____ From equation 2___ Vertical acceleration of point mass rocket

So the state-space matrix which is $\dot{x}$ can be written as $\dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ \bar{F}cos(\theta) \\ \bar{F}sin(\theta) - g \end{bmatrix}$

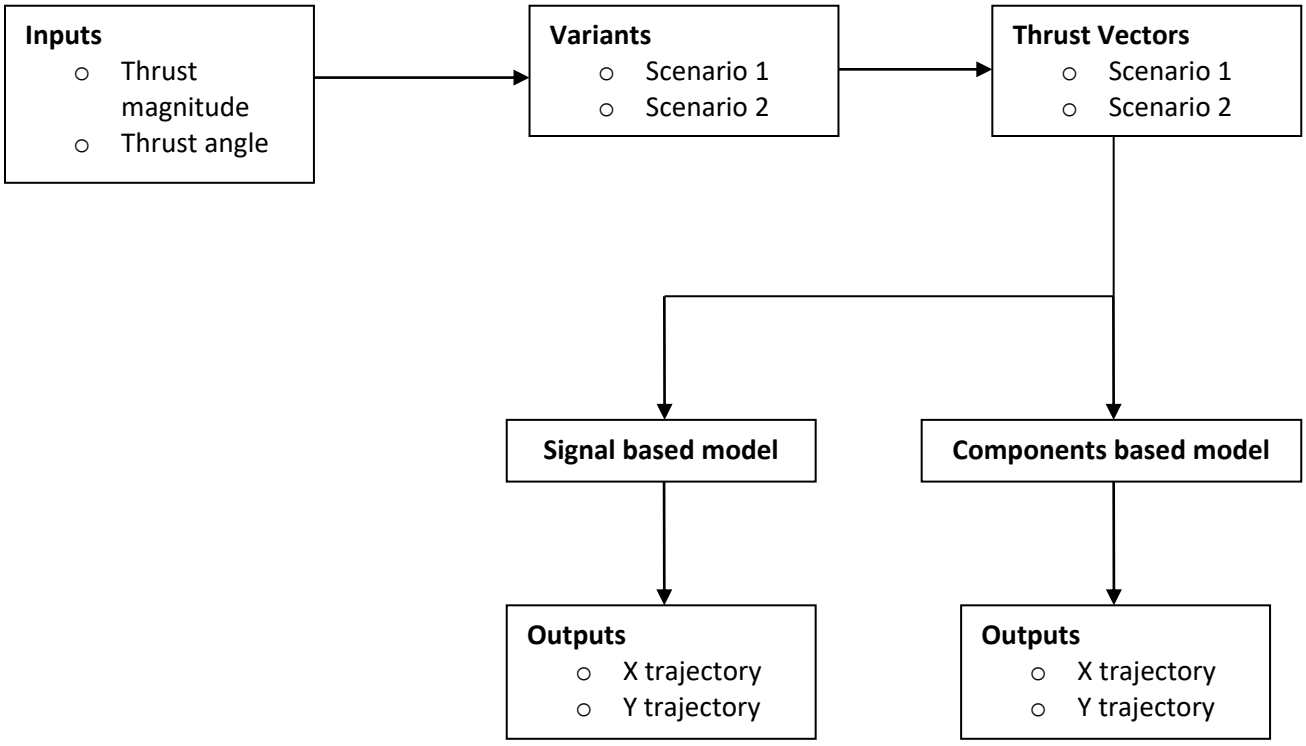
Chapter 5. Approach for simulation

Below flowchart shows the approach we would follow for simulating the system.



5.1 Creating a block diagram:

In this project, we are going to make the use of variant subsystems in order to jump from scenario 1 to scenario 2. Also, as there are different force angle values for two scenarios, it will lead to different thrust vectors for those two scenarios so we are also going to make the use of buses in Simulink for carrying those signals.
Let us create the block diagram starting from the inputs to outputs.



Chapter 6. Code description

6.1 Define Inputs, conditions & solver parameters

First of all, let us write the script for clearing the workspace, closing all open figures and opening the relevant Simulink model.
After that, start writing the parameters in the script.

Clean up

```
clearvars;  % clear the workspace
close all  % close all open figures
clc % clear the command window
```

Open Simulink model

```
open_system('rocketSim');  % open simulink model with name 'rocketSim'
```

Physical Constants

```
planet.g = 10; % This is the value of gravitational acceleration (m/s^2)
```

Rocket Parameters

```
rocket.m   = 1000;      % mass of rocket (kg)
rocket.ics = [0;0;0;0]; % initial conditions x,y,x velocity & y velocity
rocket.t1  = 5;         % Burn time for rocket in seconds after which there will
not be any thrust.
```

Thrust Input Parameters

```
thr.F = 15000; % thrust amplitude (N)
thr.tht = 56;  % thrust angle (deg)
thr.w   = 2*pi; % thrust angle frequency (rad/s) (For scenario 2 only)
thr.amp = 20; % amplitude of time varying component of thrust ang (deg) (For
scenario 2 only)
```

Solver Parameters

```
simPrm.h      = 0.0001; % This is time step value in seconds.
simPrm.solTyp = 'Fixed-step'; % This is the integration solver type either fixed
step or variable step
simPrm.sol    = 'ode3'; % define integration method as an input for simulink
solver
```

Set Simulink Configuration

```
set_param('rocketSim','SolverType',simPrm.solTyp);  % set this solver type in
simulink parameters
set_param('rocketSim','Solver'    ,simPrm.sol);     % set this integration method
in simulink solver
```

6.2 Define the buses

Buses are used in Simulink to carry multiple signals which can be utilized for simulating more than one models. In our project, as we need thrust vectors for both the models, we will create the bus for carrying the appropriate signals which are thrust vector for both scenarios, thrust magnitude and thrust angle.

Below is the structure of the buses which we have defined in our model

Bus name: Thruster ← Main top level bus
Bus elements: Thrust_vector
Bus name : ThrustMagandAng ← 2nd level bus inside top bus
Bus elements:
Bus name : Scenario 1 ← 3rd level bus inside 2nd bus
Bus elements : Thrust Magnitude
Thrust Angle
Bus name : Scenario 2 ← 3rd level bus inside 2nd bus
Bus elements : Thrust Magnitude
Thrust Angle

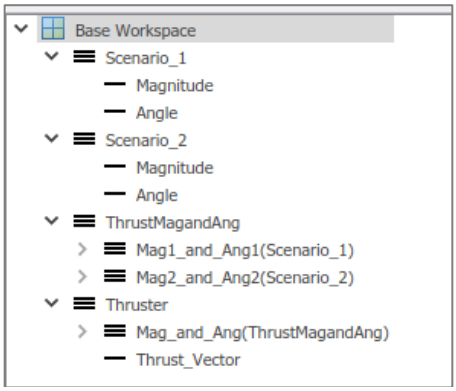


Figure 3: Bus definitions in workspace

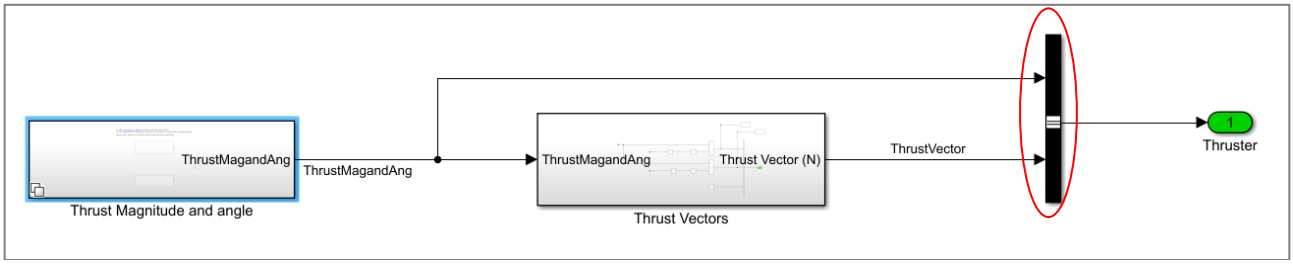


Figure 4: Top level bus creator in the model

We will save these bus definitions in .mat format and then load it before running the simulation by entering the command in MatLab script.

