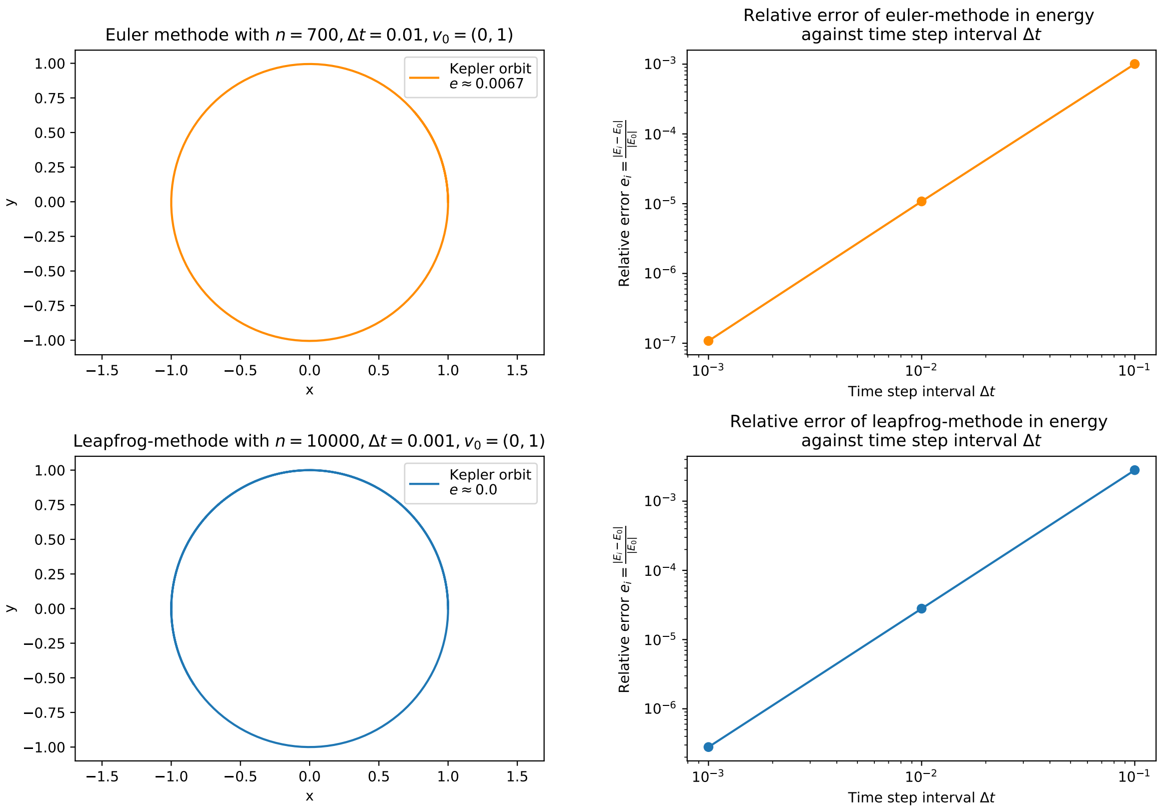
Exercises sheet 1

by Maximilian Richter and Christian Heppe

Exercise 5

a) (Orange) and b) (Blue)



In order to test the euler and the leapfrog methode, we evaluated the energy and compared the final energy with the initial energy for 3 different eccentricites (in our case we did that in a for-loop with different steps and timeintervalls (stepsizes)).

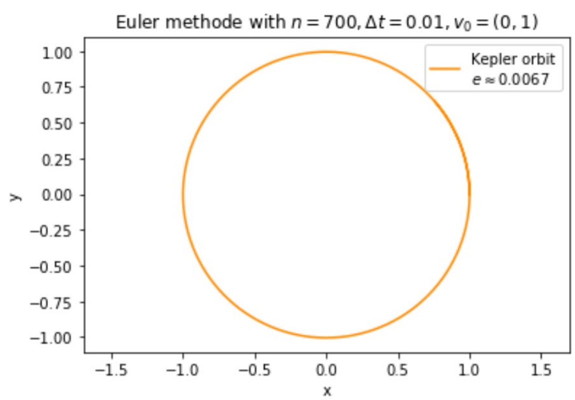
Both methodes satisfy our expectations pretty good, as expected, the relative error of the energy vanishes exponentially as we decrease the stepsize Δt logarithmically. For further details see the python-code in the appendix.

