Table of Contents

Thrust Interpolation Model

```
%Clean up
close all; clear; clc;
```

Reading in data

```
thrustMat = readmatrix('trial5');
thrustMat = thrustMat(4:end, :);
```

Creating thrust function

```
%Code for truncation provided by static test stand data
%Credit for this section to Nolan Stevenson

freq = 1.652*1000; % fixed
tStep = 1/freq;
massProp = 1; % 1 kg
g0 = 9.81; % m^2/s

summedT = thrustMat(:,3);
timeVec = [0:1:length(summedT)-1]*tStep;

    % first avg 4 pts for all, gets rid of some noise for cutoffs
    for i = 1:(length(summedT)/4 - 1)
        T1 = summedT(2*i);
        T2 = summedT(2*i + 1);
        T3 = summedT(2*i + 2);
        T4 = summedT(2*i + 3);
        Time1 = timeVec(2*i);
        Time2 = timeVec(2*i + 1);
        Time3 = timeVec(2*i + 2);
        Time4 = timeVec(2*i + 3);
        avgT(i) = mean([T1 T2 T3 T4]);
        avgTime(i) = mean([Time1 Time2 Time3 Time4]);
    end
```

```
    % now get cutoffs by checking for when the differene between avgs
 is > 0.5
    % from either side
    checker = false;
    i = 1;
    while checker == false
        if (abs(avgT(i) - avgT(i+1)) > 2.5) % larger diffference
 neededd as climb spikes and lots of noise in a few data trials
            firstCut_i = i;
            firstCut = avgTime(i);
            checker = true;
        end
        i = i + 1;
    end
    % flip it to get first change from otherside
    flippedT = flip(avgT);
    checker = false;
    i = 1;
    while checker == false
        if abs(flippedT(i) - flippedT(i+1)) > .5
            lastCutIndex = i;
            checker = true;
        end
        i = i + 1;
    end
    flippedTime = flip(avgTime);
    lastCut = flippedTime(lastCutIndex);

    cutOff = [firstCut lastCut];
    indexCut = timeVec > cutOff(1) & timeVec < cutOff(2);

    timeVec = timeVec(indexCut);
    thrustVec = summedT(indexCut)*4.44822; % Convert the lbf to N

    timeVec = timeVec - timeVec(1); %starting at the beginning t=0
```

Integration

```
%Provided constants and initial values
rhoWater = 1000;                                %kg/m^3
VolWaterInit = 0.001;                           %m^3

g = 9.81;
m0 = 0.125 + 1;
mf = 0.125;
Cd = 0.2;
DBottle = 0.105;
rhoAirAmb = 0.961;
startAngle = 45;
mu = 0.2;

constVec = [g;m0;mf;Cd;DBottle;rhoAirAmb;startAngle;mu];
```

```
wind = (10 / 2.237) * [cosd(270);sind(270);0]; %Zero wind baseline, 7
 mph SSW from 30 NE
%wind = [0;0;0];

%Getting initial parameters
x0 = 0;
y0 = 0;
z0 = 0.25;
vx0 = 0;
vy0 = 0;
vz0 = 0;

%Initial State Vector
initStateODE = [x0;y0;z0;vx0;vy0;vz0;m0];
tspan = [0 6];


%state are variable to the handle
ROCfunc = @(t,state) ROC(t,state,constVec,wind,thrustVec,timeVec);

%Creating ode options to have a more accurate calculation
options = odeset('RelTol', 1e-8, 'AbsTol',1e-10);

%Calling ode45
[finalTime, finalMat] = ode45(ROCfunc, tspan, initStateODE, options);

% %Extracting integrated values from the ode45 output
xPosition = finalMat(:,1);
yPosition = finalMat(:,2);
zPosition = finalMat(:,3);

baseLineImpact = [xPosition(end), yPosition(end)];
```