Load Bus Defs

```
load busDefs.mat  % extract & load the bus definitions in workspace
```

6.3 Create the variant sub-systems

As earlier said, since there are two scenarios we need to create the thrust vector for both scenarios and make use of both for deriving the outputs. The best possible way in this case is two create variant subsystems and switch them accordingly the command given in MatLab script.

6.3.1 What is variant sub-system?

Variant sub-system is simply a block which is used to create a single model that caters the multiple simulation requirements. For example, if we want to simulate a system for two or more types of inputs while allowing to switch between them then we create a variant sub-system which includes the blocks of all those inputs and are executed as per the specific command or scenario defined in the MatLab script.

In our project, we have two scenarios for thrust magnitude and angle which we will name as Constant_F_VSS and Varying_F_VSS which are used for solving both models. So, we will create the variant subsystem involving both scenario inputs. Below figure shows the variant subsystem created.

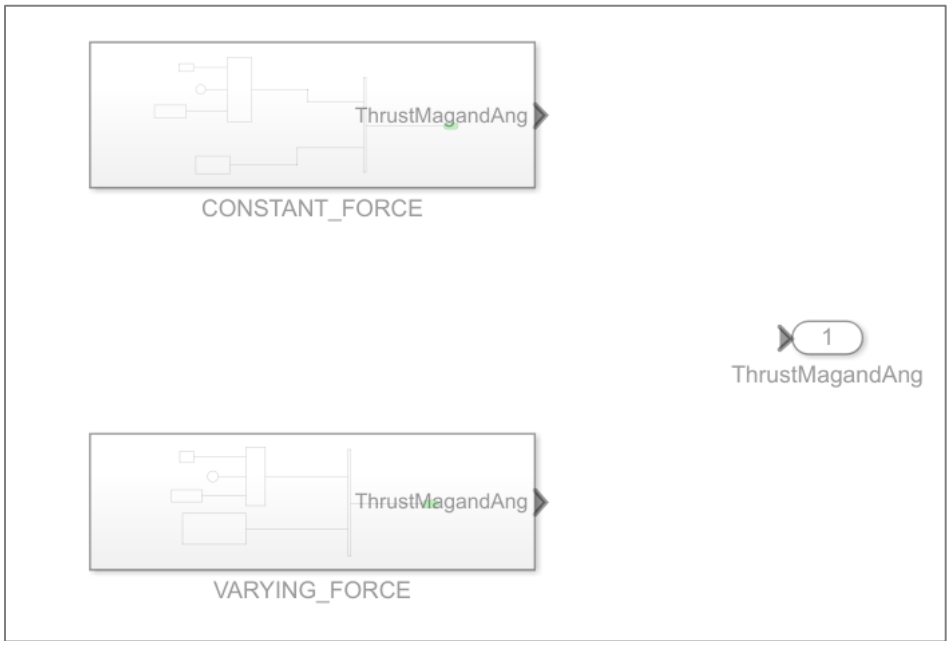


Figure 5: Variant sub-system in the model

Variant subsystems for different thrust scenarios

```
 % SCENARIO = 1; % constant thrust, constant angle
 % SCENARIO = 2; % varying thrust, varying angle

 CONSTANT_F_VSS = Simulink.Variant('SCENARIO==1'); % define scneario 1 as constant
force input
 VARYING_F_VSS = Simulink.Variant('SCENARIO==2');  % define scenario 2 as variable
force input
```

6.4 Signal-based math model

Below figure shows the signal-based model. It has a matlab function which takes the input as Force (Thrust magnitude and angle from the buses) and time clock. It gives the output as $\dot{x}$ matrix which we discussed earlier.

Thus, there are four outputs, x , y, v_x v_y

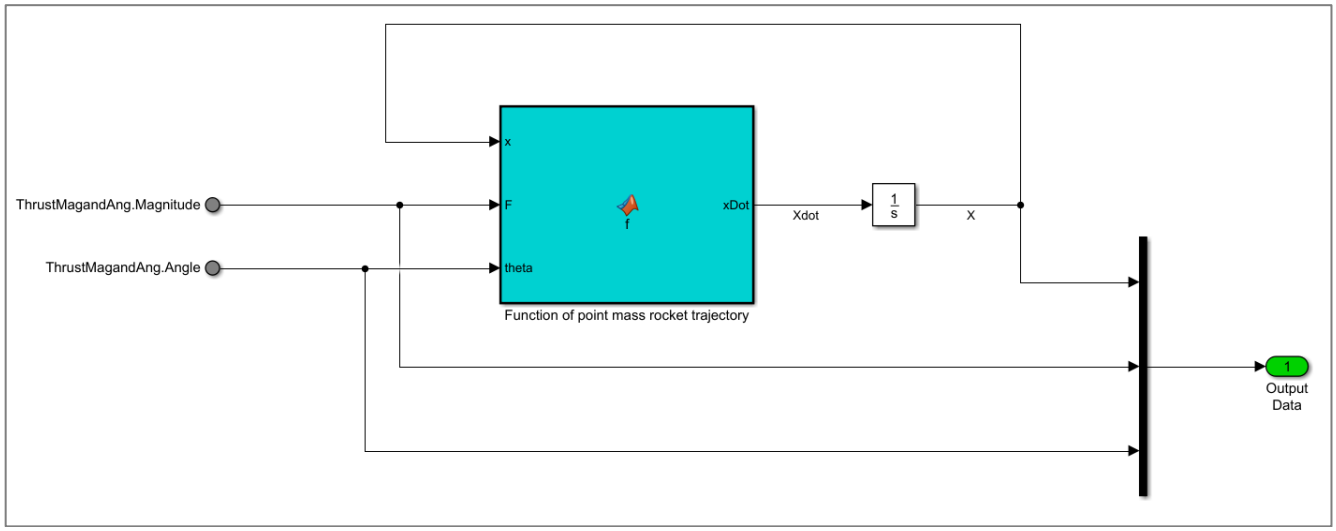


Figure 6: Signals based Simulink model

6.5 Components-based physical model

Below figure shows the component-based model. It takes the input as thrust vector.

It gives four outputs, x , y, v_x v_y.

