ASEN 3128 - Assignment 1

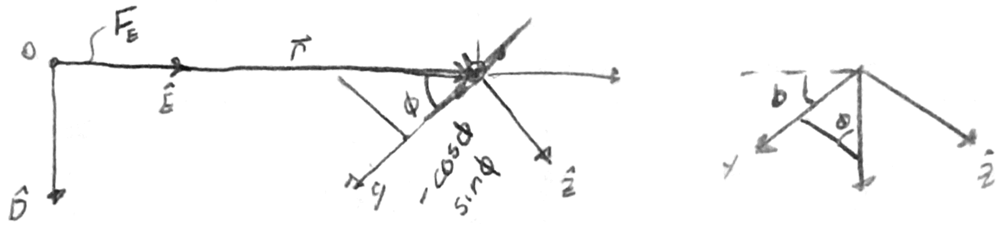
Seth Hill
Gregory Kondor
Connor Ott
Group 14 - Station 7b

January 29, 2018

University of Colorado - Boulder

Assignment #1 | ASEN 3128 | Connor Ott

Problem 1



$\boxed{\vec{r} = (0\hat{N} + 1\hat{E} + 0\hat{D})\ km}$ This is a position vector connecting the origin of F_E + the c.G. body of the plane

$\vec{r}_E$ is $\vec{r}$ written in F_E coordinates $F_E = (O, \hat{N}, \hat{E}, \hat{D})$. It is a coordinate representation of $\vec{r}$

$$\boxed{\vec{r}_E = \begin{bmatrix} 0 \\ 1km \\ 0 \end{bmatrix}}$$

$\vec{r}_B$ is a coordinate representation of $\vec{r}$ in $F_B = (\hat{x}, \hat{y}, \hat{z})$

$$\boxed{\vec{r}_B = \begin{bmatrix} 0 \\ -\cos\phi \\ \sin\phi \end{bmatrix} km}$$

Problem 2

$\vec{V}^E$ is the inertial velocity vector of the plane

$$\boxed{\vec{V}^E = -100\hat{N} + 0\hat{E} + 0\hat{D}\ m/s}$$

$\vec{V}^E_E$ is $\vec{V}^E$ represented in the F_E coordinate frame $F_E = (\hat{N}, \hat{E}, \hat{D})$

$$\boxed{\vec{V}^E_E = \begin{bmatrix} -100 \\ 0 \\ 0 \end{bmatrix} m/s}$$

$\vec{V}^E_B$ is $\vec{V}^E$ represented in the F_B coordinate frame, $F_B = (\hat{x}, \hat{y}, \hat{z})$

$$\boxed{\vec{V}^E_B = \begin{bmatrix} 100 \\ 0 \\ 0 \end{bmatrix} m/s}$$

$\Longrightarrow$

Assignment #1 | 3128 | Connor Ott

Problem 3

$\vec{\omega}^{EB}$ is the angular velocity vector of the B frame as seen from the E frame. This is the inertial angular velocity of the B frame

$$\vec{\omega}^{EB}_{ER} = \cancel{\begin{bmatrix} 0 \\ 0 \\ \frac{V^2}{r} \end{bmatrix}} = \cancel{\begin{bmatrix} 0 \\ 0 \\ \frac{\;}{\;} \end{bmatrix}} \qquad \boxed{\vec{\omega}^{EB} = (0\hat{N} + 0\hat{E} + 0.1\hat{D})\ \text{rad/s}}$$

$\vec{\omega}^{EB}_{E}$ is the angular velocity of the plane represented in F_E coordinates. $F_E = (\hat{N}, \hat{E}, \hat{D})$

$$\boxed{\vec{\omega}^{EB}_{E} = \begin{bmatrix} 0 \\ 0 \\ 0.1\ \text{rad/s} \end{bmatrix}}$$

$\vec{\omega}^{EB}_{B}$ is the coordinate representation of $\vec{\omega}^{EB}$ in F_B coords $F_B = (\hat{x}, \hat{y}, \hat{z})$

$$\boxed{\vec{\omega}^{EB}_{B} = \begin{bmatrix} 0 \\ 0.1 \sin\phi\ \text{rad/s} \\ 0.1 \cos\phi\ \text{rad/s} \end{bmatrix}}$$

