Cleaning

```
clear
clc
```

Declare Variables

```
% Constant variables
altitude = 25500; % Desired altitude of 25000 [m]
payloadMass = 500; % Desired payload mass of 500 [kg]
FS = [1:1:100]'; % Self-selected number in range 1:5
Pgage = 10; % Gage pressure [Pa]
gravity = 9.81; % Acceleration of gravity [m/s^2]

% Variable variables (o_o )
    % Measurements based on information found in:
    % http://www.matweb.com/search/datasheet_print.aspx?matguid=bcd1ad1e1a7445aca5321d26f00bdf1
materialThickness = 3.81 * 10^-5; % Balloon material thickness [m]
YS = 15.5 * 10^6; % Yield strength of balloon material [Pa]
materialDensity = 924; % Density of material [kg/m^3]
gasConst = 8.31447e3; % Universal gas constant [J/kg*K]
gasConstHe = 2.0769e3; % Gas constant of Helium [J/kg*K]

% Create table for data
data = table(FS, 'VariableNames', {'FactorSafety'});
```

Declare equations

```
% Calculate sphere volume given radius
sphereVol = @(r) (4/3) * pi * (r.^3)
```

```
sphereVol = function_handle with value:
    @(r)(4/3)*pi*(r.^3)
```

```
% Calculate yield strength given factor of safety, gage pressure, radius,
% and thickness
yieldStrength = @(F_S, Pgage, r, t) (F_S .* (Pgage * r)) ./ (2 * t)
```

```
yieldStrength = function_handle with value:
    @(F_S,Pgage,r,t)(F_S.*(Pgage*r))./(2*t)
```

```
% Calculate stress given gage pressure, radius, and thickness
stress = @(Pgage, r, t) (Pgage * r) / (2 * t)
```

```
stress = function_handle with value:
    @(Pgage,r,t)(Pgage*r)/(2*t)
```

```
% Calculate radius given density of air, pressure of air, factor of safety,
% yield strength of material, and temperature
radius = @(denAir, pressureAir, FS, YS, temp) abs(payloadMass ./ ( ((4/3) * denAir * pi) - (((m
```

```
radius = function_handle with value:
```

```
    @(denAir,pressureAir,FS,YS,temp)abs(payloadMass./(((4/3)*denAir*pi)-(((materialDensity*2*pi*Pgage).*FS)./YS)-(((
```

```
% Calculate buoyancy force given fluid density, gravity, and volume
buoyancy = @(density, volume) density * gravity * volume
```

```
buoyancy = function_handle with value:
    @(density,volume)density*gravity*volume
```

```
% Calculate force of gravity given mass
weight = @(mass) mass * gravity
```

```
weight = function_handle with value:
    @(mass)mass*gravity
```

```
% Calculate thickness of balloon material
thick = @(FS, radius) (FS .* Pgage .* radius) ./ (2 .* YS);
```

Calculate approximate balloon radius

```
bisonTopSpeed = 35 * 0.44704 % [m/s]
```

```
bisonTopSpeed = 15.6464
```

Begin calculations

```
% Find atmospheric conditions based on 1976 model
[temp, ~, pressure, densityAir] = atmoscoesa(altitude)
```

```
temp = 222.1500
pressure = 2.3250e+03
densityAir = 0.0365
```

```
% Find radius of balloon
data.balloonRadius = radius(densityAir, pressure, data.FactorSafety, YS, temp); % Radius [m]

% Find volume of balloon
data.balloonVolume = sphereVol(data.balloonRadius); % Volume [m^3]

% Find buoyancy force
data.forceBuoyancy = buoyancy(densityAir, data.balloonVolume); % Buoyancy force [N]

% Find force of gravity (weight of balloon)
data.thickness = thick(data.FactorSafety, data.balloonRadius); % Thickness [m]
data.amtBalloonMaterial = data.balloonVolume - sphereVol(data.balloonRadius - data.thickness);
data.massHeFBE = (densityAir * data.balloonVolume) - payloadMass - (materialDensity * data.amtB
data.totalMass = (data.amtBalloonMaterial * materialDensity) + payloadMass + data.massHeFBE; %
data.forceGravity = weight(data.totalMass); % Force of gravity of the system [N] --> should be

% Find mass of He from PV=nRT equation
massHePVNRT = (pressure .* data.balloonVolume) ./ (gasConst * temp);
```

Find effects of temperature change

```
% Given values
alpha_sb = 0.6;
epsilon_b = 0.8;
alpha_eb = epsilon_b;
q_sun = 1353; % [W/m^2]
q_earth = 237; % [W/m^2]

% Calculate max and min temperatures
data.Q_dot_solar = (alpha_sb * q_sun * pi) .* (data.balloonRadius .^ 2); % Change in thermal er
data.Q_dot_earth = (alpha_eb * q_earth * pi) .* (data.balloonRadius .^ 2); % Change in  thermal
Tday = ( (data.Q_dot_solar + data.Q_dot_earth) ./ ( (4 * pi * 5.67e-8 * .8) * (data.balloonRadi
Tnight = ( data.Q_dot_earth ./ ( (4 * pi * 5.67e-8 * .8) * (data.balloonRadius .^ 2) ) ) .^ (1/
Tday = Tday(1)
```

```
Tday = 272.5641
```

```
Tnight = Tnight(1)
```

```
Tnight = 179.7945
```

```
% Calculate the density of helium at different temperatures
denHeBase = (pressure + Pgage) / (gasConstHe * temp) % Density of helium at conditions at 25 km
```

```
denHeBase = 0.0051
```

```
denHeNight = (pressure + Pgage) / (gasConstHe * Tnight) % Density of helium at night temp
```

```
denHeNight = 0.0063
```

```
denHeDay = (pressure + Pgage) / (gasConstHe * Tday) % Density of helium at day temp
```

```
denHeDay = 0.0041
```

```
% Calculate the new volumes corresponding to the new densities
volHeBase = (data.massHeFBE ./ denHeBase); % Volume [m^3]
data.volHeNight = (data.massHeFBE ./ denHeNight); % Volume [m^3]
data.volHeDay = (data.massHeFBE ./ denHeDay); % Volume [m^3]

% Calculate the density of surrounding air at new temperatures
denAirBase = data.totalMass ./ volHeBase; % Density [kg/m^3]
denAirNight = data.totalMass ./ data.volHeNight; % Density [kg/m^3]
denAirDay = data.totalMass ./ data.volHeDay; % Density [kg/m^3]
denAirBase = denAirBase(1)
```

```
denAirBase = 0.0366
```

```
denAirNight = denAirNight(1)
```

```
denAirNight = 0.0452
```

```
denAirDay = denAirDay(1)
```

```
denAirDay = 0.0298
```

```
% Calculate altitude corresponding to air densities
altAirBase(i) = densityalt(denAirBase);
altAirNight(i) = densityalt(denAirNight);
altAirDay(i) = densityalt(denAirDay);
```

Calculate amount of ballast necessary to achieve flight parameters

```
% Find atmospheric properties at high and low altitudes
    % Need to take into account pressure and density variations at the
    % altitudes above and below the mean altitude
[~, ~, pressureMax, densityAirMax] = atmoscoesa(26000) % Find atmospheric information at max al
```

```
pressureMax = 2.1531e+03
densityAirMax = 0.0337
```

```
[~, ~, pressureMin, densityAirMin] = atmoscoesa(24000) % Find atmospheric information at min al
```

```
pressureMin = 2.9305e+03
densityAirMin = 0.0463
```

```
% Calculate amount of ballast necessary for neutral buoyancy within
% altitude limit
data.massBallastDay = densityAirMax .* data.volHeDay - data.totalMass; % Mass [kg]

% Check that balloon with extra mass will be within range
newTotalMass = data.massBallastDay + data.totalMass; % Mass of system + mass of ballast [kg]
denAirTest = newTotalMass ./ data.volHeDay% Density of air at maximum altitude with new total w
```

```
denAirTest = 100×1
    0.0337
    0.0337
    0.0337
    0.0337
    0.0337
    0.0337
    0.0337
    0.0337
    0.0337
    0.0337
      ⋮
```

```
altAirTest = zeros(length(data.FactorSafety), 1);
for i = 1:length(denAirTest)
    altAirTest(i) = densityalt(abs(denAirTest(i))); % Calculate altitude given densities
end

% Complete tests to make find factors of safety that result in correct
% altitude expectations
altAirTest
```

```
altAirTest = 100×1
10^4 ×
    2.6000
    2.6000
```

```
    2.6000
    2.6000
    2.6000
    2.6000
    2.6000
    2.6000
    2.6000
    2.6000
      ⋮
```

```
test1 = altAirTest <= 26000 & altAirTest > altAirNight % Test to see when within range and abov
```

```
test1 = 100×1 logical array
   1
   1
   1
   1
   1
   1
   1
   1
   1
   1
      ⋮
```

```
goodFS = data.FactorSafety(test1) % Find factors of safety that meet requirements
```

```
goodFS = 38×1
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
      ⋮
```

```
data
```

```
data = 100×14 table
```

...

