Ph20 Lab 4: Makefiles

Neil McBlane: 2050386

February 18, 2017

Euler Code

```
import numpy as np
import matplotlib.pyplot as plt

#Hello, git!

#Takes the current value of x, v
#and the step size to return the
#updated x value
def x_step(x_i, v_i, h, method):
    if (method=="Implicit"):
        return (x_i + h*v_i)/(1 + h**2)
    elif (method=="Explicit"):
        return x_i + h*v_i
    elif (method=="Symplectic"):
        return x_i + h*v_i

#Takes the current value of x, v
#and the step size to return the
#updated v value
def v_step(x_i, x_ii, v_i, h, method):
    if (method=="Implicit"):
        return (v_i - h*x_i)/(1 + h**2)
    elif (method=="Explicit"):
        return v_i - h*x_i
    elif (method=="Symplectic"):
        return v_i - h*x_ii

#Takes the initial position x_o and
#initial velocity v_o and performs
#the euler method for N steps of
#size h.
def trajectory_calc(x_o, v_o, h, N, method):
    #implicit initial time of t=0
    ts = np.arange(0, h*(N+1), h)

    #perform the Euler method
    #starting with initial conditions
    xs = np.empty(N+1)
    xs[0] = x_o
    vs = np.empty(N+1)
    vs[0] = v_o
    for i in range(N):
        xs[i+1] = x_step(xs[i],vs[i],h, method)
        vs[i+1] = v_step(xs[i],xs[i+1],vs[i],h, method)

    return (xs, vs, ts)
```

```
#Produces a plot of x and
#v vs t and returns the arrays
def trajectory_plot(x_o, v_o, h, N, method, filename):

    xs, vs, ts = trajectory_calc(x_o, v_o, h, N, method)

    #plot x and v vs t
    fig, ax = plt.subplots(nrows=1, ncols=1)
    ax.plot(ts, xs, label="x(t)")
    ax.plot(ts, vs, label="v(t)")
    ax.set_xlabel("Time")
    ax.set_title("Position and Velocity for the {} Euler Method".format(method))
    ax.legend()
    fig.savefig(filename)
    plt.close(fig)

#Compares the euler estimated
#trajectors of the spring to
#the analytic solution. Assumes
#default initial conditions are
#used.
def error_calc(x_o, v_o, h, N, method):
    #perform euler estimation for comparison
    x_euler, v_euler, ts = trajectory_calc(x_o, v_o, h, N, method)

    #calculate analytic values, implicit is t_o = 0
    x_analytic =      x_o*np.cos(ts) + v_o*np.sin(ts)
    v_analytic = (-1)*x_o*np.sin(ts) + v_o*np.cos(ts)
    Es = x_euler**2 + v_euler**2

    #calculate element-wise errors
    x_errs = x_analytic - x_euler
    v_errs = v_analytic - v_euler
    E_errs = Es - Es[0]

    return (ts, x_errs, v_errs, E_errs)

def error_plot(x_o, v_o, h, N, method, filename, x_plot=True, v_plot=True, E_plot=False):

    ts, x_errs, v_errs, E_errs = error_calc(x_o, v_o, h, N, method)

    #plot x and v errors vs t
    fig, ax = plt.subplots(nrows=1, ncols=1)

    if x_plot:
        ax.plot(ts, x_errs, label="x(t) error")
    if v_plot:
        ax.plot(ts, v_errs, label="v(t) error")
    if E_plot:
        ax.plot(ts, E_errs, label="E(t) error")

    ax.set_xlabel("Time")
    ax.set_title("Error vs Time for the {} Euler Method".format(method))
    ax.legend(loc=2)

    fig.savefig(filename)
    plt.close(fig)

#Plots the relation between the maximal
#error vs the size of step taken for a
```

```
#range of step sizes. Each is performed
#up to the same time.