Problem 4

$\frac{d^B}{dt}\vec{r}$ is the time rate of change of the $\vec{r}$ vector as seen by the body-fixed frame

$$\boxed{\frac{d^B}{dt}\vec{r} = (0\hat{x} + 0\hat{y} + 0\hat{z})\ \text{m/s}}$$

$\left(\frac{d^B}{dt}\vec{r}\right)_E$ is the body-fixed derivate of $\vec{r}$ represented in F_E coordinates

$$\boxed{\left(\frac{d^B}{dt}\vec{r}\right)_E = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\ \text{m/s}}$$

Assignment # 1 | 3128 | Connor Ott

Problem 4 cont

$\left(\frac{d^B}{dt}\vec{r}\right)_B$ is the body-fixed time derivative of $\vec{r}$ represented in F_B coordinates. $F_B = (\hat{x}, \hat{y}, \hat{z})$

$$\boxed{\left(\frac{d^B}{dt}\vec{r}\right)_B = \begin{bmatrix}0\\0\\0\end{bmatrix} m/s}$$

Problem 5

$\left(\frac{d^E}{dt}\vec{V}^E\right)_E$ is the inertial Earth-fixed time derivative of the inertial velocity vector of the plane represented in F_E coordinates. $F_E = (\hat{N}, \hat{E}, \hat{D})$

$$\left(\frac{d^E}{dt}\vec{V}^E\right)_E = \begin{bmatrix}0\\ -\frac{|\vec{V}|^2}{r}\\ 0\end{bmatrix} = \begin{bmatrix}0\\ -\frac{100^2}{1000}\\ 0\end{bmatrix} = \boxed{\begin{bmatrix}0\\-10\\0\end{bmatrix} m/s^2 = \left(\frac{d^E}{dt}\vec{V}^E\right)_E}$$

$\left(\frac{d^B}{dt}\vec{V}^E\right)_B$ is the body-fixed time derivative of the inertial velocity vector of the plane expressed in F_B coordinates

$F_B = (\hat{x}, \hat{y}, \hat{z})$

$$\left(\frac{d^B}{dt}\vec{V}^E\right)_B = \frac{d}{dt}\underbrace{\begin{bmatrix}100\\0\\0\end{bmatrix}}_{\vec{V}_B^E} = \boxed{\begin{bmatrix}0\\0\\0\end{bmatrix} m/s^2 = \left(\frac{d^B}{dt}\vec{V}^E\right)_B}$$

Problem 6

$$\frac{d^E}{dt}\vec{V}^E = \dot{\vec{V}}^E = \underbrace{\dot{\vec{V}}^B}_{\vec{0}} + \vec{\omega}^{EB} \times \vec{V}^E$$

$$\dot{\vec{V}}^E = (0\hat{N} + 0\hat{E} + 0.1\hat{D}) \times (-100\hat{N} + 0\hat{E} + 0\hat{D})$$

$$\boxed{\dot{\vec{V}}^E = (0\hat{N} - 10\hat{E} + 0\hat{D})\, m/s^2}$$

Assignment #1 | 3128 | Connor Ott

Problem 7

$\vec{f}$ is the inertial force vector acting on the plane

$\vec{f} = m \frac{d^E}{dt} \vec{v}^E = \boxed{-10m\,\hat{E}\ N = \vec{f}}$ where m = mass of plane

$\vec{f}_E$ is the coordinate representation of $\vec{f}$ in F_E coordinates

$F_E = (\hat{N}, \hat{E}, \hat{D})$

$$\boxed{\vec{f}_E = \begin{bmatrix} 0 \\ -10m \\ 0 \end{bmatrix} N}$$

$\vec{f}_B$ is the coordinate representation of $\vec{f}$ in F_B coordinates

$F_B = (\hat{x}, \hat{y}, \hat{z})$

$$\boxed{\vec{f}_B = \begin{bmatrix} 0 \\ 10\cos\phi m \\ -10\sin\phi m \end{bmatrix} N}$$

