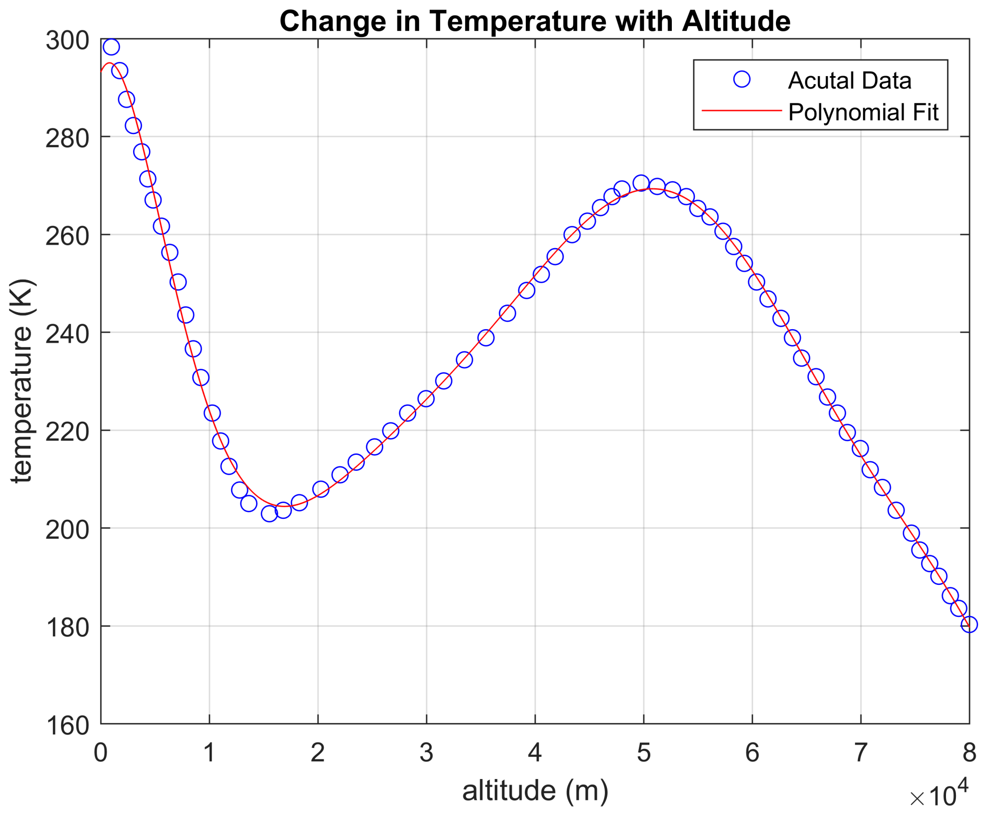
Homework 6 - Geoffrey Meier

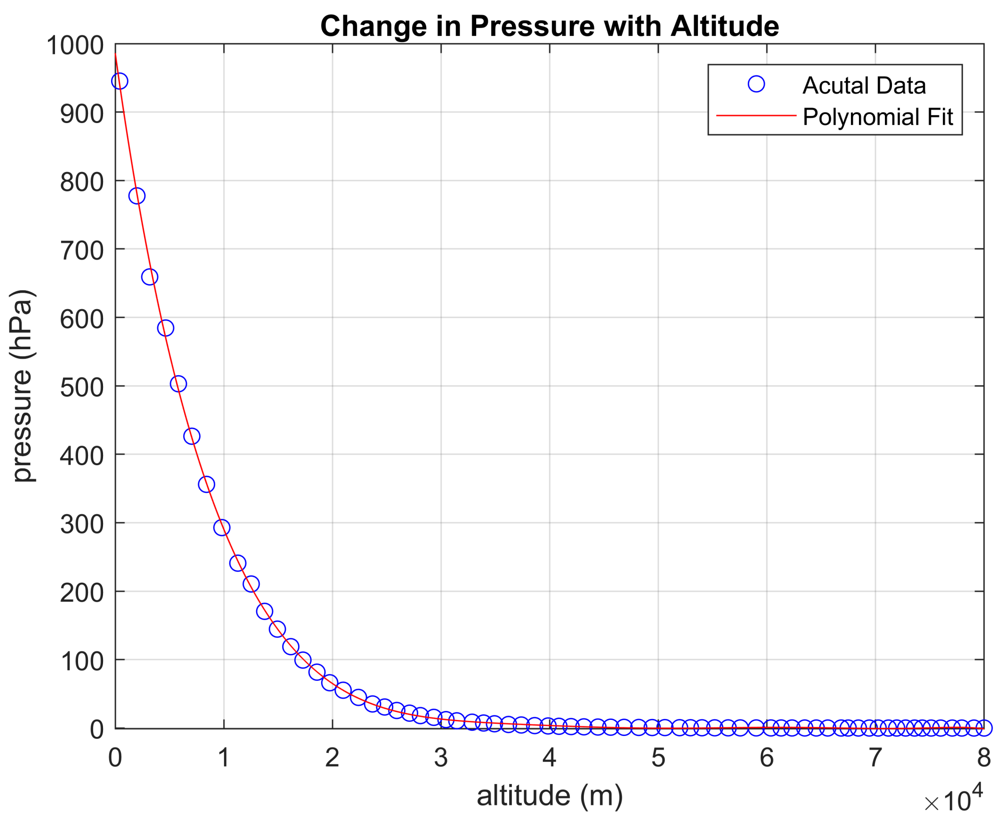
Problem 1 - Felix Baumgartner the Daredevil

