# Lab - 2 : Mine Crafting

**Mario Dajani Caceres**

*PHYS 265*

## Introduction

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import scipy
```

In [2]:
```
#Constants
G=6.6743e-11 #m^3/kg/s^2
g0=9.81 #m/s^2 approx
Me=5.972e24 #kg
Mm=7.35e22 #kg
Re=6378.1e3 #m
Rm=1738.1e3 #m
Rot=7.272e-5 #rad/s
```

## Part 1: The Ideal Case

### Problem 1

In [3]:
```
#Constants
alpha=0
s=4e3 #m


#1
#s = v0 t + (1/2) a t^2
#s=(1/2) a t^2

tff=np.sqrt(2*s/g0)
print(f'the time it takes for it to reach the bottom is {tff} seconds')
```

```
the time it takes for it to reach the bottom is 28.55686245854129 seconds
```

### Problem 2

$$\frac{d^2y}{dt^2} = -g - \alpha\left(\frac{dy}{dt}\right)^\gamma$$

$$\frac{dy}{dt} = v$$

$$\frac{dv}{dt} = -g - \alpha(v)^\gamma$$

# Problem 3

In [4]:
```
#3
#Defining the functions
g=g0
a=0
vdep=2
def E1(t,m):
    y,v=m
    dydt=v
    dvdt=-g+a*v**vdep
    return [dydt,dvdt]

#thing 2: time
t0,tf= 0, 35
t_eval = np.linspace(t0,tf, 500)


#thing 3:IC
y0 = 0 #m
v0=0 #m/s
ic=[y0,v0]


#solve_ivp
sol=solve_ivp(fun=E1,t_span=(t0,tf),y0=ic,t_eval=t_eval)

# extract data
y=sol.y[0]
v=sol.y[1]
time=sol.t

#plot
fig, ax1 = plt.subplots()
ax1.plot(time,y,label='position')
ax2=ax1.twinx()
ax2.plot(time,v,label='velocity', color='red')
ax1.axhline(y=-4000,color='black',ls='--')

ax1.set_title('Fig 1: Ideal Motion graph')
ax1.set_ylabel('Y Position (m) from surface of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1.set_xlabel('Time (s)')
ax1.legend(loc='lower left')
ax2.legend(loc='upper right');
```

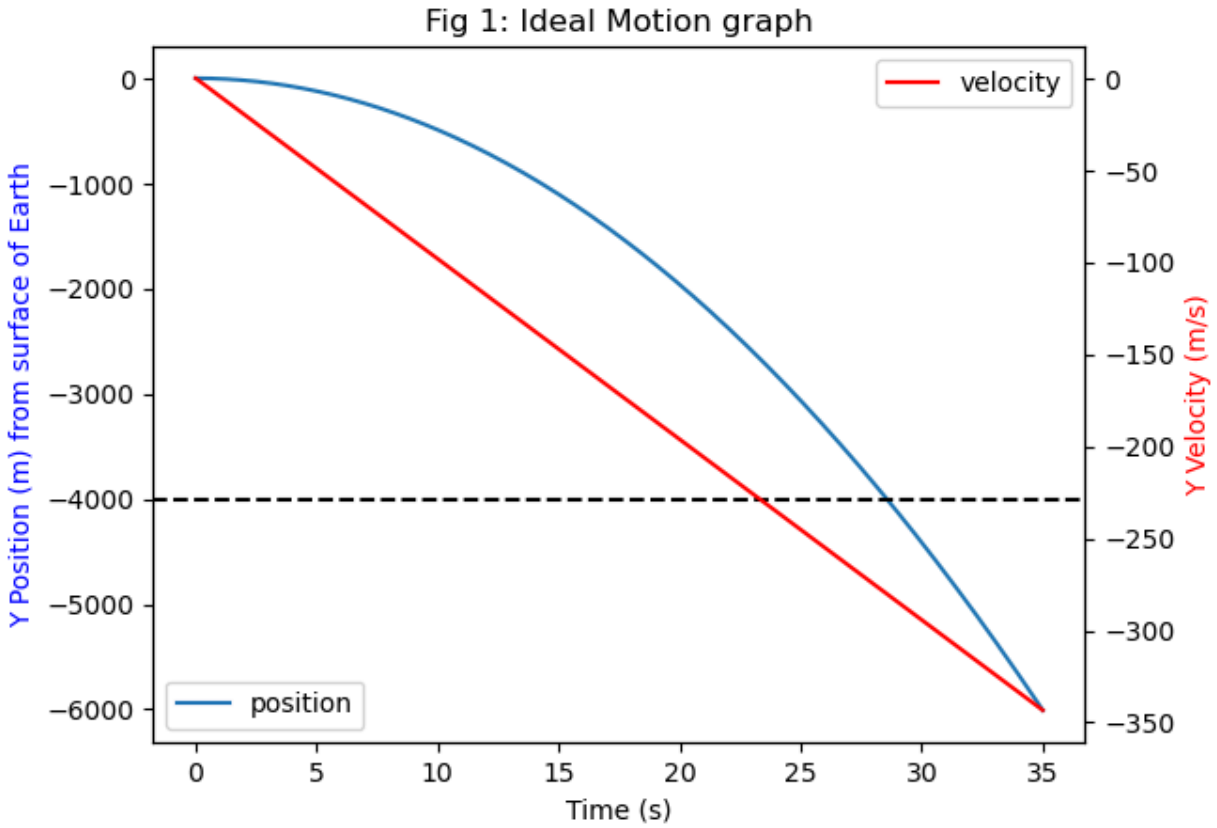


Fig 1: In this graph we can see the plots of the velocity and and position. The blue graph and the left axis represents the position, and we can see how the mass is falling downwards at an increasing speed. The red plot and the right axis represents the velocity, where we can see that the velocity is negative and decreasing linearly. The dashed black line represents the bottom of the mineshaft which is at -4000 m.

## Problem 4

In [5]:
```
#thing 4

y_stop=-4000
def stopping(t,m):
    y,v=m
    return y-y_stop

stopping.terminal=True
sol2 = solve_ivp(fun=E1,
                         t_span=(t0,tf), #tuple
                         y0=ic, #as array,
                                t_eval=t_eval,events=stopping
                               )
t_event=sol2.t_events[0][0]
print(f'Using the events detection of solve_ivp, we find that the time it hits the bot
```

```
Using the events detection of solve_ivp, we find that the time it hits the bottom aft
er using a variable g is 28.55686245854127 seconds
```

### Part 2: Including drag and variable g

# Problem 1

In [6]:
```
#1

#Defining the functions

a=0
vdep=2

def g(y):
    gr=g0*y/Re
    return gr
def E2(t,m):
    y,v=m
    dydt=v
    dvdt=-g(y)+a*v**vdep
    return [dydt,dvdt]

#thing 2: time
t0,tf= 0, 35
t_eval = np.linspace(t0,tf, 500)


#thing 3:IC
y0 = Re #m
v0=0 #m/s
ic=[y0,v0]


#solve_ivp
stopping.terminal=False
sol=solve_ivp(fun=E2,t_span=(t0,tf),y0=ic,t_eval=t_eval)

# extract data
y2=sol.y[0]
v2=sol.y[1]
time2=sol.t

#plot
fig, ax1 = plt.subplots()
ax1.plot(time2,y2,label='position')
#ax1.set_ylim(Re-5000,Re+1000)
ax2=ax1.twinx()
ax2.plot(time2,v2,label='velocity', color='red')
ax1.axhline(y=Re-4000,color='black',ls='--')

