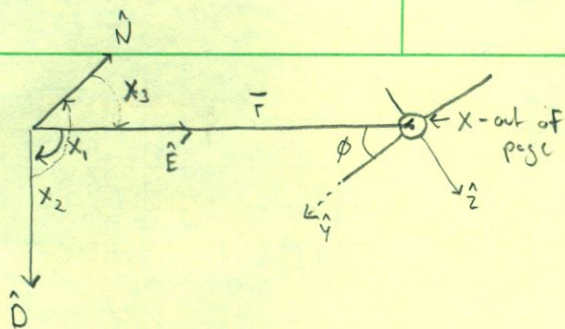


# ASEN 3128 - Assignment 1

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

January 29, 2018  
*University of Colorado - Boulder*



$$|\vec{r}| = 1 \text{ km} = 1000 \text{ m}$$

$$\text{ground speed} = |\vec{v}| = 100 \text{ m/s} = \vec{v}^E$$

$\vec{r}$ : is a position vector from the center of the  $F_E$  frame to the location 1 km due east aka reference from the origin to the center of gravity (CG) of the aircraft

$$\vec{r} = (0\hat{N} + 1\hat{E} + 0\hat{D}) (\text{km})$$

$\vec{r}_E$ : coordinate representation of  $\vec{r}$  in the E, earth fixed inertial frame

$$\vec{r}_E = [0 \ 1 \ 0]^T (\text{km})$$

" $\vec{r}$  in E"

Steps to make transformation matrix from E to B

- we start with the E axis frame + B axis frame aligned with each others relative axis
- rotate about the  $\hat{D}$  axis by  $+180^\circ$
- rotate about the  $\hat{X}$  axis by  $+\phi$

$$\begin{aligned} \vec{L}_3(X_3) &= \begin{matrix} \text{about } \hat{D} \\ \begin{bmatrix} \cos 180^\circ & \sin 180^\circ & 0 \\ -\sin 180^\circ & \cos 180^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \uparrow \\ \text{about } \hat{Z} \end{matrix} \quad \vec{L}_1(X_1) = \begin{matrix} \text{about } \hat{N} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos X_1 & \sin X_1 \\ 0 & -\sin X_1 & \cos X_1 \end{bmatrix} \\ \uparrow \\ \text{about } \hat{X}, \phi \end{matrix} \quad \vec{L}_2(X_2) = \begin{matrix} \text{about } \hat{E} \\ \begin{bmatrix} \cos X_2 & 0 & -\sin X_2 \\ 0 & 1 & 0 \\ \sin X_2 & 0 & \cos X_2 \end{bmatrix} \\ \uparrow \\ \text{about } \hat{Y} \end{matrix} \end{aligned}$$

$$\vec{L}_{BE} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos 180^\circ & \sin 180^\circ & 0 \\ -\sin 180^\circ & \cos 180^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\cos \phi & \sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix}$$

$\vec{r}_B$ : coordinate representation of  $\vec{r}$  in the B, body fixed frame

" $\vec{r}$  in B"

$$\vec{r}_B = \vec{L}_{BE} \vec{r}_E = \begin{bmatrix} \vec{L}_{BE} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \vec{r}_B = \begin{bmatrix} 0 \\ -\cos \phi \\ \sin \phi \end{bmatrix} (\text{km})$$



2  $\bar{V}^E = \frac{d^E}{dt} \bar{r}$ : the time rate of change of the position vector  $\bar{r}$  as observed from the E frame aka the inertial velocity vector of the aircraft

$$\bar{V}^E = \frac{d^E}{dt} \bar{r} = (-100 \hat{N} + 0 \hat{E} + 0 \hat{O}) [m/s]$$

$\bar{V}_E^E$ : the inertial velocity vector expressed/represented in the inertial coordinate frame

$$\bar{V}_E^E = \begin{bmatrix} -100 \\ 0 \\ 0 \end{bmatrix} [m/s]$$

$\bar{V}_B^E$ : the inertial velocity vector expressed/represented in the body fixed coordinate frame

$$\bar{V}_B^E = \begin{bmatrix} 100 \\ 0 \\ 0 \end{bmatrix} [m/s]$$

3  $\bar{\omega}^{EB}$ : the inertial angular velocity vector of the aircraft relative to the inertial frame

$$\bar{\omega}^{EB} = \frac{V}{r} = \frac{100 \text{ m/s}}{1000 \text{ m}} \hat{O} = 0.1 \hat{O} [\text{rad/s}]$$