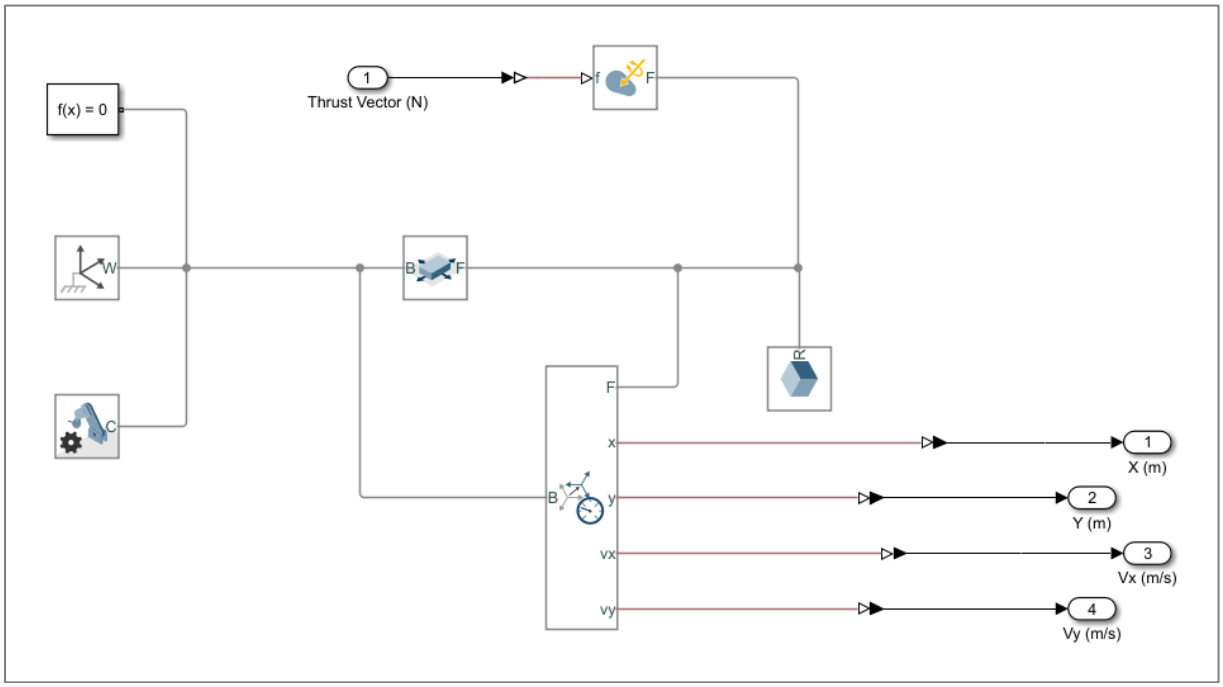


Figure 7: Components based Simscape model

6.6 Running the simulation scenarios & extracting output data

Now, we will run the simulation for both scenarios and then extract the output data which will be useful for plotting the figures and verification purpose.
But before that, we need to calculate the simulation stop time t2 which will be the maximum impact time of both scenarios.

Calculate Sim Stop Time t2 & values at burnout time

```
 % For scenario 1
 fBar = thr.F/rocket.m;                                  % normalized thrust having
unit same as that of acceleration
 x11 = 1/2*rocket.t1^2*fBar*cosd(thr.tht);               % horizontal position of
rocket at burnout time
 x21 = 1/2*rocket.t1^2*(fBar*sind(thr.tht)-planet.g); % vertical position of
rocket at burnout time
 x31 =  fBar*cosd(thr.tht)*rocket.t1;                    % horizontal speed of rocket
at burnout time
 x41 = (fBar*sind(thr.tht)-planet.g)*rocket.t1;          % vertical speed of rocket
at burnout time
 kk1 = sqrt(x41^2 + 2*planet.g*x21);                      % intermediate variable
 tau1 = (x41+kk1)/planet.g;                               % time elapsed from
burnout(t1) to impact(t2-1)
 Impact.t1 = rocket.t1+tau1;                              % impact time in seconds
 t21 = Impact.t1 + 1;                                     % Simulation stop time in
seconds

 % For scenario 2
 fBar = thr.F/rocket.m;                                  % normalized thrust having
unit same as that of acceleration
 thr.tht2 = thr.tht + thr.amp*cosd(thr.w*rocket.t1);   % Thrust angle at burnout
time
 x12 = (1/2*(rocket.t1^2))*fBar*cosd(thr.tht2);          % horizontal position of
rocket at burnout time
 x22 = (1/2*(rocket.t1^2))*(fBar*sind(thr.tht2)-planet.g); % vertical position of
rocket at burnout time
 x32 =  fBar*cosd(thr.tht2)*rocket.t1;                   % horizontal speed of
rocket at burnout time
 x42 = (fBar*sind(thr.tht2)-planet.g)*rocket.t1;         % vertical speed of rocket
at burnout time
 kk2 = sqrt(x42^2 + 2*planet.g*x21);                     % intermediate variable
 tau2 = (x41+kk2)/planet.g;                               % time elapsed from
burnout(t1) to impact(t2-1)
 Impact.t2 = rocket.t1+tau2;                              % impact time in seconds
 t22 = Impact.t2 + 1;                                     % Simulation stop time in
seconds
 t2 = max(t21,t22);    % Take maximum of simulation stop time from both scenarios
as end time.
```

Simulate Both Thrust Scenarios

```
 SCENARIO = 1; % constant thrust and angle variant model
 rocketSimOut1 = sim('rocketSim','SignalLoggingName','sdat'); % run the simulation
and save data in rocketSimOut1 struct.

 SCENARIO = 2; % constant thrust, time-varying angle variant model
 rocketSimOut2 = sim('rocketSim','SignalLoggingName','sdat'); % run the simulation
and save data in rocketSimOut2 struct.
```

