hw2_ex16

February 6, 2020

[2]:
```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  5 10:04:27 2020

@author: akswa
"""

# pendul - Program to compute the motion of a simple pendulum
# using the Euler or Verlet method

# Select the numerical method to use: Euler or Verlet


import numpy as np
import matplotlib.pyplot as plt
from scipy.special import ellipk



def period_pend(theta0,g_over_L):
# function to return the exact period for a pendulum of length L
# usage: period = exact_period(theta0,g_over_L)
# where: theta0 = inital angle in degrees
#         g_over_L = ratio g to the length of the pendulum
#         note  -earlier version has a bug as it x sqrt(g/l) not divided 9/11

# note the squaring of the argument in the elliptic function
# matlab uses a different normalization than the book

    period = 4/np.sqrt(g_over_L)*ellipk((np.sin(theta0*np.pi/180./2.))**2)
    return period


def pend(theta0,tau,nstep,NumericalMethod,plotting = False,verbose = False):
    """
    Describes and plots the motion of a simple pendulum,
    with options for several different numerical solvers
```

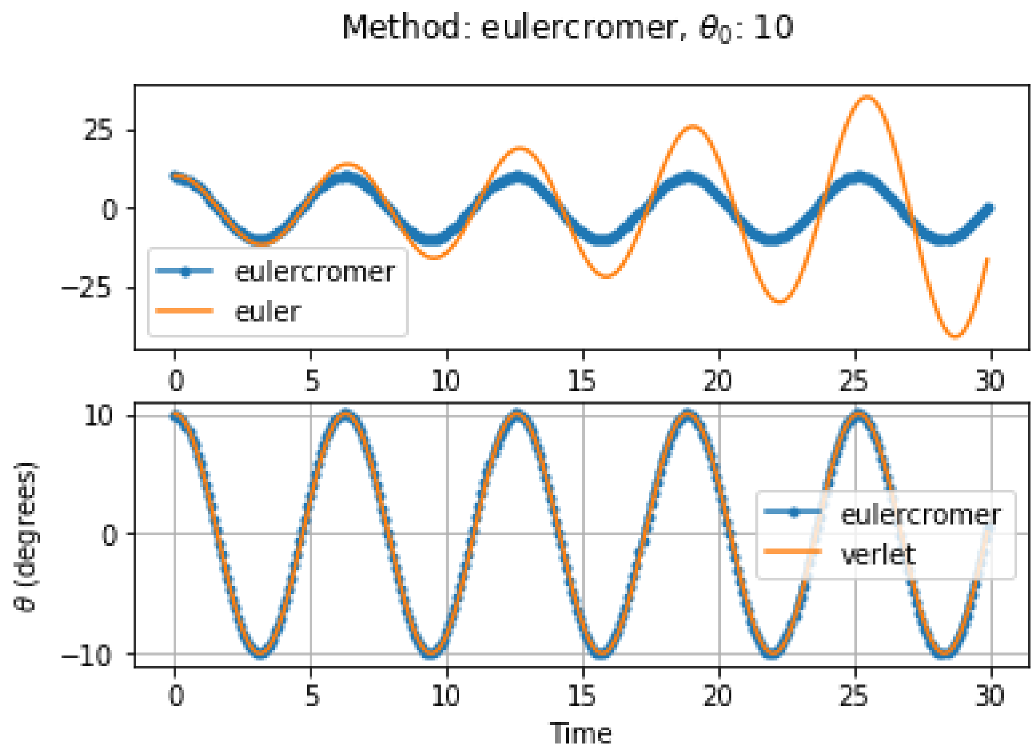
```
    Parameters
    ----------
    theta0 : Float
        Intitial Angle of pendulum.
    tau : Float
        Timestep size (seconds).
    nstep : Int
        Number of timesteps.
    NumericalMethod : string
        Type of numerical method.
        Options: "euler", "verlet", "eulercromer", "leapfrog", "midpoint"
    plotting : Bool, optional
        Toggles Plotting. The default is False.
    verbose : Bool, optional
        Toggles Informative Text output. The default is False.

    Raises
    ------
    SyntaxError
        If NumericalMethod input is an unsupported string.

    Returns
    -------
    t_plot : numpy array
        Array of time to plot with th_plot.
    th_plot : numpy array
        Array of values of theta.
    period : numpy array
        Records time of each complete period.

    """

    # Set initial position and velocity of pendulum
    theta = theta0*np.pi/180  # Convert angle to radians
    omega = 0                 # Set the initial velocity
    # Set the physical constants and other variables
    g_over_L = 1              # The constant g/L
    time = 0                  # Initial time
    irev = 0                  # Used to count number of reversals

    # Take one backward step to start Verlet
    accel = -g_over_L*np.sin(theta)    # Gravitational acceleration
    theta_old = theta - omega*tau + 0.5*tau**2*accel

    # Loop over desired number of steps with given time step
    #    and numerical method
```

```
    #nstep = int(input("Enter number of time steps: "))
    # initialize arrays
    t_plot=np.array([])
    th_plot=np.array([])
    period=np.array([])

    for istep in range(0,nstep):

        # Record angle and time for plotting
        t_plot = np.append(t_plot,time)
        th_plot = np.append(th_plot,theta*180/np.pi)   # Convert angle to␣
↪degrees
        time = time + tau

        # Compute new position and velocity using Euler or Verlet method
        accel = -g_over_L*np.sin(theta);    # Gravitational acceleration
        if NumericalMethod == "euler":