$\bar{\omega}_E^{EB}$ : the inertial angular velocity vector expressed/represented in the inertial Earth fixed coordinate frame, E.

$$\bar{\omega}_E^{EB} = \begin{bmatrix} 0 \\ 0 \\ .1 \end{bmatrix} [\text{rad/s}]$$

$\bar{\omega}_B^{EB}$ : the inertial angular velocity vector expressed/represented in the body fixed coordinate frame

$$\bar{\omega}_B^{EB} = \bar{L}_{BE} \bar{\omega}_E^{EB} = \begin{bmatrix} \bar{L}_{BE} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ .1 \end{bmatrix} = \bar{\omega}_B^{EB} = \begin{bmatrix} 0 \\ .1 \sin \phi \\ .1 \cos \phi \end{bmatrix} [\text{rad/s}]$$



4)  $\left(\frac{d^B \bar{r}}{dt}\right)$ : the body-fixed frame derivative of the position vector  $\bar{r}$ , looking out the window of the plane the  $\bar{r}$  position vector never changes

$$\left(\frac{d^B \bar{r}}{dt}\right) = (0 \hat{x} + 0 \hat{y} + 0 \hat{z}) [\text{m/s}]$$

$$(x_B \hat{x} + y_B \hat{y} + z_B \hat{z})$$

$\left(\frac{d^B \bar{r}}{dt}\right)_E$ : the body-fixed frame derivative of the position vector  $\bar{r}$ , expressed/represented in the inertial Earth fixed frame E

$$\left(\frac{d^B \bar{r}}{dt}\right)_E = [0 \ 0 \ 0]^T [\text{m/s}]$$

$\left(\frac{d^B \bar{r}}{dt}\right)_B$ : the body-fixed frame derivative of the position vector  $\bar{r}$ , expressed/represented in the body fixed frame, B

$$\left(\frac{d^B \bar{r}}{dt}\right)_B = [0 \ 0 \ 0]^T [\text{m/s}]$$

5)  $\left(\frac{d^E \bar{v}^E}{dt}\right)_E$ : the inertial fixed frame derivative of the inertial velocity vector  $\bar{v}^E$ , of the aircraft expressed/represented in the inertial Earth fixed frame E.

Acceleration has two components tangential + centripetal

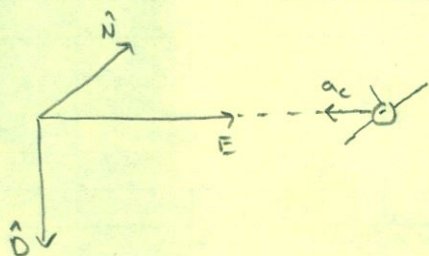
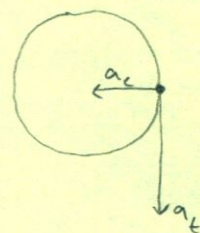
$$a_t = 0 \quad a_c = \frac{v^2}{r}$$

$$\bar{v}^E = (-100 \hat{N} + 0 \hat{E} + 0 \hat{D}) \quad |\bar{v}^E| = \sqrt{(-100)^2 + 0^2 + 0^2} = 100$$

$$\bar{r} = (0 \hat{N} + 1000 \hat{E} + 0 \hat{D}) \quad |\bar{r}| = \sqrt{0^2 + 1000^2 + 0^2} = 1000$$

$$a_c = \frac{100^2}{1000} = 10 \text{ m/s}^2$$

$$\left(\frac{d^E \bar{v}^E}{dt}\right)_E = [0 \ -10 \ 0]^T [\text{m/s}^2]$$





5  $\left( \frac{d^B}{dt} \bar{V}^E \right)_B$ : the body-fixed frame derivative of the inertial velocity vector  $\bar{V}^E$  of the aircraft expressed/represented in the body-fixed frame B

$$\left( \frac{d^B}{dt} \bar{V}^E \right)_B = \left( \frac{d^B}{dt} (0\hat{N} + 1000\hat{E} + 0\hat{S}) \right)_B = [0 \ 0 \ 0]^T [m/s^2]$$

6 Calculate  $\frac{d^E}{dt} \bar{V}^E$  using the velocity rule

$$\begin{aligned} \frac{d^E}{dt} \bar{V}^E &= \frac{d^B}{dt} \bar{V} + \bar{\omega}^{EB} \times \bar{V} \\ &= 0 + 0.1 \hat{O} \times (-100) \hat{N} \end{aligned}$$