ax1.set_title('Fig 2: Motion graph with variable g')
ax1.set_ylabel('Y Position (m) from center of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1.set_xlabel('Time (s)')
ax1.legend(loc='lower left')
ax2.legend(loc='upper right');
```

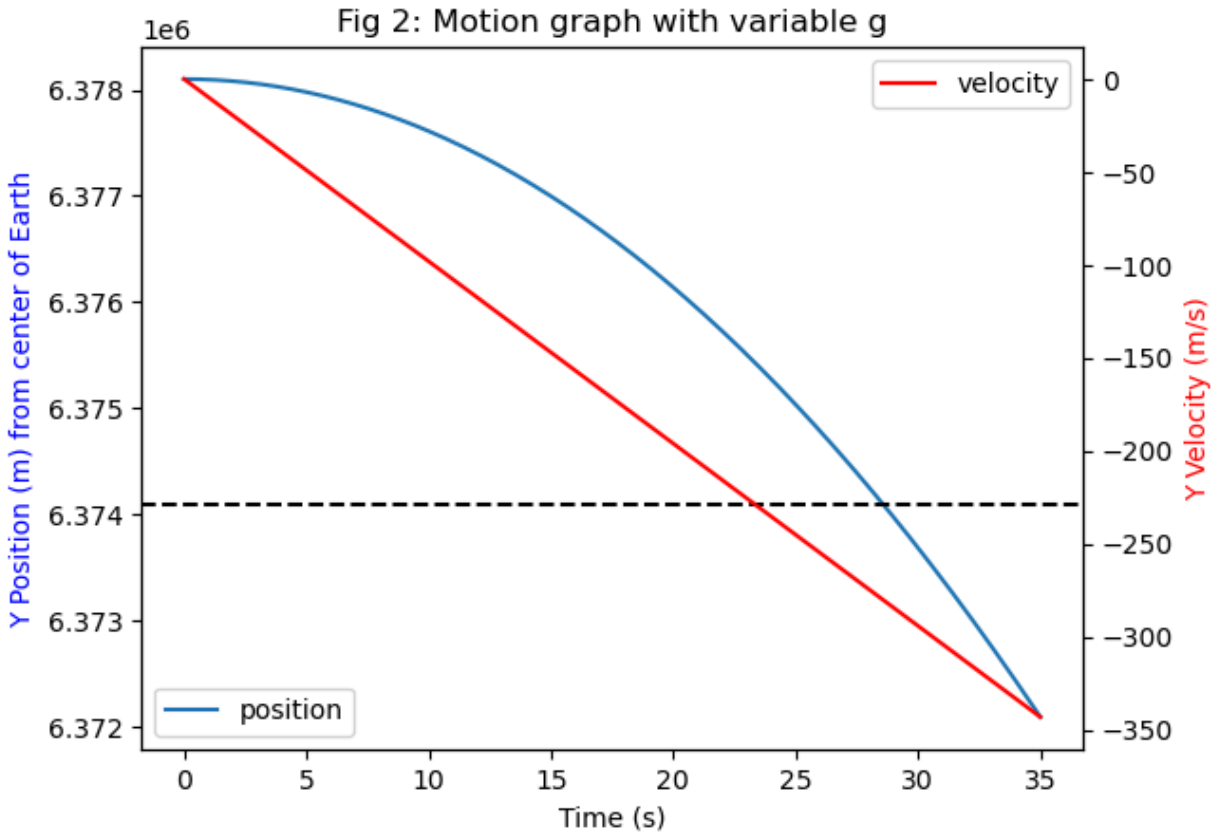


Fig 2: In this graph, we are taking into account a variable gravity that decreses linearly to the distance of the mass in relation to the center of Earth.As we can see, at t=0 seconds, the position starts at Earth's radius, and it decreases at an increasing speed. The velocity graph shows this as we can see the velocity is negative and decreasing almost linearly.

## Problem 2

In [7]:
```
y_stop=Re-4000
def stopping(t,m):
    y,v=m
    return y-y_stop

stopping.terminal=True
sol2 = solve_ivp(fun=E2,
                         t_span=(t0,tf), #tuple
                         y0=ic, #as array,
                                t_eval=t_eval,events=stopping
                               )
#print(sol2)
t_event2=sol2.t_events[0][0]
print(f'Using the events detection of solve_ivp, we find that the time it hits the bot
```

```
Using the events detection of solve_ivp, we find that the time it hits the bottom is
28.558355114121994 seconds
```

The fall time changes by 0.0015 seconds when we use a height dependent y. This is a very small change, which makes sense, since we are only dropping it 4km down, and the Earth's radius is 6378.1 km. This means that the ratio between the minshaft and the Earths radius is very small

which is shown by the fact that the time using the height dependent formula doesn't really change in comparison to the original ideal fall time.

## Problem 3

In [8]:
```
#finding drag

#0=-g -a v^2
#a=g/v^2
v_terminal=50 #m/s
alpha= g0/v_terminal**2
print(f'The value of alpha is {alpha} s/m^2' )
```

```
The value of alpha is 0.003924 s/m^2
```

In [9]:
```
#Defining the functions

a=alpha
vdep=2

def g(y):
    gr=g0*(y)/Re
    return gr
def E3(t,m):
    y,v=m
    dydt=v
    dvdt=-g(y)+a*v**vdep
    return [dydt,dvdt]

#thing 2: time
t0,tf= 0, 85
t_eval = np.linspace(t0,tf, 500)


#thing 3:IC
y0 = Re #m
v0=0 #m/s
ic=[y0,v0]


#solve_ivp
stopping.terminal=False
sol=solve_ivp(fun=E3,t_span=(t0,tf),y0=ic,t_eval=t_eval)

# extract data
y3=sol.y[0]
v3=sol.y[1]
time3=sol.t

#plot
fig, ax1 = plt.subplots()
ax1.plot(time3,y3,label='position')
ax1.set_ylim(Re-5000,Re+1000)
ax2=ax1.twinx()
ax2.plot(time3,v3,label='velocity', color='red')
ax1.axhline(y=Re-4000,color='black',ls='--')

ax1.set_title('Fig 2(b): Motion graph with variable g AND drag')
```

```
ax1.set_ylabel('Y Position (m) from center of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1.set_xlabel('Time (s)')
ax1.legend(loc='upper center')
ax2.legend(loc='upper right');
```

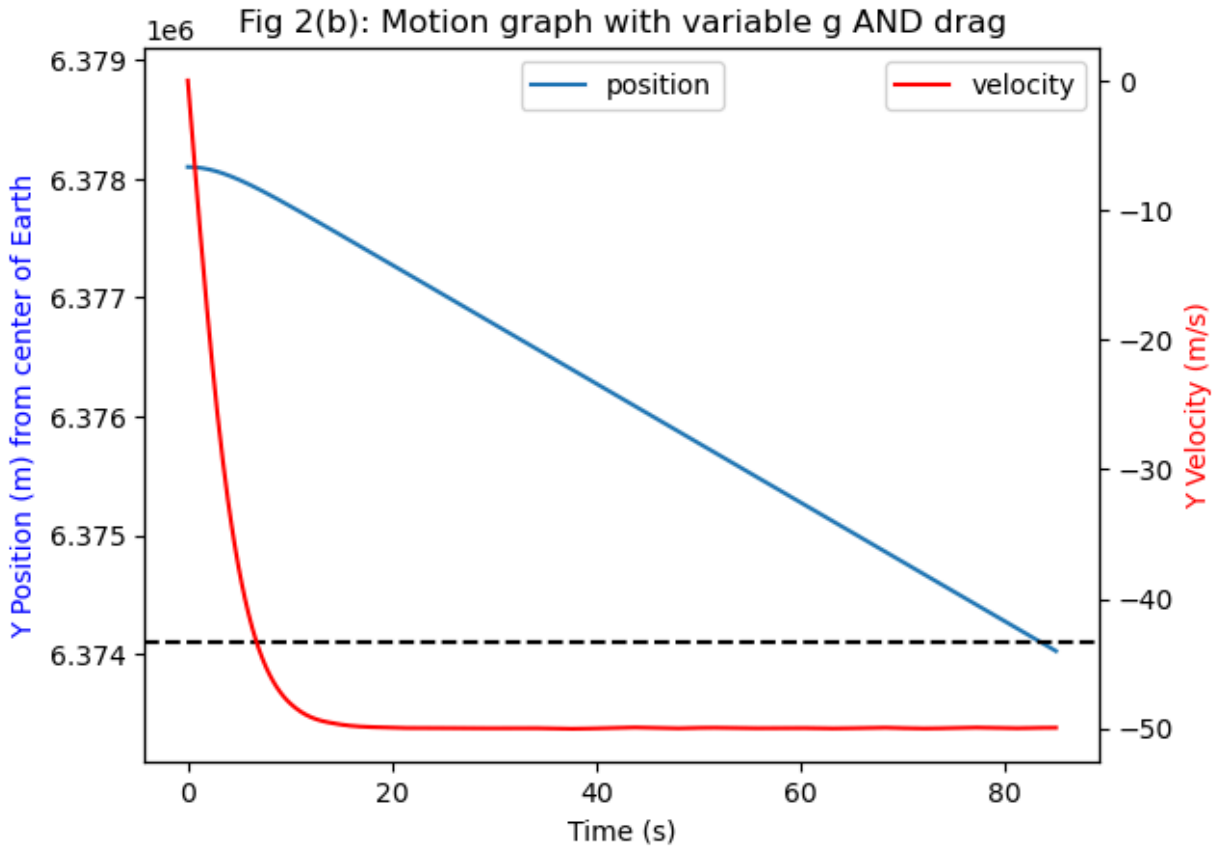


Fig 2(b): In this plot, we plot the y position from the center of Earth, and velocity, but we include drag in a way which the terminal velocity is 50 m/s. This graph looks different to the past graphs. We can see how the position is decreasing exponentially, at the start of the drop, but it starts to decrease more lineraly over time, until it reaches aroung 15 sec where it decerases fully linearly. The velocity plot agrees with this, since we can see how the velocity is negative and starts decreasing, but it starts leveling out at -50 m/s since that is when the drag force is equal to the grafivity force, therefore there is no more acceleration and the velocity remains constant.

In [10]:
```
y_stop=Re-4000
def stopping(t,m):
    y,v=m
    return y-y_stop

stopping.terminal=True
sol3 = solve_ivp(fun=E3,
                         t_span=(t0,tf), #tuple
                         y0=ic, #as array,
                                t_eval=t_eval,events=stopping
                               )
t_event3=sol3.t_events[0][0]
print(f'Using the events detection of solve_ivp, we find that the time it hits the bot
```