            if istep == 0 and verbose: print("using euler method")
            theta_old = theta               # Save previous angle
            theta = theta + tau*omega       # Euler method
            omega = omega + tau*accel

        elif NumericalMethod == "verlet":
            if istep == 0 and verbose: print("using verlet method")
            theta_new = 2*theta - theta_old + tau**2*accel
            theta_old = theta                                   # Verlet method
            theta = theta_new

        elif NumericalMethod == "eulercromer":
            if istep == 0 and verbose: print("using eulercromer method")
            theta_old = theta               # Save previous angle
            omega = omega + tau*accel       # Euler-cromer method
            theta = theta + tau*omega

        elif NumericalMethod == "leapfrog":
            if istep == 0: # On first step, use euler method to get first omega
                if verbose: print("using leapfrog method")
                omega_old = omega
                omega = omega_old+(1/2)*tau*accel
            theta_old = theta
            omega_new = omega_old + tau*accel # Leapfrog method (formulated as␣
↪half-step method)
            theta = theta_old + tau*omega_new
            omega_old = omega_new

        elif NumericalMethod == "midpoint":
            if istep == 0 and verbose: print("using midpoint method")
```

```
            theta_old = theta               # Save previous angle and velocity
            omega_old = omega

            omega = omega + tau*accel
            theta = theta + (1/2)*(omega+omega_old)*tau       # Midpoint method

        else:
            raise SyntaxError("Numerical method name entered is not a valid␣
↪choice")


        # Test if the pendulum has passed through theta = 0;
        #    if yes, use time to estimate period
        if theta*theta_old < 0: # Test position for sign change
            if verbose:
                print("Turning point at time t= %f" %time) ;
            if irev == 0:           # If this is the first change,
                time_old = time     # just record the time
            else:
                period = np.append(period,2*(time - time_old))
                time_old = time
            irev = irev + 1       # Increment the number of reversals

    if verbose:
        if irev > 1:
            # Estimate period of oscillation, including error bar
            AvePeriod = np.mean(period)
            ErrorBar = np.std(period)/np.sqrt(irev)
            print("Average period = %g +/- %g" %(AvePeriod,ErrorBar))
        else:
            print('Pendulum program could not complete a period, time =%g'%time)

        print("Exact period = %g" %period_pend(theta0,g_over_L))

    # Graph the oscillations as theta versus time
    if plotting:
        fig,ax = plt.subplots(2,1) # Make subplots for comparing each method to␣
↪euler and verlet methods
        ax[0].plot(t_plot,th_plot,'.-')
        ax[1].plot(t_plot,th_plot,'.-')

        # Compare with Euler and Verlet Methods
        t_plot_e,th_plot_e,period_e = pend(theta0,tau,nstep,'euler')
        t_plot_v,th_plot_v,period_v = pend(theta0,tau,nstep,'verlet')

        ax[0].plot(t_plot_e,th_plot_e,'-')
```