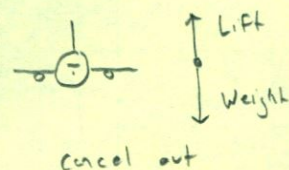
$$\frac{d^E}{dt} \bar{V}^E = -10 \hat{E} [m/s^2]$$

7  $\bar{F}$ : is the resultant/net of all forces acting on the aircraft

$$\bar{F} = m \ddot{\bar{r}} = m \dot{\bar{v}} = m a_c$$

$$F_{cent} = m \frac{v^2}{r}$$

$$\bar{F} = m(-10 \hat{E}) [N]$$



$\bar{F}_E$ : the resultant force vector expressed in the inertial Earth fixed frame E.

$$\bar{F}_E = m [0 \ -10 \ 0]^T [N]$$

$\bar{F}_B$ : the resultant force vector expressed in the body fixed frame B

$$\bar{F}_B = \bar{L}_{BE} \bar{F}_E = \begin{bmatrix} \bar{L}_{BE} \end{bmatrix} \begin{bmatrix} 0 \\ -10 \\ 0 \end{bmatrix} = \bar{F}_B = m \begin{bmatrix} 0 \\ 10 \cos \phi \\ -10 \sin \phi \end{bmatrix} [N]$$



8) Given:  $\vec{W}_E = [10 \ 20 \ -5]^T [m/s] = [W_x \ W_y \ W_z]^T$   
 $\vec{V}_E = [-100 \ 0 \ 0]^T [m/s] = [\dot{x}_E \ \dot{y}_E \ \dot{z}_E]^T$

$$\vec{V}_E^E = \vec{V}_E + \vec{W}_E$$

$\uparrow$  Inertial Velocity       $\uparrow$  Velocity of CG to air "relative velocity"       $\uparrow$  Wind velocity relative to  $E$

Find: Relative wind vector  $\vec{V}_B$   
 Which is  $\vec{V}_B = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$

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

$$\vec{V}_E = \begin{bmatrix} -100 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 10 \\ 20 \\ -5 \end{bmatrix} = \begin{bmatrix} -110 \\ -20 \\ 5 \end{bmatrix} [m/s] \quad \vec{V} = (-110 \hat{N} - 20 \hat{E} + 5 \hat{D}) [m/s]$$

$$\vec{V}_B = \vec{L}_{BE} \vec{V}_E = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\cos \phi & \sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} -110 \\ -20 \\ 5 \end{bmatrix} = \begin{bmatrix} 110 \\ 20 \cos \phi + 5 \sin \phi \\ -20 \sin \phi + 5 \cos \phi \end{bmatrix} = \vec{V}_B$$

If defined in E frame, transform to B frame  $\vec{L}_{BE} \leftarrow$  from  $\hat{X} = -\hat{N}$

$$\hat{y} = -\cos \phi \hat{E} + \sin \phi \hat{D}$$

$$\hat{z} = \sin \phi \hat{E} + \cos \phi \hat{D}$$

$$\vec{r}_E, \vec{v}_E, \vec{\omega}_E^{EB} = \begin{bmatrix} \hat{N} \\ \hat{E} \\ \hat{D} \end{bmatrix}$$

If defined in B, transform to E  $\vec{L}_{EB}$

$$\hat{N} = -\hat{x}$$

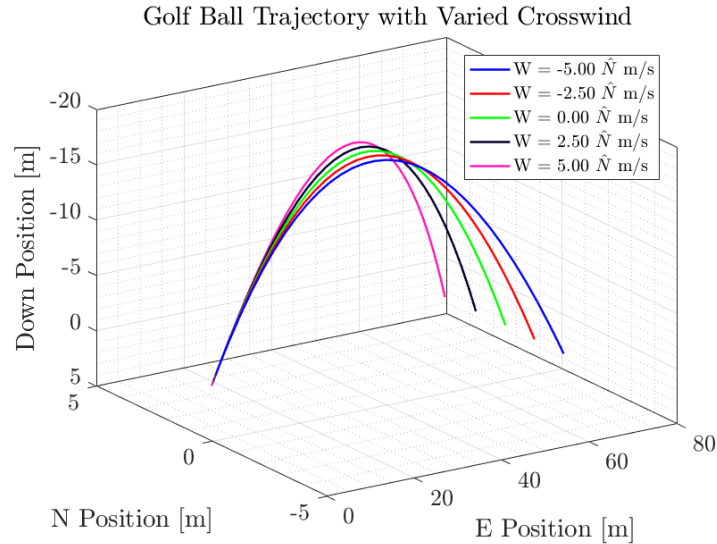
$$\hat{E} = -\cos \phi \hat{y} + \sin \phi \hat{z}$$

$$\hat{D} = \sin \phi \hat{y} + \cos \phi \hat{z}$$

$$\vec{r}_B, \vec{v}_B, \vec{\omega}_B^{EB} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix}$$