Extract Data

```
 RS = 5;  % Variable for extracting the output data of signal based model
 SX = 3;  % Variable for extracting the X position data from simscape model
 SY = 4;  % Variable for extracting the X position data from simscape model
 VX = 1;  % Variable for extracting the X velocity data from simscape model
 VY = 2;  % Variable for extracting the Y velocity data from simscape model

 % Scenario 1 Results (Constant Thrust, Constant Angle)
 s1.t = rocketSimOut1.sdat{RS}.Values.Time;  % Save simulation time values in s1.t
array
 s1.x = rocketSimOut1.sdat{RS}.Values.Data(:,1); % Save horizontal position values
of signal based model in s1.x array
 s1.y = rocketSimOut1.sdat{RS}.Values.Data(:,2); % Save vertical position values
of signal based model in s1.y array
 s1.xd = rocketSimOut1.sdat{RS}.Values.Data(:,3); % Save horizontal speed values
of signal based model in s1.xd array
 s1.yd = rocketSimOut1.sdat{RS}.Values.Data(:,4); % Save vertical speed values of
signal based model in s1.yd array
 s1.ff = rocketSimOut1.sdat{RS}.Values.Data(:,5);  % Save thrust magnitude values
of signal based model in s1.ff array
 s1.aa = rocketSimOut1.sdat{RS}.Values.Data(:,6)*180/pi; % Save thrust angle
values of signal based model in s1.aa array
 s1.sx = rocketSimOut1.sdat{SX}.Values.Data(:,1); % Save horizontal position
values of simscape model in s1.sx array
 s1.sy = rocketSimOut1.sdat{SY}.Values.Data(:,1); % Save vertical position values
of simscape model in s1.sy array
 s1.sxd = rocketSimOut1.sdat{VX}.Values.Data(:,1); % Save horizontal velocity
values of simscape model in s1.sxd array
 s1.syd = rocketSimOut1.sdat{VY}.Values.Data(:,1); % Save vertical velocity values
of simscape model in s1.syd array
```

```
 % % Scenario 2 Results (Constant Thrust, Time-Varying Angle)
 s2.t = rocketSimOut2.sdat{RS}.Values.Time;  % Save simulation time values in s2.t
array
 s2.x = rocketSimOut2.sdat{RS}.Values.Data(:,1); % Save horizontal position values
of signal based model in s2.x array
 s2.y = rocketSimOut2.sdat{RS}.Values.Data(:,2); % Save vertical position values
of signal based model in s2.y array
 s2.xd = rocketSimOut2.sdat{RS}.Values.Data(:,3); % Save horizontal speed values
of signal based model in s2.xd array
 s2.yd = rocketSimOut2.sdat{RS}.Values.Data(:,4);  % Save vertical speed values of
signal based model in s2.yd array
 s2.ff = rocketSimOut2.sdat{RS}.Values.Data(:,5); % Save thrust magnitude values
of signal based model in s2.ff array
 s2.aa = rocketSimOut2.sdat{RS}.Values.Data(:,6)*180/pi; % Save thrust angle
values of signal based model in s2.aa array
 s2.sx = rocketSimOut2.sdat{SX}.Values.Data(:,1); % Save horizontal position
values of simscape model in s2.sx array
 s2.sy = rocketSimOut2.sdat{SY}.Values.Data(:,1); % Save vertical position values
of simscape model in s2.sy array
 s2.sxd = rocketSimOut2.sdat{VX}.Values.Data(:,1); % Save horizontal velocity
values of simscape model in s2.sxd array
 s2.syd = rocketSimOut2.sdat{VY}.Values.Data(:,1); % Save vertical velocity values
of simscape model in s2.syd array
 %
 % % Extract indices at burnout and impact for both runs
 s1.idBurnOut = find(s1.ff==0,1);   % Find the index at burnout for scenario 1
 s1.tBurnOut  = s1.t(s1.idBurnOut); % Find the value of time at burnout index for
scenario 1
 s1.idImpact  = find(s1.y<0,1) - 1; % Find the index at impact for scenario 1
 s1.tImpact   = s1.t(s1.idImpact);  % Find the value of time at impact index for
scenario 1

 s2.idBurnOut = find(s2.ff==0,1);   % Find the index at burnout for scenario 2
 s2.tBurnOut  = s2.t(s2.idBurnOut); % Find the value of time at burnout index for
scenario 2
 s2.idImpact  = find(s2.y<0,1) - 1; % Find the index at impact for scenario 2
 s2.tImpact   = s2.t(s2.idImpact);  % Find the value of time at impact index for scenario 2
```

6.7 Calculations and printing the results

After, we run the simulation and extract the data with the help of the indices, we will make several calculations which are –

- Energy at burnout and impact for both scenarios
- Total energy during entire trajectory
- Error in signals at burnout called burnout errors

Calculate Energy at Burnout (EO) and Impact (Ef)

```
s1.E0 = 1/2*rocket.m*( (s1.xd(s1.idBurnOut))^2 + (s1.yd(s1.idBurnOut))^2 ) + ...
    rocket.m*planet.g*s1.y(s1.idBurnOut);   % Energy at burnout for scenario 1
s1.Ef = 1/2*rocket.m*( (s1.xd(s1.idImpact))^2 + (s1.yd(s1.idImpact))^2 ) + ...
    rocket.m*planet.g*s1.y(s1.idImpact);    % Energy at impact for scenario 1

s2.E0 = 1/2*rocket.m*( (s2.xd(s2.idBurnOut))^2 + (s2.yd(s2.idBurnOut))^2 ) + ...
    rocket.m*planet.g*s2.y(s2.idBurnOut);   % Energy at burnout for scenario 2
s2.Ef = 1/2*rocket.m*( (s2.xd(s2.idImpact))^2 + (s2.yd(s2.idImpact))^2 ) + ...
    rocket.m*planet.g*s2.y(s2.idImpact);    % Energy at impact for scenario 2
```

Calculate Total Energy During Entire Trajectory for both scenarios

```
s1.E = 1/2*rocket.m*( (s1.xd).^2 + (s1.yd).^2 ) + ...
    rocket.m*planet.g*(s1.y);   % Total Energy calculated for scneario 1

s2.E = 1/2*rocket.m*( (s2.xd).^2 + (s2.yd).^2 ) + ...
    rocket.m*planet.g*(s2.y);   % Total Energy calculated for scneario 2
```

Calculate burn out errors

```
e.e11 = x11 - s1.x(s1.idBurnOut);  % Error in horizontal position at burnout time
e.e21 = x21 - s1.y(s1.idBurnOut);  % Error in vertical position at burnout time
e.e31 = x31 - s1.xd(s1.idBurnOut); % Error in horizontal velocity at burnout time
e.e41 = x41 - s1.yd(s1.idBurnOut); % Error in vertical velocity at burnout time
fprintf('<strong>------------------------------------------------------------------
-------------------------------------------------</strong>\n');
fprintf('<strong>Burnout Errors </strong>\n'); % Print the heading as burnout
errors
fprintf('x  Error = %f \n',e.e11);  % Print horizontal position error
fprintf('y  Error = %f \n',e.e21);  % Print vertical position error
fprintf('Vx Error = %f \n',e.e31);  % Print horizontal velocity error
fprintf('Vy Error = %f \n',e.e41);  % Print vertical velocity error
```

Results:

Burnout Errors	
Error in X – horizontal displacement	-5.75e-11 m
Error in Y – Vertical displacement	-1.07e-11 m
Error in Vx – horizontal velocity	-2.94e-11 m/s
Error in Vy – Vertical velocity	7.97e-13 m/s

6.8 Plotting the results

First of all, we will compare the vertical motion for both scenarios and then plot it in the figure.

Compare vertical motion for both scenarios

```
% plot figure 1 as time vs vertical motion of both scenarios.
figure(1);plot(s1.t(s1.idBurnOut:s1.idImpact),s1.y(s1.idBurnOut:s1.idImpact),...)
  s2.t(s2.idBurnOut:s2.idImpact),s2.y(s2.idBurnOut:s2.idImpact))
xlabel('Time (sec)');  % Give X label as Time in seconds
ylabel('Y (m)');       % Give Y label as Y in meters
% Define the legends
legend('Scenario = 1 = Const F and Ang','Scenario = 2 = Const F, Varying Ang');
title("Vertical motions for both scenarios"); % Give the title
fprintf('<strong>-------------------------------------------------------------------
-------------------------------------------------</strong>\n');
fprintf('<strong>Refer Figure 1 for comparison of vertical motion of both
scenarios</strong>\n');  % Print this heading
```