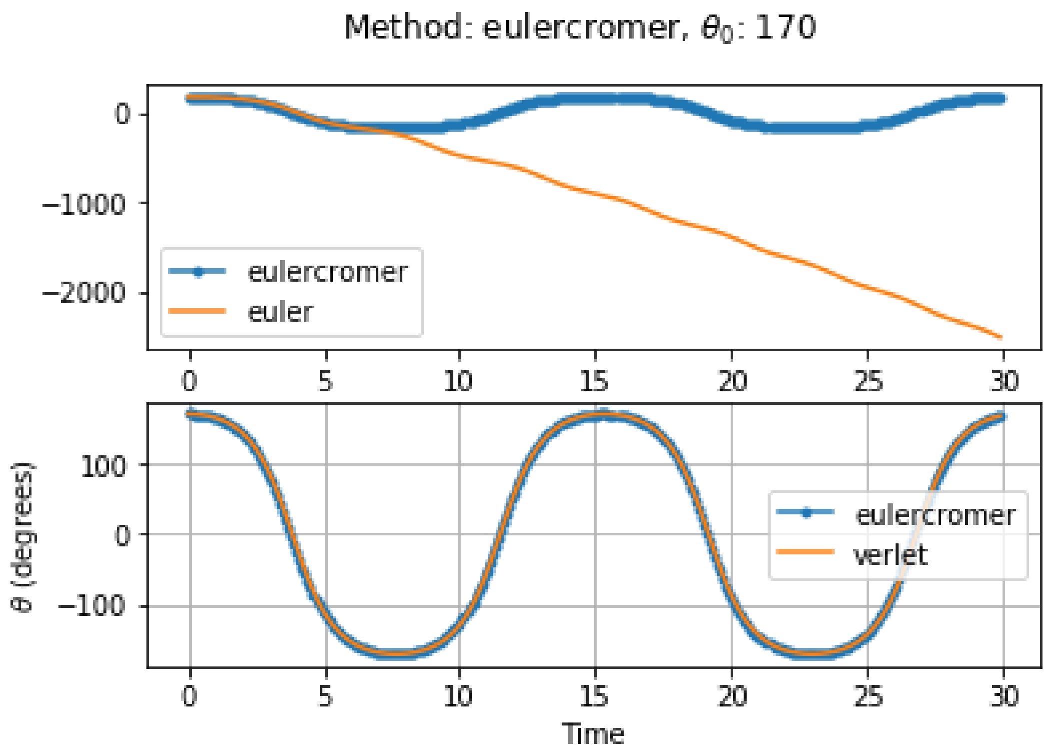
```
        ax[1].plot(t_plot_v,th_plot_v,'-')
        ax[0].legend([NumericalMethod,"euler"])
        ax[1].legend([NumericalMethod,"verlet"])

        fig.suptitle(r"Method: %s, $\theta_0$: %s" % (NumericalMethod,theta0) )
        plt.xlabel('Time')
        plt.ylabel(r'$\theta$ (degrees)') # the 'r' means raw strings for latex
        plt.grid()
        plt.show()

    return t_plot,th_plot,period


if __name__ == "__main__":
    # Part a
    # Figure 2.7
    print("Part a: Fig 2.7")
    a,b,c = pend(10,.1,300,"eulercromer",plotting = True)

    # Figure 2.8
    print("Part a: Fig 2.8")
    a,b,c = pend(170,.1,300,"eulercromer",plotting = True)

    # Part b
    # Figure 2.7
    print("Part b: Fig 2.7")
    a,b,c = pend(10,.1,300,"leapfrog",plotting = True)

    # Figure 2.8
    print("Part b: Fig 2.8")
    a,b,c = pend(170,.1,300,"leapfrog",plotting = True)

    # Part c
    # Figure 2.7
    print("Part c: Fig 2.7")
    a,b,c = pend(10,.1,300,"midpoint",plotting = True)

    # Figure 2.8
    print("Part c: Fig 2.8")
    a,b,c = pend(170,.1,300,"midpoint",plotting = True)
```

```
Part a: Fig 2.7
```