def trunc_err(x_o, v_o, h_o, t, method, filename):
    #avoid integer division
    h_o = float(h_o)
    t = float(t)

    #create range of step sizes to try
    hs = [h_o/(2.0**n) for n in range(5)]
    errs = []

    #perform error analysis for each step size
    for h in hs:
        #determine the number of steps reqired to reach t
        N = int(t/h)   #integer required for indexing

        #perform the estimation
        x_errs = error_calc(x_o, v_o, h, N, method)[1]

        #store the maximum value
        errs.append(max(x_errs))

    #plot the relationship between error and step size
    fig, ax = plt.subplots(nrows=1, ncols=1)
    ax.plot(hs, errs)
    ax.set_xlabel("Step size")
    ax.set_ylabel("Error")
    ax.set_title("Position Error vs Step Size for the {} Euler Method".format(method))

    fig.savefig(filename)
    plt.close(fig)

#Plot the phase space of all methods
#specified on the same plot.
def phase_space(x_o, v_o, h, N, methods, filename):
    #initialise the plot
    fig, ax = plt.subplots(nrows=1, ncols=1)

    #perform the estimation for each method
    for method in methods:
        xs, vs, ts = trajectory_calc(x_o, v_o, h, N, method)
        ax.plot(xs, vs, label=method)

    #determine the analytic solution
    xs =      x_o*np.cos(ts) + v_o*np.sin(ts)
    vs = (-1)*x_o*np.sin(ts) + v_o*np.cos(ts)
    #plot this too
    ax.plot(xs, vs, label="Analytic")

    ax.set_xlabel("Position")
    ax.set_ylabel("Momentum")
    ax.set_title("Phase Space for all Euler Methods")
    ax.legend()

    fig.savefig(filename)
    plt.close(fig)


#Plot the last few waves of a
#long running simulation of the
```

```
#symplectic method vs the analytic
#method to view the phase lag
def phase_lag(x_o, v_o, h, N, filename):
    x_euler, v_euler, ts = trajectory_calc(x_o, v_o, h, N, "Symplectic")

    #determine the analytic solution
    x_analytic =      x_o*np.cos(ts) + v_o*np.sin(ts)
    v_analytic = (-1)*x_o*np.sin(ts) + v_o*np.cos(ts)

    #select the last 10%
    x_euler = x_euler[(9*N/10):]
    x_analytic = x_analytic[(9*N/10):]
    ts = ts[(9*N/10):]

    #plot vs t
    fig, ax = plt.subplots(nrows=1, ncols=1)
    ax.plot(ts, x_euler, label="Symplectic Euler")
    ax.plot(ts, x_analytic, label="Analytic")
    ax.set_xlabel("Time")
    ax.set_title("Symplectic Solution Phase Lag")
    ax.legend()
    fig.savefig(filename)
    plt.close(fig)
```

Assignment Script

```
import euler

#Directory for images
path = "images/"

#Q1
euler.trajectory_plot(0,1,0.01,10000,"Explicit",path+"Sec1Plt1.pdf")

#Q2
euler.error_plot(0,1,0.01,10000,"Explicit",path+"Sec1Plt2.pdf")

#Q3
euler.trunc_err(0,1,0.01,1000,"Explicit",path+"Sec1Plt3.pdf")

#Q4
euler.error_plot(0,1,0.01,10000,"Explicit",path+"Sec1Plt4.pdf",E_plot=True)

#Q5
euler.error_plot(0,1,0.01,10000,"Implicit",path+"Sec1Plt5.pdf",E_plot=True)

#Q6
euler.phase_space(0,1,0.01,1000,["Implicit","Explicit"],path+"Sec2Plt1.pdf")

#Q7
euler.phase_space(0,1,0.2,500,["Symplectic"],path+"Sec2Plt2.pdf")

#Q8
euler.error_plot(0,1,0.01,10000,"Symplectic",path+"Sec2Plt3.pdf",E_plot=True)

#Q9
euler.phase_lag(0,1,0.2,500,path+"Sec2Plt4.pdf")
```