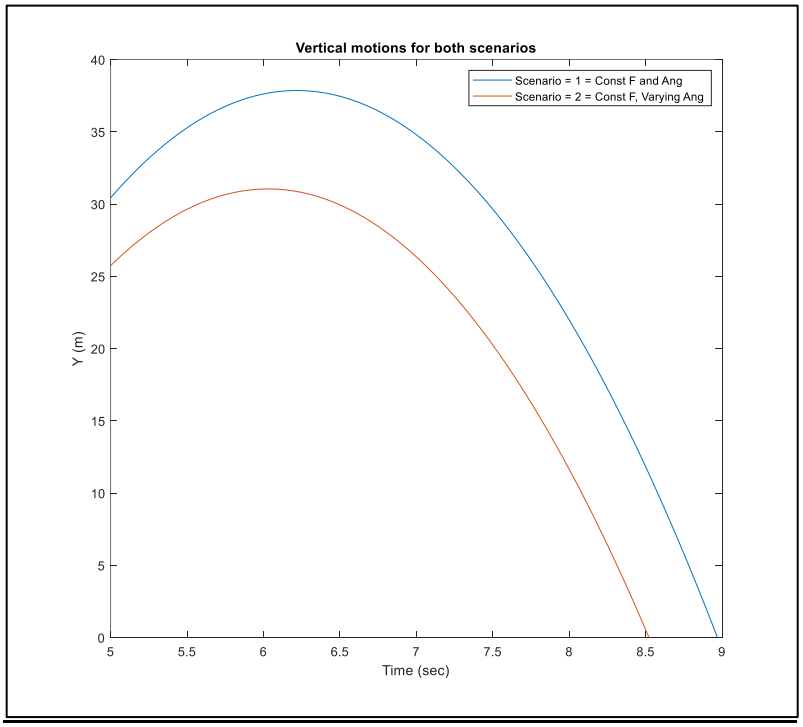


Figure 8: Vertical motion of both scenarios

Conclusion:

From the figure, it can be inferred that the point mass rocket has lesser horizontal range in scenario 2 as compared to scenario 1. The maximum height reached by point mass rocket is more in scenario 1 as compared to scenario 2.

Ideally, the total energy from burnout to impact must be conserved. We can verify this phenomenon by examining the energy error from burnout to impact.

Plot Energy Error from burnout to impact

```
 % plot figure 2 as time vs energy error from burn out to impact
 figure(2);
 plot(s1.t(s1.idBurnOut:s1.idImpact),abs(s1.E(s1.idBurnOut:s1.idImpact)-
s1.E0),'k');
 hold on
 plot(s2.t(s2.idBurnOut:s2.idImpact),abs(s2.E(s2.idBurnOut:s2.idImpact)-
s2.E0),'r');
 xlabel('Time (sec)');       % Give X label as Time in seconds
 ylabel('Energy Error (J)'); % Give Y label as Energy error in joules
 axis([0 10 0 4e-7]);     % specify X and Y limits
 legend('Scenario 1','Scenario 2'); % Give the legends
 title("Energy error from burnout to impact"); % Give the title
 fprintf('<strong>----------------------------------------------------------------
----------------------------------------------------</strong>\n');
 fprintf('<strong>Refer Figure 2 for energy errors from burnout to
impact</strong>\n');  % Print this heading
```

Result:

Max error in energy from burnout to impact = 2.7660e-07 joules.

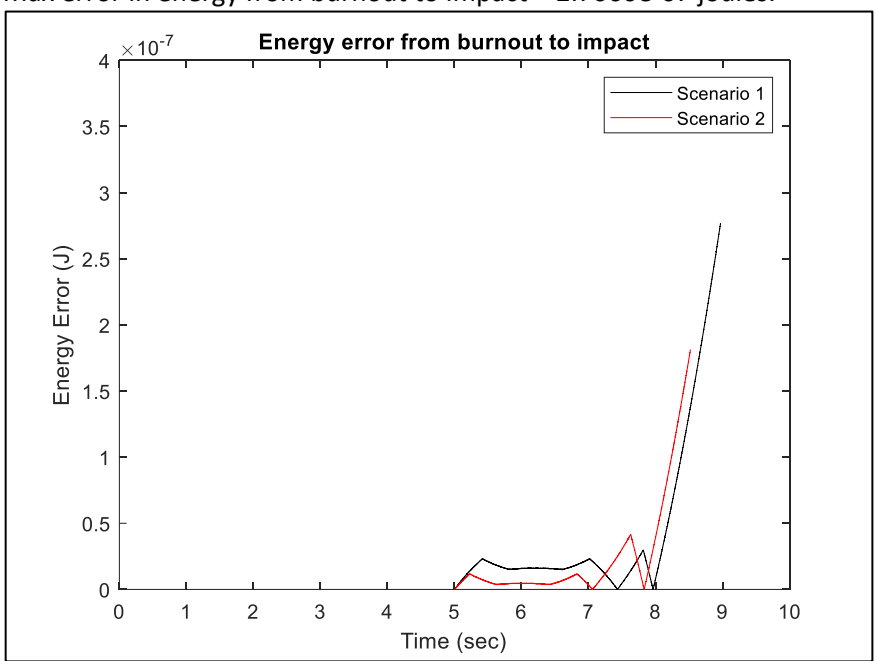


Figure 9: Energy error from burnout to impact

Conclusion:

As the error in energy from burnout to impact is very negligible we can say that energy is conserved from burnout to impact.

Chapter 7. Verification

As we discussed earlier, we are verifying our models and results with three different verification methods.

7.1 Task 2a – Comparison with closed form solution

In this task, we are generating the closed form solution for scenario 1 and comparing the results with Simulink results. Let us define the code for exact (closed form) solution, calculating the error and plotting the results.