```
Using the events detection of solve_ivp, we find that the time it hits the bottom whe
n we take into account the variable g and drag is 83.5428725861718 seconds
```

# Part 3: The Coriolis Force

## Problem 1

To take into account the coriolis effect we must use three dimensions instead of one. Before we were using the y direction as "down into the mineshaft", but now we have to take into account forces going int he perpendicular plane of the mineshaft. We will set the x_vector to be pointing East, and the z_vector to point North. Now we have $\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$. Something worth noting is that the z direction is not affected by the coriolis effect, therefore it doesn't make sense to include it in our data since it is unnecesary.

The coriolis effect equation is $\vec{F}_c = -2m(\vec{\Omega} x \vec{v})$. Where $\Omega$ is the Earth's rotation, $m$ is the mass, and $\vec{v}$ is the velocity vector. Using this equation, we find that:

$\vec{F_{cx}} = +2m\Omega\vec{v_y}$

$\vec{F_{cy}} = -2m\Omega\vec{v_x}.$

Since $F = ma$ or $a = \frac{F}{m}$, we can use the Coriolis equations to get:

$\vec{a_{cx}} = +2\Omega\vec{v_y}$

$\vec{a_{cy}} = -2\Omega\vec{v_x}.$

In [11]:
```
a=0
def E4(t,r_vector):
    x,y,vx,vy=r_vector
    dxdt=vx
    dydt=vy
    dvxdt=2*Rot*vy
    dvydt=-g(y)+a*vy**vdep-2*Rot*vx
    return dxdt,dydt,dvxdt,dvydt
```

## Problem 2

In [12]:
```
a=0
#thing 2: time
t0,tf= 0, 35
t_eval = np.linspace(t0,tf, 500)


#thing 3:IC
x0=0 #m
y0 = Re #m
vy0=0
vx0=0 #m/s
```

```
ic=[x0,y0,vx0,vy0]

#stopping
def stopping(t,r_vector):
    x,y,vx,vy=r_vector
    return x+2.5


#solve_ivp
stopping.terminal=False
sol4=solve_ivp(fun=E4,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)
#print(sol)

# extract data
x4=sol4.y[0]
y4=sol4.y[1]
vx4=sol4.y[2]
vy4=sol4.y[3]
time4=sol4.t
#plot
fig, ax1 = plt.subplots()
ax1.scatter(y4[::20],x4[::20],label='position')
ax1.axvline(x=Re-4000,ls='--',color='black',label='Bottom of shaft')
#ax1.set_ylim(Re-5000,Re+1000)
#ax2=ax1.twinx()
#ax2.plot(time3,v3,label='velocity', color='red')
#ax1.axhline(y=-5,color='black',ls='--')
#ax1.axhline(y=0,color='black',ls='--')


ax1.set_title('Fig 3: Transverse position with Coriolis effect (with variable g, no dr
ax1.set_ylabel('X Position (m)')
#ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1.set_xlabel('Y position to the center of earth (m)')
ax1.legend(loc='upper left');
```

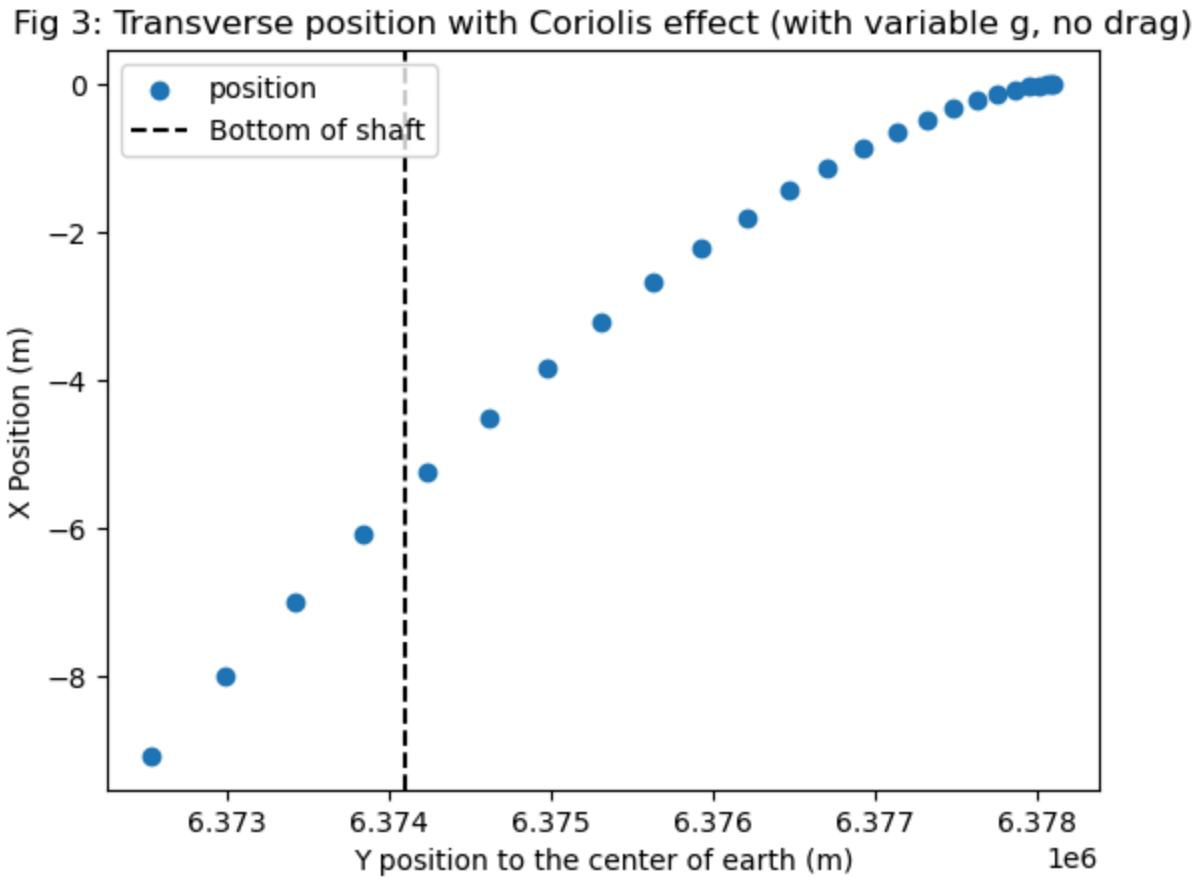


Fig 3: This graph shows the y and x positions inside the mineshaft. The y position is the position from the center of the Earth, and the x position is the x displacement in relation to where it is initially dropped. As we can see, the x position of the mass strays further from the starting point the further down it goes.

## Problem 3

In [13]:
```
#plot
fig, ax1 = plt.subplots()
ax1.scatter(y4[::20],x4[::20],label='position')

#ax1.set_ylim(Re-5000,Re+1000)
#ax2=ax1.twinx()
#ax2.plot(time3,v3,label='velocity', color='red')

ax1.plot([Re,Re-4000],[2.5,2.5],color='black',ls='--', label='edge of shaft')
ax1.plot([Re,Re-4000],[-2.5,-2.5],color='black',ls='--')
ax1.plot([Re-4000,Re-4000],[2.5,-2.5],color='black',ls='--')

#ax1.axvline(x=Re-4000,ls='--',color='black')
#ax1.axhline(y=-5,color='black',ls='--')
#ax1.axhline(y=0,color='black',ls='--')


ax1.set_title('Fig 3(b): Representation of the mineshaft\nand tranverse position with
ax1.set_ylabel('X Position (m)')
#ax2.set_ylabel('Y Velocity (m/s)', color='red')
```

