In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from matplotlib.patches import Circle
from matplotlib.animation import FuncAnimation

from scipy.integrate import solve_ivp
```

In [ ]:
```
# some constants

# astronomical unit, gives the mean distance between sun and earth
AU = 1.496e11 # in meters
AU_inv = 1.0 / AU

# gravitational constant
G = 6.6743e-11

# nearest position relativ to the sun
d_perihel = 1.4709e11
v_perihel = 30290.0

# furthest position relativ to the sun
d_amphel = 1.5210e11
v_amphel = 29290.0

M_sun = 1.989e30
R_sun = 6.96342e8

m_earth = 5.9722e24

M_total = M_sun + m_earth
# reduced mass
mu = M_sun * m_earth / M_total

# day in seconds
dis = 24.0 * 60.0 * 60.0
# year in seconds
yis = 365.25 * dis
```

Part a)

The equation of motion (in cartesian coordinates) is given by

$$m\ddot{\mathbf{r}} = -\frac{k}{r^2}\frac{\mathbf{r}}{r},$$

where $k = G\,M_{\text{sun}}\,m_{\text{earth}} = G\,M\,m$. The gravitational constant G is saved in the variable `G`, the total mass M in the variable `M_total` and the reduced mass m in the variable `mu`.

In [ ]:
```
def eq_of_motion(t: float, y_init):
    # y_init contains [x, y, d/dt x, d/dt y]

    # you can use this command to split the input array in the posit
    # and the velocity v = [d/dt x, d/dt y]
    # r, v = np.split(y_init, 2)

    # remove the pass keyword before you start your implementation
    pass

    # you should return [d/dt x, d/dt y, d^2/dt^2 x, d^2/dt^2 y]
```

Part c)

We want to perturb the Kepler potential by a perturbation of the form

$$V_1(r) = \frac{\gamma}{r^2},$$

which gives us a total potential of

$$V(r) = -\frac{k}{r} + \frac{\gamma}{r^2}.$$

Calculate the new system of differential equations for this system ($m\ddot{\mathbf{r}} = \ldots$) and implement it. A "good" value for γ is already given via the variable `gamma`.

In [ ]:
```
def eom_perturbed(t: float, y_init):
    # y_init contains [x, y, d/dt x, d/dt y]

    # you can use this command to split the input array in the posit
    # and the velocity v = [d/dt x, d/dt y]
    # r, v = np.split(y_init, 2)

    # remove the pass keyword before you start your implementation
    pass

    # you should return [d/dt x, d/dt y, d^2/dt^2 x, d^2/dt^2 y]
```

Part b)

The analytic solution for the radius of Kepler orbit is given by

$$r(\varphi) = \frac{p}{1 + \varepsilon\cos(\varphi)}, \quad \text{where} \quad p = \frac{\ell^2}{m\,k} \quad , \varepsilon = \sqrt{1 + \frac{2E\ell^2}{mk^2}}.$$

Since the input variables `r0` and `v0` are in cartesian representation, you can calculate the the angular momentun as

$$\boldsymbol{\ell} = \mathbf{r} \times m\mathbf{v}.$$

(Hint: You only need the z component of the cross product, which is equal to $|\boldsymbol{\ell}|$). The energy is calculated via

$$E = \frac{1}{2}m\mathbf{v}^2 + V(\sqrt{x^2 + y^2}).$$

In [ ]:
```
def radius_elliptic_orbit(r0, v0, phi):
    # remove the pass keyword before you start your implementation
    pass
```

In [ ]:
```
# set up the system

r_init = np.array([d_perihel, 0.0])
v_init = np.array([0.0, 0.5 * v_perihel])

y_init = np.concatenate((r_init, v_init))

t_span = (0.0, 5.0 * yis)
dt = 0.001 * dis
t_eval = np.arange(0.0, t_span[-1], dt)
```

In [ ]:
```
# calculate the solution of our differential equation system
sol = solve_ivp(eq_of_motion, t_span, y_init, t_eval=t_eval, dense_o

x, y = sol.y[:2, :]
```

This cell plots the result of **part a)**:

In [ ]:
```
fig, ax = plt.subplots()
ax.set_aspect('equal')