	FactorSafety	balloonRadius	balloonVolume	forceBuoyancy	thickness
1	1	15.7544	1.6379e+04	5.8584e+03	5.0821e-06
2	2	15.9113	1.6874e+04	6.0352e+03	1.0265e-05
3	3	16.0747	1.7399e+04	6.2230e+03	1.5556e-05
4	4	16.2450	1.7957e+04	6.4228e+03	2.0961e-05
5	5	16.4227	1.8553e+04	6.6359e+03	2.6488e-05
6	6	16.6084	1.9190e+04	6.8637e+03	3.2145e-05
7	7	16.8029	1.9872e+04	7.1076e+03	3.7942e-05

	FactorSafety	balloonRadius	balloonVolume	forceBuoyancy	thickness
8	8	17.0068	2.0604e+04	7.3695e+03	4.3889e-05
9	9	17.2210	2.1393e+04	7.6514e+03	4.9996e-05
10	10	17.4464	2.2244e+04	7.9558e+03	5.6279e-05
11	11	17.6840	2.3165e+04	8.2854e+03	6.2750e-05
12	12	17.9352	2.4166e+04	8.6435e+03	6.9427e-05
13	13	18.2013	2.5258e+04	9.0339e+03	0.0001
14	14	18.4839	2.6453e+04	9.4613e+03	0.0001
15	15	18.7849	2.7766e+04	9.9311e+03	0.0001
16	16	19.1066	2.9217e+04	1.0450e+04	0.0001
17	17	19.4514	3.0828e+04	1.1026e+04	0.0001
18	18	19.8226	3.2627e+04	1.1670e+04	0.0001
19	19	20.2239	3.4648e+04	1.2393e+04	0.0001
20	20	20.6598	3.6937e+04	1.3211e+04	0.0001
21	21	21.1358	3.9550e+04	1.4146e+04	0.0001
22	22	21.6590	4.2560e+04	1.5222e+04	0.0002
23	23	22.2381	4.6066e+04	1.6476e+04	0.0002
24	24	22.8847	5.0202e+04	1.7956e+04	0.0002
25	25	23.6136	5.5154e+04	1.9727e+04	0.0002
26	26	24.4454	6.1190e+04	2.1886e+04	0.0002
27	27	25.4082	6.8708e+04	2.4575e+04	0.0002
28	28	26.5432	7.8334e+04	2.8018e+04	0.0002
29	29	27.9128	9.1096e+04	3.2582e+04	0.0003
30	30	29.6174	1.0882e+05	3.8923e+04	0.0003
31	31	31.8331	1.3512e+05	4.8329e+04	0.0003
32	32	34.9078	1.7818e+05	6.3729e+04	0.0004
33	33	39.6704	2.6151e+05	9.3534e+04	0.0004
34	34	48.9490	4.9127e+05	1.7571e+05	0.0005
35	35	98.8507	4.0460e+06	1.4471e+06	0.0011
36	36	53.7050	6.4883e+05	2.3207e+05	0.0006
37	37	41.5437	3.0033e+05	1.0742e+05	0.0005
38	38	35.9973	1.9539e+05	6.9884e+04	0.0004
39	39	32.5752	1.4479e+05	5.1788e+04	0.0004
40	40	30.1683	1.1501e+05	4.1136e+04	0.0004
41	41	28.3448	9.5391e+04	3.4118e+04	0.0004

	FactorSafety	balloonRadius	balloonVolume	forceBuoyancy	thickness
42	42	26.8949	8.1489e+04	2.9146e+04	0.0004
43	43	25.7025	7.1124e+04	2.5439e+04	0.0004
44	44	24.6969	6.3098e+04	2.2568e+04	0.0004
45	45	23.8322	5.6700e+04	2.0280e+04	0.0003
46	46	23.0772	5.1480e+04	1.8413e+04	0.0003
47	47	22.4095	4.7140e+04	1.6860e+04	0.0003
48	48	21.8130	4.3475e+04	1.5550e+04	0.0003
49	49	21.2754	4.0338e+04	1.4428e+04	0.0003
50	50	20.7870	3.7624e+04	1.3457e+04	0.0003
51	51	20.3407	3.5252e+04	1.2609e+04	0.0003
52	52	19.9304	3.3162e+04	1.1861e+04	0.0003
53	53	19.5513	3.1305e+04	1.1197e+04	0.0003
54	54	19.1995	2.9645e+04	1.0603e+04	0.0003
55	55	18.8717	2.8153e+04	1.0069e+04	0.0003
56	56	18.5652	2.6803e+04	9.5866e+03	0.0003
57	57	18.2777	2.5577e+04	9.1481e+03	0.0003
58	58	18.0072	2.4458e+04	8.7480e+03	0.0003
59	59	17.7521	2.3433e+04	8.3814e+03	0.0003
60	60	17.5108	2.2491e+04	8.0443e+03	0.0003
61	61	17.2821	2.1621e+04	7.7332e+03	0.0003
62	62	17.0649	2.0816e+04	7.4453e+03	0.0003
63	63	16.8583	2.0069e+04	7.1781e+03	0.0003
64	64	16.6613	1.9374e+04	6.9294e+03	0.0003
65	65	16.4732	1.8725e+04	6.6973e+03	0.0003
66	66	16.2933	1.8118e+04	6.4803e+03	0.0003
67	67	16.1210	1.7550e+04	6.2769e+03	0.0003
68	68	15.9558	1.7016e+04	6.0859e+03	0.0003
69	69	15.7972	1.6513e+04	5.9062e+03	0.0004
70	70	15.6447	1.6039e+04	5.7368e+03	0.0004
71	71	15.4979	1.5592e+04	5.5768e+03	0.0004
72	72	15.3565	1.5169e+04	5.4255e+03	0.0004
73	73	15.2201	1.4769e+04	5.2822e+03	0.0004
74	74	15.0884	1.4389e+04	5.1463e+03	0.0004
75	75	14.9611	1.4028e+04	5.0172e+03	0.0004

	FactorSafety	balloonRadius	balloonVolume	forceBuoyancy	thickness
76	76	14.8381	1.3684e+04	4.8944e+03	0.0004
77	77	14.7190	1.3357e+04	4.7775e+03	0.0004
78	78	14.6036	1.3046e+04	4.6660e+03	0.0004
79	79	14.4918	1.2748e+04	4.5597e+03	0.0004
80	80	14.3833	1.2464e+04	4.4580e+03	0.0004
81	81	14.2780	1.2192e+04	4.3608e+03	0.0004
82	82	14.1757	1.1932e+04	4.2678e+03	0.0004
83	83	14.0762	1.1683e+04	4.1786e+03	0.0004
84	84	13.9796	1.1444e+04	4.0931e+03	0.0004
85	85	13.8855	1.1214e+04	4.0110e+03	0.0004
86	86	13.7939	1.0994e+04	3.9321e+03	0.0004
87	87	13.7046	1.0782e+04	3.8563e+03	0.0004
88	88	13.6177	1.0578e+04	3.7833e+03	0.0004
89	89	13.5329	1.0381e+04	3.7131e+03	0.0004
90	90	13.4501	1.0192e+04	3.6454e+03	0.0004
91	91	13.3694	1.0010e+04	3.5802e+03	0.0004
92	92	13.2906	9.8337e+03	3.5172e+03	0.0004
93	93	13.2135	9.6638e+03	3.4564e+03	0.0004
94	94	13.1383	9.4996e+03	3.3977e+03	0.0004
95	95	13.0647	9.3409e+03	3.3409e+03	0.0004
96	96	12.9928	9.1874e+03	3.2860e+03	0.0004
97	97	12.9224	9.0389e+03	3.2329e+03	0.0004
98	98	12.8535	8.8951e+03	3.1815e+03	0.0004
99	99	12.7860	8.7558e+03	3.1317e+03	0.0004
100	100	12.7200	8.6208e+03	3.0834e+03	0.0004

Pick a factor of safety and gather measurements

```
% Selected factor of safety of 11
out = data(11, :)
```

out = 1×14 table

...

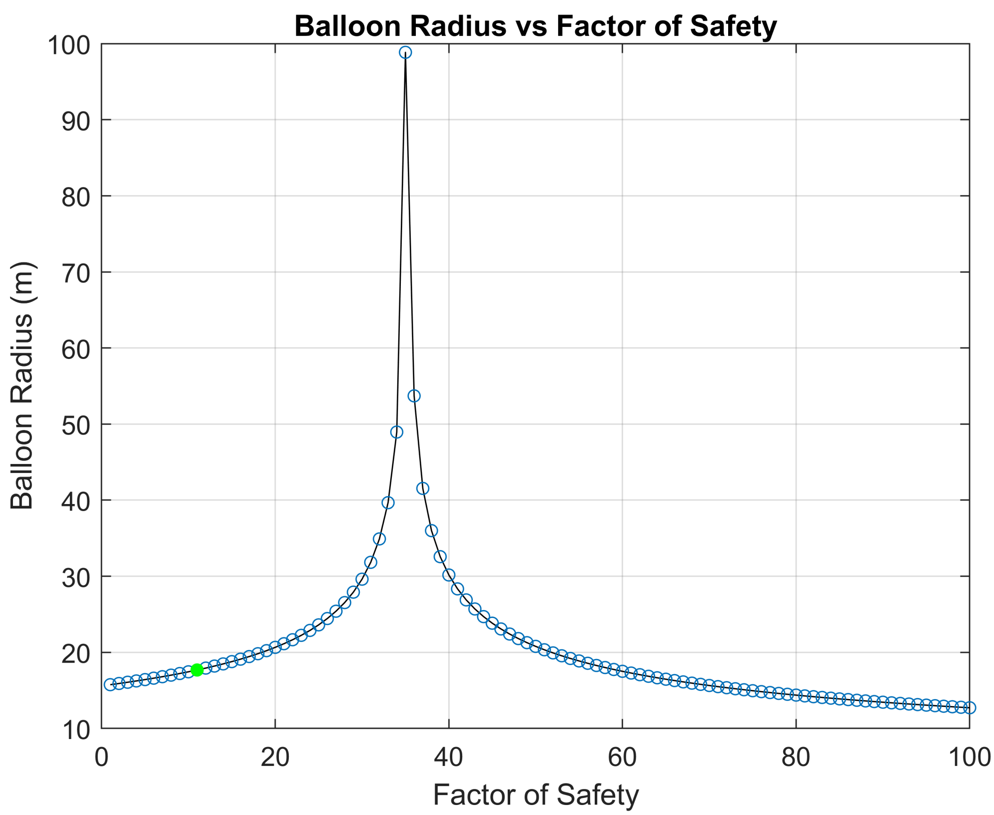
	FactorSafety	balloonRadius	balloonVolume	forceBuoyancy	thickness
1	11	17.6840	2.3165e+04	8.2854e+03	6.2750e-05

```
costHe = 4.29 * (out.massHeFBE / .16931) % Cost of Helium [$]
```

```
costHe = 2.9578e+03
```