```
ax1.set_xlabel('Y position to the center of earth (m)')
ax1.legend();
```

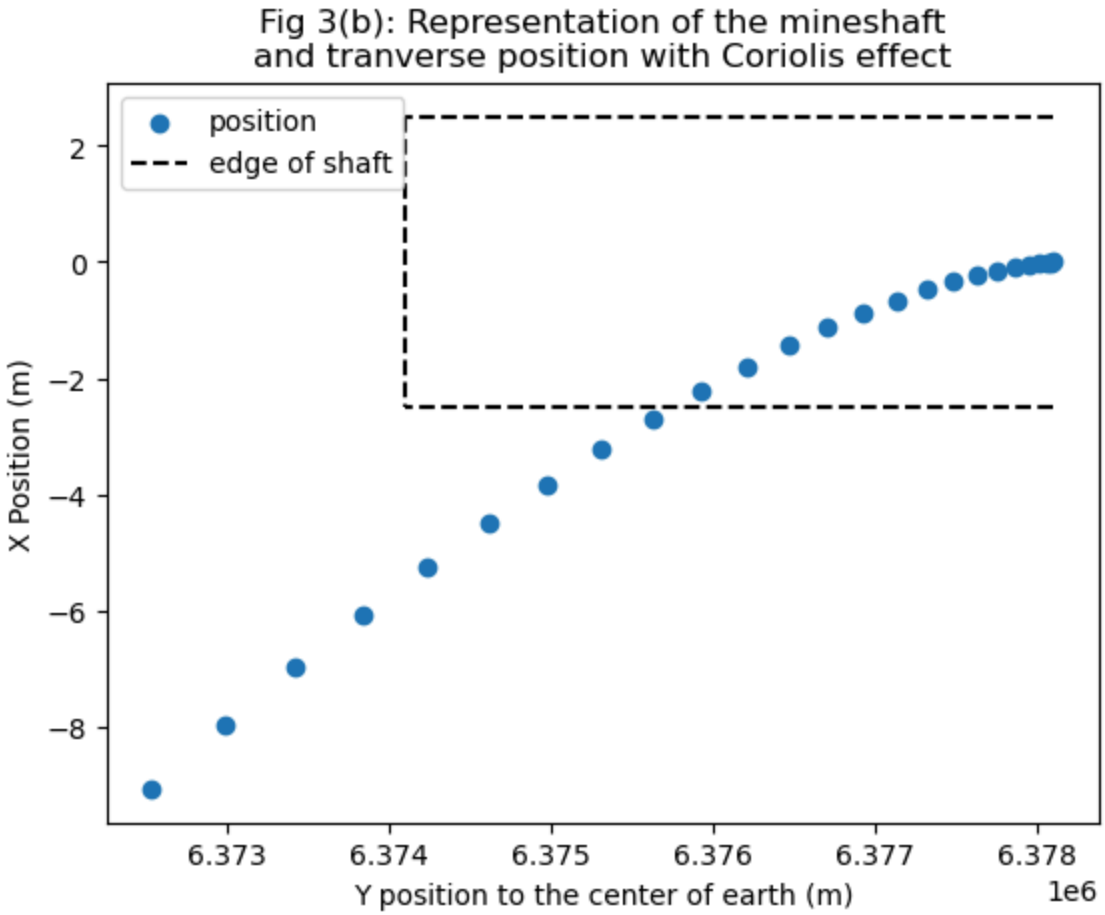


Fig 3(b): This graph is related to Part 3, Problem 3. If we graph a 4000 m deep and 5 m wide mineshaft with the transverse position of the mass, we can see that the mass hits the side of the shaft before it reaches the bottom.

In [14]:
```
#Time to reach other  side
#print(sol4)
t_event4_side=sol4.t_events[0][0]
t_event4_side_p=sol4.y_events[0][0][1]
print(f'Using the events detection of solve_ivp, we find that the time it reaches to h
```

```
Using the events detection of solve_ivp, we find that the time it reaches to hit a si
de of the mineshaft is 21.90711407103461 seconds at 2353.8688868256286 meters
```

## Problem 4

In [15]:
```
a=alpha

#thing 2: time
t0,tf= 0, 90
t_eval = np.linspace(t0,tf, 500)


#thing 3:IC
x0=0 #m
y0 = Re #m
```

```
vy0=0
vx0=0 #m/s
ic=[x0,y0,vx0,vy0]

#solve_ivp
sol5=solve_ivp(fun=E4,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)


# extract data
x5=sol5.y[0]
y5=sol5.y[1]
vx5=sol5.y[2]
vy5=sol5.y[3]
time5=sol.t
#plot
fig, ax1 = plt.subplots()
ax1.scatter(y4[::20],x4[::20],label='Position without drag',color='b')
ax1.scatter(y5[::20],x5[::20],label='Position WITH drag',color='r')

ax1.plot([Re,Re-4000],[2.5,2.5],color='black',ls='--',label='edge of shaft')
ax1.plot([Re,Re-4000],[-2.5,-2.5],color='black',ls='--')
ax1.plot([Re-4000,Re-4000],[2.5,-2.5],color='black',ls='--')
#ax1.axvline(x=Re-4000,ls='--',color='black')
#ax1.set_ylim(Re-5000,Re+1000)
#ax2=ax1.twinx()
#ax2.plot(time3,v3,label='velocity', color='red')

#ax1.axhline(y=0,color='black',ls='--')
#ax1.axhline(y=-5,color='black',ls='--')
#ax1.axhline(y=0,color='black',ls='--')


ax1.set_title('Fig 3(c): Transverse position with Coriolis effect\n(with variable g AN
ax1.set_ylabel('X Position (m)')
#ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1.set_xlabel('Y position to the center of earth (m)')
ax1.legend();
```

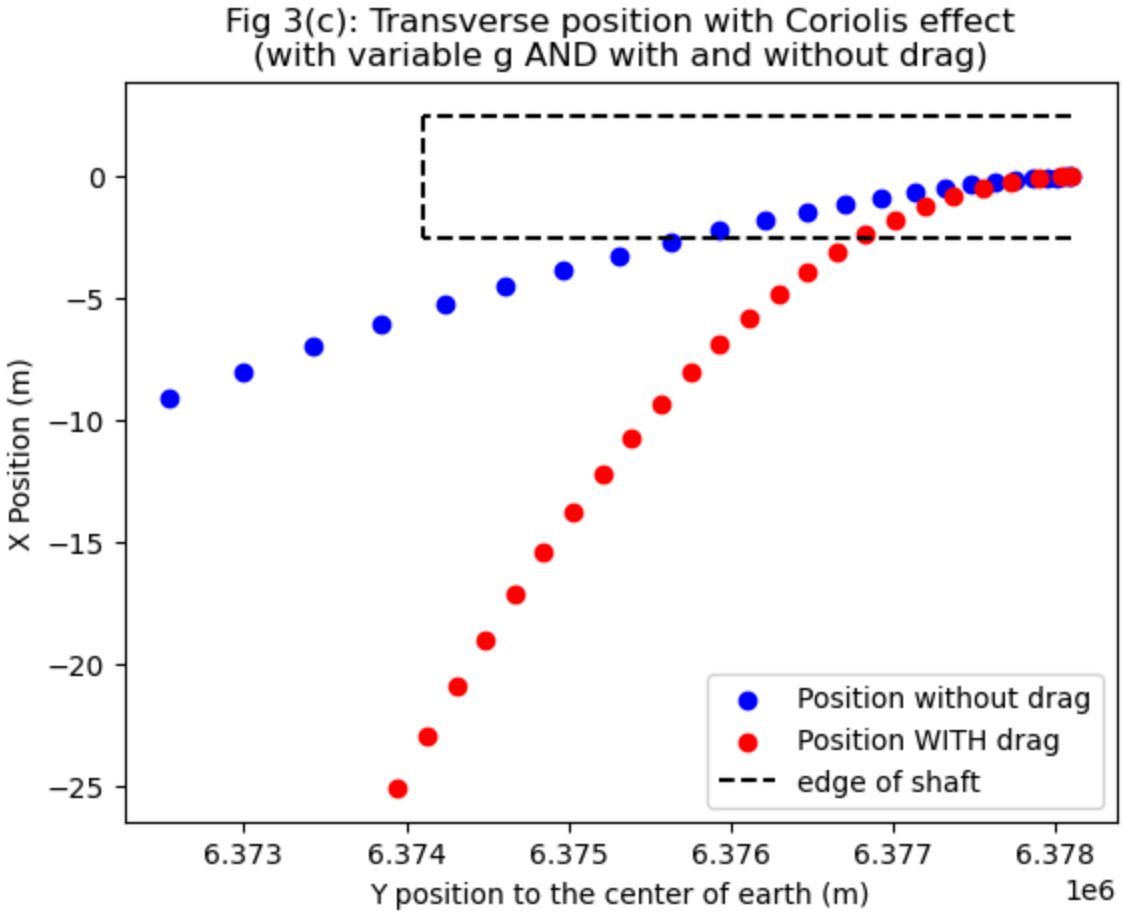


Fig 3(c): Here we have plotted the position graph without drag in blue, and the position graph with drag in red. As we can see, the plot with drag hits the side of the shaft much closer to the top than the plot without drag.

In [16]:
```
#Time to reach other side with drag
#print(sol4)
t_event5_side=sol5.t_events[0][0]
t_event5_side_p=sol5.y_events[0][0][1]
print(f'Using the events detection of solve_ivp, we find that the time it reaches to h
```