In [87]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

#Create global arrays for error-plot
dE = np.zeros(3)
time = np.zeros(3)
```

In [88]:
```
#Defining function to do step-by-step euler integration
#n = steps
#x = initial x coord.
#y = initial y coord.
#w_x = initial x velocity
#w_y = initial y velocity
#h = stepsize/timeinterval
#title = name for saving the figure
def euler(n,x,y,w_x,w_y,h,title):

    #Create empty arrays in order to save the trajectory coordinates
    X = np.zeros(n)
    Y = np.zeros(n)

    #For-loop for integrating
    for i in range(n):

        r = np.sqrt(x**2+y**2) #Distance between the two bodies

        X[i] = x #save value to array
        w_x = w_x - x/(r)**3 * h #calculate new velocity
        x = x + w_x * h #calculate new coordinate

        Y[i] = y #save value to array
        w_y = w_y - y/(r)**3 * h #calculate new velocity
        y = y + w_y * h #calculate new coordinate

        #Calculate eccentricity
        s = np.array([x,y,0]) #3D position vector to do cross-product
        w = np.array([w_x,w_y,0]) #3D velocity vector
        e = np.cross(w,np.cross(s,w))-s #Runge-Lenz-vector
        if i==n-1: print("Eccentrity ~",np.round(np.linalg.norm(e),6)) #Print value

        #Calculate energy
        E = (w_x**2+w_y**2)/2+1/np.sqrt(x**2+y**2) #Energy
        e_i=np.abs(E-E_0)/np.abs(E_0) #Relative error
        if i==n-1: print("Energy =",E,", e_i =",e_i) #Print out

    #Plot
    plt.plot(X,Y, color="darkorange", label="Kepler orbit\n$e\\approx${}".format(np.round(
np.linalg.norm(e),4)))
    plt.title("Euler methode with $n={}, \\Delta t={}, v_0=(0,{})$".format(n,h,w_0))
    plt.axis("equal")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.savefig(title)
```

In [89]:
```
#Initializing values
title = "euler.pdf"
n = 700
x = 1
y = 0
w_x = 0
w_y = 1
h = 0.01
e=0 #Intialize eccentricity
w_0=w_y #Initialize velocity

#Calculate initial Energy
E_0 = (w_x**2+w_y**2)/2+1/np.sqrt(x**2+y**2)
print("Initial energy E_0 =",E_0)

euler(n,x,y,w_x,w_y,h,title)
```

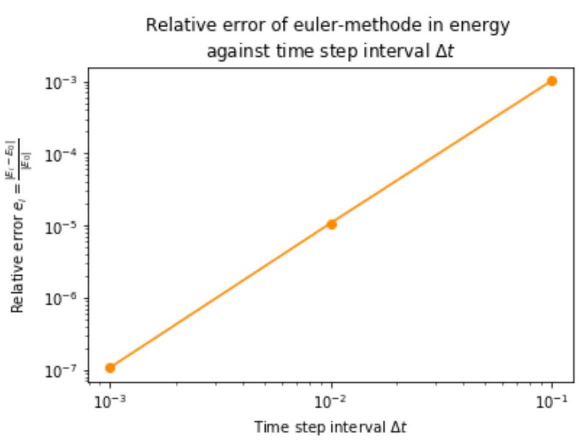
```
Initial energy E_0 = 1.5
Eccentrity ~ 0.006688
Energy = 1.5065798384798426 , e_i = 0.004386558986561706
```



In [90]:
```
#Function to calculate relative error in energy
#t = loop-variable
def error(n,x,y,w_x,w_y,h,t):

    for i in range(n):
        #Step-by-step euler
        r = np.sqrt(x**2+y**2)

        w_x = w_x - x/(r)**3 * h
        x = x + w_x * h

        w_y = w_y - y/(r)**3 * h
        y = y + w_y * h

        #Eccentricity
        s = np.array([x,y,0])
        w = np.array([w_x,w_y,0])
        e = np.cross(w,np.cross(s,w))-s
        if i==n-1: print("Eccentrity ~",np.round(np.linalg.norm(e),6))