Verification part - Task 2a

```
 % Generate the exact solution
 FBar1.X = [fBar*cosd(thr.tht), 0];  % Horizontal component of nomralized thrust
 FBar1.Y = [fBar*sind(thr.tht)-planet.g, -planet.g];  % Vertical component of
nomralized thrust
 i = 1;
 for t1 = 0:simPrm.h:10  % t1 goes from 0 to t2 with step of h
     if t1 <=5  % if t1 is less than or equal to 5
     exact.x1(i,:) = rocket.ics(1) + rocket.ics(3)*t1 + 1/2*t1^2*FBar1.X(1);  %
define exact value of horizontal position
     exact.xd1(i,:) = rocket.ics(3) + FBar1.X(1)*t1;   % define exact value of
horizontal velocity
     exact.y1(i,:) = rocket.ics(2) + rocket.ics(4)*t1 + 1/2*t1^2*FBar1.Y(1);  %
define exact value of vertical position
     exact.yd1(i,:) = rocket.ics(4) + FBar1.Y(1)*t1;  % define exact value of
vertical velocity
     bout = i;  % make the values at burnout equal to latest value
     else
     exact.x1(i,:) = exact.x1(bout) + (exact.xd1(bout))*(t1-rocket.t1) + 1/2*(t1-
rocket.t1)^2*FBar1.X(2); % define exact value of horizontal position
     exact.xd1(i,:) = exact.xd1(bout) + FBar1.X(2)*(t1-rocket.t1);  % define exact
value of horizontal velocity
     exact.y1(i,:) = exact.y1(bout) + (exact.yd1(bout))*(t1-rocket.t1) + 1/2*(t1-
rocket.t1)^2*FBar1.Y(2); % define exact value of vertical position
     exact.yd1(i,:) = exact.yd1(bout) + FBar1.Y(2)*(t1-rocket.t1);  % define exact
value of vertical velocity
     exact.f1(i,:) = 0;
     end
     exact.t1(i,:) = t1;
     i = i+1;  % Repeat the loop for generating next value
 end

 % Calculate, print and plot errors for all states between exact solution and
simulink solver
 ts_error.X = abs(exact.x1(:,1)-s1.x(:,1));  % error in horizontal position
between exact solution and simulink solver
```

```
 ts_error.Y = abs(exact.y1(:,1)-s1.y(:,1));  % error in vertical position between
exact solution and simulink solver
 ts_error.Xd = abs(exact.xd1(:,1)-s1.xd(:,1));  % error in horizontal velocity
between exact solution and simulink solver
 ts_error.Yd = abs(exact.yd1(:,1)-s1.yd(:,1));  % error in vertical velocity
between exact solution and simulink solver

 figure(3);  % plot figure 3 as time vs error in exact solution vs simulation
 plot(exact.t1,ts_error.X,'k');  % plot t vs error in horizontal position
 hold on
 plot(exact.t1,ts_error.Y,'r');  % plot t vs error in vertical position
 plot(exact.t1,ts_error.Xd,'b');  % plot t vs error in vertical position
 plot(exact.t1,ts_error.Yd,'m');  % plot t vs error in vertical position
 axis([0 10 0 5e-10]);
 legend('X-error','Y-error','Vx-error','Vy-error'); % Give the legends
 title({                                            % Give the title
      ('Error in values between')
       ('exact solution and simulink solver')
       });
 xlabel('Time (sec)');  % Give X label as Time in seconds
 ylabel('error');  % Give Y label as error

 fprintf('<strong>----------------------------------------------------------------
-------------------------------------------------</strong>\n');
 fprintf('<strong>Verification part 1 - Erros between exact solution and simulink
solver; Refer Figure 3</strong>\n');  % Print this heading
 fprintf('Max X error for exact solution vs simulink solver = %f
\n',max(ts_error.X)); % Print maximum of error in X for exact vs simulink
 fprintf('Max Y error for exact solution vs simulink solver = %f
\n',max(ts_error.Y)); % Print maximum of error in Y for exact vs simulink
 fprintf('Max Vx error for exact solution vs simulink solver = %f
\n',max(ts_error.Xd)); % Print maximum of error in Xd for exact vs simulink
 fprintf('Max Vy error for exact solution vs simulink solver = %f
\n',max(ts_error.Yd)); % Print maximum of error in Yd for exact vs simulink

 if max(ts_error.X) < 10e-6  && max(ts_error.Y) < 10e-6  && max(ts_error.Xd) <
10e-6 && max(ts_error.Yd) < 10e-6
     fprintf('<strong>Simulated and closed solution results are
matching</strong>\n');
     fprintf('<strong>Integration method and time step is
appropriate</strong>\n');
 else
     fprintf('<strong>Simulated and closed solution results are not
matching</strong>\n');
     fprintf('<strong>Integration method and time step is not
appropriate</strong>\n');
 end
```

Result:

Errors between exact solution and Simulink solver		Less than threshold $1x10^{-5}$
Error in X – horizontal displacement	3.70e-10 m	Yes
Error in Y – Vertical displacement	1.38e-11 m	Yes
Error in Vx – horizontal velocity	2.94e-11 m/s	Yes
Error in Vy – Vertical velocity	1.56e-11 m/s	Yes

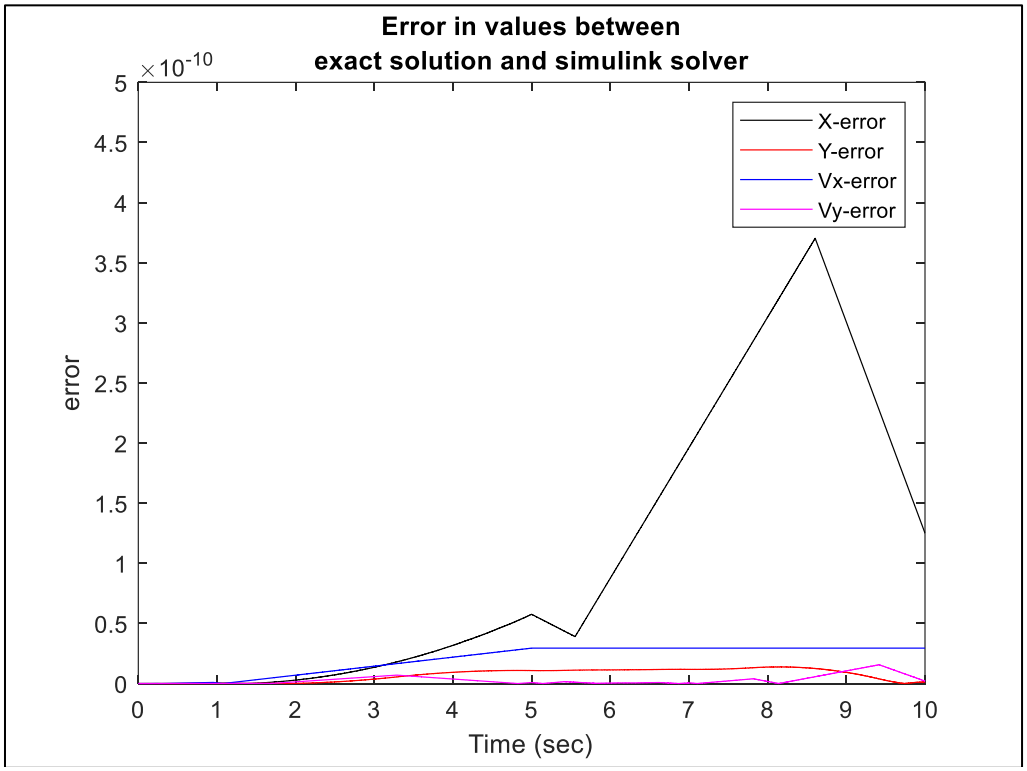


Figure 10: Output error between exact solution and Simulink solver

Conclusion:

From the result tables, it can be seen that error in all signals is less than the threshold value and thus it can be inferred that there is close approximation of Simulink solver results with the closed form solution results.

This confirms that the value of simulation integration time step size and the method of integration chosen are appropriate.

Finalized simulation solver parameters	
Integration method	Fixed step – ode3
Integration step size	0.0001

7.2 Task 2b – Comparison of results of two models

In this task, we are comparing the output results from signal-based model with component based Simscape model. Let us define the code for comparing output results of two models.