Problem 8

$\vec{W} = (10\hat{N} + 20\hat{E} - 5\hat{D})\ m/s$

$\vec{V} = \vec{v}^E - \vec{W}$

$= -100\hat{N} - (10\hat{N} + 20\hat{E} - 5\hat{D})$

$\boxed{\vec{V} = (-110\hat{N} - 20\hat{E} + 5\hat{D})\ m/s \quad \text{relative wind } \vec{V}}$

$\vec{V}_B$ is the relative wind $\vec{V}$ represented in F_B coordinates

$\hat{N} = -\hat{x} \quad \hat{E} = -\cos\phi\,\hat{y} + \sin\phi\,\hat{z} \quad \hat{D} = \sin\phi\,\hat{y} + \cos\phi\,\hat{z}$

$$\boxed{\vec{V}_B = \begin{bmatrix} 110 \\ 20\cos\phi + 5\sin\phi \\ -20\sin\phi + 5\cos\phi \end{bmatrix} m/s}$$

Problem 9

This problem simulated the trajectory of a golf ball under the effects of drag and gravity. Furthermore, crosswind speed and mass of the ball were varied to better understand the relationship these perturbances could have on the flight of the ball. Figure 1 indicates how crosswind speed can change the trajectory of the golf ball. Under variations between -5 to 5 m/s in the $\hat{N}$ direction, it was determined that the sensitivity of the ball to windspeed is approximately 0.514 m deflection per m/s windspeed.

Golf Ball Trajectory with Varied Crosswind

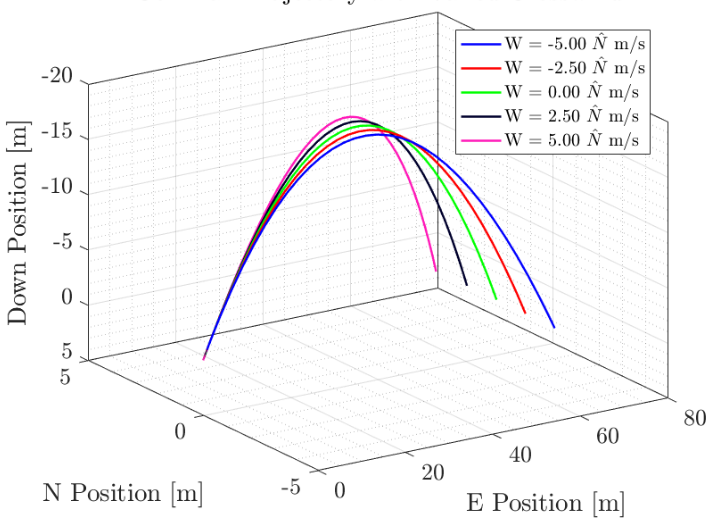


Figure 1: Golf Ball Wind Variation

Figure 2 indicates how changes in mass effect the flight of the golf ball. For this simulation, the mass of the golf ball was altered by 20% from 0.05 kg while keeping the kinetic energy of the ball constant. This resulted in increased velocity for the lighter ball and decreased velocity for the heavier ball. This simulation indicated that, in this case, a the lighter ball performed better, with a range of approximately 90 m as opposed to 67 and 50 m for the original and heavier ball. It should be noted that using too light of a ball may result in drag playing too great a role to achieve the greater distance seen with this single lighter ball.