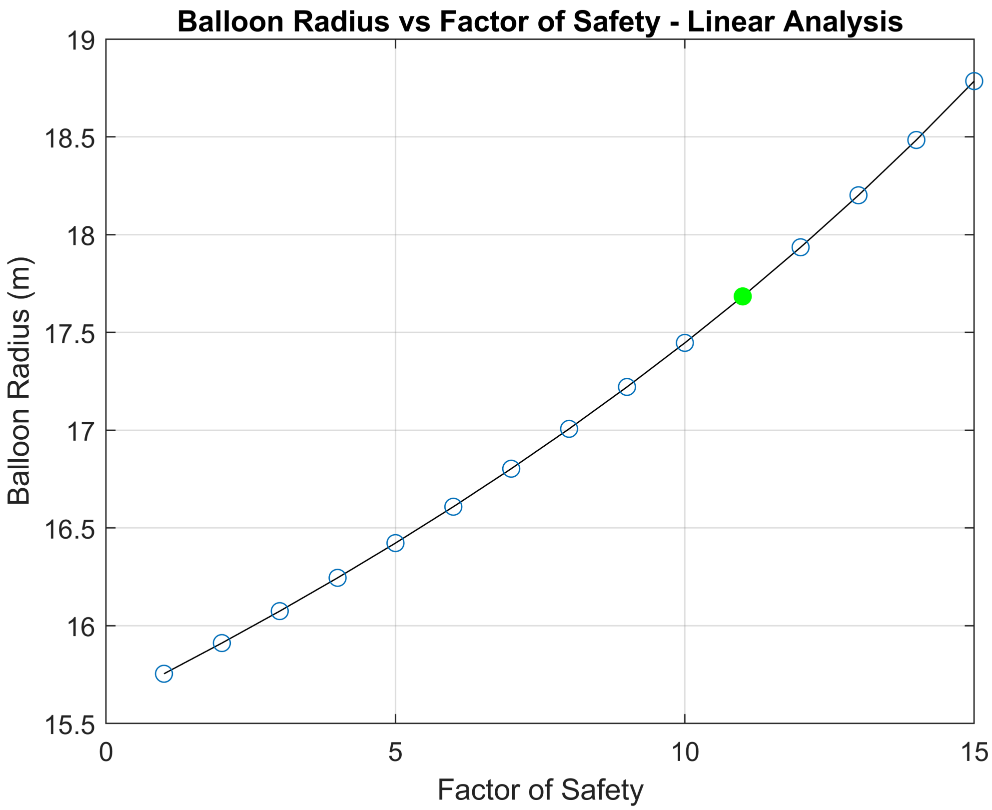
Graphs of features vs Factor of Safety

```
% Plotted so far:
% - Balloon radius
% - Balloon volume
% - Force of buoyancy
% - Thickness
% - Mass of helium
% - Ballast mass

% ------------------------------------------------------------------------------------------- %

plot(data.FactorSafety, data.balloonRadius, 'k')
hold on
scatter(data.FactorSafety, data.balloonRadius, 20, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.balloonRadius(11), 20, 'filled', 'MarkerFaceColor', 'g', 'M
hold off
grid on
title("Balloon Radius vs Factor of Safety")
xlabel("Factor of Safety")
ylabel("Balloon Radius (m)")
```

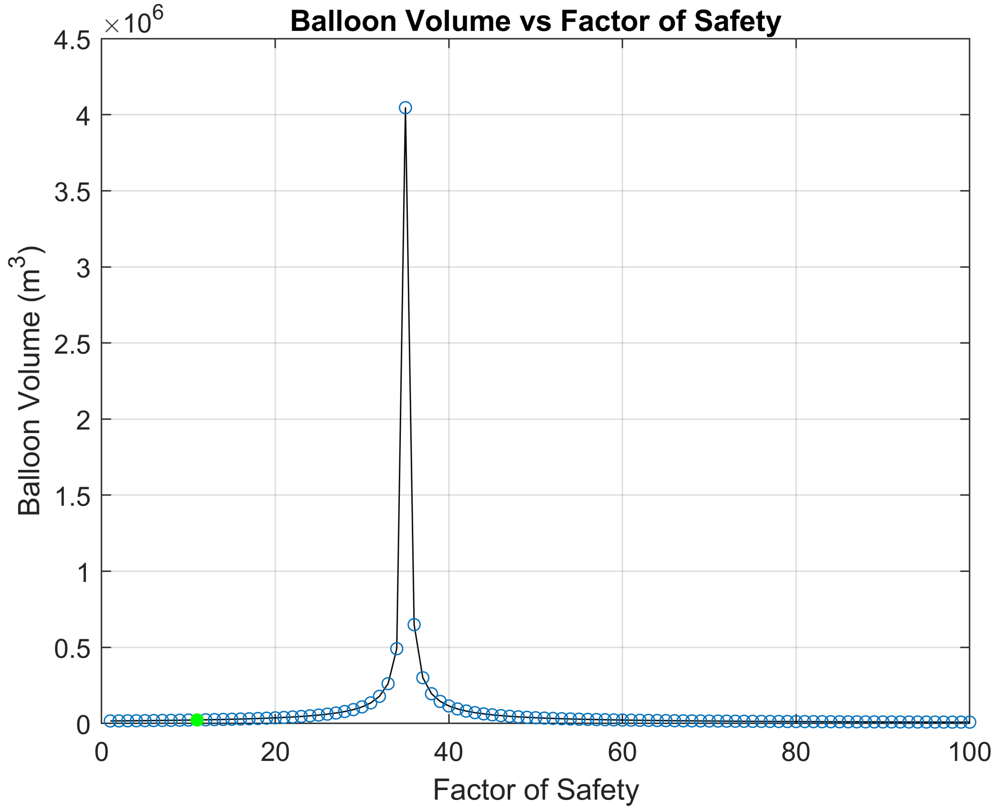



```
% Linear analysis
plot(data.FactorSafety(1:15), data.balloonRadius(1:15), 'k')
```

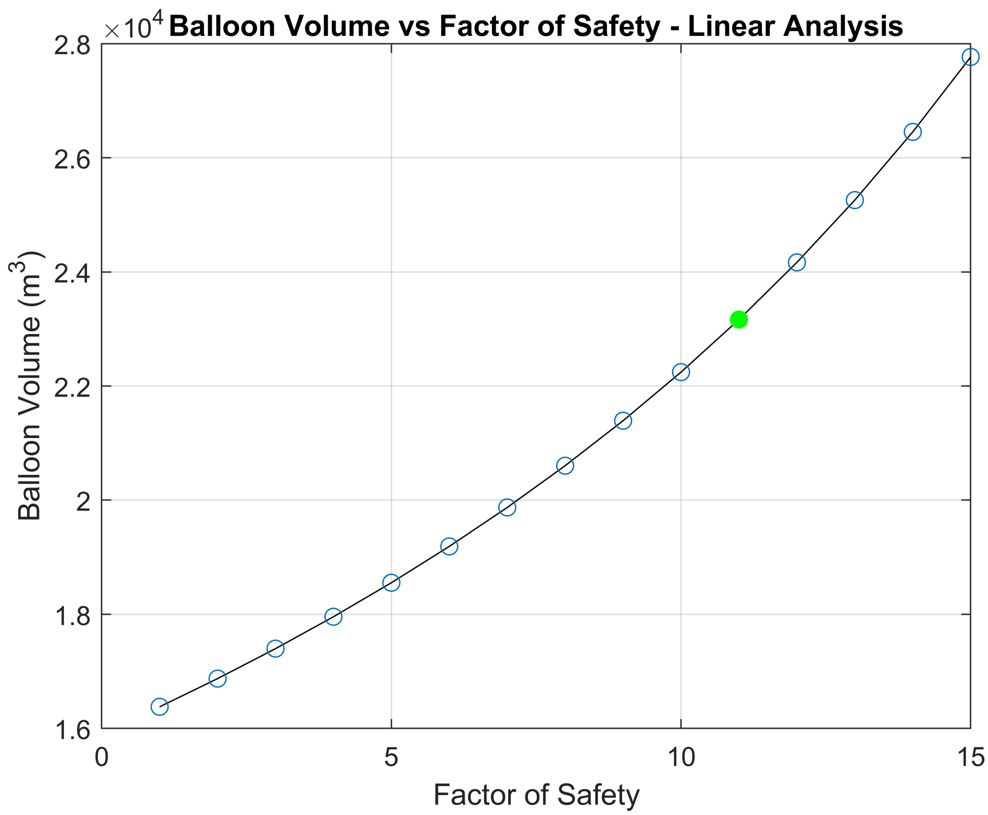
```
hold on
scatter(data.FactorSafety(1:15), data.balloonRadius(1:15), 40, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.balloonRadius(11), 40, 'filled', 'MarkerFaceColor', 'g', 'M
hold off
grid on
title("Balloon Radius vs Factor of Safety - Linear Analysis")
xlabel("Factor of Safety")
ylabel("Balloon Radius (m)")
```



