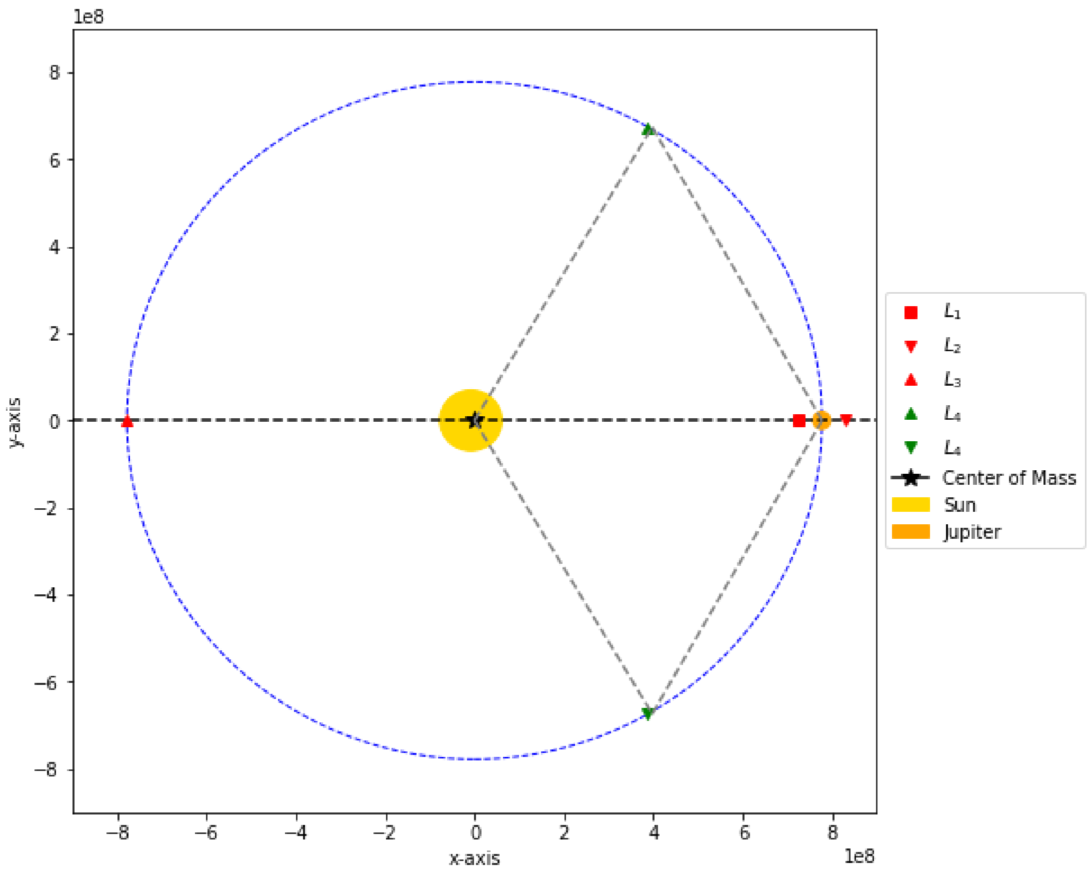
In [121]:
```python
import math
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt
from vpython import *
```

In [122]:
```python
# variables that we'll need
G = 6.67408e-20

# mass of the sun
m_sun = 1.9891e30 # kg

# radius of the sun (not to scale)
r_sun = 6.9634e5 #km

# mass of jupiter
m_jupiter = 1.898e27 # kg

# radius of jupiter
r_jupiter = 6.9911e4 # km

# distance from sun to jupiter:
dis = 7.78e8 # km

# distance of sun and jupiter from center of mass
d_sun = (m_jupiter * dis)/(m_sun + m_jupiter)
d_jupiter = (dis * m_sun) / (m_sun + m_jupiter)

# object velocities:
v_sun = math.sqrt(G*m_jupiter*d_sun/dis**2)
v_jupiter = math.sqrt(G*m_sun*d_jupiter/dis**2)

# w = angular velocity
w = v_jupiter/d_jupiter # rad/sec
```