```
Using the events detection of solve_ivp, we find that the time it reaches to hit a si
de of the mineshaft is 29.583292530666125 seconds at 1302.5383943542838 meters
```

# Part 4: An infinitely deep mine

## Problem 1

In [17]:
```
a=0
def g(y):
    gr=g0*(y)/Re
    return gr
def E6(t,m):
    y,v=m
    dydt=v
    dvdt=-g(y)+a*v**vdep
    return [dydt,dvdt]
```

```
#thing 2: time
t0,tf= 0, 7000
t_eval = np.linspace(t0,tf, 10000)


#thing 3:IC
y0 = Re #m
v0=0 #m/s
ic=[y0,v0]

#stop
def stopping(t, m):
    y, v = m
    return y + Re


stopping.terminal = True
stopping.direction = -1

#Solve_ivp
sol6 = solve_ivp(fun=E6, t_span=(t0, tf), y0=ic, t_eval=t_eval, events=stopping)

# Extract
y6 = sol6.y[0]
v6 = sol6.y[1]
time6 = sol6.t

# Plot
fig, ax1 = plt.subplots()
ax1.plot(time6, y6, label='Position', color='blue')
ax1.set_ylim(-1.1*Re, 1.1*Re)
ax1.axhline(y=0, color='black', linestyle='--')
ax1.axhline(y=Re, color='black', linestyle='--')
ax1.axhline(y=-Re, color='black', linestyle='--')
ax2 = ax1.twinx()
ax2.plot(time6, v6, label='Velocity', color='red')
ax2.set_ylabel('Y Velocity (m/s)', color='red')


ax1.set_title('Fig 4: Infinite Mineshaft Plot with Variable g (No Drag)')
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Y Position (m) from center of Earth', color='blue')
ax1.legend(loc='upper left')
ax2.legend();
```

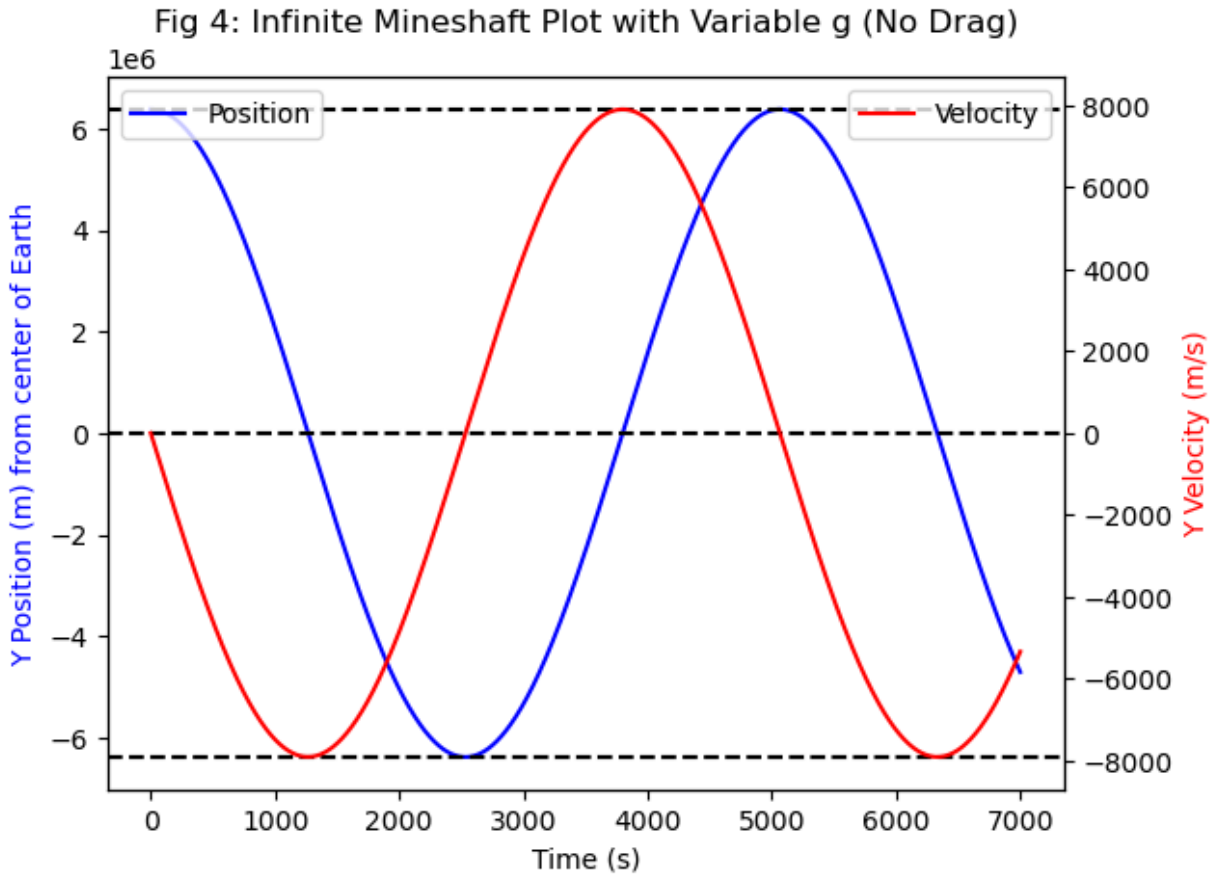


Fig 4: In this graph we can see the position and velocity graph of an object going through a uniform density earth. We can see how it reaches the other side of Earth and comes back up sinosoudly.

## Problem 2

In [18]:
```
#Time to reach other  side

def stopping(t,m):
    y,v=m
    return v

stopping.terminal=False
sol6 = solve_ivp(fun=E6,
                         t_span=(t0,tf), #tuple
                         y0=ic, #as array,
                                t_eval=t_eval,events=stopping
                               )
#print(sol6)
t_event6_os=sol6.t_events[0][1]
print(f'Using the events detection of solve_ivp, we find that the time it reaches the
```

```
Using the events detection of solve_ivp, we find that the time it reaches the other s
ide when we take into account the variable g is 2532.6120147256643 seconds
```

In [19]:
```
#Time to reach center of earth
y_stop=0

def stopping(t,m):
```

```
    y,v=m
    return y

stopping.terminal=True

sol6 = solve_ivp(fun=E6,
                         t_span=(t0,tf), #tuple
                         y0=ic, #as array,
                                t_eval=t_eval,events=stopping
                               )
#print(sol6)
t_event6_ce_t=sol6.t_events[0][0]
t_event6_ce_v=sol6.y_events[0][0][1]

print(f'Using the events detection of solve_ivp, we find that the time it reaches the
```

```
Using the events detection of solve_ivp, we find that the time it reaches the center
of the Earth when we take into account the variable g is 1266.473469558803 seconds, a
nd it passes it with a speed of 7910.759872777071 m/s downwards.
```

## Problem 3

In [20]:
```
#speeds
orbit_v=np.sqrt(G*Me/Re)
print(f'The orbital speed is {orbit_v} m/s. This value is incredibly close to the spee
```

```
The orbital speed is 7905.277129890415 m/s. This value is incredibly close to the spe
ed when the mass is at the center of Earth which, using solve_ivp event detection, is
7910.759872777071 m/s.
```

In [21]:
```
#orbit times
orbit_t=np.pi*2*Re/orbit_v
print(f'The crossing time when the ball is at the center of the Earth is very close to
```

```
The crossing time when the ball is at the center of the Earth is very close to 1/4 of
the orbital period, since 4 times the crossing time is 5065.893878235212 seconds and
the orbital period is 5069.371199675785 seconds.
```

# Part 5: A non-uniform Earth

## Problem 1

In [22]:
```
#Plot normalised density profile

def p_0(y):
    pr=1*(1-(y/Re)**2)**0
    return pr
def p_1(y):
    pr=1*(1-(y/Re)**2)**1
    return pr
def p_2(y):
    pr=1*(1-(y/Re)**2)**2
    return pr
def p_9(y):
    pr=1*(1-(y/Re)**2)**9
    return pr
```