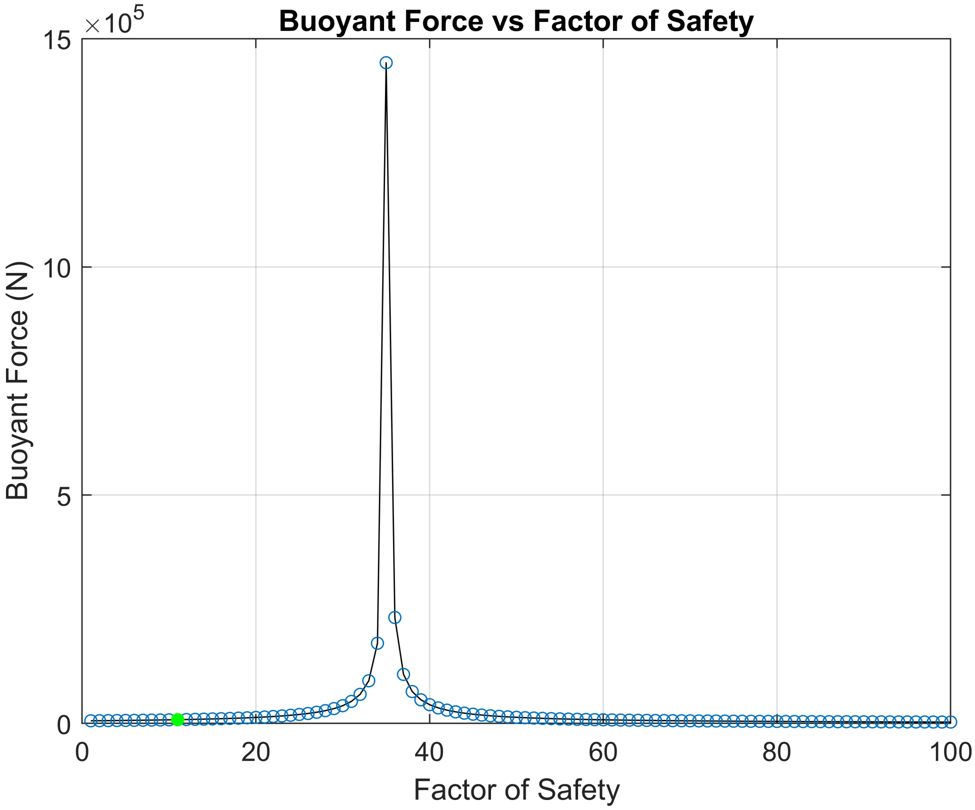

```
% ------------------------------------------------------------------------------------------ %

plot(data.FactorSafety, data.balloonVolume, 'k')
hold on
scatter(data.FactorSafety, data.balloonVolume, 20, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.balloonVolume(11), 20, 'filled', 'MarkerFaceColor', 'g', 'M
hold off
grid on
title("Balloon Volume vs Factor of Safety")
xlabel("Factor of Safety")
ylabel("Balloon Volume (m^3)")
```



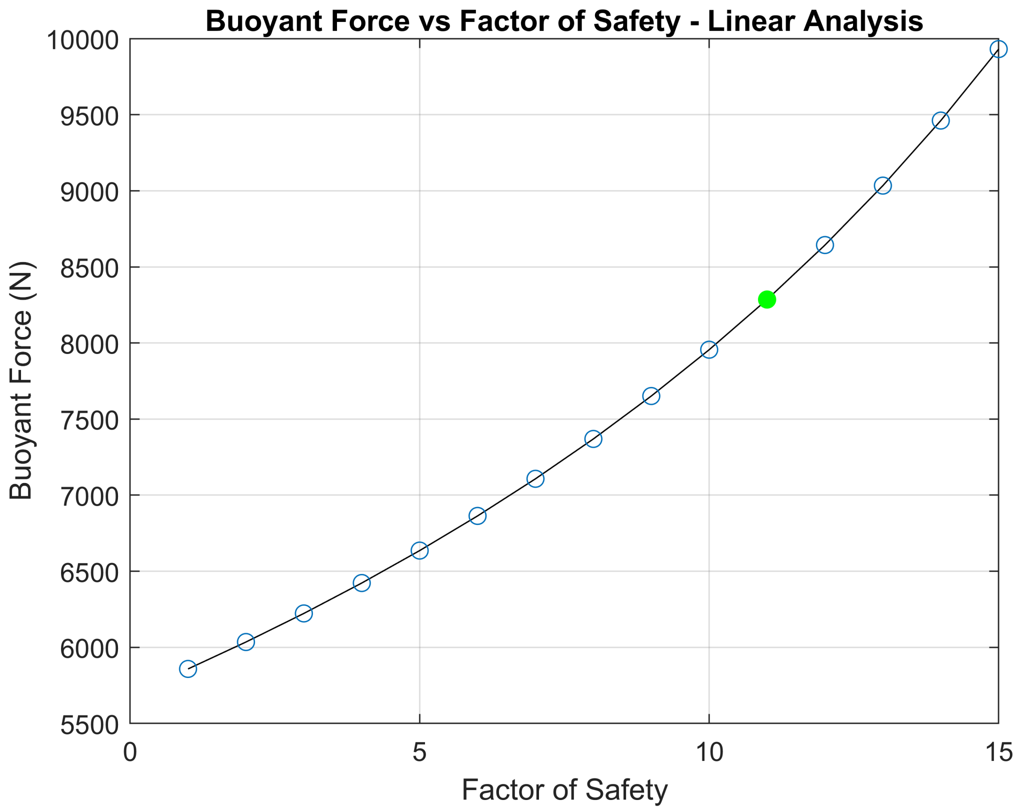

```
% Linear analysis
plot(data.FactorSafety(1:15), data.balloonVolume(1:15), 'k')
hold on
scatter(data.FactorSafety(1:15), data.balloonVolume(1:15), 40, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.balloonVolume(11), 40, 'filled', 'MarkerFaceColor', 'g', 'M
hold off
grid on
title("Balloon Volume vs Factor of Safety - Linear Analysis")
xlabel("Factor of Safety")
ylabel("Balloon Volume (m^3)")
```



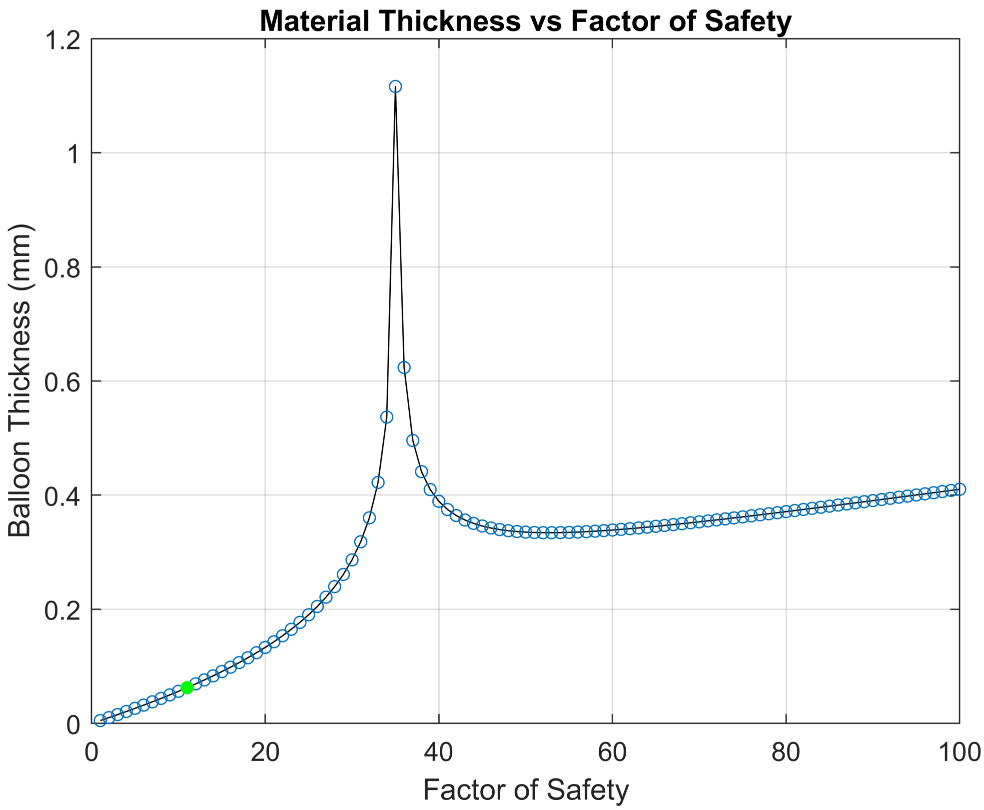

```
% ------------------------------------------------------------------------------------------- %

plot(data.FactorSafety, data.forceBuoyancy, 'k')
hold on
scatter(data.FactorSafety, data.forceBuoyancy, 20, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.forceBuoyancy(11), 20, 'filled', 'MarkerFaceColor', 'g', 'M
hold off
grid on
title("Buoyant Force vs Factor of Safety")
xlabel("Factor of Safety")
ylabel("Buoyant Force (N)")
```