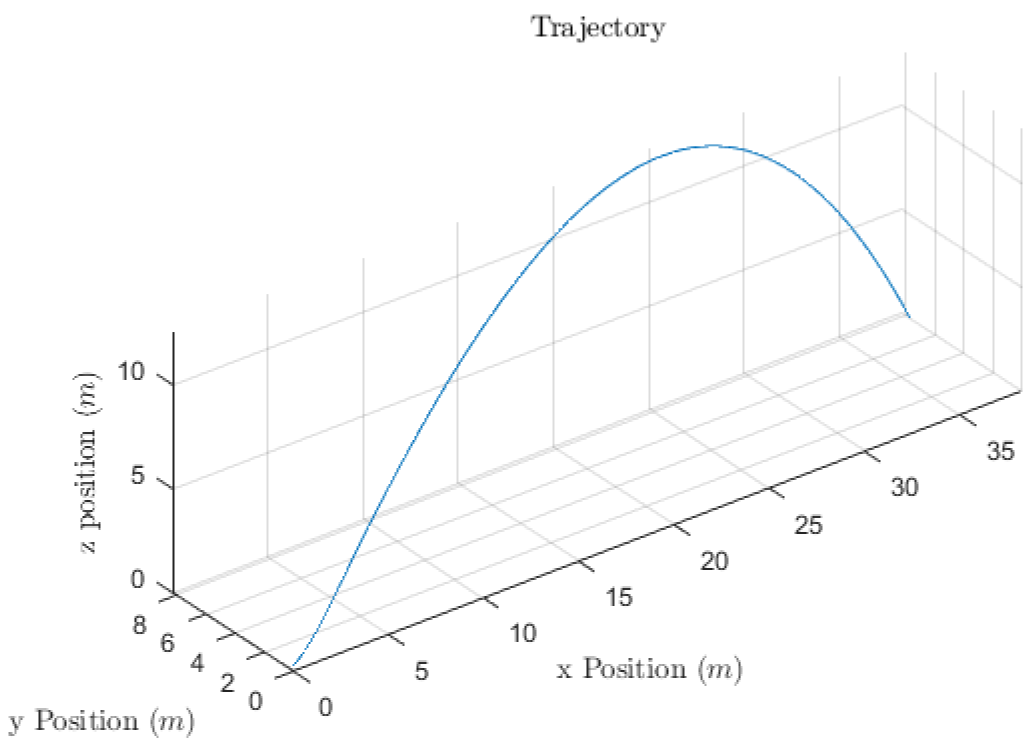
Monte Carlo Simulation

```
%Variables to vary

stdAngle = 1;
stdWater = 0.5 / 1000;
stdCd = 0.05;
stdRhoAirAmb = 0.05;
stdMu = 0.05;
stdWind = 5 / 2.237;

%Performing 100 simulations
monteCell = cell(1,500);
impactMat = zeros(500, 2);

for j = 1:500
    %Calculating random variation for the uncertain values
    randAngle = (2 * rand(1) - 1) * stdAngle;
    randWater = (2 * rand(1) - 1) * stdWater;
    randCd = (2 * rand(1) - 1) * stdCd;
```

```
    randRhoAirAmb = (2 * rand(1) - 1) * stdRhoAirAmb;
    randMu = (2 * rand(1) - 1) * stdMu;
    randWindx = (2 * rand(1) - 1) * stdWind; %rand between -3 and 3
    randWindy = (2 * rand(1) - 1) * stdWind;

    %Constant Values
    m0 = 0.125 + 1 + randWater;
    mf = 0.125;
    Cd = 0.2 + randCd;
    rhoAirAmb = 0.961 + randRhoAirAmb;
    startAngle = 45 + randAngle;
    mu = 0.4 + randMu;

    constVec = [g;m0;mf;Cd;DBottle;rhoAirAmb;startAngle;mu];

    %Wind and Isp
    windMonte = wind + [randWindx; randWindy; 0];

    %Performing integration
    %Initial State Vector
    initStateODE = [x0;y0;z0;vx0;vy0;vz0;m0];
    tspan = [0 6];

    %state are variable to the handle
    ROCfunc = @(t,state)
 ROC(t,state,constVec,windMonte,thrustVec,timeVec);

    %Creating ode options to have a more accurate calculation
    options = odeset('RelTol', 1e-8, 'AbsTol',1e-10);

    %Calling ode45
    [finalTime, finalMat] = ode45(ROCfunc, tspan, initStateODE,
 options);

    monteCell{j} = finalMat;

    %Getting the impact location
    impactMat(j,:) = finalMat(end, 1:2);
end
```