```
fig,ax=plt.subplots()
rr=np.linspace(0,Re,1000)
p0=p_0(rr)
p1=p_1(rr)
p2=p_2(rr)
p9=p_9(rr)
ax.plot(rr,p0,label='n=0')
ax.plot(rr,p1,label='n=1')
ax.plot(rr,p2,label='n=2')
ax.plot(rr,p9,label='n=9')

ax.set_ylabel('Normalized Density ρ(r) (kg/m**3)')
ax.set_title('Fig 5: Normalized Density Profile for Different Exponents (n)')
ax.set_xlabel('Position from the center of the Earth (m)')
ax.legend();
```

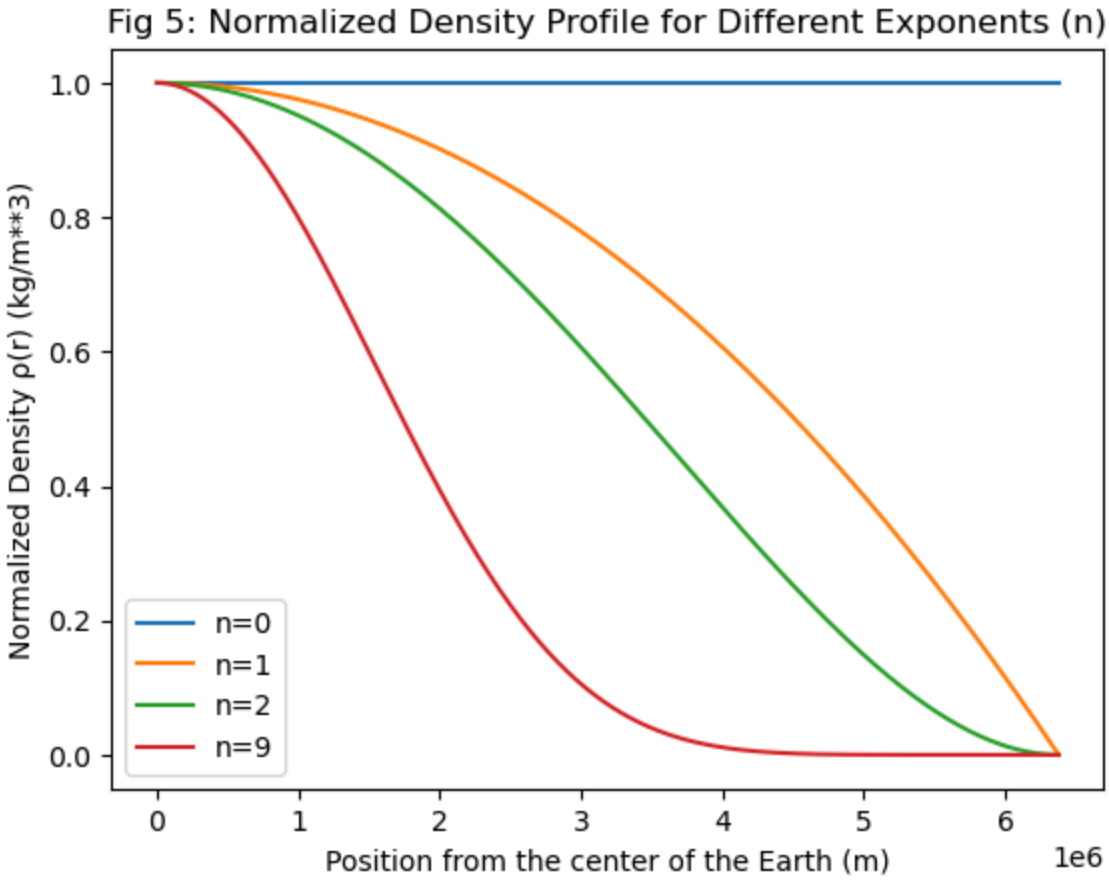


Fig 5: In this graph we can see how density changes depending on the density for different n=0,1,2,9.

## Problem 2

The simple model for density can be represented as:

$$\rho(r) = \rho_n(1 - r^2)^n$$

The mass of earth can be represented as:

$$M = \int_V \rho(r)\, dV = \int_0^{2\pi} \int_0^{\pi} \int_0^{R_\oplus} \rho(r) r^2 \sin\phi \, dr \, d\phi \, d\theta = 4\pi \int_0^{R_\oplus} \rho(r) r^2 \, dr$$

And to solve for the density constant for each n we will reorganize the equation to get:

$$\rho_n = \frac{M}{4\pi \int_0^{R_\oplus} (1-(r/R_e)^2) r^2 \, dr}$$

In [23]:
```
#density
pr0=lambda r:(1 - (r/Re)**2)**0
pr1=lambda r:(1 - (r/Re)**2)**1
pr2=lambda r:(1 - (r/Re)**2)**2
pr9=lambda r:(1 - (r/Re)**2)**9

#normalization factor
pn0=Me/( 4*np.pi*scipy.integrate.quad((lambda r: pr0(r)*(r)**2),0,Re )[0])
pn1=Me/( 4*np.pi*scipy.integrate.quad((lambda r: pr1(r)*(r)**2),0,Re )[0])
pn2=Me/( 4*np.pi*scipy.integrate.quad((lambda r: pr2(r)*(r)**2),0,Re )[0])
pn9=Me/( 4*np.pi*scipy.integrate.quad((lambda r: pr9(r)*(r)**2),0,Re )[0])
#print(pn0,pn1,pn2,pn9)

def F0(y):
    Mr=pn0*4*np.pi*scipy.integrate.quad((lambda r: pr0(r)*r**2),0,y )[0]
    Fr=G*Mr/y**2
    return Fr
def F1(y):
    Mr=pn1*4*np.pi*scipy.integrate.quad((lambda r: pr1(r)*r**2),0,y )[0]
    Fr=G*Mr/y**2
    return Fr
def F2(y):
    Mr=pn2*4*np.pi*scipy.integrate.quad((lambda r: pr2(r)*r**2),0,y )[0]
    Fr=G*Mr/y**2
    return Fr
def F9(y):
    Mr=pn9*4*np.pi*scipy.integrate.quad((lambda r: pr9(r)*r**2),0,y )[0]
    Fr=G*Mr/y**2
    return Fr
yy=np.linspace(0.01,Re,1000)

F0_graph=np.array([F0(y) for y in yy])
F1_graph=np.array([F1(y) for y in yy])
F2_graph=np.array([F2(y) for y in yy])
F9_graph=np.array([F9(y) for y in yy])

fig,ax=plt.subplots()
ax.plot(yy,F0_graph,label='n=0')
ax.plot(yy,F1_graph,label='n=1')
ax.plot(yy,F2_graph,label='n=2')
ax.plot(yy,F9_graph,label='n=9')

ax.set_xlabel('Radius from the center of the Earth (m)')
ax.set_ylabel('Force (N)')
ax.set_title('Fig 6: Force Profile for Different Exponents (n)')
ax.legend();
```

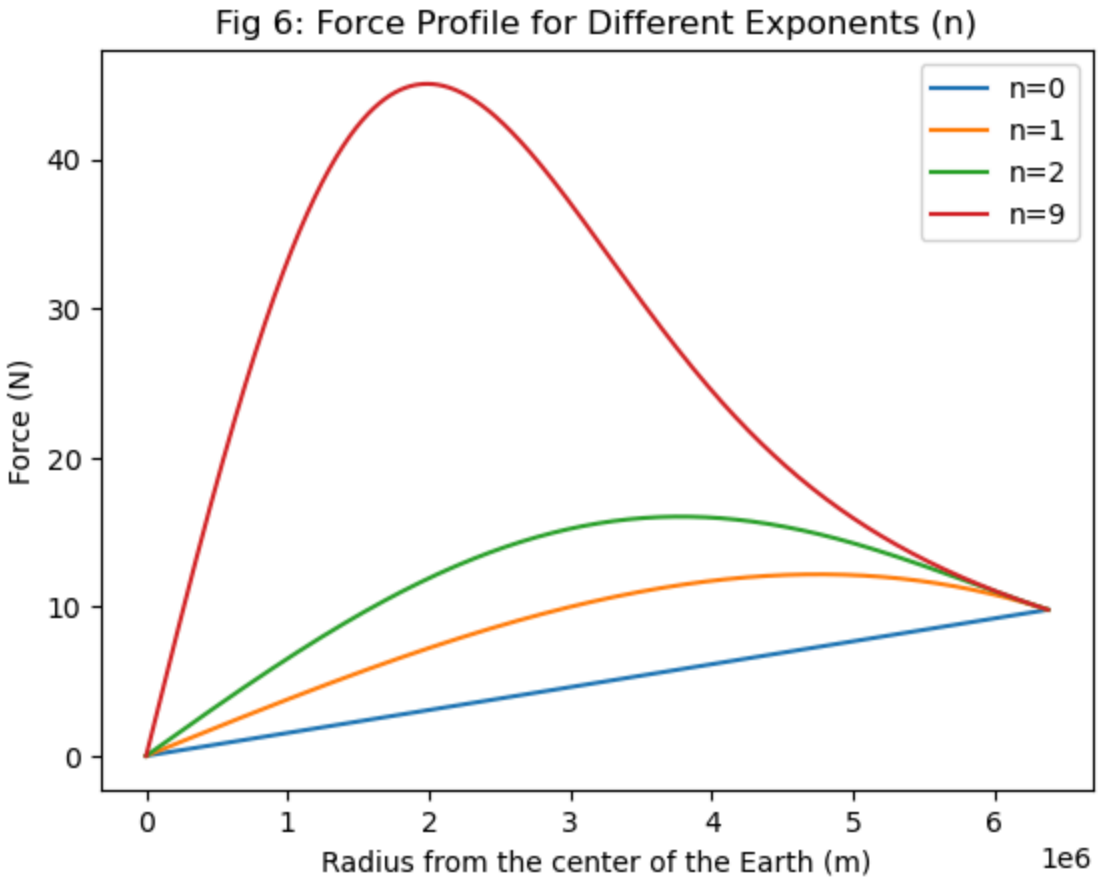


Fig 6: Here we can see how much force is applied to an object at different distances from the center of Earth for different density distributions.

## Problem 3

In [24]:
```
#Defining the functions