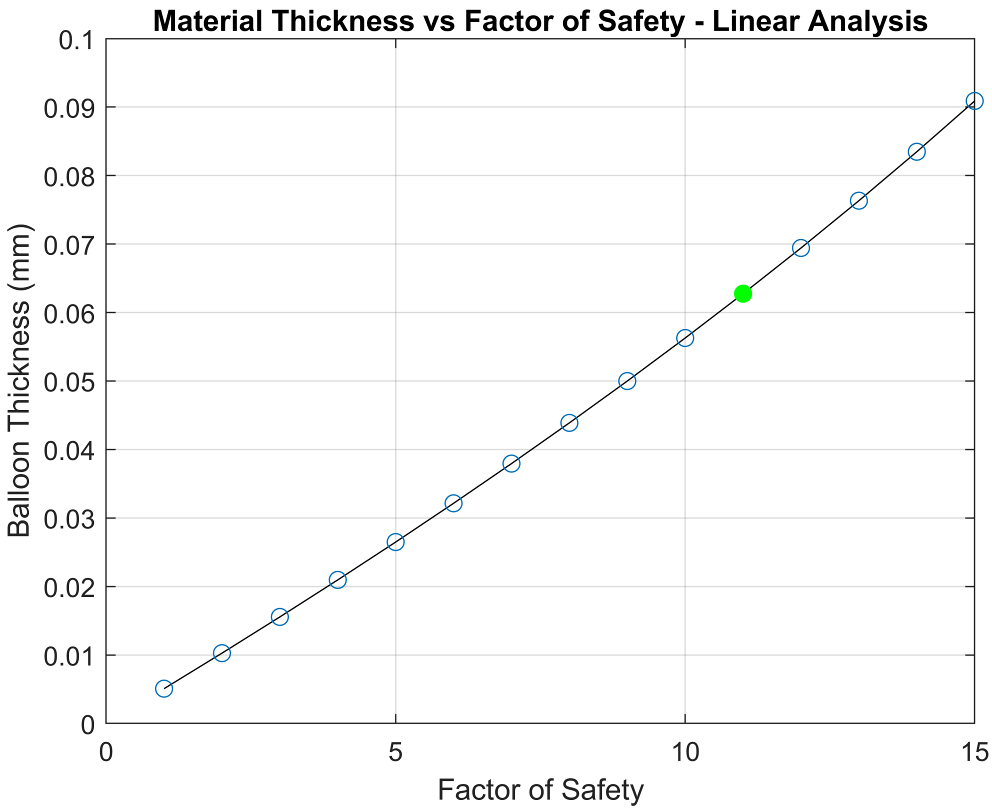

```
% Linear analysis
plot(data.FactorSafety(1:15), data.forceBuoyancy(1:15), 'k')
hold on
scatter(data.FactorSafety(1:15), data.forceBuoyancy(1:15), 40, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.forceBuoyancy(11), 40, 'filled', 'MarkerFaceColor', 'g', 'M
hold off
grid on
title("Buoyant Force vs Factor of Safety - Linear Analysis")
xlabel("Factor of Safety")
ylabel("Buoyant Force (N)")
```



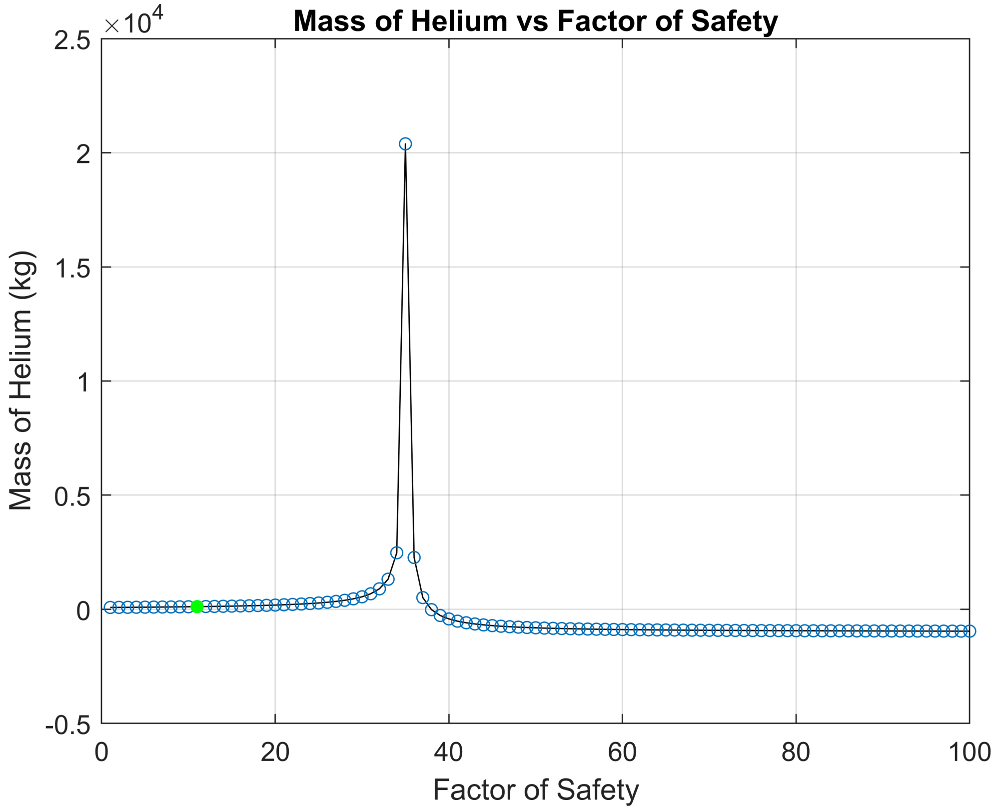

```
% --------------------------------------------------------------------------------------- %

plot(data.FactorSafety, data.thickness .* 1000, 'k')
hold on
scatter(data.FactorSafety, data.thickness .* 1000, 20, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.thickness(11) .* 1000, 20, 'filled', 'MarkerFaceColor', 'g'
hold off
grid on
title("Material Thickness vs Factor of Safety")
xlabel("Factor of Safety")
ylabel("Balloon Thickness (mm)")
```



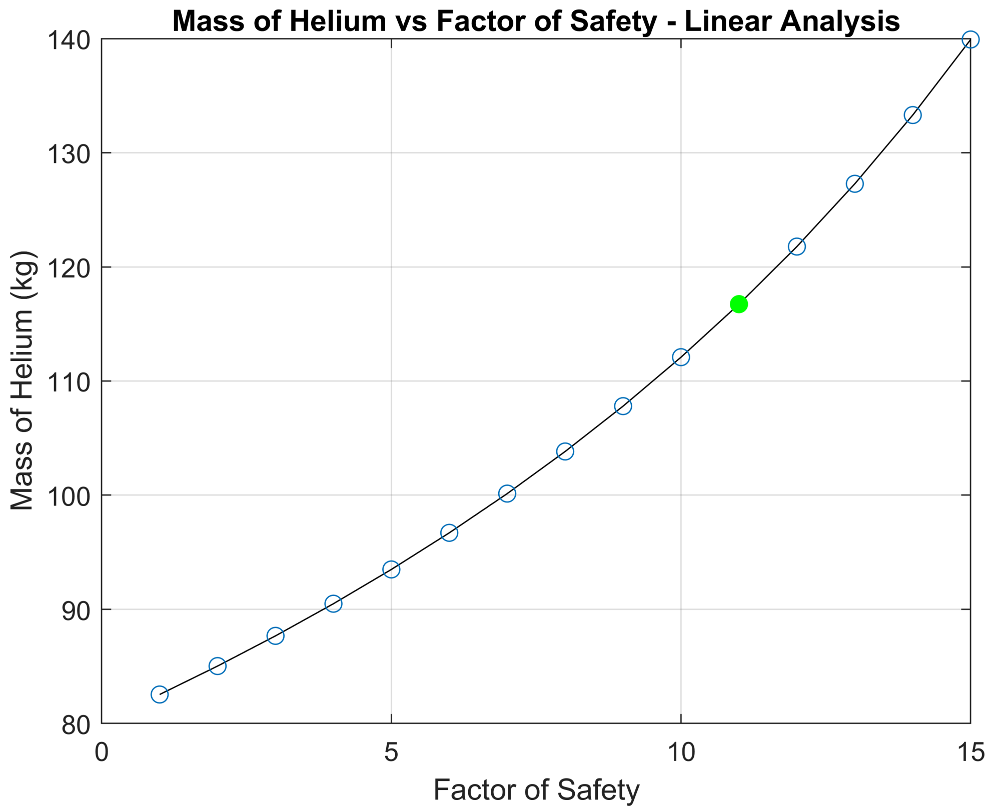

```
% Linear analysis
plot(data.FactorSafety(1:15), data.thickness(1:15) .* 1000, 'k')
hold on
scatter(data.FactorSafety(1:15), data.thickness(1:15) .* 1000, 40, 'MarkerEdgeColor', '#0072BD'
scatter(data.FactorSafety(11), data.thickness(11) .* 1000, 40, 'filled', 'MarkerFaceColor', 'g'
hold off
grid on
title("Material Thickness vs Factor of Safety - Linear Analysis")
xlabel("Factor of Safety")
ylabel("Balloon Thickness (mm)")
```