Verification part - Task 2b

```
 % Calculate, print and plot errors for all signals between simscape multibody and
signal based simulink model
 sc_error.X = abs(s2.sx(:,1)-s2.x(:,1));  % error in horizontal position between
simscape and signal based
 sc_error.Y = abs(s2.sy(:,1)-s2.y(:,1));  % error in vertical position between
simscape and signal based
 sc_error.Xd = abs(s2.sxd(:,1)-s2.xd(:,1));  % error in horizontal velocity
between simscape and signal based
 sc_error.Yd = abs(s2.syd(:,1)-s2.yd(:,1));  % error in vertical velocity between
simscape and signal based
 figure(4);  % plot figure 4 as time vs error in simscape vs signal based simulink
model
 plot(s1.t,s2.x(:,1),'k'); % X trajectory for signal based model
 hold on
 plot(s1.t,s2.sx(:,1),'r'); % X trajectory for simscape based model
 plot(s1.t,s2.y(:,1),'b'); % Y trajectory for signal based model
 plot(s1.t,s2.sy(:,1),'m'); % Y trajectory for simscape based model
 legend('X-signal based','X-Comppnent based','Y-signal based','Y-Comppnent
based'); % Give the legends
 title("X & Y trajectories for signal based and component based models"); % Give
the title
 xlabel('Time (sec)');  % Give X label as Time in seconds
 ylabel('X & Y');  % Give Y label as X & Y
 fprintf('<strong>-----------------------------------------------------</strong>\n');
 % Print this heading
 fprintf('<strong>Verification part 2 - Identical trajectory for simscape
multibody and signal based simulink model; Refer Figure 4</strong>\n');
 fprintf('Max X error for component based vs signal based model = %f
\n',max(sc_error.X)); % Print maximum of error in X for simscape vs simulink
 fprintf('Max Y error for component based vs signal based model = %f
\n',max(sc_error.Y)); % Print maximum of error in Y for simscape vs simulink
 fprintf('Max Vx error for component based vs signal based model = %f
\n',max(sc_error.Xd)); % Print maximum of error in Xd for simscape vs simulink
 fprintf('Max Vy error for component based vs signal based model = %f
\n',max(sc_error.Yd)); % Print maximum of error in Yd for simscape vs simulink

 if max(sc_error.X) < 10e-6  && max(sc_error.Y) < 10e-6  && max(sc_error.Xd) <
10e-6 && max(sc_error.Yd) < 10e-6
     fprintf('<strong>Both trajectories are identical so both models are
implemented correctly; Refer figure 4</strong>\n');
 else
     fprintf('<strong>Both trajectories are not identical</strong>\n');
 end
```

Result:

Errors between exact solution and Simulink solver		Less than threshold $1x10^{-5}$
Error in X – horizontal displacement	5.68e-14m	**Yes**
Error in Y – Vertical displacement	2.30e-14 m	**Yes**
Error in Vx – horizontal velocity	7.10e-15 m/s	**Yes**
Error in Vy – Vertical velocity	3.55e-15 m/s	**Yes**

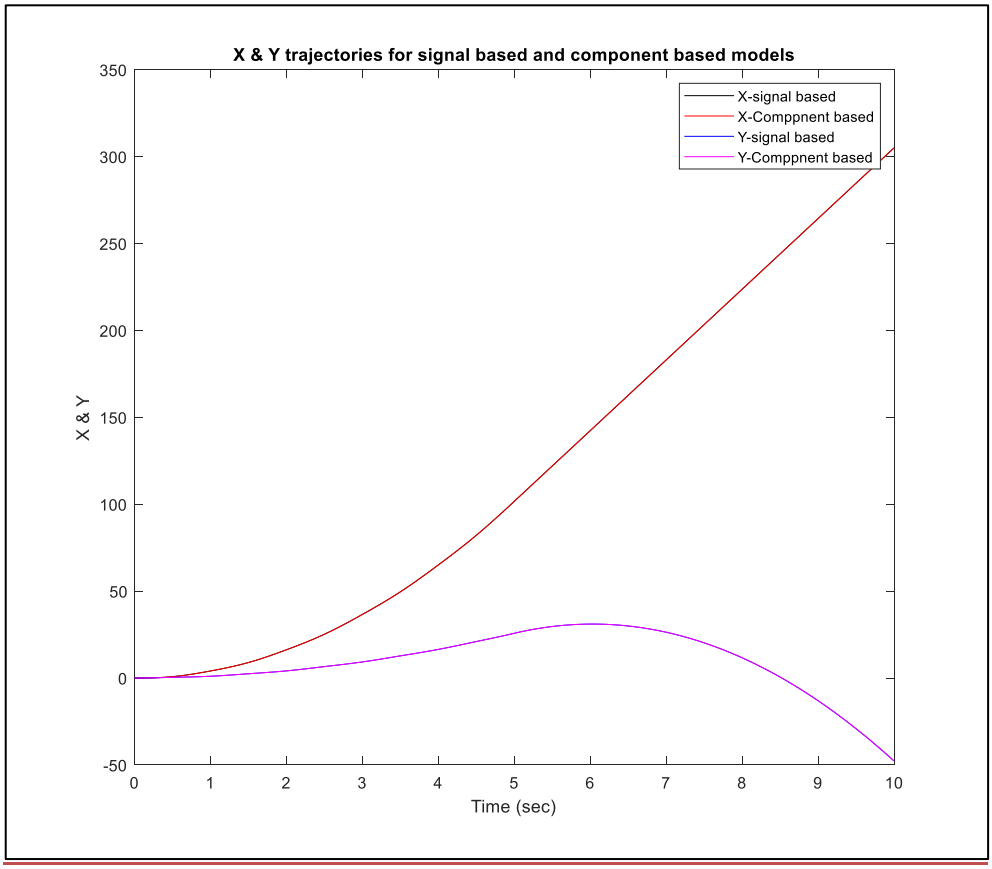


Figure 11: X & Y trajectories for signal based and component-based models

Conclusion:

From the result tables, it can be seen that error in all signals is less than the threshold value.

From the plot, it can be seen that the X & Y trajectories of scenario 2 for signal-based model are overlapped with those of component-based model and thus it can be inferred that the signal-based modeling is correct and thus it can be utilized now on.

7.3 Task 2c – Verification of principle of conservation of energy

In this task, we are verifying that total energy remains constant from burnout to impact.
Let us define the code to plot the total energy from burnout to impact with respect to time.

Verification part - Task 2c

```
 % Calculate & print change in energy from burnout to impact and plot total energy
vs time to prove that energy is conserved
 energychange.s1 = abs(s1.E(s1.idBurnOut)-s1.E(s1.idImpact));   % Calculate change
in total energy for scenario 1
 energychange.s2 = abs(s2.E(s2.idBurnOut)-s2.E(s2.idImpact));  % Calculate change
in total energy for scenario 2
 fprintf('<strong>----------------------------------------------------------------
-------------------------------------------------</strong>\n');
 fprintf('<strong>Verification part 3 - Energy Conservation from burnout to
impact; Refer Figure 5</strong>\n');  % Print this heading
 fprintf('Change in total energy from burnout to impact for scenario 1 = %f
\n',energychange.s1); % Print change in total energy for scenario 1
 fprintf('Change in total energy from burnout to impact for scenario 2 = %f
\n',energychange.s2); % Print change in total energy for scenario 2

 % Plot the total energy vs time from burn out to impact to show that energy
 % changes by very very small amount which confirms that energy is conserved
 figure(5);  % plot figure 5 as time vs energy and energy errors
 plot(s1.t(s1.idBurnOut:s1.idImpact),s1.E(s1.idBurnOut:s1.idImpact),'k'); % error
in horizontal position between simscape and signal based
 hold on
 plot(s2.t(s2.idBurnOut:s2.idImpact),s2.E(s2.idBurnOut:s2.idImpact),'r'); % error
in vertical position between simscape and signal based
 axis([rocket.t1 Impact.t2 1.1e6 1.3e6]);
 legend('Total energy for scenario 1','Total energy for scenario 2'); % Give the
legends
 title("Verification of energy conservation"); % Give the title
 xlabel('Time (sec)');  % Give X label as Time in seconds
 ylabel('Energy (Joules)');  % Give Y label as error
 if max(energychange.s1) < 10e-6  && max(energychange.s2) < 10e-6
     fprintf('<strong>Total energy is constant from burnout to impact thus it is
conserved; Refer figure 5</strong>\n');
 else
     fprintf('<strong>Total energy is changing from burnout to
impact</strong>\n');
 end
```