a=0
vdep=2


def E7_0(t,m):
    y,v=m
    dydt=v
    dvdt=-F0(y)+a*v**vdep
    return [dydt,dvdt]

def E7_1(t,m):
    y,v=m
    dydt=v
    dvdt=-F1(y)+a*v**vdep
    return [dydt,dvdt]

def E7_2(t,m):
    y,v=m
    dydt=v
    dvdt=-F2(y)+a*v**vdep
    return [dydt,dvdt]
```

```
def E7_9(t,m):
    y,v=m
    dydt=v
    dvdt=-F9(y)+a*v**vdep
    return [dydt,dvdt]

#thing 2: time
t0,tf= 0, 10000
t_eval = np.linspace(t0,tf, 50000)


#thing 3:IC
y0 = Re #m
v0=0 #m/s
ic=[y0,v0]

#stopping
def stopping(t,m):
    y,v=m
    return y


fig, ax1 = plt.subplots(4,1,figsize=(10,15))
#n=0
stopping.terminal=False
sol7_0=solve_ivp(fun=E7_0,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)
y7_0=sol7_0.y[0]
v7_0=sol7_0.y[1]
time7_0=sol7_0.t
ax1[0].plot(time7_0,y7_0,label='position')
ax2=ax1[0].twinx()
ax2.plot(time7_0,v7_0,label='velocity', color='red')
ax1[0].set_title('Fig 7: Motion graph with variable density distribution n=0')
ax1[0].set_ylabel('Y Position (m) from center of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1[0].set_xlabel('Time (s)')

#n=1
sol7_1=solve_ivp(fun=E7_1,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)
y7_1=sol7_1.y[0]
v7_1=sol7_1.y[1]
time7_1=sol7_1.t
ax1[1].plot(time7_1,y7_1,label='position')
ax2=ax1[1].twinx()
ax2.plot(time7_1,v7_1,label='velocity', color='red')
ax1[1].set_title('Fig 7: Motion graph with variable density distribution n=1')
ax1[1].set_ylabel('Y Position (m) from center of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1[1].set_xlabel('Time (s)')

#n=2
sol7_2=solve_ivp(fun=E7_2,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)
y7_2=sol7_2.y[0]
v7_2=sol7_2.y[1]
time7_2=sol7_2.t

ax1[2].plot(time7_2,y7_2,label='position')
ax2=ax1[2].twinx()
ax2.plot(time7_2,v7_2,label='velocity', color='red')
ax1[2].set_title('Fig 7: Motion graph with variable density distribution n=2')
```

```
ax1[2].set_ylabel('Y Position (m) from center of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1[2].set_xlabel('Time (s)')

#n=9
sol7_9=solve_ivp(fun=E7_9,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)
y7_9=sol7_9.y[0]
v7_9=sol7_9.y[1]
time7_9=sol7_9.t
ax1[3].plot(time7_9,y7_9,label='position')
ax2=ax1[3].twinx()
ax2.plot(time7_9,v7_9,label='velocity', color='red')
ax1[3].set_title('Fig 7: Motion graph with variable density distribution n=9')
ax1[3].set_ylabel('Y Position (m) from center of Earth', color='blue')
ax2.set_ylabel('Y Velocity (m/s)', color='red')
ax1[3].set_xlabel('Time (s)')

fig.tight_layout();
```

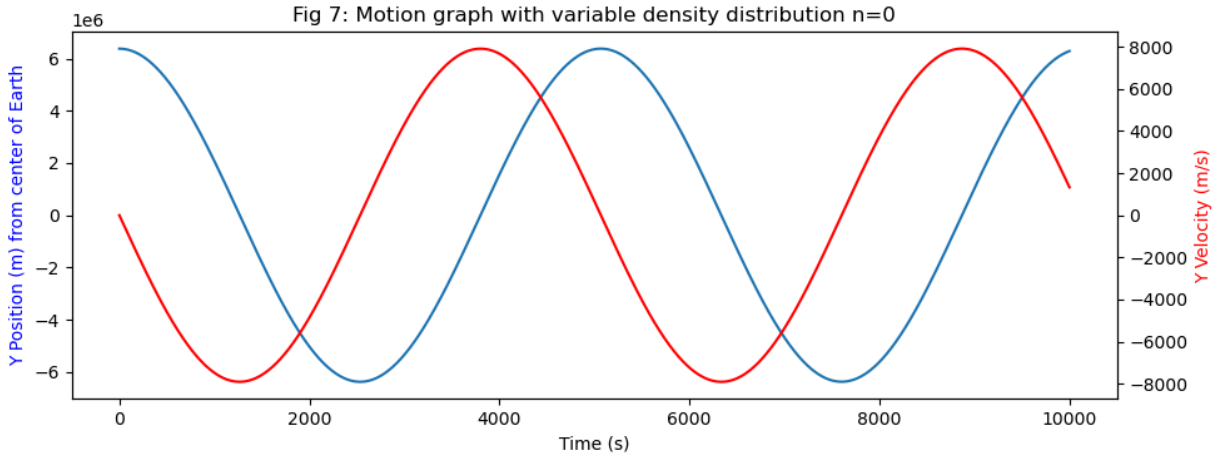
Fig 7: Motion graph with variable density distribution n=0
1e6
Y Position (m) from center of Earth
Y Velocity (m/s)
Time (s)

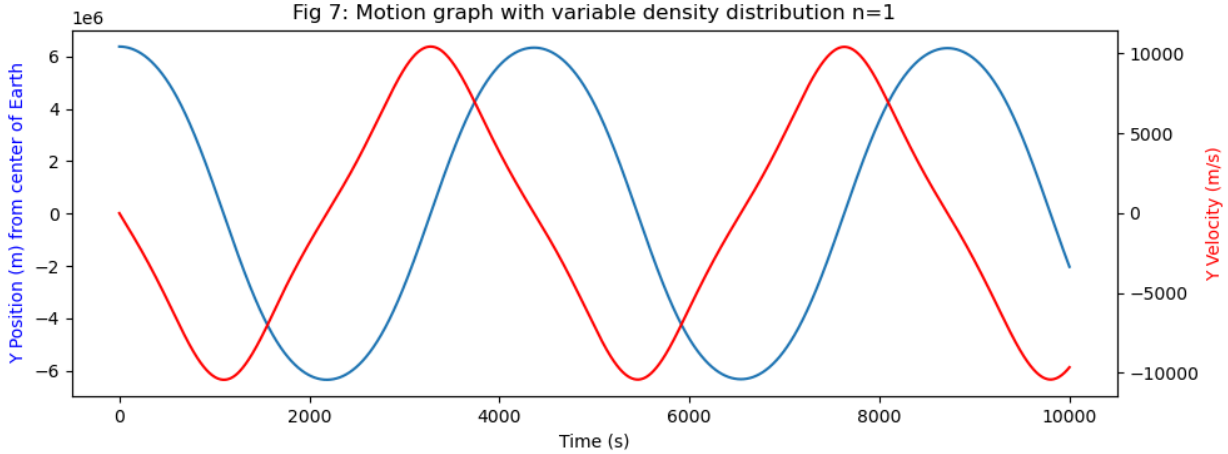
Fig 7: Motion graph with variable density distribution n=1
1e6
Y Position (m) from center of Earth
Y Velocity (m/s)
Time (s)

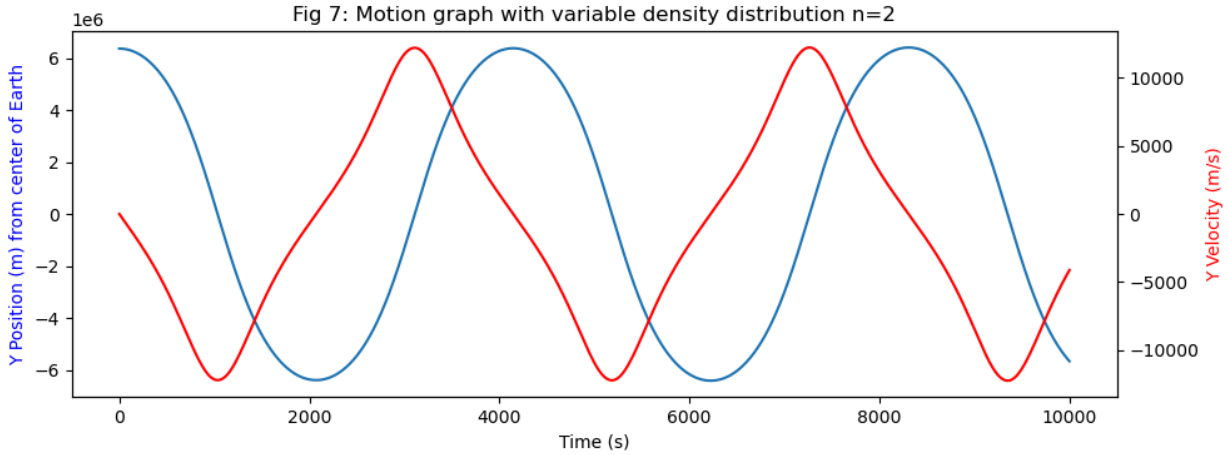
Fig 7: Motion graph with variable density distribution n=2
1e6
Y Position (m) from center of Earth
Y Velocity (m/s)
Time (s)

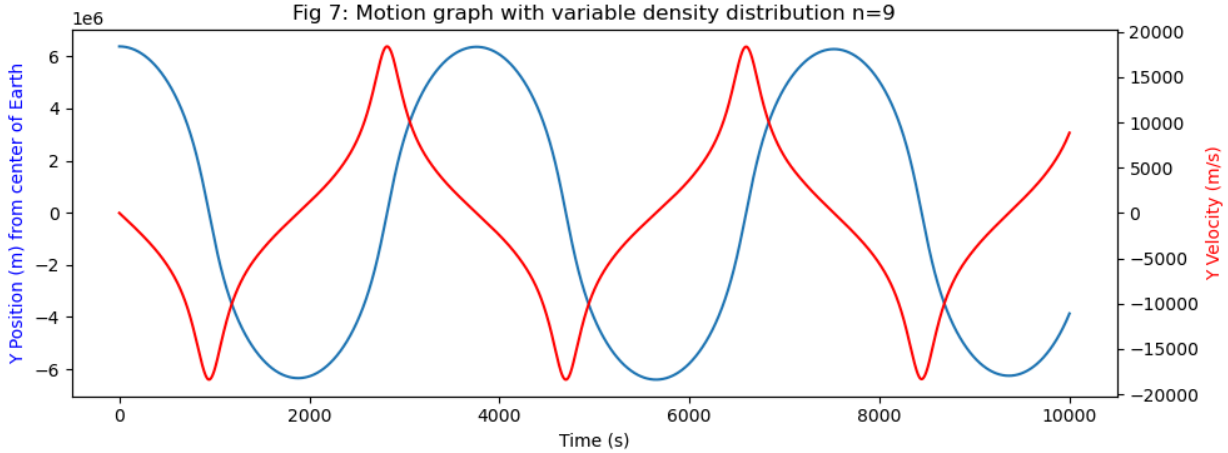
Fig 7: Motion graph with variable density distribution n=9
1e6
Y Position (m) from center of Earth
Y Velocity (m/s)
Time (s)

Fig 7: Here we can see the position and velocity graphs for dfferent variable gravity that correspond with a different density distribution for n=0,1,2,9.

## Problem 4

In [29]:
```
#Time to reach center of earth