Golf Ball Trajectory with Varied Mass

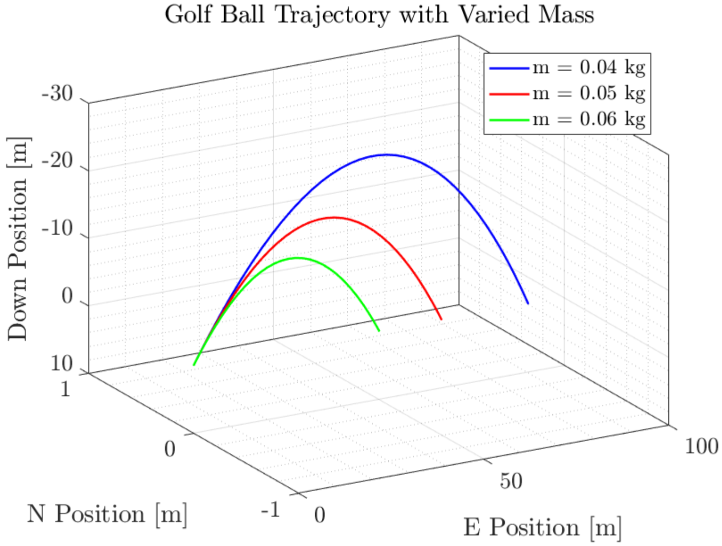


Figure 2: Golf Ball mass Variation

Problem 10

This problem aimed to determine a relationship between the volume of helium in a balloon and windspeed that would keep its angle of ascent at less than 45° from the vertical. That is, what volume is need to keep the balloon from straying too far from its launch position. For this experiment, a validation case was first created using a balloon with 1 m^3 of He, 0.5 kg payload mass, and prevailing wind of 4 m/s N and 2 m/s East. This case is shown in Figure 3.

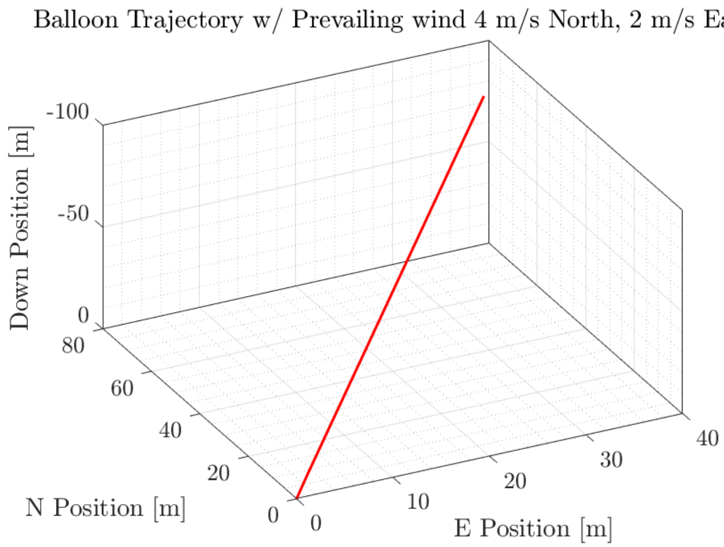


Figure 3: Balloon Validation case.

The wind speed and balloon volume was then varied from 0 to 20 m/s North and 0 to 10,000 m^3 respectively. The line labeled "45" in the contour plot shown in Figure 4 indicates that, in order to keep the angle to the vertical at or below 45°, increased balloon volume is needed for increased wind speed.

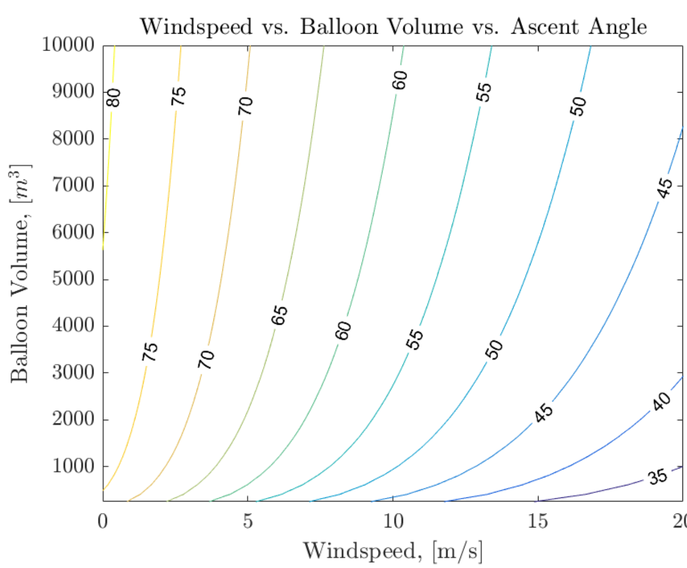


Figure 4: Wind and volume variation versus ascent angle.