Finding Rho

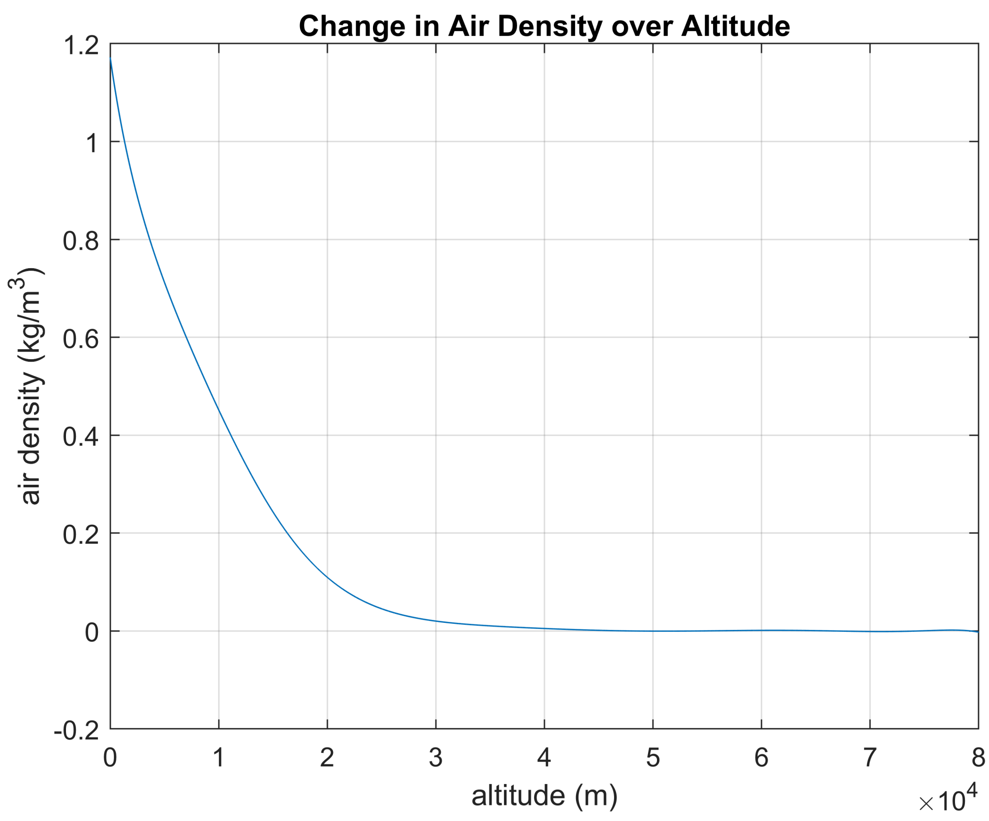
```
clear all
clf

% First, define the variables for the problem
y0 = 38969; %m - initial position
yp0 = 0; %m/s - initial velocity
y_chute = 2000; %m - altitude where parachute opens
h = 0.1; %timestep
z0 = [yp0; y0]; %cast initial values as vector

% Find the function rho from the given data
% pressure and temperature variable manually loaded into file

T = importfile('temperature.csv',2,70);
P = importfile('pressure.csv',2,72);
scale = 100; %value by which to scale values

T_x = T(:,2);
T_y = T(:,1);
P_x = P(:,2);
P_y = P(:,1);
deg = 9; % degree of polynomial

[cT,~,muT] = polyfit(T_x,T_y,deg);
[cP,~,muP] = polyfit(P_x,P_y,deg);
T_fun = @(x) polyval(cT,x,[],muT);
P_fun = @(x) polyval(cP,x,[],muP);


x = 0:80000; %values to test functions on

% plot temperature against altitude
plot(T_x,T_y,'bo',x,T_fun(x),'r-')
xlabel('altitude (m)')
ylabel('temperature (K)')
title('Change in Temperature with Altitude')
grid on
legend('Acutal Data','Polynomial Fit')
```



```
% plot pressure against altitude
plot(P_x,P_y,'bo',x,P_fun(x),'r-')
xlabel('altitude (m)')
ylabel('pressure (hPa)')
title('Change in Pressure with Altitude')
grid on
legend('Acutal Data','Polynomial Fit')
```