print(sol7_0)



print(f'In the graph where n=0, the time it reaches the center of the Earth is {sol7_0
print(f'In the graph where n=1, the time it reaches the center of the Earth is {sol7_1
print(f'In the graph where n=2, the time it reaches the center of the Earth is {sol7_2
print(f'In the graph where n=9, the time it reaches the center of the Earth is {sol7_9
```

```
  message: The solver successfully reached the end of the integration interval.
  success: True
   status: 0
        t: [ 0.000e+00  2.000e-01 ...  1.000e+04  1.000e+04]
        y: [[ 6.378e+06  6.378e+06 ...  6.287e+06  6.287e+06]
            [ 0.000e+00 -1.960e+00 ...  1.327e+03  1.325e+03]]
      sol: None
 t_events: [array([ 1.267e+03,  3.801e+03,  6.335e+03,  8.869e+03])]
 y_events: [array([[ 0.000e+00, -7.906e+03],
                   [ 1.572e-09,  7.905e+03],
                   [ 6.985e-09, -7.907e+03],
                   [ 4.657e-09,  7.908e+03]])]
     nfev: 128
     njev: 0
      nlu: 0
In the graph where n=0, the time it reaches the center of the Earth is 1267.240953460
1416 seconds, and it passes it with a speed of 7905.971248072909 m/s downwards.

In the graph where n=1, the time it reaches the center of the Earth is 1096.891482002
5393 seconds, and it passes it with a speed of 10435.238778034956 m/s downwards.

In the graph where n=2, the time it reaches the center of the Earth is 1035.138775425
4736 seconds, and it passes it with a speed of 12200.745609087493 m/s downwards.

In the graph where n=9, the time it reaches the center of the Earth is 943.8748065601
649 seconds, and it passes it with a speed of 18391.996831517798 m/s downwards.
```

# Part 6: A lunar mine shaft

## Problem 1

In [26]:
```
#Defining the functions

a=0
vdep=2
def g_moon(y):
    g0_moon=G*Mm/Rm**2
    gr=g0_moon*y/Rm
    return gr
```

```
def E8(t,m):
    y,v=m
    dydt=v
    dvdt=-g_moon(y)
    return [dydt,dvdt]

#thing 2: time
t0,tf= 0, 2000
t_eval = np.linspace(t0,tf, 50000)


#thing 3:IC
y0 = Rm #m
v0=0 #m/s
ic=[y0,v0]


#solve_ivp
def stopping(t,m):
    y,v=m
    return y
stopping.terminal=False

sol8=solve_ivp(fun=E8,t_span=(t0,tf),y0=ic,t_eval=t_eval,events=stopping)
#print(sol8)
# extract data
y8=sol8.y[0]
v8=sol8.y[1]
time8=sol8.t



print(f'The time it reaches the center of the Moon is {sol8.t_events[0][0]} seconds, a
```

```
The time it reaches the center of the Moon is 1624.9065288236459 seconds, and it pass
es it with a speed of 1679.9477528483299 m/s downwards.
```

## Problem 2

In [27]:

```
moon_d=Mm/(4/3*np.pi*Rm**3)

earth_d=Me/(4/3*np.pi*Re**3)

print(f'The density of the Earth is {earth_d} kg/m^3, and the density of the Moon is {
```

```
The density of the Earth is 5494.867409551201 kg/m^3, and the density of the Moon is
3341.7538038703183 kg/m^3. As we can see, the Earth is more dense than the Moon. It i
s the Earth is 1.6443064726034609 denser than the Moon.
```

## Problem 3

We will be using different equations to find a relationship between fall/orbit time vs density.

It is important to note that fall time to the center of a planet is 1/4th of the orbit time, so we will be calculating the orbit time and the fall time to the center is just a fourth of this time.

The centripetal acceleration (aka gravity) can be represented by:

$$a_{centripetal} = g = \frac{v_{orbit}^2}{R}$$

Where orbit velocity is:

$$v_{orbit} = \frac{2\pi R}{T}$$

And another way to write gravity is:

$$g = \frac{GM}{R^2}$$

Now, to include density in these equations we will rewrite the Mass as:

$$M = \frac{4}{3}\pi R^3 \rho$$

We can use the other equations to get:

$$g = \frac{4\pi^2 R}{T^2} = \frac{4}{3}\pi G R \rho$$

And we can rearrange this to get:

$$T = \sqrt{\frac{3\pi}{G\rho}}$$

And from this equation we can say that the relationship between orbital time and fall time and density is:

$$T \propto \sqrt{\frac{1}{\rho}}$$

In [28]:
```
orbit_t_p=np.sqrt(3*np.pi/(G*earth_d))

print(f'The orbit time on earth is {orbit_t} sec, and using our equation, the orbit ti
```

```
The orbit time on earth is 5069.371199675785 sec, and using our equation, the orbit t
ime is 5069.371199675784 sec
```

In [ ]:
```

```