Acknowledgements

Seth Hill - Problems 1-8 Gregory Kondor - Problems 1-8 Connor Ott - Problems 9-10

MATLAB Code

```
%{
Determines trajectory of a golf ball with a given initial velocity. Varies
mass and crosswind speed.

Created: 1/23/18 - Connor Ott
Last Modified: 1/27/18 - Connor Ott
%}

clear variables
close all
clc

nums = numbers;
%% Getting the Initial V for the trajectory of the ball.
vInit = [nums.V_xi, 0, nums.V_zi]; % [m/s]

%% Calling ode45 with initial velocity and position

V_0x = vInit(1);
V_0y = vInit(2);
V_0z = vInit(3);

[X_0x, X_0y, X_0z] = deal(0); % m - Inital position at (0,0,0)

t0 = 0;
tf = 5; % s

%% Wind Variation
windVels = linspace(-5, 5, 5);
colors = distinguishable_colors(length(windVels));
for i = 1:length(windVels)

    legStr{i} = sprintf('W = %.2f $\\hat{N}$ m/s', windVels(i));

    initialVals = [V_0x, V_0y, V_0z, 0, 0, 0, windVels(i), nums.m];
    [~, Fwind{i}] = ode45('golfBallODE', [t0 tf], initialVals);

    figure(1)
    hold on; grid on; grid minor;
    plot3(Fwind{i}(:, 4), Fwind{i}(:, 5), -Fwind{i}(:, 6), ...
          'linewidth', 1.2, 'color', colors(i, :));
    xlabel('E Position [m]')
    ylabel('N Position [m]')
    zlabel('Down Position [m]')
    title('Golf Ball Trajectory with Varied Crosswind')
    set(gca, 'TickLabelInterpreter', 'latex',...
         'fontsize', 13, ...
         'box', 'on',...
         'Zdir','reverse');
```

```
    deflect(i) = Fwind{i}(end, 5);
end
leg = legend(legStr);
set(leg, 'interpreter', 'latex', ...
         'fontsize', 10);

tempFit = polyfit(windVels, deflect, 1);
% Ball sensitivity to crosswind
wSense = tempFit(1); % m deflect / m/s crosswind Vel
fprintf('Ball sensitivity to crosswind: %.3f m/m/s\n', wSense)

%% Mass Variation
% Varying mass by 20% in either direction
massVec = linspace(nums.m*0.8, nums.m*1.2, 3);
kE_max = 0.5 * nums.m * sqrt(2) * nums.V_xi; % max kinetic energy
colors = distinguishable_colors(length(massVec)); % plotting
for i = 1:length(massVec)
    V_0x = kE_max * 2 / (massVec(i) * sqrt(2));
    V_0z = V_0x;
    legStr{i} = sprintf('m = %.2f kg', massVec(i));
    initialVals = [V_0x, V_0y, V_0z, 0, 0, 0, 0, massVec(i)];
    [~, Fmass{i}] = ode45('golfBallODE', [t0 tf], initialVals);

    figure(2)
    hold on; grid on; grid minor;
    plot3(Fmass{i}(:, 4), Fmass{i}(:, 5), -Fmass{i}(:, 6), ...
          'linewidth', 1.2, 'color', colors(i, :));
    xlabel('E Position [m]')
    ylabel('N Position [m]')
    zlabel('Down Position [m]')
    title('Golf Ball Trajectory with Varied Mass')
    set(gca, 'TickLabelInterpreter', 'latex',...
         'fontsize', 13, ...
         'box', 'on',...
         'Zdir','reverse');

       % Max ranges, indicates LIGHTER Ball will work best
    rangeVec(i) = Fmass{i}(end, 4);
end
leg = legend(legStr);
set(leg, 'interpreter', 'latex');
fprintf('Maximum Range and ball mass: %.2fm, %.2fkg\n', ...
         rangeVec(1), massVec(1));
```

```
function [ dfdt ] = golfBallODE( t, f )

% Importing number library
nums = numbers;
dfdt = zeros(6, 1);

V_g = [f(1), f(2), f(3)]; % Ground Speed [x, y, z] m/s
R = [f(4), f(5), f(6)]; % Positon [x, y, z] m
windy = f(7);
mass = f(8);

% Relative wind speed (x, y, z) m/s
```

```
V_rel = [V_g(1) - 0, V_g(2) - windy, V_g(3) - 0];

head = V_rel/norm(V_rel); % Heading vector (direction of rocket)