# add the sun as a circle, it is enlarged 5 times for better visibil
c_sun = Circle(xy=(0.0, 0.0),
               radius=5.0 * R_sun * AU_inv,
               facecolor='#FFFF00')
ax.add_patch(c_sun)


ax.plot(x * AU_inv, y * AU_inv, '--')
```

This cell adds the analytic solution of **part b)** to the plot, so you can check if your implementation of the differential equations is correct:

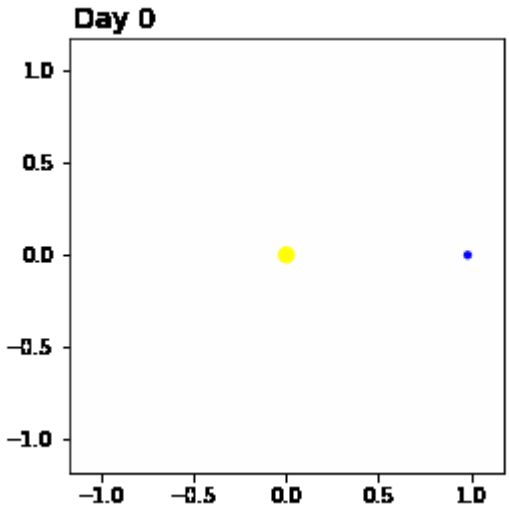
In [ ]:
```
phi = np.linspace(0.0, 2.0 * np.pi, 5000)
r_ellipse = radius_elliptic_orbit(r_init, v_init, phi)

# convert the polar coordinates to cartesi
x_ellip = [-AU_inv * r * np.cos(p) for r, p in zip(r_ellipse, phi)]
y_ellip = [ AU_inv * r * np.sin(p) for r, p in zip(r_ellipse, phi)]

ax.plot(x_ellip, y_ellip, '--r')

# display the figure again, this time with the ellipse added
fig
```

Now we add the perturbed orbit of **part c)** to the plot:

In [ ]:
```
# calculate the solution of the perturbed equation of motion
sol_p = solve_ivp(eom_perturbed, t_span, y_init, t_eval=t_eval, dens

x_p, y_p = sol_p.y[:2, :]

ax.plot(x_p * AU_inv, y_p * AU_inv, '--k', lw=2)

# display the figure again, this time with the perturbed orbit added
fig
```

To finish, we do a little animation of the perturbed orbit and save it in the file `perturbed_orbit.gif`. Have fun :)

In [ ]:
```
# animate the perturbed system
figa, axa = plt.subplots()
skip_frames = int(dis / dt)
x_ani = x_p[::skip_frames] * AU_inv
y_ani = y_p[::skip_frames] * AU_inv

num_frames = len(x_ani)

axa.set_aspect('equal')
ax_lim = 1.2 * max(max(np.amax(x_ani), np.amax(y_ani)),
                   abs(max((np.amin(x_ani), np.amin(y_ani)))))
axa.set_xlim(-ax_lim, ax_lim)
axa.set_ylim(-ax_lim, ax_lim)

earth_orbit, = axa.plot(x_ani[0], y_ani[0], 'k', lw=2.0)

# radius chosen for better visibility
c_earth = Circle(xy=(x_ani[0], y_ani[0]),
                 radius=5.0 * R_sun * AU_inv,
                 facecolor='b')
axa.add_patch(c_earth)

c_sun = Circle(xy=(0.0, 0.0),
               radius=10.0 * R_sun * AU_inv,
               facecolor='#FFFF00')
axa.add_patch(c_sun)

text_day = axa.text(0.01, 1.02, 'Day 0', fontsize=14, transform=axa.

def frame_update(frame: int):
    if (frame % 500 == 0):
        print('>>> Frame {:d} of {:d}'.format(frame, num_frames))
    earth_orbit.set_xdata(x_ani[:frame])
    earth_orbit.set_ydata(y_ani[:frame])
    c_earth.set_center((x_ani[frame], y_ani[frame]))
    text_day.set_text('Day {:d}'.format(frame))
    return (earth_orbit, c_earth, text_day)

ani = FuncAnimation(figa, frame_update, frames=num_frames, interval=
ani.save('perturbed_orbit.gif', fps=60)

plt.close(figa)
```


Day 0
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0