        #Energy
        E = np.linalg.norm(w)**2/2+1/np.linalg.norm(w)
        e_i=np.abs(E-E_0)/np.abs(E_0)
        if i==n-1: print("Energy =",E,", e_i =",e_i)
        dE[t] = e_i #Save energy in an array
        time[t] = h #Save stepsize in an array
```

In [91]:
```
#Initializing values
n=70
w_y=1
h=0.1

#Iterate over different steps and stepsizes
for t in range(3):
    E_0 = np.linalg.norm(np.array([w_x,w_y]))**2/2+1/np.linalg.norm(np.array([x,y]))
    print("Initial energy E_0 =",E_0)
    error(n*10**t,x,y,w_x,w_y,h*10**-t,t)

#Print values
print(time,dE)

#Plot
plt.plot(time,dE, "o-", color="darkorange")
plt.xscale("log")
plt.yscale("log")
plt.title("Relative error of euler-methode in energy \nagainst time step interval $\\Delta t$")
plt.xlabel("Time step interval $\\Delta t$")
plt.ylabel("Relative error $e_i=\\frac{\\|E_i-E_0\\|}{\\|E_0\\|}$")
plt.savefig("error_euler.pdf")
```

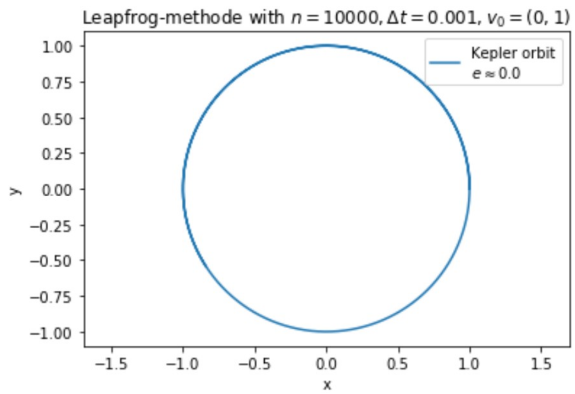
```
Initial energy E_0 = 1.5
Eccentrity ~ 0.065072
Energy = 1.501506931353755 , e_i = 0.001004620902503343
Initial energy E_0 = 1.5
Eccentrity ~ 0.006688
Energy = 1.5000161773479164 , e_i = 1.0784898610912327e-05
Initial energy E_0 = 1.5
Eccentrity ~ 0.000668
Energy = 1.5000001618634093 , e_i = 1.0790893956169612e-07
[0.1   0.01  0.001] [1.00462090e-03 1.07848986e-05 1.07908940e-07]
```



In [92]:
```
#Leapfrog methode
#n = steps
#x0 = initial x coord.
#y0 = initial y coord.
#v_x0 = initial x velocity
#v_y0 = initial y velocity
#dt = stepsize/time-interval
def leapfrog(n,x0,y0,v_x0,v_y0,dt):

    #Empty arrays
    x = np.zeros(n)
    y = np.zeros(n)
    v_x = np.zeros(n)
    v_y = np.zeros(n)
    E = np.zeros(n)
    r = np.zeros(n)

    #Giving inital values
    x[0] = x0
    y[0] = y0
    v_x[0] = v_x0
    v_y[0] = v_y0

    #Leapfrog-implementation
    for i in range(n-1):

        #Distance between the bodies
        r[i] = np.sqrt(x[i]**2+y[i]**2)