D = nums.rho/2 * norm(V_rel)^2 * nums.C_D * nums.A * head; % N - Drag force

F_g = [0, 0, -nums.g * mass];

% Sum of forces divided by mass to get acceleration
dV_gdt = (-D + F_g)/mass;

dRdt = V_g; % Vector of Ground Speed [x, y, z] m/s

% Once the rocket hits the ground, it should no longer travel in the x and
% z directions.
if R(3) <= 0 && t > 2
   dRdt = 0;
end

dfdt(1:3) = dV_gdt;
dfdt(4:6) = dRdt;
dfdt(7) = 0;
dfdt(8) = 0;
end
```

```
classdef numbers
   properties (Constant)
       m = 0.05;    % [kg]
       d = 0.03;    % [m]
       C_D = 0.5;   % []
       rho = 1.0;   % [kg/m^3]
       g = 9.81;    % [m/s^2]
       V_xi = 20;   % [m/s] x initial
       V_zi = 20;   % [m/s] Up initial
   end
   properties (SetAccess  = immutable)
       A            % [m^2]
   end
   properties (SetAccess = public)
       windVec = [0, -5, 0];
   end
   methods
       function obj = numbers()
           obj.A = pi * obj.d^2 / 4;
       end
   end
end
```

```
function colors = distinguishable_colors(n_colors,bg,func)
% DISTINGUISHABLE_COLORS: pick colors that are maximally perceptually distinct
%
% When plotting a set of lines, you may want to distinguish them by color.
% By default, Matlab chooses a small set of colors and cycles among them,
% and so if you have more than a few lines there will be confusion about
% which line is which. To fix this problem, one would want to be able to
% pick a much larger set of distinct colors, where the number of colors
% equals or exceeds the number of lines you want to plot. Because our
```

```
% ability to distinguish among colors has limits, one should choose these
% colors to be "maximally perceptually distinguishable."
%
% This function generates a set of colors which are distinguishable
% by reference to the "Lab" color space, which more closely matches
% human color perception than RGB. Given an initial large list of possible
% colors, it iteratively chooses the entry in the list that is farthest (in
% Lab space) from all previously-chosen entries. While this "greedy"
% algorithm does not yield a global maximum, it is simple and efficient.
% Moreover, the sequence of colors is consistent no matter how many you
% request, which facilitates the users' ability to learn the color order
% and avoids major changes in the appearance of plots when adding or
% removing lines.
%
% Syntax:
%   colors = distinguishable_colors(n_colors)
% Specify the number of colors you want as a scalar, n_colors. This will
% generate an n_colors-by-3 matrix, each row representing an RGB
% color triple. If you don't precisely know how many you will need in
% advance, there is no harm (other than execution time) in specifying
% slightly more than you think you will need.
%
%   colors = distinguishable_colors(n_colors,bg)
% This syntax allows you to specify the background color, to make sure that
% your colors are also distinguishable from the background. Default value
% is white. bg may be specified as an RGB triple or as one of the standard
% "ColorSpec" strings. You can even specify multiple colors:
%     bg = {'w','k'}
% or
%     bg = [1 1 1; 0 0 0]
% will only produce colors that are distinguishable from both white and
% black.
%
%   colors = distinguishable_colors(n_colors,bg,rgb2labfunc)
% By default, distinguishable_colors uses the image processing toolbox's