Result:

Maximum change in energy from burnout to impact	
Scenario 1	2.76e-07 joules
Scenario 2	1.81e-07 joules

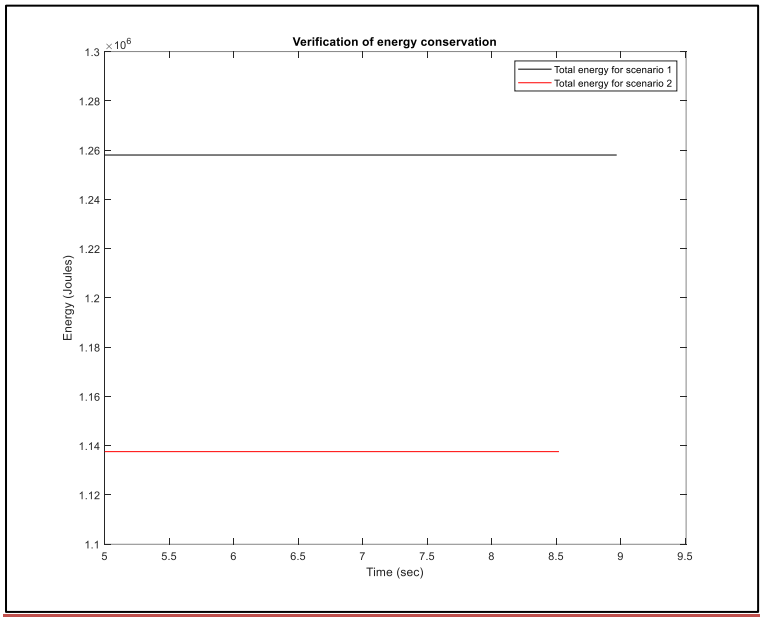


Figure 12: Verification of energy conservation – Total energy vs Time

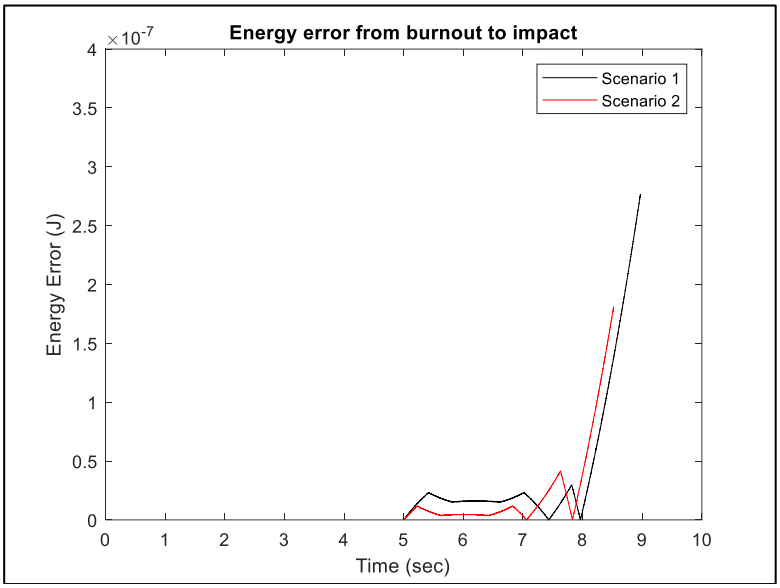


Figure 13: Verification of energy conservation – Energy error vs Time

Conclusion:

From the plot, it can be seen that total energy in both scenarios is remaining almost constant thus the principle of conservation of energy is satisfied.

Chapter 8. What will be there in command window after running the simulation

Once the simulation run is completed, we will see following data in the command window which is printed from the matlab script.
The statements in bold letters are the decisions made in the script based upon the comparison of magnitude of errors with the threshold values.

- Errors for different signals at burnout time
- Errors between closed form solution and simulation along with the conclusion (verification 1)
- Comparison of trajectory for Simscape and Simulink model and conclusion (verification 2)
- Verification of energy conservation by displaying the energy change errors along with the conclusion (Verification 3)

```
Command Window
-------------------------------------------------------------------------------------------------------------------------
Burnout Errors
x  Error = -0.000000
y  Error = -0.000000
Vx Error = -0.000000
Vy Error = 0.000000
-------------------------------------------------------------------------------------------------------------------------
Refer Figure 1 for comparison of vertical motion of both scenarios
-------------------------------------------------------------------------------------------------------------------------
Refer Figure 2 for energy errors from burnout to impact
-------------------------------------------------------------------------------------------------------------------------
Verification part 1 - Erros between exact solution and simulink solver; Refer Figure 3
Max X error for exact solution vs simulink solver = 0.000000
Max Y error for exact solution vs simulink solver = 0.000000
Max Vx error for exact solution vs simulink solver = 0.000000
Max Vy error for exact solution vs simulink solver = 0.000000
Simulated and closed solution results are matching
Integration method and time step is appropriate
-------------------------------------------------------------------------------------------------------------------------
Verification part 2 - Identical trajectory for simscape multibody and signal based simulink model; Refer Figure 4
Max X error for component based vs signal based model = 0.000000
Max Y error for component based vs signal based model = 0.000000
Max Vx error for component based vs signal based model = 0.000000
Max Vy error for component based vs signal based model = 0.000000
Both trajectories are identical so both models are implemented correctly; Refer figure 4
-------------------------------------------------------------------------------------------------------------------------
Verification part 3 - Energy Conservation from burnout to impact; Refer Figure 5
Change in total energy from burnout to impact for scenario 1 = 0.000000
Change in total energy from burnout to impact for scenario 2 = 0.000000
Total energy is constant from burnout to impact thus it is conserved; Refer figure 5
```

Chapter 9. Conclusion

- From the comparison of results of closed form solution of scenario 1 with the results of Simulink solver, it can be seen that the maximum absolute error in all signal values is less than the threshold value. The resulting error could be due to truncation and round off for selected integration method. Thus, it can be concluded that the integration method and step size we have chosen for integration is appropriate**.**

- From the graph of X & Y trajectories of signal-based Simulink model and components-based Simscape model, it is observed that both the trajectories overlap on each other and hence the models are giving exactly same results. This means that the matlab function inside signal-based model is correct and is verified.

- From the plot of total energy vs time, it is observed that the total energy for both scenarios from burnout to impact time changes by very small amount over the range of time. The maximum change in energy for both scenarios is very less which means the principle of conservation of energy is satisfied.

References:

1. **Canvas Notes-MEEM 4730 – Dr. Gordon Parker, Michigan Technological University**

2. Kinematics of particles – Kaustubh Dasgupta, Indian Institute of Technology, Guwahati

3. System Dynamics by Katsuhiko Ogata