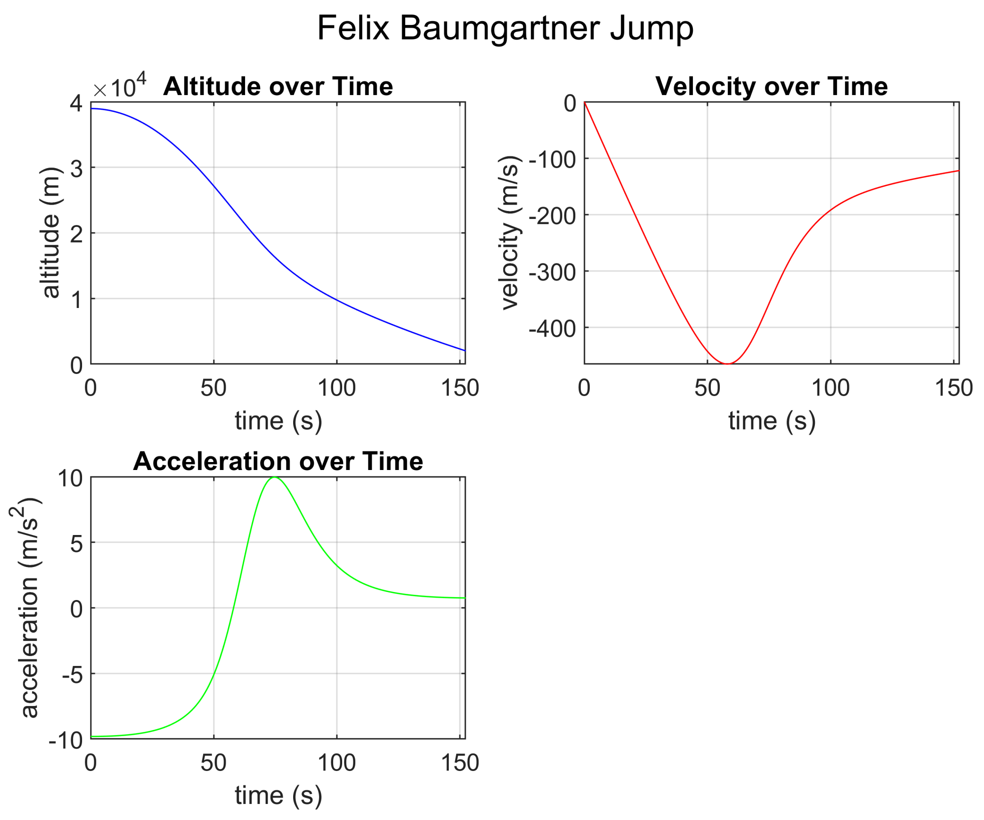
```
% Calculate air density using the functions for pressure and temperature
R_air = 287; % J/(kg*K) - gas constant for dry air
rho = @(y) 100*P_fun(y)./(R_air.*T_fun(y)); % gas law, and accounting for hPa instead of Pa

% plot air density against altitude
plot(x,rho(x))
xlabel('altitude (m)')
ylabel('air density (kg/m^3)')
title('Change in Air Density over Altitude')
grid on
```



Part A

```
cDA = 0.18; %m^2
fun = @(t,z) daredevil(z,cDA,rho);

z(:,1) = z0; %assign intial vector
t(1) = 0; %initialize time
zp(:,1) = fun(t(1),z0); %assign initial z' vector
i = 1; %initialize counter
y(1) = y0; %initialize position

%keeping going until parachute opens
while y(i)>y_chute
    z(:,i+1) = rk4_step(fun,z(:,i),t(i),h); %make one step of RK4 method
    t(i+1) = t(i) + h; %increment time
    zp(:,i+1) = fun(t(i+1),z(:,i+1)); %increment z prime
    y(i+1) = z(2,i+1); %extract position
    i = i+1;
end

% plot altitude
subplot(2,2,1)
plot(t,y,'b')
xlabel('time (s)')
ylabel('altitude (m)')
title('Altitude over Time')
```

```
grid on
% plot velocity
subplot(2,2,2)
plot(t,z(1,:),'r')
xlabel('time (s)')
ylabel('velocity (m/s)')
title('Velocity over Time')
grid on
% plot acceleration
subplot(2,2,3)
plot(t,zp(1,:),'g')
xlabel('time (s)')
ylabel('acceleration (m/s^2)')
title('Acceleration over Time')
grid on
sgtitle('Felix Baumgartner Jump')
```