In [123]:
```python
# expected values relative to the center of mass:
L1 = (7.25e8, 0)
L2 = (8.32e8, 0)
L3 = (-7.78e8, 0)
L4 = (3.88e8, 6.74e8)
L5 = (3.88e8, -6.74e8)
```

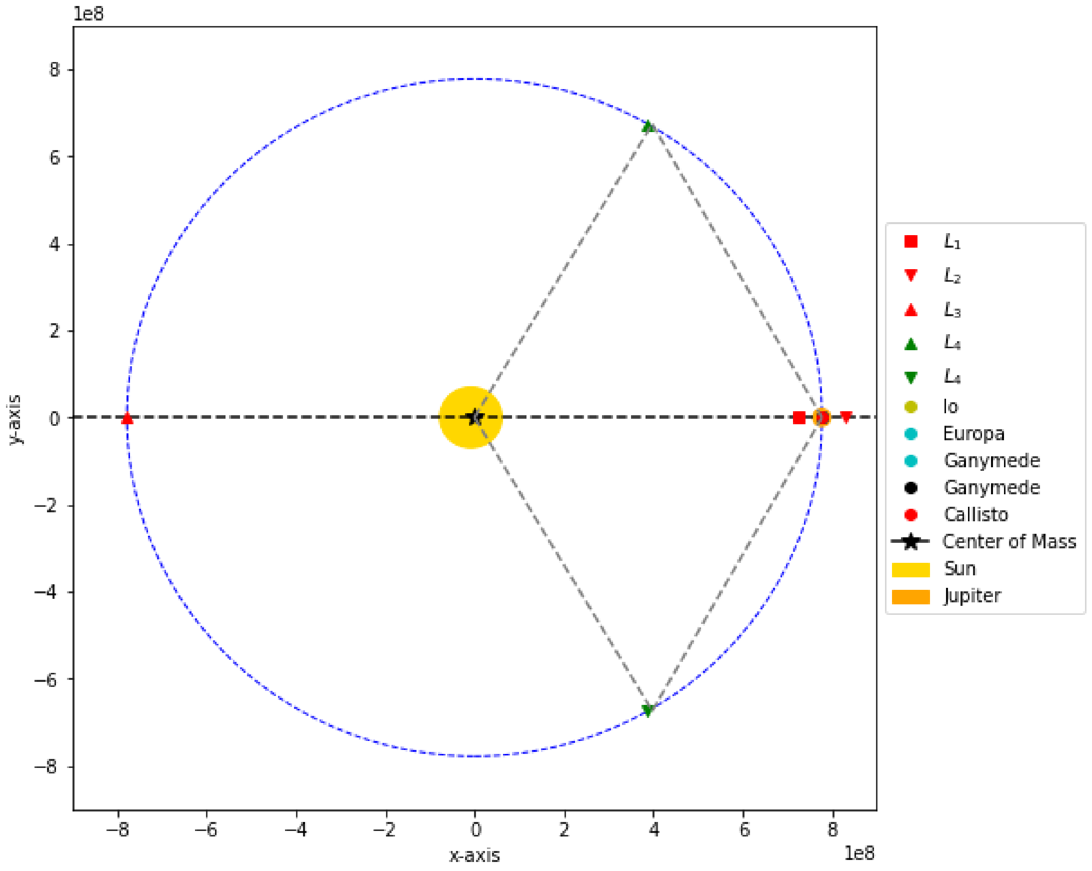
In [124]:
```
figure, ax = plt.subplots(figsize=(8,8))
ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.axhline(0, color='k', linestyle = "--")

# set axis limits
ax.set_xlim((-9e8, 9e8))
ax.set_ylim((-9e8, 9e8))

# orbit trail
orbit = plt.Circle((0, 0), 7.78e8, color='b', fill=False, linestyle = "--")

# plot the sun (not to scale), moved more in the x-axis to show sun is not at the ori
sun = plt.Circle((-8e6, 0), 7e7, color='gold', fill=True, label = "Sun")

# plot Jupiter (not to scale)
Jupiter = plt.Circle((d_jupiter, 0), 2e7, color = "orange", fill = True, label = "Jup

# add the lagrange points
ax.plot(L1[0], L1[1], 'rs', label="$L_1$")
ax.plot(L2[0], L2[1], 'rv', label="$L_2$")
ax.plot(L3[0], L3[1], 'r^', label="$L_3$")
ax.plot(L4[0], L4[1], 'g^', label="$L_4$")
ax.plot(L5[0], L5[1], 'gv', label="$L_4$")