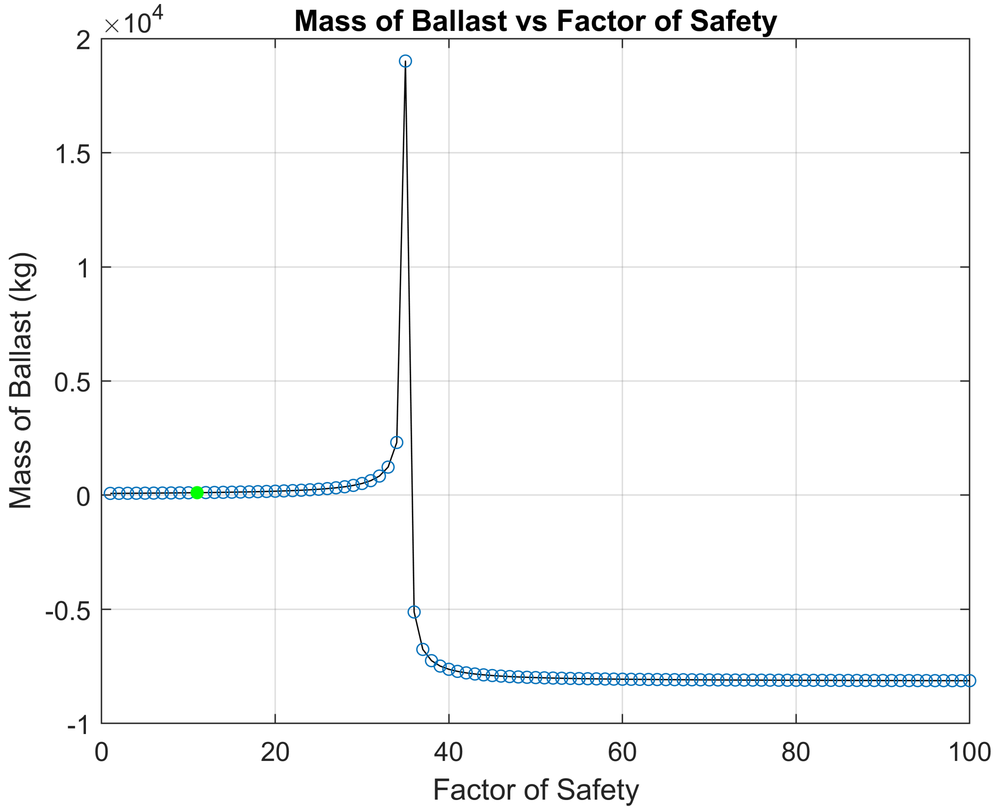

```
% ------------------------------------------------------------------------------------------ %

plot(data.FactorSafety, data.massHeFBE, 'k')
hold on
scatter(data.FactorSafety, data.massHeFBE, 20, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.massHeFBE(11), 20, 'filled', 'MarkerFaceColor', 'g', 'Marke
hold off
grid on
title("Mass of Helium vs Factor of Safety")
xlabel("Factor of Safety")
ylabel("Mass of Helium (kg)")
```



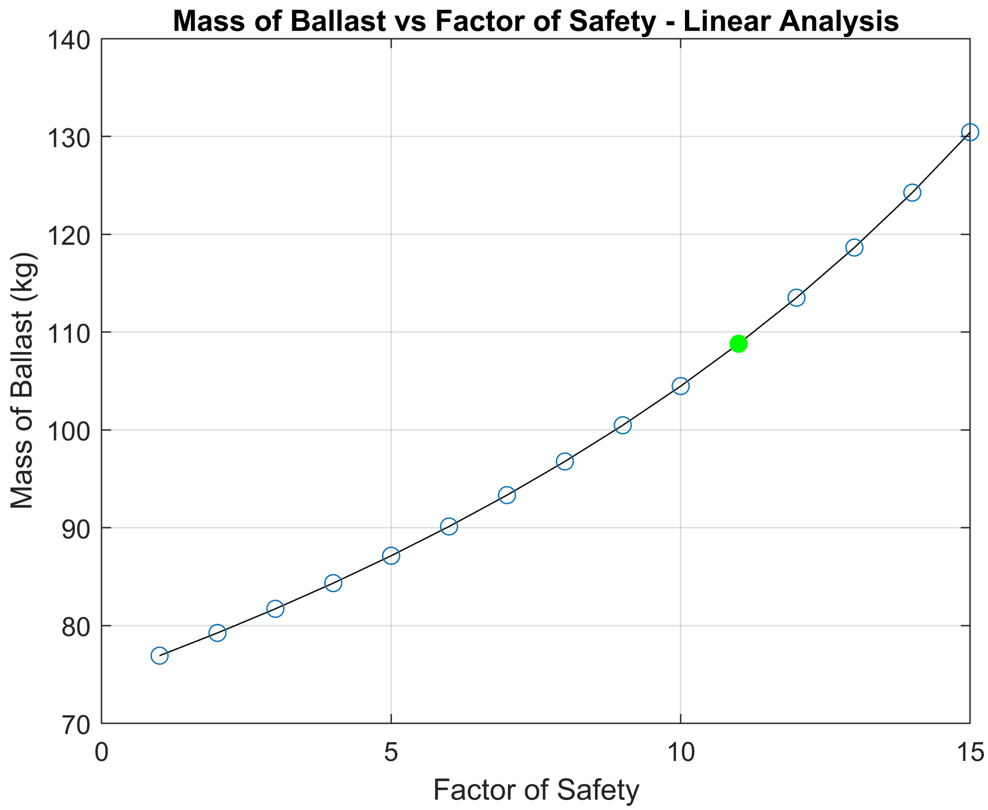

```
% Linear analysis
plot(data.FactorSafety(1:15), data.massHeFBE(1:15), 'k')
hold on
scatter(data.FactorSafety(1:15), data.massHeFBE(1:15), 40, "MarkerEdgeColor", '#0072BD')
scatter(data.FactorSafety(11), data.massHeFBE(11), 40, 'filled', 'MarkerFaceColor', 'g', 'Marke
hold off
grid on
title("Mass of Helium vs Factor of Safety - Linear Analysis")
xlabel("Factor of Safety")
ylabel("Mass of Helium (kg)")
```




```
% ------------------------------------------------------------------------------------------ %

plot(data.FactorSafety, data.massBallastDay, 'k')
hold on
scatter(data.FactorSafety, data.massBallastDay, 20, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.massBallastDay(11), 20, 'filled', 'MarkerFaceColor', 'g', '
hold off
grid on
title("Mass of Ballast vs Factor of Safety")
xlabel("Factor of Safety")
ylabel("Mass of Ballast (kg)")
```




```
% Linear analysis
plot(data.FactorSafety(1:15), data.massBallastDay(1:15), 'k')
hold on
scatter(data.FactorSafety(1:15), data.massBallastDay(1:15), 40, 'MarkerEdgeColor', '#0072BD')
scatter(data.FactorSafety(11), data.massBallastDay(11), 40, 'filled', 'MarkerFaceColor', 'g', '
hold off
grid on
title("Mass of Ballast vs Factor of Safety - Linear Analysis")
xlabel("Factor of Safety")
ylabel("Mass of Ballast (kg)")
```