Analysis

As his altitude decreases, my model predicts that his velocity will rapidly increase (the negative values for velocity just indicate a downward direction), then will rapidly increase before beginning to taper off. His acceleration will start out negative, change to positive, and then start stabilizing close to zero near the end of his fall. This makes sense with regards to the change in air density, which will start very low and then become much higher at about 20,000 meters.

```
max_vel = abs(min(z(1,:))) %m/s
```

```
max_vel = 465.1648
```

```
max_vel_exact = 1357.64*5/18 %m/s (originally given in km/h)
```

```
max_vel_exact = 377.1222
```

The velocity that my model predicts is 465.16 m/s. This is significantly greater than Felix Baumgartner's actual max velocity, which was 377.12 m/s*.

```
time = t(end) %s
```

```
time = 152.4000
```

```
time_exact = 4*60+19 %s (originally given in m:s)
```

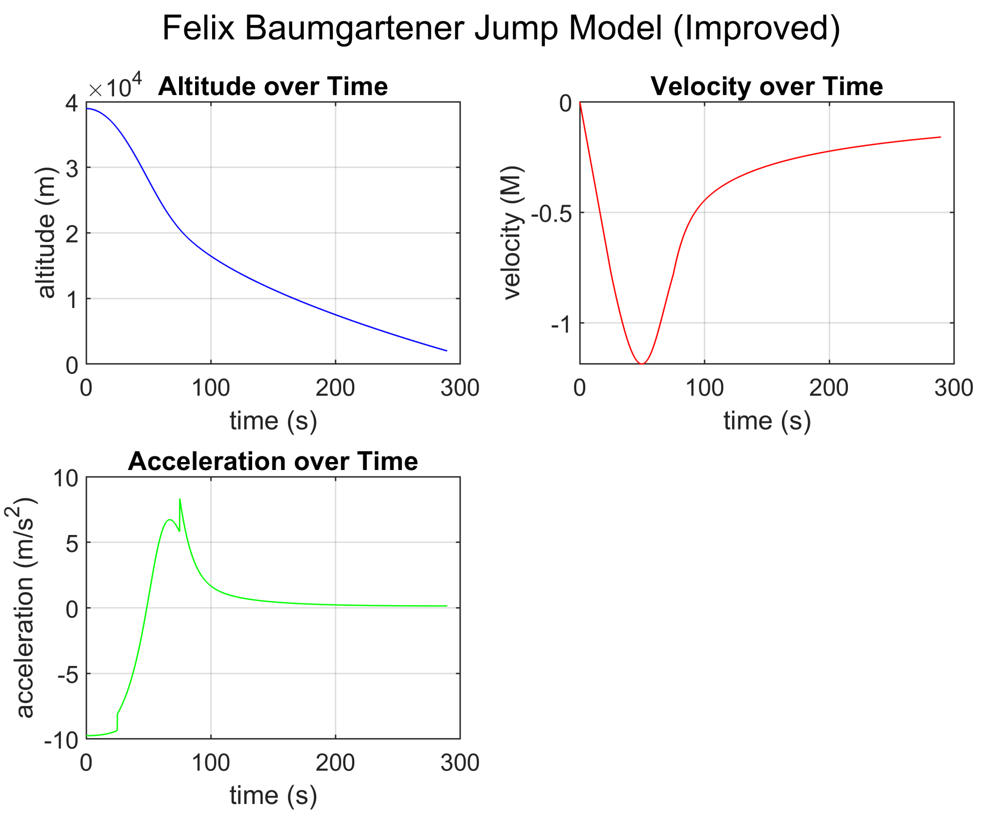
```
time_exact = 259
```

The time that my model predicts is about 152 seconds. This is significantly less than Felix's actual free fall time, which was 259 seconds*.