# mark center of mass
ax.plot(0, 0, 'k', marker="*", markersize=10, label = "Center of Mass")

x_half = np.linspace(0, 7.78e8/2, 1000)
y_half = np.linspace(0, 6.74e8/2, 1000)

# plot guidelines for L4 and L5
ax.plot([0, 3.98e8], [0, 6.74e8], linestyle = "--", color = "gray")
ax.plot([3.98e8, 7.78e8], [6.74e8, 0], linestyle = "--", color = "gray")
ax.plot([0, 3.98e8], [0, -6.74e8], linestyle = "--", color = "gray")
ax.plot([3.98e8, 7.78e8], [-6.74e8, 0], linestyle = "--", color = "gray")

# Plot the orbits
ax.add_patch(orbit)
ax.add_patch(sun)
ax.add_patch(Jupiter)

# add legend
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

Out[124]:
```
<matplotlib.legend.Legend at 0x127bd1a80>
```

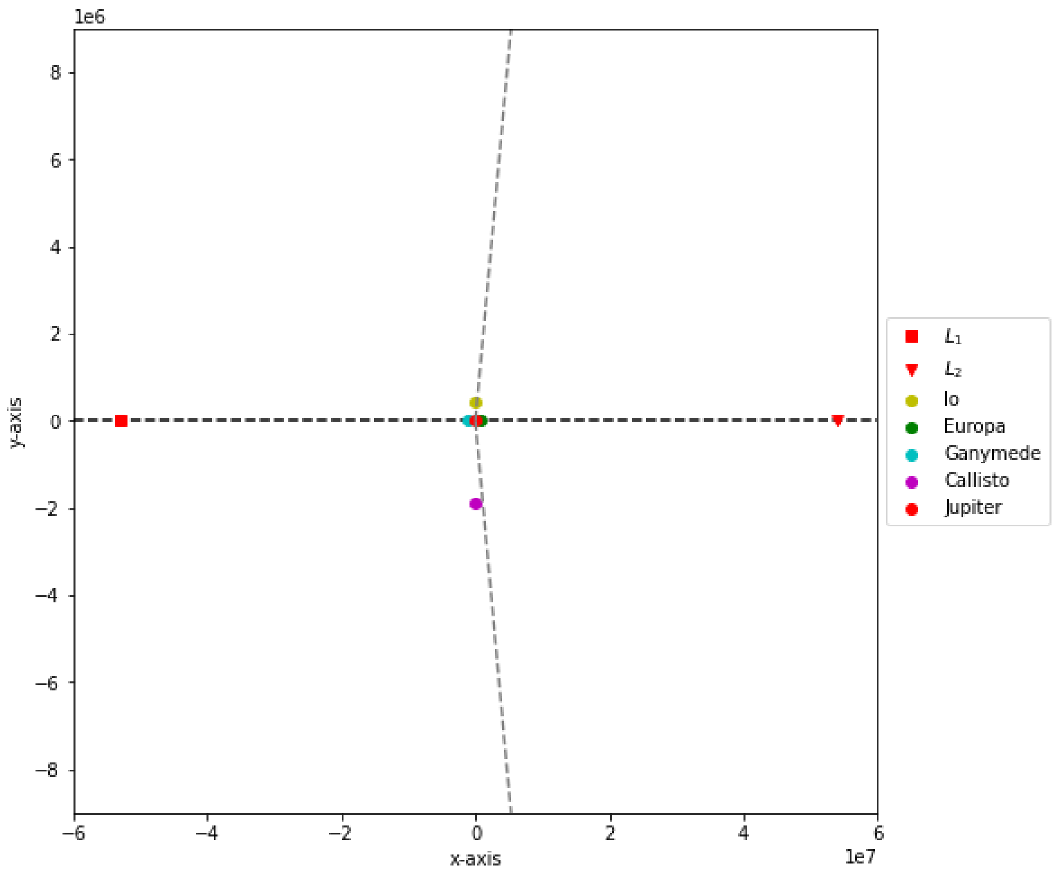

1e8
8
6
4
2
0
−2
−4
−6
−8
y-axis
−8
−6
−4
−2
0
2
4
6
8
x-axis
1e8
L_1
L_2
L_3
L_4
L_4
Center of Mass
Sun
Jupiter

In [125]:
```
figure, ax = plt.subplots(figsize=(8,8))
ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.axhline(0, color='k', linestyle = "--")