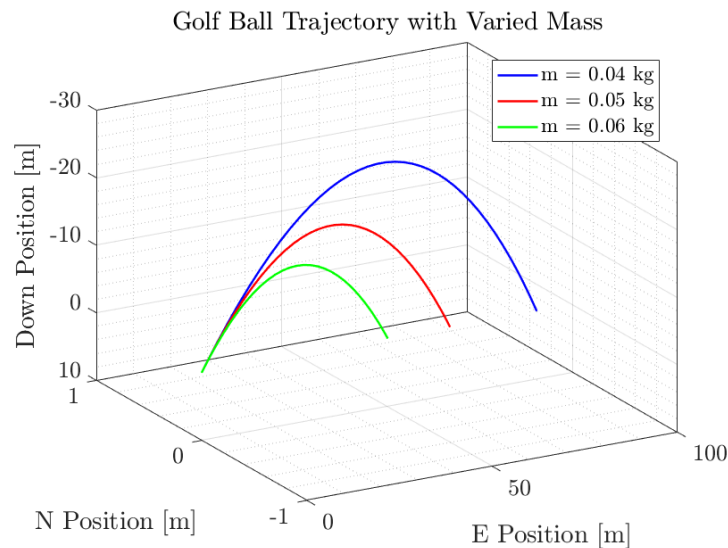
## Problem 9

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.



**Figure 1:** Golf Ball Wind Variation

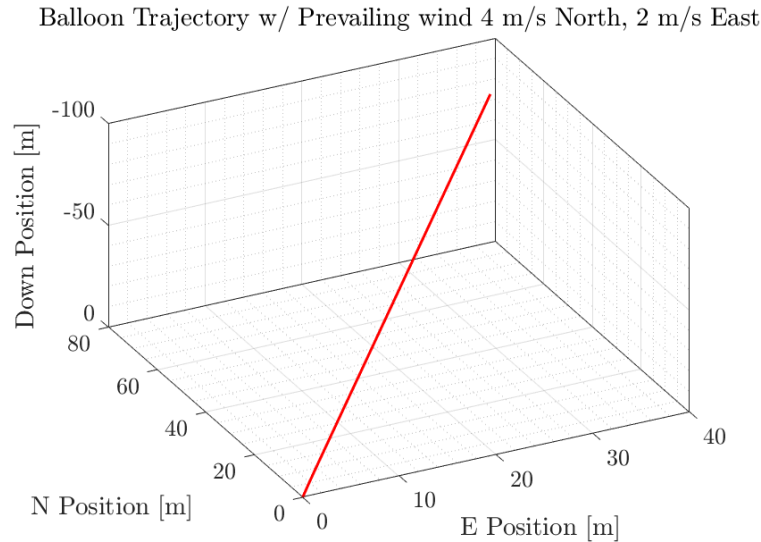
Figure 2 indicates how changes in mass effect the flight of the golf ball. For this simulation, the mass of the golf ball was altered by 20% from 0.05 kg while keeping the kinetic energy of the ball constant. This resulted in increased velocity for the lighter ball and decreased velocity for the heavier 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.