Part B - Extra Credit

```
% clear unwanted values from last part
clear z t zp y
clf

fun1 = @(t,z) daredevil1(t,z,rho);

z(:,1) = z0; %assign intial vector
t(1) = 0; %initialize time
zp = fun1(t(1),z0); %assign initial z' vector
i = 1; %initialize counter
y(1) = y0; %initialize position

%keeping going until parachute opens
while y(i)>y_chute
    z(:,i+1) = rk4_step(fun1,z(:,i),t(i),h); %make one step of RK4 method
    t(i+1) = t(i) + h; %increment time
    zp(:,i+1) = fun1(t(i+1),z(:,i+1)); %increment z prime
    y(i+1) = z(2,i+1); %extract position
    i = i+1;
end

kappa = 1.4; %adiabatic index
to_mach = @(v,y) v./sqrt(kappa*R_air*T_fun(y));
v_mach = to_mach(z(1,:),y);

% plot altitude
subplot(2,2,1)
plot(t,y,'b')
xlabel('time (s)')
ylabel('altitude (m)')
title('Altitude over Time')
```

```
grid on
% plot velocity
subplot(2,2,2)
plot(t,v_mach,'r')
xlabel('time (s)')
ylabel('velocity (M)')
title('Velocity over Time')
grid on
% plot acceleration
subplot(2,2,3)
plot(t,zp(1,:),'g')
xlabel('time (s)')
ylabel('acceleration (m/s^2)')
title('Acceleration over Time')
grid on
sgtitle('Felix Baumgartener Jump Model (Improved)')
```



Analysis

```
max_vel = abs(min(v_mach)) %M
```

```
max_vel = 1.1854
```

```
max_vel_exact = to_mach(1357.64*5/18,y(find(min(v_mach)))) %M (originally given in km/h)
```

```
max_vel_exact = 1.1925
```

My model predicts a max velocity of Mach 1.185, which is fairly close to Felix's actual max velocity of Mach 1.1925.

```
time = t(end) %s
```

```
time = 289.4000
```

```
time_exact = 4*60+19 %s (originally given in m:s)
```

```
time_exact = 259
```

My model predicts that Felix will be in free fall for about 289 seconds, which is fairly close to his actual free fall time of 259 seconds.

```
t_ind = @(sec) find(abs(t-sec)<10^-3);
mach_25 = abs(v_mach(t_ind(25)))
```

```
mach_25 = 0.7741
```

```
mach_75 = abs(v_mach(t_ind(75)))
```

```
mach_75 = 0.7782
```

In his actual jump, Felix hit Mach 0.8 at 25 seconds and 75 seconds. My model is very close to this, predicting that he would be at Mach 0.77 at 25 seconds and 0.78 at 75 seconds.

Functions

```
function y2=rk4_step(fun,y1,t1,h)
%%RK4_STEP makes one time-step of the Runge-Kutta 4 method

k1 = fun(t1,     y1        ); %slope 1 at the origin point
k2 = fun(t1+h/2, y1+h/2*k1 ); %slope 2 at the predicted midpoint
k3 = fun(t1+h/2, y1+h/2*k2 ); %slope 3 at updated prediction of midpoint
k4 = fun(t1+h,   y1+h*k3   ); %slope 4 at full step predicted using slope k3

y2 = y1 + h*1/6*(k1 + 2*k2 + 2*k3 + k4); %take full step using a linear combination of slopes
end

function FD = drag(z,cDA,rho)
%% DRAG determines the amount of drag depending on whether or not the parachute is open

FD = 0.5*cDA*rho(z(2))*z(1)^2; %drag

end

function zp = daredevil(z,cDA,rho)
%% DAREDEVIL returns the system of equations that will solve the ODE

g = 9.81; %m/s^2
```

```
m = 118; %kg - Felix mass

z1p = -g +drag(z,cDA,rho)/m; %acceleration
z2p = z(1); %velocity

zp = [z1p; z2p];
end

function zp = daredevil1(t,z,rho)
%% DAREDEVIL1 improved functions that returns the system of equations that will solve the ODE

m = 118; %kg - Felix mass

% calculate gravitational force
G = 6.67e-11; %Nm^2/kg^2 - Gravitational constant
ME = 5.98e24; %kg - Mass of Earth
R = 6356.8e3; %m - Radius of Earth
g = G*ME/(R+z(2))^2;

% determine drag coefficient
if t<=25
    cDA = 0.18;
elseif t>25 && t<=75
    cDA = 0.73;
else
    cDA = 0.85;
end

z1p = -g +drag(z,cDA,rho)/m; %acceleration
z2p = z(1); %velocity
zp = [z1p; z2p];
end
```