        #Calculate new coordinates
        x[i+1] = x[i]+v_x[i]*dt-0.5*x[i]/r[i]**3*dt**2
        y[i+1] = y[i]+v_y[i]*dt-0.5*y[i]/r[i]**3*dt**2

        #calculate new distance
        r[i+1] = np.sqrt(x[i+1]**2+y[i+1]**2)

        #calculate new velocity
        v_x[i+1] = v_x[i]+0.5*(-x[i]/r[i]**3-x[i+1]/r[i+1]**3)*dt
        v_y[i+1] = v_y[i]+0.5*(-y[i]/r[i]**3-y[i+1]/r[i+1]**3)*dt

        #Calculate energy
        E[i] = (v_x[i]**2+v_y[i]**2)-1/np.sqrt(x[i]**2+y[i]**2)

        #Calculate eccentricity
        s = np.array([x[i],y[i],0])
        w = np.array([v_x[i],v_y[i],0])
        e = np.cross(w,np.cross(s,w))-s
        if i==n-2: print("Eccentrity ~",np.round(np.linalg.norm(e),6))

    #Plot
    plt.plot(x,y, label="Kepler orbit\n$e\\approx${}".format(np.round(np.linalg.norm(e),4)
))
    plt.title("Leapfrog-methode with $n={}, \\Delta t={}, v_0=(0,{})$".format(n,dt,v_y0))
    plt.axis("equal")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.savefig("leap.pdf")
```

In [96]:
```
#Initializing values
n = 10000
x0 = 1
y0 = 0
v_x0 = 0
v_y0 = 1
dt = 0.001

leapfrog(n,x0,y0,v_x0,v_y0,dt)
```

```
Eccentrity ~ 1e-06
```



In [94]:
```
#Function to calculate error in leapfrog-methode
def error_leap(n,x0,y0,v_x0,v_y0,dt):

    #Empty arrays
    x = np.zeros(n)
    y = np.zeros(n)
    v_x = np.zeros(n)
    v_y = np.zeros(n)
    E = np.zeros(n)
    r = np.zeros(n)

    #Setting initial values
    x[0] = x0
    y[0] = y0
    v_x[0] = v_x0
    v_y[0] = v_y0

    #Leapfrog
    for i in range(n-1):
        E[i] = (v_x[i]**2+v_y[i]**2)/2-1/np.sqrt(x[i]**2+y[i]**2)

        r[i] = np.sqrt(x[i]**2+y[i]**2)

        x[i+1] = x[i]+v_x[i]*dt-0.5*x[i]/r[i]**3*dt**2
        y[i+1] = y[i]+v_y[i]*dt-0.5*y[i]/r[i]**3*dt**2

        r[i+1] = np.sqrt(x[i+1]**2+y[i+1]**2)

        v_x[i+1] = v_x[i]+0.5*(-x[i]/r[i]**3-x[i+1]/r[i+1]**3)*dt
        v_y[i+1] = v_y[i]+0.5*(-y[i]/r[i]**3-y[i+1]/r[i+1]**3)*dt

    dE[s]=np.abs(E[n-2]-E[0])/np.abs(E[0]) #Calculate relative error and save to array
    time[s]=dt #save timestep to array
```

In [95]:
```
#Iterate for different steps and stepsize
for s in range(3):
    error_leap(100*10**s,1,0,0,1.2,0.1*10**-s)

#Print out values
print(time)
print(dE)

#Plot
plt.plot(time,dE,"o-")
plt.xscale("log")
plt.yscale("log")
plt.title("Relative error of leapfrog-methode in energy \nagainst time step interval $\\De
lta t$")
plt.xlabel("Time step interval $\\Delta t$")
plt.ylabel("Relative error $e_i=\\frac{\\|E_i-E_0\\|}{\\|E_0\\|}$")
plt.savefig("error_leap.pdf")
```

```
[0.1   0.01  0.001]
[2.80696651e-03 2.80006008e-05 2.80009355e-07]
```