% https://www.mathworks.com/matlabcentral/fileexchange/28135-standard-atmosphere-functions

```
function h = densityalt(rho,varargin)
% DENSITYALT  Returns altitude corresponding to the given array of air densities
%   in the standard or non-standard atmosphere.
%
%   H = DENSITYALT(RHO) returns altitude, h, as a function of air density, rho.
%
%   The input RHO can be followed by parameter/value pairs for further control
%   of DENSITYALT. Possible parameters are:
%     inputUnits     - String for units of input RHO, either kg/m³ or slug/ft³.
%                      [{SI}|kg/m3|kg/m^3  |  US|slug/ft3|slug/ft^3]
%     outputUnits    - String for units of output H, either meters or feet.
%                      [{SI}|m|meters  |  US|ft|feet]
%     atmosphereFunc - String determining atmosphere function to be used.
%                      [{atmos}|tropos|atmosisa|atmoscoesa|atmosnonstd]
%     atmosphereArgs - Cell array of additional arguments to pass to
%                      atmosphereFunction after the density input (e.g. for non-
%                      standard atmospheres).
%     method         - Method used for either searching for or interpolating a
%                      solution (the equations that define the standard
%                      atmosphere cannot be inverted in terms of density).
%                      Search: [fzero | bisection] (fzero only for scalar case)
%                      Interpolate: any method accepted by interp1.
%                      Default method is 'pchip' for interpolation.
%     options        - Options used for fzero or bisection methods.
%     hMin           - For search: lower search interval bound in meters.
```

```
%                       For interpolation: start of generated interpolation grid.
%                       Default hMin = 0.
%     hMax           - For search: upper search interval bound in meters.
%                       For interpolation: end of generated interpolation grid.
%                       Default hMax = 86000.
%     spacing        - Spacing of of interpolation grid in meters.
%                       Default spacing = 50.
%
%   If the input RHO is a DimVar, inputUnits and outputUnits will be ignored and
%   the output will be a DimVar.
%
%   DENSITYALT is valid for the entire standard atmosphere up through the
%   mesopause (86 km height). It assumes that all that is known is air
%   density. If pressure or temperature and density are known, there exist
%   more straightforward methods for calculating density altitude:
%       P = rho*R*T; h = h0 * (1 - P^0.190284), where P is in atmospheres and
%           h0 = 145366.45 ft or 44307.694 m.
%       (http://www.srh.noaa.gov/images/epz/wxcalc/pressureAltitude.pdf)
%
%   Example: Plot altitude as a function of air density for a cold (-15°C),
%   standard, and hot (+15°C) day (leverages vectorization of bisection).
%       rho = 1.225:-0.01:0.025; % kg/m^3
%       tempOffset = [-15 0 15]; % °C
%       [rho, tempOffset] = meshgrid(rho,tempOffset);
%       h = densityalt(rho,'method','bisection','outputUnits','ft',...
%               'atmosphereArgs',{tempOffset},'hMin',-5000);
%       plot(rho',h'/1000);
%       xlabel('Air density (kg/m^3)'); ylabel('Altitude (kft)'); grid on
%       legend('Cold','Standard','Hot')
%
%   See also ATMOSISA, ATMOSNONSTD, ATMOSCOESA, INTERP1, FZERO,
%     BISECTION - http://www.mathworks.com/matlabcentral/fileexchange/28150,
%     ATMOS     - http://www.mathworks.com/matlabcentral/fileexchange/28135,
%     U, UNITS  - http://www.mathworks.com/matlabcentral/fileexchange/38977.
%
%   H = DENSITYALT(RHO,Param1,Val1,Param2,Val2,...)
%   Copyright 2012-2013, 2015 Sky Sartorius
%   Author contact: mathworks.com/matlabcentral/fileexchange/authors/101715
% The 'pchip' method is smooth, preserves the shape, and has very tiny errors
% for most of the domain. However, the errors around the discontinuous
% transisions between atmospheres (e.g. between troposphere and tropopause) can
% be on the order of 1/100th of the spacing of given H. For spacing of 50 m,
% errors are never more than 813 mm. The absolute errors at these
% discontinuities are an order of magnitude lower for the 'linear' method but
% the error is present across the entire domain.
%% Parse inputs:
p = inputParser;
fName = 'densityalt';
p.FunctionName = fName;
p.addRequired('rho',@(x) validateattributes(x,{'numeric','DimVar'},...
    {'positive'},'','density, rho,'));
% Ignored if input is DimVar:
p.addParameter('inputUnits','SI');
p.addParameter('outputUnits','SI');
```

```
p.addParameter('atmosphereFunction','atmos');
p.addParameter('atmosphereArgs',{},@iscell);
p.addParameter('method','pchip');
% Possibilities are exact matches of 'fzero' or 'bisection', or an interp1
% method. Validation by interp1.
p.addParameter('hMin',0,@(x) validateattributes(x,{'numeric'},...
    {'scalar','finite','real'},'','altitude minimum h (m)'));
p.addParameter('hMax',86000,@(x) validateattributes(x,{'numeric'},...
    {'scalar','finite','real'},'','altitude maximum h (m)'));
% Used only with fzero/bisection search method:
p.addParameter('options',{})
% Used only with an interp method:
p.addParameter('spacing',50,@(x) validateattributes(x,{'numeric'},...
    {'positive','scalar','finite','real'},'','altitude grid spacing (m)'));
parse(p,rho,varargin{:});
i = p.Results;
rho = i.rho;
if ~iscell(i.options)
    i.options = {i.options};
end
% Validate inputs strings:
i.inputUnits = validatestring(i.inputUnits,...
    {'SI','kg/m3','kg/m^3','US','slug/ft3','slug/ft^3'},fName,'inputUnits');
i.outputUnits = validatestring(i.outputUnits,...
    {'SI','m','meters','US','ft','feet'},fName,'outputUnits');
i.atmosphereFuntion = validatestring(i.atmosphereFunction,...
    {'atmos','tropos','atmosisa','atmoscoesa','atmosnonstd'},...
    fName,'atmosphereFunction');
%% Process input density and return density in units of kg/m^3:
dimVarOut = false; % Flag to convert output to a DimVar.
if isa(rho,'DimVar')
    rho = rho/(u.kg/u.m^3);
    dimVarOut = true;
elseif any(strcmpi(i.inputUnits,{'US','slug/ft3','slug/ft^3'}))
    % Convert from imperial units.
    rho = rho * 515.3788183931961;
end
% Otherwise, input rho is already in kg/m^3.
%% Build atmosphere model:
% Build an atmosphere function that takes only altitude (in meters) as input and
% returns density (in kg/m^3) as the first and only output.
function rho = myAtmo(h)
    switch i.atmosphereFunction
        case {'atmos' 'tropos'}
            rho = feval(i.atmosphereFunction,h,i.atmosphereArgs{:});
        otherwise
            % Other functions return density as the fourth output.
            [~,~,~,rho] = feval(i.atmosphereFunction,h,i.atmosphereArgs{:});
    end
end
%% Find the altitude:
% Switch between search and interpolation methods:
switch i.method
    case 'fzero'
```

```
        h = fzero(@(h) rho-myAtmo(h),[i.hMin,i.hMax],i.options{:});

    case 'bisection'
        h = bisection(@myAtmo,i.hMin,i.hMax,rho,i.options{:});

    otherwise % interp1
        H = (i.hMin:i.spacing:i.hMax)';
        RHO = myAtmo(H);
        h = interp1(RHO,H,rho,i.method);
end
%% Process output altitude from meters into desired units:
if dimVarOut
    h = h*u.m;
elseif any(strcmpi(i.outputUnits,{'US','ft','feet'}))
    % Convert to imperial units if necessary.
    h = h / 0.3048;
end
% Otherwise, output h is already in meters.
end

function varargout = atmos(h,varargin)
%  ATMOS  Find gas properties in the 1976 Standard Atmosphere.
%   [rho,a,T,P,nu,z,sigma] = ATMOS(h,varargin)
%
%   ATMOS by itself gives atmospheric properties at sea level on a standard day.
%
%   ATMOS(h) returns the properties of the 1976 Standard Atmosphere at
%   geopotential altitude h, where h is a scalar, vector, matrix, or ND array.
%
%   The input h can be followed by parameter/value pairs for further control of
%   ATMOS. Possible parameters are:
%     tOffset      - Returns properties when the temperature is tOffset degrees
%                    above or below standand conditions. h and tOffset must be
%                    the same size or else one must be a scalar. Default is no
%                    offset. Note that this is an offset, so when converting
%                    between Celsius and Fahrenheit, use only the scaling factor
%                    (dC/dF = dK/dR = 5/9).
%     tAbsolute    - Similar to tOffest, but an absolute air temperature is
%                    provided (°K or °R) instead of an offset from the standard
%                    temperature. Supersedes tOffset if both are provided.
%     altType      - Specify type of input altitude, either 'geopotential' (h)
%                    or 'geometric' (z). Default altType = 'geopotential'.
%     structOutput - When set, ATMOS produces a single struct output with fields
%                    rho, a, T, P, nu, and either z or h (whichever complements
%                    input altType). Default structOutput = false.