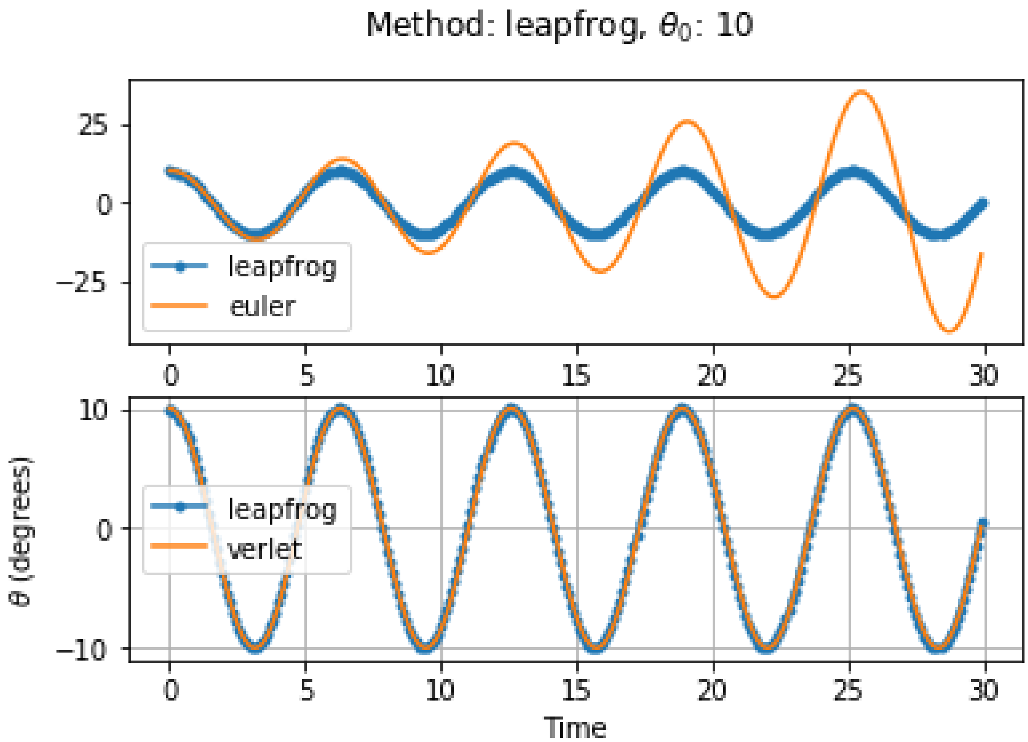
Method: eulercromer, θ_0: 10
eulercromer
euler
eulercromer
verlet
θ (degrees)
Time