**Figure 2:** Golf Ball mass Variation

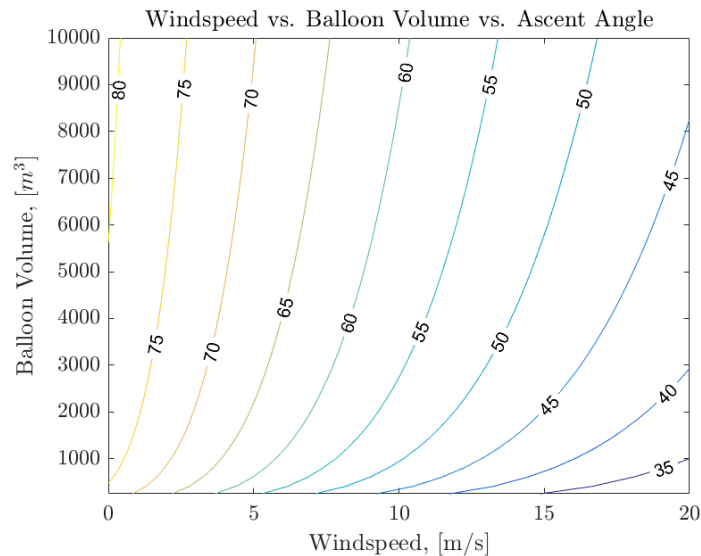
## Problem 10

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  $45^\circ$  from the vertical. That is, what volume is needed 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  $1\text{ m}^3$  of He,  $0.5\text{ kg}$  payload mass, and prevailing wind of  $4\text{ m/s}$  N and  $2\text{ m/s}$  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  $20\text{ m/s}$  North and  $0$  to  $10,000\text{ m}^3$  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  $45^\circ$ , 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

```
1  %{
2  Determines trajectory of a golf ball with a given initial velocity. Varies
3  mass and crosswind speed.
4
5  Created: 1/23/18 – Connor Ott
6  Last Modified: 1/27/18 – Connor Ott
7  %}
8
9  clear variables
10 close all
11 clc
12
13 nums = numbers;
14 %% Getting the Initial V for the trajectory of the ball.
15 vInit = [nums.V_xi, 0, nums.V_zi]; % [m/s]
16
17 %% Calling ode45 with initial velocity and position
18
19 V_0x = vInit(1);
20 V_0y = vInit(2);
21 V_0z = vInit(3);
22
23 [X_0x, X_0y, X_0z] = deal(0); % m – Initial position at (0,0,0)
24
25 t0 = 0;
26 tf = 5; % s
27
28 %% Wind Variation
29 windVels = linspace(-5, 5, 5);
30 colors = distinguishable_colors(length(windVels));
31 for i = 1:length(windVels)
32
33     legStr{i} = sprintf('W = %.2f  $\hat{N}$  m/s', windVels(i));
34
35     initialVals = [V_0x, V_0y, V_0z, 0, 0, 0, windVels(i), nums.m];
36     [~, Fwind{i}] = ode45('golfBallODE', [t0 tf], initialVals);
37
38     figure(1)
39     hold on; grid on; grid minor;
40     plot3(Fwind{i}(:, 4), Fwind{i}(:, 5), -Fwind{i}(:, 6), ...
41         'linewidth', 1.2, 'color', colors(i, :));
42     xlabel('E Position [m]')
43     ylabel('N Position [m]')
44     zlabel('Down Position [m]')
45     title('Golf Ball Trajectory with Varied Crosswind')
46     set(gca, 'TickLabelInterpreter', 'latex', ...
47         'fontsize', 13, ...
48         'box', 'on', ...
49         'Zdir', 'reverse');
```

```

50     deflect(i) = Fwind{i}(end, 5);
51 end
52 leg = legend(legStr);
53 set(leg, 'interpreter', 'latex', ...
54       'fontsize', 10);
55
56 tempFit = polyfit(windVels, deflect, 1);
57 % Ball sensitivity to crosswind
58 wSense = tempFit(1); % m deflect / m/s crosswind Vel
59 fprintf('Ball sensitivity to crosswind: %.3f m/m/s\n', wSense)
60
61 %% Mass Variation
62 % Varying mass by 20% in either direction
63 massVec = linspace(nums.m*0.8, nums.m*1.2, 3);
64 kE_max = 0.5 * nums.m * sqrt(2) * nums.V_xi; % max kinetic energy
65 colors = distinguishable_colors(length(massVec)); % plotting
66 for i = 1:length(massVec)
67     V_0x = kE_max * 2 / (massVec(i) * sqrt(2));
68     V_0z = V_0x;
69     legStr{i} = sprintf('m = %.2f kg', massVec(i));
70     initialVals = [V_0x, V_0y, V_0z, 0, 0, 0, massVec(i)];
71     [~, Fmass{i}] = ode45('golfBallODE', [t0 tf], initialVals);
72
73     figure(2)
74     hold on; grid on; grid minor;
75     plot3(Fmass{i}(:, 4), Fmass{i}(:, 5), -Fmass{i}(:, 6), ...
76           'linewidth', 1.2, 'color', colors(i, :));
77     xlabel('E Position [m]')
78     ylabel('N Position [m]')
79     zlabel('Down Position [m]')
80     title('Golf Ball Trajectory with Varied Mass')
81     set(gca, 'TickLabelInterpreter', 'latex', ...
82           'fontsize', 13, ...
83           'box', 'on', ...
84           'Zdir', 'reverse');
85
86     % Max ranges, indicates LIGHTER Ball will work best
87     rangeVec(i) = Fmass{i}(end, 4);
88 end
89 leg = legend(legStr);
90 set(leg, 'interpreter', 'latex');
91 fprintf('Maximum Range and ball mass: %.2fm, %.2fkg\n', ...
92       rangeVec(1), massVec(1));