# set axis limits
ax.set_xlim((-9e8, 9e8))
ax.set_ylim((-9e8, 9e8))

# orbit trail
orbit = plt.Circle((0, 0), 7.78e8, color='b', fill=False, linestyle = "--")

# plot the sun (not to scale), moved more in the x-axis to show sun is not at the ori
sun = plt.Circle((-8e6, 0), 7e7, color='gold', fill=True, label = "Sun")

# plot Jupiter (not to scale)
Jupiter = plt.Circle((d_jupiter, 0), 2e7, color = "orange", fill = True, label = "Jup

# add the lagrange points
ax.plot(L1[0], L1[1], 'rs', label="$L_1$")
ax.plot(L2[0], L2[1], 'rv', label="$L_2$")
ax.plot(L3[0], L3[1], 'r^', label="$L_3$")
ax.plot(L4[0], L4[1], 'g^', label="$L_4$")
ax.plot(L5[0], L5[1], 'gv', label="$L_4$")
ax.plot(7.78e8 - 4.22e5, 0, 'yo', label = "Io")
ax.plot(7.78e8 - 6.71e5, 0, 'oc', label = "Europa")
ax.plot(7.78e8 - 1.07e6, 0, 'co', label = "Ganymede")
ax.plot(7.78e8 - 1.07e6, 0, 'ko', label = "Ganymede")
ax.plot(7.78e8 - 1.88e6, 0, 'ro', label = "Callisto")

# mark center of mass
ax.plot(0, 0, 'k', marker="*", markersize=10, label = "Center of Mass")

x_half = np.linspace(0, 7.78e8/2, 1000)
y_half = np.linspace(0, 6.74e8/2, 1000)

# plot guidelines for L4 and L5
ax.plot([0, 3.98e8], [0, 6.74e8], linestyle = "--", color = "gray")
ax.plot([3.98e8, 7.78e8], [6.74e8, 0], linestyle = "--", color = "gray")
ax.plot([0, 3.98e8], [0, -6.74e8], linestyle = "--", color = "gray")
ax.plot([3.98e8, 7.78e8], [-6.74e8, 0], linestyle = "--", color = "gray")

# Plot the orbits
ax.add_patch(orbit)
ax.add_patch(sun)
ax.add_patch(Jupiter)

# add legend
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

Out[125]:
```
<matplotlib.legend.Legend at 0x127fe5540>
```


1e8
8
6
4
2
0
−2
−4
−6
−8
y-axis
−8
−6
−4
−2
0
2
4
6
8
x-axis
1e8
L_1
L_2
L_3
L_4
L_4
Io
Europa
Ganymede
Ganymede
Callisto
Center of Mass
Sun
Jupiter

```
In [126]: figure, ax = plt.subplots(figsize=(8,8))
          ax.set_xlabel("x-axis")
          ax.set_ylabel("y-axis")
          ax.axhline(0, color='k', linestyle = "--")

          # set axis limits
          ax.set_xlim((-6e7, 6e7))
          ax.set_ylim((-9e6, 9e6))

          # add the moons and L1/L2 Lagrange Point
          ax.plot(L1[0] - 7.78e8, 0, 'rs', label="$L_1$")
          ax.plot(L2[0] -7.78e8, 0, 'rv', label="$L_2$")
          ax.plot(0, 4.22e5, 'yo', label = "Io")
          ax.plot(6.71e5, 0, 'go', label = "Europa")
          ax.plot(-1.07e6, 0, 'co', label = "Ganymede")
          ax.plot(0, -1.88e6, 'mo', label = "Callisto")

          # plot Jupiter (not to scale)
          ax.plot(0,0, "ro", label = "Jupiter")

          x_half = np.linspace(0, 7.78e8/2, 1000)
          y_half = np.linspace(0, 6.74e8/2, 1000)

          # plot guidelines for L4 and L5
          ax.plot([0, 3.98e8], [0, 6.74e8], linestyle = "--", color = "gray")
          ax.plot([3.98e8, 7.78e8], [6.74e8, 0], linestyle = "--", color = "gray")
          ax.plot([0, 3.98e8], [0, -6.74e8], linestyle = "--", color = "gray")
          ax.plot([3.98e8, 7.78e8], [-6.74e8, 0], linestyle = "--", color = "gray")

          # add legend
          ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

Out[126]: <matplotlib.legend.Legend at 0x128468790>
```