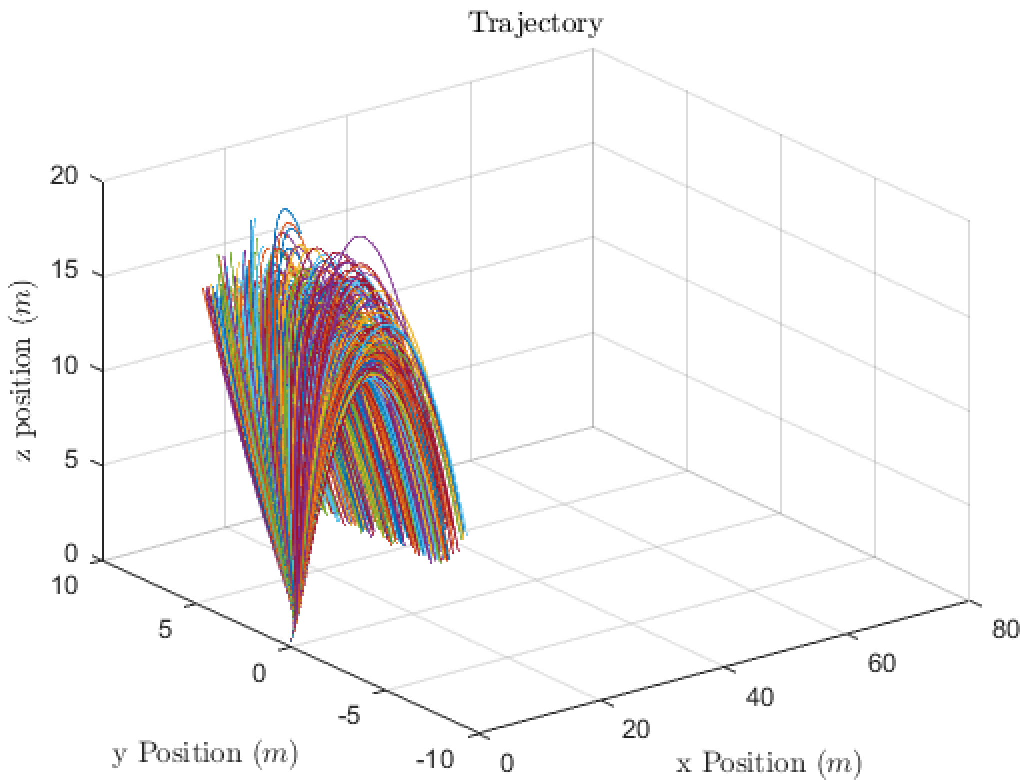
Trajectory Plot

```
figure(1)
plot3(xPosition, yPosition, zPosition);
grid on
xlim([0 50]);
ylim([-6 1]);
zlim([0 25]);
axis equal
title('Trajectory');
xlabel('x Position ($m$)');
ylabel('y Position ($m$)');
zlabel('z position ($m$)');
```