```

```

1 function [ dfdt ] = golfBallODE( t, f )
2
3 % Importing number library
4 nums = numbers;
5 dfdt = zeros(6, 1);
6
7 V_g = [f(1), f(2), f(3)]; % Ground Speed [x, y, z] m/s
8 R = [f(4), f(5), f(6)]; % Positon [x, y, z] m
9 windy = f(7);
10 mass = f(8);
11
12 % Relative wind speed (x, y, z) m/s

```



```

13 V_rel = [V_g(1) - 0, V_g(2) - windy, V_g(3) - 0];
14
15 head = V_rel/norm(V_rel); % Heading vector (direction of rocket)
16 D = nums.rho/2 * norm(V_rel)^2 * nums.C_D * nums.A * head; % N - Drag force
17
18 F_g = [0, 0, -nums.g * mass];
19
20 % Sum of forces divided by mass to get acceleration
21 dV_gdt = (-D + F_g)/mass;
22
23 dRdt = V_g; % Vector of Ground Speed [x, y, z] m/s
24
25 % Once the rocket hits the ground, it should no longer travel in the x and
26 % z directions.
27 if R(3) <= 0 && t > 2
28     dRdt = 0;
29 end
30
31 dfdt(1:3) = dV_gdt;
32 dfdt(4:6) = dRdt;
33 dfdt(7) = 0;
34 dfdt(8) = 0;
35 end

1 classdef numbers
2     properties (Constant)
3         m = 0.05; % [kg]
4         d = 0.03; % [m]
5         C_D = 0.5; % []
6         rho = 1.0; % [kg/m^3]
7         g = 9.81; % [m/s^2]
8         V_xi = 20; % [m/s] x initial
9         V_zi = 20; % [m/s] Up initial
10    end
11    properties (SetAccess = immutable)
12        A % [m^2]
13    end
14    properties (SetAccess = public)
15        windVec = [0, -5, 0];
16    end
17    methods
18        function obj = numbers()
19            obj.A = pi * obj.d^2 / 4;
20        end
21    end
22 end

1 function colors = distinguishable_colors(n_colors,bg,func)
2 % DISTINGUISHABLE_COLORS: pick colors that are maximally perceptually distinct
3 %
4 % When plotting a set of lines, you may want to distinguish them by color.
5 % By default, Matlab chooses a small set of colors and cycles among them,
6 % and so if you have more than a few lines there will be confusion about
7 % which line is which. To fix this problem, one would want to be able to
8 % pick a much larger set of distinct colors, where the number of colors
9 % equals or exceeds the number of lines you want to plot. Because our

```

```

10 % ability to distinguish among colors has limits, one should choose these
11 % colors to be "maximally perceptually distinguishable."
12 %
13 % This function generates a set of colors which are distinguishable
14 % by reference to the "Lab" color space, which more closely matches
15 % human color perception than RGB. Given an initial large list of possible
16 % colors, it iteratively chooses the entry in the list that is farthest (in
17 % Lab space) from all previously-chosen entries. While this "greedy"
18 % algorithm does not yield a global maximum, it is simple and efficient.
19 % Moreover, the sequence of colors is consistent no matter how many you
20 % request, which facilitates the users' ability to learn the color order
21 % and avoids major changes in the appearance of plots when adding or
22 % removing lines.
23 %
24 % Syntax:
25 %   colors = distinguishable_colors(n_colors)
26 % Specify the number of colors you want as a scalar, n_colors. This will
27 % generate an n_colors-by-3 matrix, each row representing an RGB
28 % color triple. If you don't precisely know how many you will need in
29 % advance, there is no harm (other than execution time) in specifying
30 % slightly more than you think you will need.
31 %
32 %   colors = distinguishable_colors(n_colors,bg)
33 % This syntax allows you to specify the background color, to make sure that
34 % your colors are also distinguishable from the background. Default value
35 % is white. bg may be specified as an RGB triple or as one of the standard
36 % "ColorSpec" strings. You can even specify multiple colors:
37 %       bg = {'w','k'}
38 % or
39 %       bg = [1 1 1; 0 0 0]
40 % will only produce colors that are distinguishable from both white and
41 % black.
42 %
43 %   colors = distinguishable_colors(n_colors,bg,rgb2labfunc)
44 % By default, distinguishable_colors uses the image processing toolbox's
45 % color conversion functions makeform and applyform. Alternatively, you
46 % can supply your own color conversion function.
47 %
48 % Example:
49 %   c = distinguishable_colors(25);
50 %   figure
51 %   image(reshape(c,[1 size(c)]))
52 %
53 % Example using the file exchange's 'colspace':
54 %   func = @(x) colorspace('RGB->Lab',x);
55 %   c = distinguishable_colors(25,'w',func);
56 %
57 % Copyright 2010-2011 by Timothy E. Holy
58 %
59 % Parse the inputs
60 if (nargin < 2)
61     bg = [1 1 1]; % default white background
62 else
63     if iscell(bg)
64         % User specified a list of colors as a cell array