In [127]:
```
# normal newton's method
def newton(f, f_prime, x, parameters, tol):
    while (abs(f(x, parameters)) > tol):
        x = x - (f(x, parameters)/f_prime(x, parameters))
    return (float(x))
```

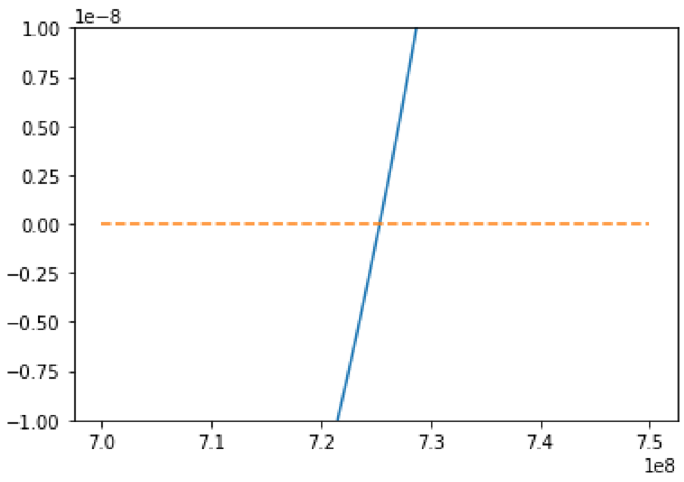
In [128]:
```
rt time
 solve for first 3 Lagrange Points:
se Newton's root findng method, where y=0 so only solve for x
meters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]
f(x, parameters):
G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
output = -((G*m_sun*(x+d_sun))/(((x+d_sun)**2)**(3/2))) - ((G*m_jupiter*(x-d_jupiter)
return(output)

f_prime(x, parameters):
G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
output = ((2*G*m_sun)/((x+d_sun)**2 * abs(x + d_sun))) + ((2*G*m_jupiter)/((x-d_jupite
return(output)

s = [7e8, 8e8, -7e8]
= 1e-20
i in range(len(guess)):
x = guess[i]
# our newton's method
start_time1 = time.time()
L1_newt = newton(f, f_prime, x, parameters, tol)
end_time1 = time.time()
# scipy newton's method
start_time2 = time.time()
L1_scipy = scipy.optimize.fsolve(f, x, parameters)
end_time2 = time.time()

print(f"L{i+1}: ({L1_newt}, 0). Time to calculate: {end_time1 - start_time1}s.")
print(f"L{i+1}: ({L1_scipy[0]}, 0). Time to calculate: {end_time2 - start_time2}s.")
```

```
L1: (725391231.1852783, 0). Time to calculate: 2.5033950805664062e-05s.
L1: (725391231.1852779, 0). Time to calculate: 0.00039887428283691406s.
L2: (831539264.6332624, 0). Time to calculate: 4.315376281738281e-05s.
L2: (831539264.6332624, 0). Time to calculate: 0.0006499290466308594s.
L3: (-778309025.0519384, 0). Time to calculate: 3.0040740966796875e-05s.
L3: (-778309025.0519385, 0). Time to calculate: 0.0004668235778808594s.
```