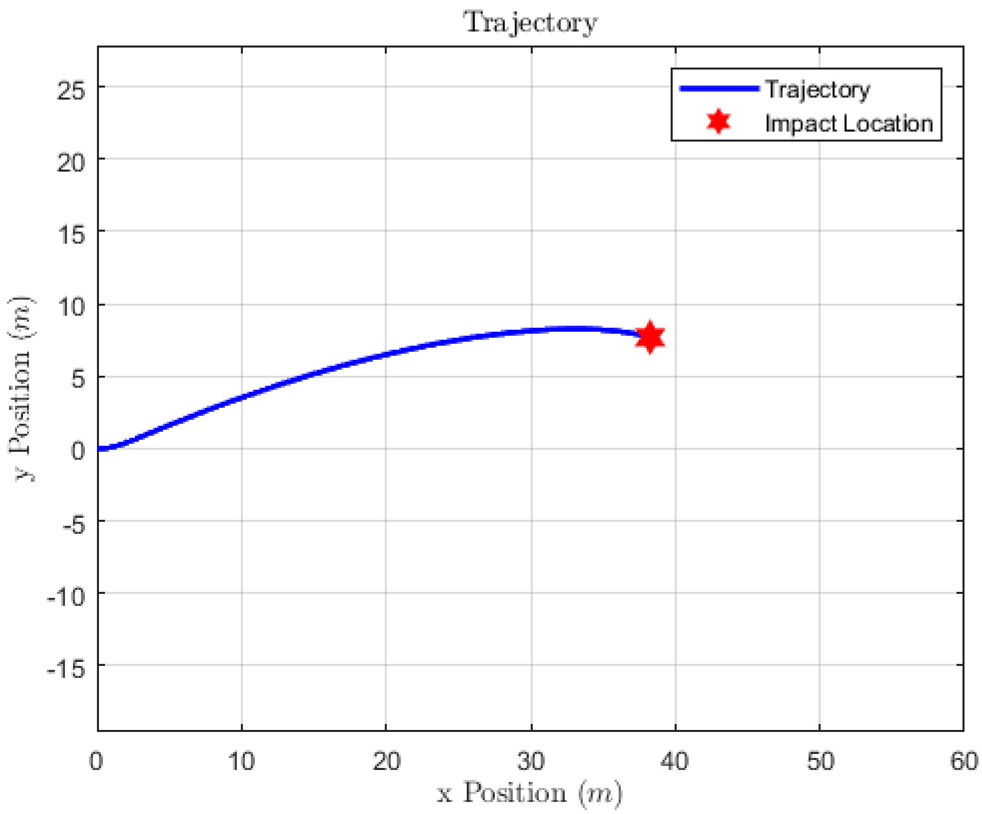
Monte Carlo Plot

```
figure(2)
set(0, 'defaultTextInterpreter', 'latex');
mat = monteCell{1};
xPos = mat(:,1);
yPos = mat(:,2);
zPos = mat(:,3);
plot3(xPos, yPos, zPos);
grid on
hold on
for j = 2:500
   mat = monteCell{j};
   xPos = mat(:,1);
   yPos = mat(:,2);
   zPos = mat(:,3);
   plot3(xPos, yPos, zPos);
end
xlim([0 80]);
ylim([-10 10]);
zlim([0 20]);
title('Trajectory');
xlabel('x Position ($m$)');
ylabel('y Position ($m$)');
zlabel('z position ($m$)');
```