% color conversion functions makecform and applycform. Alternatively, you
% can supply your own color conversion function.
%
% Example:
%   c = distinguishable_colors(25);
%   figure
%   image(reshape(c,[1 size(c)]))
%
% Example using the file exchange's 'colorspace':
%   func = @(x) colorspace('RGB->Lab',x);
%   c = distinguishable_colors(25,'w',func);

% Copyright 2010-2011 by Timothy E. Holy

  % Parse the inputs
  if (nargin < 2)
    bg = [1 1 1];  % default white background
  else
    if iscell(bg)
      % User specified a list of colors as a cell aray
```

```
      bgc = bg;
      for i = 1:length(bgc)
        bgc{i} = parsecolor(bgc{i});
      end
      bg = cat(1,bgc{:});
    else
      % User specified a numeric array of colors (n-by-3)
      bg = parsecolor(bg);
    end
  end

  % Generate a sizable number of RGB triples. This represents our space of
  % possible choices. By starting in RGB space, we ensure that all of the
  % colors can be generated by the monitor.
  n_grid = 30;  % number of grid divisions along each axis in RGB space
  x = linspace(0,1,n_grid);
  [R,G,B] = ndgrid(x,x,x);
  rgb = [R(:) G(:) B(:)];
  if (n_colors > size(rgb,1)/3)
    error('You can''t readily distinguish that many colors');
  end

  % Convert to Lab color space, which more closely represents human
  % perception
  if (nargin > 2)
    lab = func(rgb);
    bglab = func(bg);
  else
    C = makecform('srgb2lab');
    lab = applycform(rgb,C);
    bglab = applycform(bg,C);
  end

  % If the user specified multiple background colors, compute distances
  % from the candidate colors to the background colors
  mindist2 = inf(size(rgb,1),1);
  for i = 1:size(bglab,1)-1
    dX = bsxfun(@minus,lab,bglab(i,:)); % displacement all colors from bg
    dist2 = sum(dX.^2,2);  % square distance
    mindist2 = min(dist2,mindist2);  % dist2 to closest previously-chosen color
  end

  % Iteratively pick the color that maximizes the distance to the nearest
  % already-picked color
  colors = zeros(n_colors,3);
  lastlab = bglab(end,:);   % initialize by making the "previous" color equal to background
  for i = 1:n_colors
    dX = bsxfun(@minus,lab,lastlab); % displacement of last from all colors on list
    dist2 = sum(dX.^2,2);  % square distance
    mindist2 = min(dist2,mindist2);  % dist2 to closest previously-chosen color
    [~,index] = max(mindist2);  % find the entry farthest from all previously-
```

```
        chosen colors
    colors(i,:) = rgb(index,:);  % save for output
    lastlab = lab(index,:);  % prepare for next iteration
  end
end

function c = parsecolor(s)
  if ischar(s)
    c = colorstr2rgb(s);
  elseif isnumeric(s) && size(s,2) == 3
    c = s;
  else
    error('MATLAB:InvalidColorSpec','Color specification cannot be parsed.');
  end
end

function c = colorstr2rgb(c)
  % Convert a color string to an RGB value.
  % This is cribbed from Matlab's whitebg function.
  % Why don't they make this a stand-alone function?
  rgbspec = [1 0 0;0 1 0;0 0 1;1 1 1;0 1 1;1 0 1;1 1 0;0 0 0];
  cspec = 'rgbwcmyk';
  k = find(cspec==c(1));
  if isempty(k)
    error('MATLAB:InvalidColorString','Unknown color string.');
  end
  if k~=3 || length(c)==1,
    c = rgbspec(k,:);
  elseif length(c)>2,
    if strcmpi(c(1:3),'bla')
      c = [0 0 0];
    elseif strcmpi(c(1:3),'blu')
      c = [0 0 1];
    else
      error('MATLAB:UnknownColorString', 'Unknown color string.');