```



```

65     bgc = bg;
66     for i = 1:length(bgc)
67         bgc{i} = parsecolor(bgc{i});
68     end
69     bg = cat(1,bgc{:});
70 else
71     % User specified a numeric array of colors (n-by-3)
72     bg = parsecolor(bg);
73 end
74 end
75
76 % Generate a sizable number of RGB triples. This represents our space of
77 % possible choices. By starting in RGB space, we ensure that all of the
78 % colors can be generated by the monitor.
79 n_grid = 30; % number of grid divisions along each axis in RGB space
80 x = linspace(0,1,n_grid);
81 [R,G,B] = ndgrid(x,x,x);
82 rgb = [R(:) G(:) B(:)];
83 if (n_colors > size(rgb,1)/3)
84     error('You can''t readily distinguish that many colors');
85 end
86
87 % Convert to Lab color space, which more closely represents human
88 % perception
89 if (nargin > 2)
90     lab = func(rgb);
91     bglab = func(bg);
92 else
93     C = makecform('srgb2lab');
94     lab = applycform(rgb,C);
95     bglab = applycform(bg,C);
96 end
97
98 % If the user specified multiple background colors, compute distances
99 % from the candidate colors to the background colors
100 mindist2 = inf(size(rgb,1),1);
101 for i = 1:size(bglab,1)-1
102     dX = bsxfun(@minus,lab,bglab(i,:)); % displacement all colors from bg
103     dist2 = sum(dX.^2,2); % square distance
104     mindist2 = min(dist2,mindist2); % dist2 to closest previously-chosen
        color
105 end
106
107 % Iteratively pick the color that maximizes the distance to the nearest
108 % already-picked color
109 colors = zeros(n_colors,3);
110 lastlab = bglab(end,:); % initialize by making the "previous" color equal
        to background
111 for i = 1:n_colors
112     dX = bsxfun(@minus,lab,lastlab); % displacement of last from all colors on
        list
113     dist2 = sum(dX.^2,2); % square distance
114     mindist2 = min(dist2,mindist2); % dist2 to closest previously-chosen
        color
115     [~,index] = max(mindist2); % find the entry farthest from all previously-

```

```

        chosen_colors
116     colors(i,:) = rgb(index,:); % save for output
117     lastlab = lab(index,:); % prepare for next iteration
118 end
119 end
120
121 function c = parsecolor(s)
122     if ischar(s)
123         c = colorstr2rgb(s);
124     elseif isnumeric(s) && size(s,2) == 3
125         c = s;
126     else
127         error('MATLAB:InvalidColorSpec','Color specification cannot be parsed.');
```

```

128     end
129 end
130
131 function c = colorstr2rgb(c)
132     % Convert a color string to an RGB value.
133     % This is cribbed from Matlab's whitebg function.
134     % Why don't they make this a stand-alone function?
135     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];
136     cspec = 'rgbwcmlyk';
137     k = find(cspec==c(1));
138     if isempty(k)
139         error('MATLAB:InvalidColorString','Unknown color string.');
```

```

140     end
141     if k~=3 || length(c)==1,
142         c = rgbspec(k,:);
143     elseif length(c)>2,
144         if strcmpi(c(1:3),'bla')
145             c = [0 0 0];
146         elseif strcmpi(c(1:3),'blu')
147             c = [0 0 1];
148         else
149             error('MATLAB:UnknownColorString','Unknown color string.');
```

```

150     end
151 end
152 end

1 % {
2 % Determines relationship between wind, volume, and angle of ascent for a
3 % balloon sonde.
4
5 % Created: 1/23/18 - Connor Ott
6 % Last Modified: 1/27/18 - Connor Ott
7 % }
8
9 clear variables
10 close all
11 clc
12 set(0, 'defaulttextinterpreter', 'latex');
13
14 % Number library
15 nums = balloonNums;
16 t0 = 0;
17 tf = 20; % s