In [129]:
```
import time
# to solve for first 3 Lagrange Points:
## use Newton's root findng method, where y=0 so only solve for x
parameters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]

# system of equations functions
def f1(v, parameters):
    G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
    x = v[0][0]
    y = v[1][0]
    output = -((G*m_sun*(x+d_sun))/(((x+d_sun)**2 + y**2)**(3/2))) - ((G*m_jupiter*(x
    return(output)

def f2(v, parameters):
    G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
    x = v[0][0]
    y = v[1][0]
    output = -((G*m_sun*(y))/(((x+d_sun)**2 + y**2)**(3/2))) - ((G*m_jupiter*(y))/(((
    return(output)

F = [f1, f2]

# partial derivative functions
def f1_x(v, parameters):
    G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
    x = v[0][0]
    y = v[1][0]
    output = w**2 - ((G*m_sun)/((x+d_sun)**2 + y**2)**(3/2)) - ((G*m_jupiter)/((x-d_j
    return(output)

def f1_y(v, parameters):
    G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
    x = v[0][0]
    y = v[1][0]
    output = (3*G*m_sun*y*(d_sun+x))/(((x+d_sun)**2 + y**2)**(5/2)) - (3*G*m_jupiter*
    return(output)

def f2_x(v, parameters):
    G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
    x = v[0][0]
    y = v[1][0]
    output = (3*G*m_sun*y*(2*d_sun+2*x))/(2*((x+d_sun)**2 + y**2)**(5/2)) - (3*G*m_ju
    return(output)

def f2_y(v, parameters):
    G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
    x = v[0][0]
    y = v[1][0]
    output = w**2 - (G*m_sun)/(((x+d_sun)**2 + y**2)**(3/2)) - (G*m_jupiter/((x-d_jup
    return(output)

Fp = [[f1_x, f1_y], [f2_x, f2_y]]
```

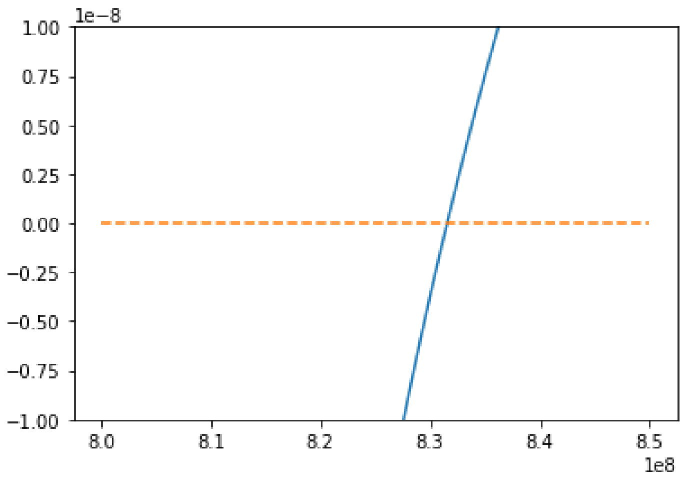
In [130]:
```
# plot points for L1
parameters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]
x = np.linspace(7e8, 7.5e8, 1000)
y = [f(x0, parameters) for x0 in x]
plt.ylim([-1e-8, 1e-8])
plt.plot(x, y)
y_line = [0 for x0 in x]
plt.plot(x, y_line, "--")
```

Out[130]:
```
[<matplotlib.lines.Line2D at 0x1284ef880>]
```

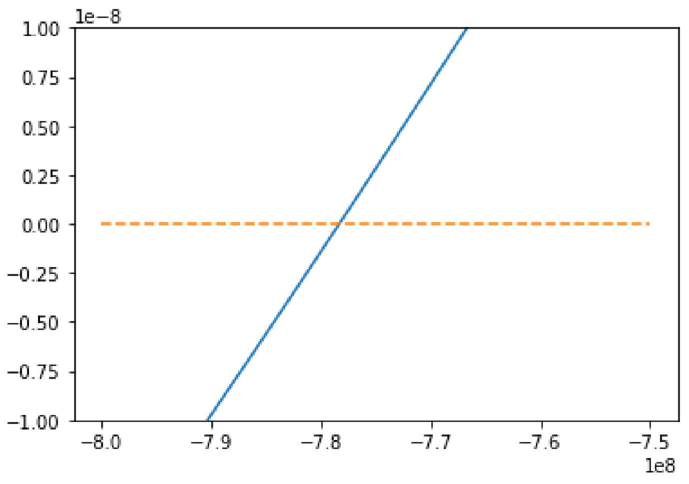


In [131]:
```
# plot points for L2
parameters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]
x = np.linspace(8e8, 8.5e8, 1000)
y = [f(x0, parameters) for x0 in x]
plt.ylim([-1e-8, 1e-8])
plt.plot(x, y)
y_line = [0 for x0 in x]
plt.plot(x, y_line, "--")
```