Part a: Fig 2.8

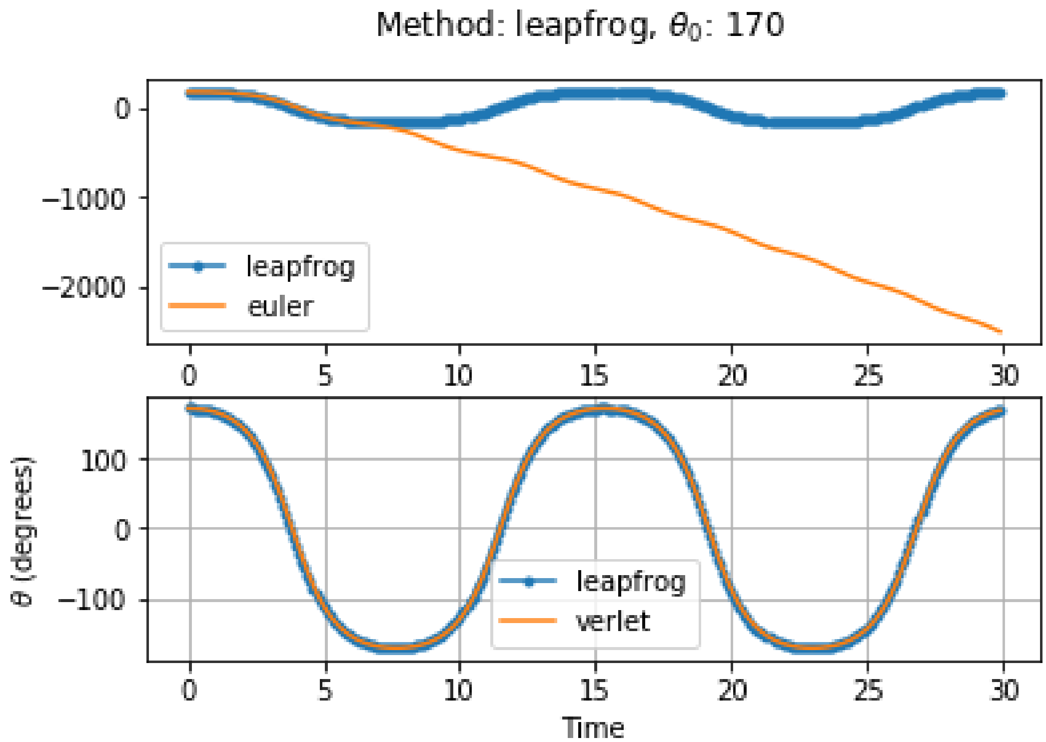


Method: eulercromer, θ_0: 170
0
−1000
−2000
eulercromer
euler
0 5 10 15 20 25 30
100
0
−100
θ (degrees)
eulercromer
verlet
0 5 10 15 20 25 30
Time