```



```

18
19 %% Validation case
20 initialVals = [0, 0, 0, 0, 0, 0, 0, 1];
21 [~, F] = ode45('balloonODE', [t0 tf], initialVals);
22
23 figure
24 hold on; grid on; grid minor;
25 plot3(F(:, 4), F(:, 5), -F(:, 6), ...
26     'r-', 'linewidth', 1.5)
27 xlabel('E Position [m]')
28 ylabel('N Position [m]')
29 zlabel('Down Position [m]')
30 title('Balloon Trajectory w/ Prevailing wind 4 m/s North, 2 m/s East')
31 set(gca, 'TickLabelInterpreter', 'latex', ...
32     'fontsize', 13, ...
33     'box', 'on', ...
34     'Zdir', 'reverse');
35
36 %% Wind and Volume Variation
37 windVels = linspace(0, 20, 20); % [m/s]
38 vols = linspace(0, 10000, 50); % [m^3]
39
40 FCell = cell(length(windVels), length(vols));
41 angles = zeros(length(windVels), length(vols));
42
43 % Varying both volume and North windspeed
44 for i = 1:length(windVels)
45     for j = 1:length(vols)
46         initialVals = [0, 0, 0, 0, 0, 0, windVels(i), vols(j)];
47         [~, FCell{i, j}] = ode45('balloonODE', [t0 tf], initialVals);
48
49         angles(i, j) = atand(FCell{i, j}(end, 6)/FCell{i, j}(end, 5));
50     end
51 end
52
53 %% Plotting Data
54 levels = 35:5:85;
55 figure
56 hold on;
57 set(gca, 'TickLabelInterpreter', 'latex', ...
58     'fontsize', 12, ...
59     'box', 'on');
60 contour(windVels, vols, angles, levels, 'showtext', 'on')
61 title('Windspeed vs. Balloon Volume vs. Ascent Angle')
62 xlabel('Windspeed, [m/s]')
63 ylabel('Balloon Volume, [m^3]')
64 axis([0, 20, 250, 10000])
65
66 %%
67
68 1 % Event function for balloon sonde simulation
69 2
70 3 function [ dfdt ] = balloonODE( t, f )
71 4
72 5 % Importing number library
73 6 nums = balloonNums;

```

```

7  dfdt = zeros(6, 1);
8
9  V_g = [f(1), f(2), f(3)]; % Ground Speed [x, y, z] m/s
10 R = [f(4), f(5), f(6)]; % Positon [x, y, z] m
11 crossWind = f(7);
12 volume = f(8);
13 mass = nums.m + volume*nums.rho_He; % total mass of sonde
14
15 % Balloon shape for calculating drag
16 r = (3/(pi*4) * volume)^(1/3);
17 A = pi * r^2;
18
19 windx = nums.WE;
20 windy = nums.WN;
21
22 % Relative wind speed (x, y, z) m/s
23 V_rel = [V_g(1) - windx, V_g(2) - windy - crossWind, V_g(3)];
24
25 head = V_rel/norm(V_rel); % Heading vector (direction of rocket)
26 D = nums.rho_air/2 * norm(V_rel)^2 * nums.CD * A * head; % N - Drag force
27
28 % Gravity
29 F_g = [0, 0, -nums.g * mass];
30
31 % Bouyant Force
32 F_b = [0, 0, nums.g * nums.rho_air * volume];
33
34 % if norm(F_b) < norm(F_g)
35 %     dV_gdt = [0, 0, 0];
36 % else
37 %     % Sum of forces divided by mass to get acceleration
38 %     dV_gdt = (-D + F_g + F_b)/mass;
39 % end
40
41 dRdt = V_g; % Vector of Ground Speed [x, y, z] m/s
42
43
44 dfdt(1:3) = dV_gdt;
45 dfdt(4:6) = dRdt;
46 dfdt(7) = 0;
47 dfdt(8) = 0;
48 end

```

```

1  % Small number library for balloon simulation
2
3  classdef balloonNums
4      properties (Constant)
5          m = 0.5; % [kg]
6          CD = 0.5; % []
7          rho_air = 1.225; % [kg/m^3]
8          rho_He = 0.164; % [kg/m^3]
9          g = 9.81; % [m/s^2]
10         WE = 2; % [m/s] x wind
11         WN = 4; % [m/s] y wind
12     end
13     properties (SetAccess = immutable)

```



```
14         A           % [m^2]
15         r           % [m]
16     end
17 end
```