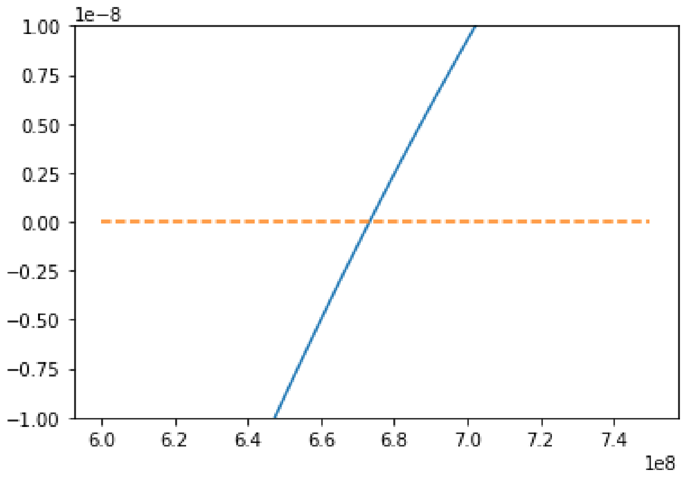
Out[131]:
```
[<matplotlib.lines.Line2D at 0x128805720>]
```



```
In [132]: # plot points for L3
          parameters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]
          x = np.linspace(-8e8, -7.5e8, 1000)
          y = [f(x0, parameters) for x0 in x]
          plt.ylim([-1e-8, 1e-8])
          plt.plot(x, y)
          y_line = [0 for x0 in x]
          plt.plot(x, y_line, "--")

Out[132]: [<matplotlib.lines.Line2D at 0x1289cd150>]
```

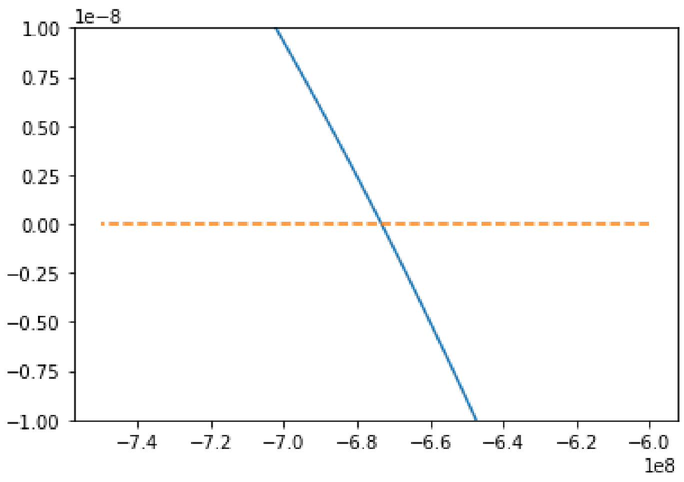


```
In [133]: # plot points for L4
          ## given x0 = 7.78e8 / 2, this plot outputs the expected y-value
          def f_v2(x, y, parameters):
              G, m_sun, d_sun, m_jupiter, d_jupiter, w = parameters
              output = -((G*m_sun*(x+d_sun))/(((x+d_sun)**2 + y**2)**(3/2))) - ((G*m_jupiter*(x
              return(output)

          parameters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]
          x0 = 7.78e8/2
          y = np.linspace(6e8,7.5e8, 100)
          y_value = [f_v2(x0, y0, parameters) for y0 in y]
          plt.plot(y, y_value)
          plt.ylim([-1e-8, 1e-8])
          y_line = [0 for y0 in y]
          plt.plot(y, y_line, "--")

Out[133]: [<matplotlib.lines.Line2D at 0x1273889d0>]
```