XY Plane Trajectory

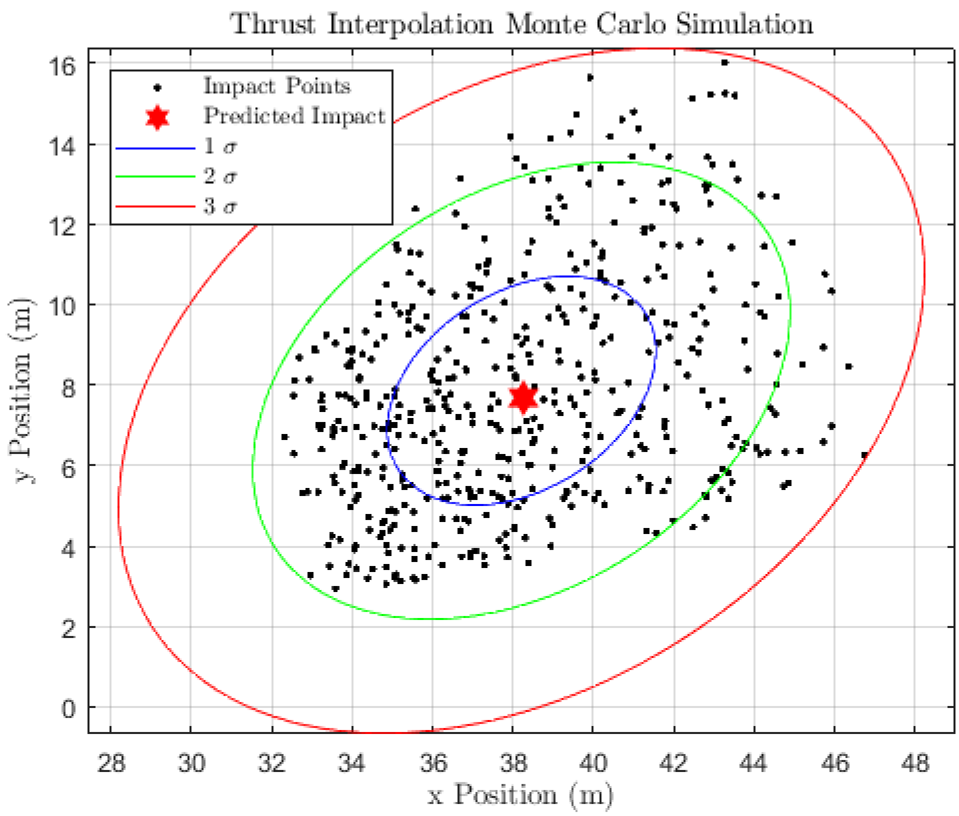
```
figure(3)
plot(xPosition, yPosition, 'LineWidth', 2, 'color', 'b');
xlim([0 60])
ylim([-20 20])
axis equal
grid on
hold on
plot(baseLineImpact(1), baseLineImpact(2), '-
h', 'LineStyle', 'none', 'color', 'r', 'MarkerSize',12, 'MarkerFaceColor', 'r');
title('Trajectory');
xlabel('x Position ($m$)');
ylabel('y Position ($m$)');
legend('Trajectory', 'Impact Location');
```



Error Ellipses

```
impacts = impactMat;
baseImpact = baseLineImpact;
xBase = baseImpact(1);
yBase = baseImpact(2);
x = impacts(:,1); % Randomly create some x data, meters
y = impacts(:,2); % Randomly create some y data, meters

figure(4); plot(x,y,'k.','markersize',6)
axis equal; grid on; xlabel('x Position (m)'); ylabel('y Position
 (m)'); hold on;
plot(xBase,yBase,'-
h', 'LineStyle', 'none', 'color', 'r', 'MarkerSize',12, 'MarkerFaceColor', 'r');

% Calculate covariance matrix
P = cov(x,y);
mean_x = mean(x);
mean_y = mean(y);

% Calculate the define the error ellipses
n=100; % Number of points around ellipse
p=0:pi/n:2*pi; % angles around a circle

[eigvec,eigval] = eig(P); % Compute eigen-stuff
```

```
xy_vect = [cos(p'),sin(p')] * sqrt(eigval) * eigvec'; % Transformation
x_vect = xy_vect(:,1);
y_vect = xy_vect(:,2);

% Plot the error ellipses overlaid on the same figure
plot(1*x_vect+mean_x, 1*y_vect+mean_y, 'b')
plot(2*x_vect+mean_x, 2*y_vect+mean_y, 'g')
plot(3*x_vect+mean_x, 3*y_vect+mean_y, 'r')

title('Thrust Interpolation Monte Carlo Simulation');
legend('Impact Points', 'Predicted Impact', '1 $\sigma$', '2 $\sigma
$', '3 $\sigma$', 'Interpreter','latex', 'location', 'northwest');
```