%     units        - String for units of inputs and outpus, either 'SI'
%                    (default) or 'US'. This is ignored if the provided input h
%                    is a DimVar, in which case all outputs are also DimVars and
%                    expected tOffset is either a DimVar or in °C/°K.
%                                 Description:         SI:           US:
%                     Input:      ---------------      -----         -----
%                       h | z     Altitude or height   m             ft
%                       tOffset   Temp. offset         °C/°K         °F/°R
%                     Output:     ---------------      -----         -----
```

```
%                       rho       Density              kg/m^3        slug/ft^3
%                       a         Speed of sound       m/s           ft/s
%                       T         Temperature          °K            °R
%                       P         Pressure             Pa            lbf/ft^2
%                       nu        Kinem. viscosity     m^2/s         ft^2/s
%                       z | h     Height or altitude   m             ft
%                       sigma     Density ratio        -             -
%
%   ATMOS returns properties the same size as h and/or tOffset (P does not vary
%   with temperature offset and is always the size of h).
%
%   Example 1: Find atmospheric properties at every 100 m of geometric height
%   for an off-standard atmosphere with temperature offset varying +/- 25°C
%   sinusoidally with a period of 4 km.
%       z = 0:100:86000;
%       [rho,a,T,P,nu,h,sigma] = atmos(z,'tOffset',25*sin(pi*z/2000),...
%                                  'altType','geometric');
%       semilogx(sigma,h/1000)
%       title('Density variation with sinusoidal off-standard atmosphere')
%       xlabel('Density ratio, \sigma'); ylabel('Geopotential altitude (km)')
%
%   Example 2: Create tables of atmospheric properties up to 30,000 ft for a
%   cold (-20°C), standard, and hot (+20°C) day with columns
%   [h(ft) z(ft) rho(slug/ft³) sigma a(ft/s) T(R) P(psf) µ(slug/ft-s) nu(ft²/s)]
%   leveraging n-dimensional array capability.
%       [~,h,dT] = meshgrid(0,-5000:1000:30000,[-20 0 20]);
%       [rho,a,T,P,nu,z,sigma] = atmos(h,'tOffset',dT*9/5,'units','US');
%       t = [h z rho sigma a T P nu.*rho nu];
%       format short e
%       varNames = {'h' 'z' 'rho' 'sigma' 'a' 'T' 'P' 'mu' 'nu'};
%       ColdTable       = array2table(t(:,:,1),'VariableNames',varNames)
%       StandardTable   = array2table(t(:,:,2),'VariableNames',varNames)
%       HotTable        = array2table(t(:,:,3),'VariableNames',varNames)
%
%   Example 3: Use the unit consistency enforced by the DimVar class to find the
%   SI dynamic pressure, Mach number, Reynolds number, and stagnation
%   temperature of an aircraft flying at flight level FL500 (50000 ft) with
%   speed 500 knots and characteristic length of 80 inches.
%       V = 500*u.kts; c = 80*u.in;
%       o = atmos(50*u.kft,'structOutput',true);
%       Dyn_Press = 1/2*o.rho*V^2;
%       M = V/o.a;
%       Re = V*c/o.nu;
%       T0 = o.T*(1+(1.4-1)/2*M^2);
%
%   This model is not recommended for use at altitudes above 86 km geometric
%   height (84852 m / 278386 ft geopotential) but will attempt to extrapolate
%   above 86 km (with a lapse rate of 0°/km) and below 0.
%
%   See also ATMOSISA, ATMOSNONSTD, TROPOS,
%     DENSITYALT - http://www.mathworks.com/matlabcentral/fileexchange/39325,
%     UNITS      - http://www.mathworks.com/matlabcentral/fileexchange/38977.
%
%   [rho,a,T,P,nu,z,sigma] = ATMOS(h,varargin)
```

```
%   Copyright 2015 Sky Sartorius
%   www.mathworks.com/matlabcentral/fileexchange/authors/101715
%
%   References: ESDU 77022; www.pdas.com/atmos.html
%% User-customizable defaults:
defaultUnits = 'SI'; % Alternate: 'US'
defaultStructOutput = false;
%% Parse inputs:
if nargin == 0
    h = 0;
end
if nargin <= 1 && ~nnz(h)
    % Quick return of sea level conditions.
    rho = 1.225;
    a = 340.2940;
    temp = 288.15;
    press = 101325;
    kvisc = 1.4607186e-05;
    ZorH = 0;
    if isa(h,'DimVar')
        rho = rho*u.kg/(u.m^3);
        if nargout == 1
            varargout = {rho};
            return
        end
        a = a*u.m/u.s;
        temp = temp*u.K;
        press = press*u.Pa;
        kvisc = kvisc*u.m^2/u.s;
        ZorH = ZorH*u.m;
    end
    varargout = {rho,a,temp,press,kvisc,ZorH,1};
    return
end

% validateattributes(h,{'DimVar' 'numeric'},{'finite' 'real'});
p = inputParser;
addParameter(p,'tOffset',0);
addParameter(p,'tAbsolute',[]);
addParameter(p,'units',defaultUnits);
addParameter(p,'altType','geopotential');
addParameter(p,'structOutput',defaultStructOutput);
parse(p,varargin{:});
tOffset = p.Results.tOffset;
tAbsolute = p.Results.tAbsolute;
if strcmpi(p.Results.units,'SI')
    convertUnits = false;
elseif strcmpi(p.Results.units,'US')
    convertUnits = true;
    % Flag if I need to convert to/from SI.
else
    error('Invalid units. Expected: ''SI'' or ''US''.')
end
if strcmpi(p.Results.altType,'geopotential')
```

```
    geomFlag = false;
elseif strcmpi(p.Results.altType,'geometric')
    geomFlag = true;
    % Flag specifying z provided as input.
else
    error('Invalid altType. Expected: ''geopotential'' or ''geometric''.')
end
structOutput = p.Results.structOutput;
%% Deal with different input types:
dimVarOut = false;
if isa(h,'DimVar')
    h = h/u.m;
    dimVarOut = true;
    convertUnits = false; % Trumps specified units.
end
if isa(tOffset,'DimVar')
    tOffset = tOffset/u.K;
    % It is allowed to mix DimVar h_in and double tOffset (or reverse).
end
if isa(tAbsolute,'DimVar')
    tAbsolute = tAbsolute/u.K;
end
if convertUnits
    h = h * 0.3048;
    tOffset   = tOffset   * 5/9;
    tAbsolute = tAbsolute * 5/9;
end
%% Constants, etc.:
%  Lapse rate Base Temp       Base Geop. Alt    Base Pressure
%   Ki (°C/m) Ti (°K)         Hi (m)            P (Pa)
D =[-0.0065   288.15          0                 101325            % Troposphere
    0         216.65          11000             22632.04059693474 % Tropopause
    0.001     216.65          20000             5474.877660660026 % Stratosph. 1
    0.0028    228.65          32000             868.0158377493657 % Stratosph. 2
    0         270.65          47000             110.9057845539146 % Stratopause
    -0.0028   270.65          51000             66.938535373039073% Mesosphere 1
    -0.002    214.65          71000             3.956392754582863 % Mesosphere 2
    0         186.94590831019 84852.04584490575 .373377242877530];% Mesopause
% Constants:
rho0 = 1.225;   % Sea level density, kg/m^3
gamma = 1.4;
g0 = 9.80665;   %m/sec^2
RE = 6356766;   %Radius of the Earth, m
Bs = 1.458e-6;  %N-s/m2 K1/2
S = 110.4;      %K
K = D(:,1); %°K/m
T = D(:,2); %°K
H = D(:,3); %m
P = D(:,4); %Pa
R = P(1)/T(1)/rho0; %N-m/kg-K
% Ref:
%   287.05287 N-m/kg-K: value from ESDU 77022
%   287.0531 N-m/kg-K:  value used by MATLAB aerospace toolbox ATMOSISA
%% Convert from geometric altitude to geopotental altitude, if necessary.
```

```
if geomFlag
    hGeop = (RE*h) ./ (RE + h);
else
    hGeop = h;
end
%% Calculate temperature and pressure:
% Pre-allocate.
temp = zeros(size(h));
press = temp;
nSpheres = size(D,1);
for i = 1:nSpheres
    % Put inputs into the right altitude bins:
    if i == 1 % Extrapolate below first defined atmosphere.
        n = hGeop <= H(2);
    elseif i == nSpheres % Capture all above top of defined atmosphere.
        n = hGeop > H(nSpheres);
    else
        n = hGeop <= H(i+1) & hGeop > H(i);
    end


    if K(i) == 0 % No temperature lapse.
        temp(n) = T(i);
        press(n) = P(i) * exp(-g0*(hGeop(n)-H(i))/(T(i)*R));
    else
        TonTi = 1 + K(i)*(hGeop(n) - H(i))/T(i);
        temp(n) = TonTi*T(i);
        press(n) = P(i) * TonTi.^(-g0/(K(i)*R)); % Undefined for K = 0.
    end
end
%% Switch between using standard temp and provided absolute temp.
if isempty(tAbsolute)
    % No absolute temperature provided - use tOffset.
    temp = temp + tOffset;
else
    temp = tAbsolute;
end
%% Populate the rest of the parameters:
rho = press./temp/R;
sigma = rho/rho0;
a = sqrt(gamma * R * temp);
kvisc = (Bs * temp.^1.5 ./ (temp + S)) ./ rho; %m2/s
if geomFlag % Geometric in, ZorH is geopotential altitude (H)
    ZorH = hGeop;
else % Geop in, find Z
    ZorH = RE*hGeop./(RE-hGeop);
end
%% Process outputs:
if dimVarOut
    rho = rho*u.kg/(u.m^3);
    a = a*u.m/u.s;
    temp = temp*u.K;
    press = press*u.Pa;
    kvisc = kvisc*u.m^2/u.s;
```

```
    ZorH = ZorH*u.m;
elseif convertUnits
    rho = rho / 515.3788;
    a = a / 0.3048;
    temp = temp * 1.8;
    press = press / 47.88026;
    kvisc = kvisc / 0.09290304;
    ZorH = ZorH / 0.3048;
end
varargout = {rho,a,temp,press,kvisc,ZorH,sigma};
if structOutput
    if geomFlag
        ZorHname = 'h';
    else
        ZorHname = 'z';
    end
    names = {'rho' 'a' 'T' 'P' 'nu' ZorHname 'sigma'};
    varargout = {cell2struct(varargout,names,2)};
end
end
```