In [134]:
```
# plot points for L5
## given x0 = 7.78e8 / 2, this plot outputs the expected y-value

parameters = [G, m_sun, d_sun, m_jupiter, d_jupiter, w]
x0 = 7.78e8/2
y = np.linspace(-6e8,-7.5e8, 100)
y_value = [f_v2(x0, y0, parameters) for y0 in y]
plt.plot(y, y_value)
plt.ylim([-1e-8, 1e-8])
y_line = [0 for y0 in y]
plt.plot(y, y_line, "--")
```

Out[134]:
```
[<matplotlib.lines.Line2D at 0x122a2b2b0>]
```



In [135]:
```
import math
# solve for L4 and L5
x = (dis / 2) * ((m_sun - m_jupiter) / (m_sun + m_jupiter))
y = math.sqrt(3)/2 * dis
L4 = (x, y)
L5 = (x, -y)
```

In [145]:
```
# given that α*(r-r_1) + β*(r - r_2) = r
## prove that L4 points is correct from the simplified derivation
r = np.array([[3.88e8], [6.74e8]])
r1 = np.array([[d_sun], [0]])
r2 = np.array([[d_jupiter], [0]])
alpha = (G*m_sun) / (w**2 * np.linalg.norm(r-r1)**3)
beta = (G*m_jupiter) / (w**2 * np.linalg.norm(r-r2)**3)
test_r = alpha * (r - r1) + beta * (r-r2)
print(alpha)
print(beta)
print(test_r)
print(r)
```

```
1.0016245555942398
0.0009520776346815713
[[3.87516858e+08]
 [6.75736651e+08]]
[[3.88e+08]
 [6.74e+08]]
```

In [144]:
```
output = (3.87516858e+08 - 3.88e+08) / 3.88e+08
print(output)
output = (6.75736651e+08 - 6.74e+08) / 6.74e+08
print(output)
```

```
-0.0012452113402061855
0.002577663353115727
```

In [137]:
```
# create a jacobian to test stability
def jacobian(F, Fp, v, parameters, tol):
    jacobian = np.array([[Fp[0][0](v, parameters), Fp[0][1](v, parameters)],[Fp[1][0]
    return (jacobian)
```

In [138]:
```
# L1 and L2 stability (two ways)
# with w = 1.6798160267387252e-08
real = w * math.sqrt(1 + 2*math.sqrt(7))
real2 = -1* w * math.sqrt(1 + 2*math.sqrt(7))
comp = 1.68e-8j * math.sqrt(2*math.sqrt(7) - 1)
comp2 = -1.68e-8j * math.sqrt(2*math.sqrt(7) - 1)

print([real, real2])
print([comp, comp2])
```

```
[4.213460349914476e-08, -4.213460349914476e-08]
[3.4802782935704155e-08j, -3.4802782935704155e-08j]
```

In [139]:
```
# L3 stability (two ways)
real = w * math.sqrt((3*m_sun)/(8*m_jupiter))
real2 = -w * math.sqrt((3*m_sun)/(8*m_jupiter))
comp = 1.68e-8j * math.sqrt(7)
comp2 = -1.68e-8j * math.sqrt(7)

print([real, real2])
print([comp, comp2])
```

```
[3.330102175939218e-07, -3.330102175939218e-07]
[4.444862202588513e-08j, -4.444862202588513e-08j]
```

In [140]:
```
# L4 and L5 stability (two ways)
## with w/2 = 8.399080133693626e-09
k = (m_sun - m_jupiter) / (m_sun + m_jupiter)
real = 8.4e-9j * math.sqrt(2 - math.sqrt(27*k**2 - 23))
real2 = -8.4e-9j * math.sqrt(2 - math.sqrt(27*k**2 - 23))
comp = 8.4e-9j * math.sqrt(2 + math.sqrt(27*k**2 - 23))
comp2 = -8.4e-9j * math.sqrt(2 + math.sqrt(27*k**2 - 23))

print([real, real2])
print([comp, comp2])
```

```
[1.3513764979138936e-09j, -1.3513764979138936e-09j]
[1.6745560055157487e-08j, -1.6745560055157487e-08j]
```

```
In [141]: # additional proof that L4 is stable
          print(k**2 >= 23/37)

          print(math.sqrt(27*k**2 - 23) <= 2)

          print(m_sun >= 25*m_jupiter*((1 + math.sqrt(1-4/625))/2))
```

```
True
True
True
```