Part b: Fig 2.7

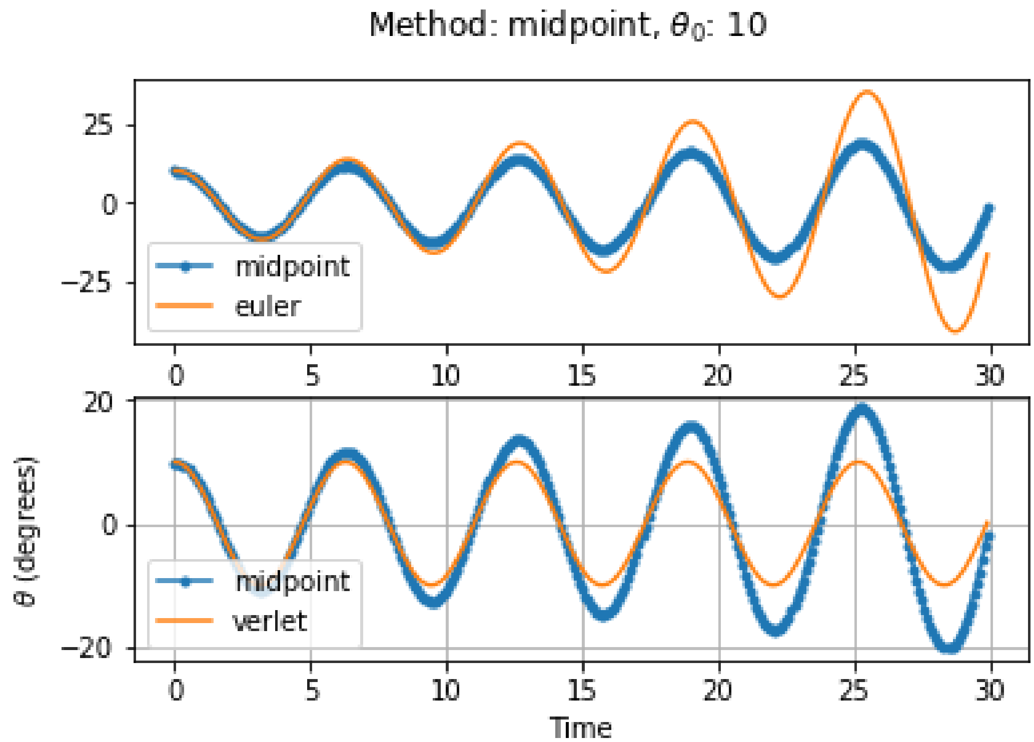

Method: leapfrog, θ_0: 10
leapfrog
euler
leapfrog
verlet
θ (degrees)
Time
25
0
−25
10
0
−10
0
5
10
15
20
25
30

Part b: Fig 2.8

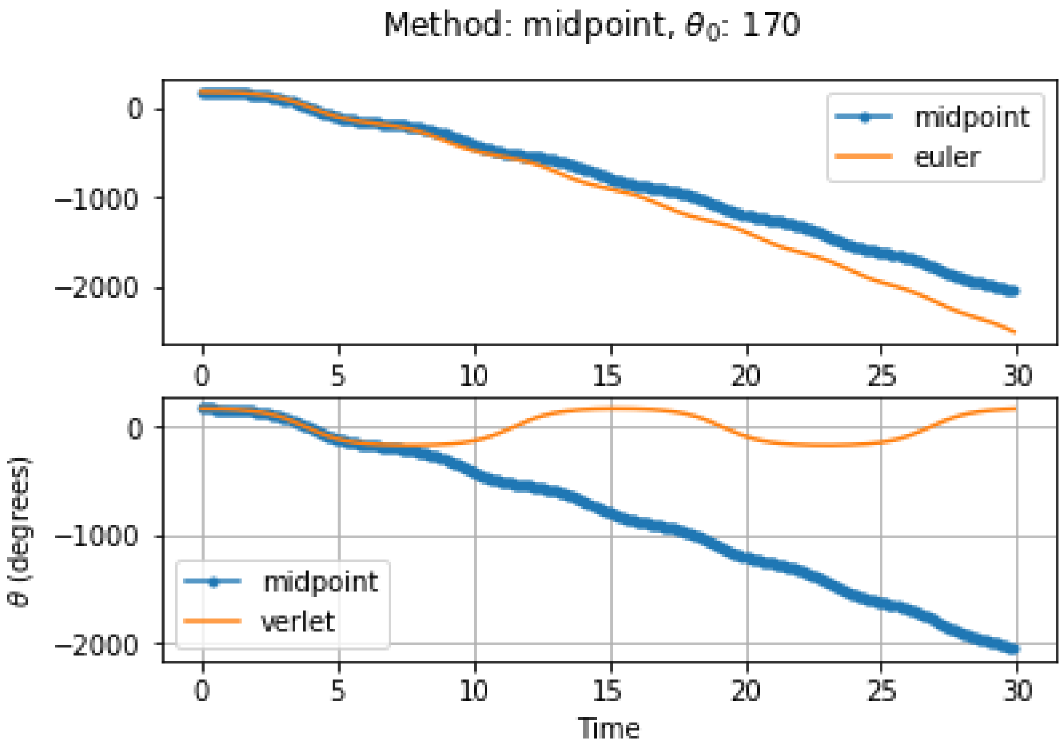


Method: leapfrog, θ_0: 170
0
−1000
−2000
leapfrog
euler
0
5
10
15
20
25
30
100
0
−100
θ (degrees)
leapfrog
verlet
Time

Part c: Fig 2.7


Method: midpoint, θ_0: 10
25
0
−25
midpoint
euler
0
5
10
15
20
25
30
20
0
−20
θ (degrees)
midpoint
verlet
Time

Part c: Fig 2.8


Method: midpoint, θ_0: 170
midpoint
euler
midpoint
verlet
θ (degrees)
Time
0
−1000
−2000
0
5
10
15
20
25
30

```
[ ]:
```