    end
  end
end
```

```
%{
Determines relationship between wind, volume, and angle of ascent for a
balloon sonde.

Created: 1/23/18 - Connor Ott
Last Modified: 1/27/18 - Connor Ott
%}

clear variables
close all
clc
set(0, 'defaulttextinterpreter', 'latex');

% Number library
nums = balloonNums;
t0 = 0;
tf = 20; % s
```

```
%% Validation case
initialVals = [0, 0, 0, 0, 0, 0, 0, 1];
[~, F] = ode45('balloonODE', [t0 tf], initialVals);

figure
hold on; grid on; grid minor;
plot3(F(:, 4), F(:, 5), -F(:, 6), ...
    'r-', 'linewidth', 1.5)
xlabel('E Position [m]')
ylabel('N Position [m]')
zlabel('Down Position [m]')
title('Balloon Trajectory w/ Prevailing wind 4 m/s North, 2 m/s East')
set(gca, 'TickLabelInterpreter', 'latex',...
         'fontsize', 13, ...
         'box', 'on',...
         'Zdir','reverse');

%% Wind and Volume Variation
windVels = linspace(0, 20, 20); % [m/s]
vols = linspace(0, 10000, 50);   % [m^3]

FCell = cell(length(windVels), length(vols));
angles = zeros(length(windVels), length(vols));

% Varying both volume and North windspeed
for i = 1:length(windVels)
    for j = 1:length(vols)
        initialVals = [0, 0, 0, 0, 0, 0, windVels(i), vols(j)];
        [~, FCell{i, j}] = ode45('balloonODE', [t0 tf], initialVals);

        angles(i, j) = atand(FCell{i, j}(end, 6)/FCell{i, j}(end, 5));
    end
end

%% Plotting Data
levels = 35:5:85;
figure
hold on;
set(gca, 'TickLabelInterpreter', 'latex',...
         'fontsize', 12, ...
         'box', 'on');
contour(windVels, vols, angles', levels, 'showtext', 'on')
title('Windspeed vs. Balloon Volume vs. Ascent Angle')
xlabel('Windspeed, [m/s]')
ylabel('Balloon Volume, [$m^3$]')
axis([0, 20, 250, 10000])

%%
```

```
% Event function for balloon sonde simulation

function [ dfdt ] = balloonODE( t, f )

% Importing number library
nums = balloonNums;
```

```
dfdt = zeros(6, 1);

V_g = [f(1), f(2), f(3)]; % Ground Speed [x, y, z] m/s
R = [f(4), f(5), f(6)]; % Positon [x, y, z] m
crossWind = f(7);
volume = f(8);
mass = nums.m + volume*nums.rho_He; % total mass of sonde

% Balloon shape for calculating drag
r = (3/(pi*4) * volume)^(1/3);
A = pi * r^2;

windx = nums.W_E;
windy = nums.W_N;

% Relative wind speed (x, y, z) m/s
V_rel = [V_g(1) - windx, V_g(2) - windy - crossWind, V_g(3)];

head = V_rel/norm(V_rel); % Heading vector (direction of rocket)
D = nums.rho_air/2 * norm(V_rel)^2 * nums.C_D * A * head; % N - Drag force

% Gravity
F_g = [0, 0, -nums.g * mass];

% Bouyant Force
F_b = [0, 0, nums.g * nums.rho_air * volume];

% if norm(F_b) < norm(F_g)
%     dV_gdt = [0, 0, 0];
% else
    % Sum of forces divided by mass to get acceleration
    dV_gdt = (-D + F_g + F_b)/mass;
% end

dRdt = V_g; % Vector of Ground Speed [x, y, z] m/s


dfdt(1:3) = dV_gdt;
dfdt(4:6) = dRdt;
dfdt(7) = 0;
dfdt(8) = 0;
end
```

```
% Small number library for balloon simulation

classdef balloonNums
    properties (Constant)
        m = 0.5;          % [kg]
        C_D = 0.5;        % []
        rho_air = 1.225;  % [kg/m^3]
        rho_He = 0.164;   % [kg/m^3]
        g = 9.81;         % [m/s^2]
        W_E = 2;          % [m/s] x wind
        W_N = 4;          % [m/s] y wind
    end
    properties (SetAccess = immutable)
```

```
            A                    % [m^2]
            r                    % [m]
    end
end
```