Rate of Change Function

```
function [dX] = ROC(t,state,const,wind,thrustVec,timeThrust)
% Purpose: To calculate the rate of change of various parameters along
% the flight trajector of a water rocket
%
% Inputs:    t = anonymous time variable
%          state = anonymous state vector containing
%             = [x, y, z, vx, vy, vz]
%            const = vector to hold constant values
%            wind = wind vector that holds the corresponding change in
 vel
%            thrustFunc = function handle of thrust polynomial fit
```

```matlab
%           initTime = time the polynomial is valid
% Outputs: dX = derivative state vector containing
%             =  [vx,vy,vz,d2x,d2y,d2z,dmdt]
%

%Constants from constant vector
g = const(1);
m0 = const(2);
mf = const(3);
Cd = const(4);
DBottle = const(5);
rhoAirAmb = const(6);
startAngle = const(7);
mu = const(8);
rhoWater = 1000;
DThroat = 0.021;

%Calculating area
areaBottle = pi*(DBottle / 2)^2;
areaNozzle = pi*(DThroat / 2)^2;

%Getting initial states
x = state(1);
y = state(2);
z = state(3);
vx = state(4);
vy = state(5);
vz = state(6);
m = state(7);

%Velocity vector and heading vector
vVec = [vx; vy; vz];
vHeading = vVec / norm(vVec);


%Calculating thrust
if(t < timeThrust(end)) %Base case, thrust is zero
    [~, ind] = min((abs(t - timeThrust))); %Finding time that is
 closest to t for ode45
    if(t <= timeThrust(1)) %First Case
        thrust = thrustVec(1);
    elseif(t < timeThrust(ind) && (ind - 1) > 0) %Interpolate with
 previous
        deltaTh = thrustVec(ind) - thrustVec(ind - 1);
        deltaT = timeThrust(ind) - timeThrust(ind - 1);
        slope = deltaTh / deltaT;
        thrust = thrustVec(ind - 1) + slope*(t - timeThrust(ind - 1));
    elseif(t > timeThrust(ind) && (ind + 1) <
 length(timeThrust)) %interpolate with next
        deltaTh = thrustVec(ind + 1) - thrustVec(ind);
        deltaT = timeThrust(ind + 1) - timeThrust(ind );
        slope = deltaTh / deltaT;
        thrust = thrustVec(ind) + slope*(t - timeThrust(ind));
        check = thrustVec(ind + 1) - thrustVec(ind);
```

```
    else %No interpolation
        thrust = thrustVec(ind);
    end
else
    thrust = 0;
end

%Calculating mass flow rate
dmdt = -1 * sqrt(rhoWater * areaNozzle * thrust);


%When the rocket is on the stand this is the heading
%It is fixed at an angle of 45 degrees, the stand is 0.5 meters long
if z < (0.25 + 0.5*cosd(45)) && t < 1
    vHeading = [cosd(startAngle); 0; sind(startAngle)];
    fGrav = [0; 0; -m * g];
    fDrag = -((1/2)*rhoAirAmb*(norm(vVec))^2*Cd*areaBottle) *
 vHeading;
    fThrust = thrust * vHeading;

    %Adding friction on the rails
    FricMag = mu * m * g * cosd(startAngle);
    fFric = -FricMag * vHeading;

elseif z <= 0 %Condition for rocket hitting the ground
    %All rates of change go to zero
    fGrav = [0; 0; 0];
    fDrag = [0; 0; 0];
    fFric = [0; 0; 0];
    fThrust = [0; 0; 0];
    vVec = [0;0;0];
else
    vRel = [vx - wind(1); vy - wind(2); vz - wind(3)];
    vHeading = vRel / norm(vRel);
    fGrav = [0; 0; -m * g];
    fDrag = -((1/2)*rhoAirAmb*(norm(vVec))^2*Cd*areaBottle) *
 vHeading;
    fThrust = thrust * vHeading;
    fFric = [0;0;0];
end

%Calculating the net force by summing all forces
fnet = fGrav + fDrag + fFric + fThrust;
a = fnet ./ m; %acceleration from F=ma

%Outputting the thrust and the rate of change vector
dX = [vVec(1); vVec(2); vVec(3); a(1); a(2); a(3); dmdt];
end
```

Published with MATLAB® R2020b