Using Makefiles and Version Control to Build Assignment

Daniel Wendt

February 25, 2019

1 Makefiles and Version Control

I used the makefile "ph20makefile" to build this pdf document, calling the code from the previous assignment to create the required plots and using the pdflatex command. The makefile also access and saves the git log recorded in this document. The text of this makefile is included below:

ph20makefile

```
wendt-assignment3.pdf : wendt-assignment3.tex outs textgitlog.txt
        pdflatex $<

textgitlog.txt :
        git log --oneline > textgitlog.txt

.PHONY : outs
outs : eulerexplicit.pdf eulerexpliciterror.pdf eulerexplicitenergy.pdf
   eulerimplicit.pdf eulerimpliciterror.pdf eulerimplicitenergy.pdf
   eulerphasespace.pdf eulerphasesymp.pdf eulersymplecticenergy.pdf

eulerexplicit.pdf eulerexpliciterror.pdf eulerexplicitenergy.pdf
   eulerimplicit.pdf eulerimpliciterror.pdf eulerimplicitenergy.pdf
   eulerphasespace.pdf eulerphasesymp.pdf eulersymplecticenergy.pdf :
   ph20numint2.py
        python $^

.PHONY : clean
clean :
        rm -f *.pdf
```

Additionally, git was used to version control this assignment during the process of creating a makefile and modifying the code from the previous assignment. The version control log shows the process of changes, including my detour into making the code take command line input, and then deciding it was too hard to figure out how to plot multiple plots. Fortunately, the progress I made on this is saved using version control, so I may return to it someday.

git log

```
3fe8824 Revert python file to original, edit makefile to make plots for
   write-up and build pdf of write-up.
aa13913 Add TeX file for original Ph20 assignment
```

```
ff182ab Makefile builds LaTeX file into pdf, includes plot generated by
   python script.
c5faf5a Test LaTeX installation, work on assignment makefile and modifying
    python script to be useable by makefile.
5541cea Ignore notebook checkpoints file, add makefile function to call
   python program to create plot.
9c4ab86 Modify ph20 python assignment to take command line arguments
dfb95cb Create .py version of ph20numint2 for use in makefile project
50bc04f Add comment to test git version control
33a986c Add assignment 3 on numerical integration from ph20 to folder
```

The python source code used is represented below. This program defines functions to create plots of different types of simulations of a harmonic oscillator, which are then called to plot and save the figures used in the second part of this assignment.

ph20numint2.py

```
import sys
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

#mass on a spring: d2x/dt2=-k/m x

def euler(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=v[i]-h*x[i]
        x[i+1]=x[i]+h*v[i]
    fig=plt.plot(t,x,label='Position (x)')
    fig=plt.plot(t,v,label='Velocity (v)')

def eulererror(x0,v0,n,h,ax):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=v[i]-h*x[i]
        x[i+1]=x[i]+h*v[i]
    xerr=np.cos(t)-x
    verr=-np.sin(t)-v

    ax[0].plot(t,xerr,label='h = '+str(h))
    ax[1].plot(t,verr,label='h = '+str(h))
```

```
def eulerenergy(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=v[i]-h*x[i]
        x[i+1]=x[i]+h*v[i]
    E = x*x+v*v

    plt.plot(t,E)

def eulerimplicit(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=(v[i]-h*x[i])/(1+h*h)
        x[i+1]=(x[i]+h*v[i])/(1+h*h)
    fig=plt.plot(t,x,label='Position (x)')
    fig=plt.plot(t,v,label='Velocity (v)')

def eulererrorimp(x0,v0,n,h,ax):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=(v[i]-h*x[i])/(1+h*h)
        x[i+1]=(x[i]+h*v[i])/(1+h*h)
    xerr=np.cos(t)-x
    verr=-np.sin(t)-v

    ax[0].plot(t,xerr,label='h = '+str(h))
    ax[1].plot(t,verr,label='h = '+str(h))

def eulerenergyimp(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
```

```
        t[i+1]=h*(i+1)
        v[i+1]=(v[i]-h*x[i])/(1+h*h)
        x[i+1]=(x[i]+h*v[i])/(1+h*h)
    E = x*x+v*v

    plt.plot(t,E)

def eulerphaseexp(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=v[i]-h*x[i]
        x[i+1]=x[i]+h*v[i]
    fig=plt.plot(x,v,label='Explicit')
    fig=plt.plot(np.cos(t),-np.sin(t),label='Analytic')

def eulerphaseimp(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        v[i+1]=(v[i]-h*x[i])/(1+h*h)
        x[i+1]=(x[i]+h*v[i])/(1+h*h)
    fig=plt.plot(x,v,label='Implicit')

def eulerphasesymp(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
        t[i+1]=h*(i+1)
        x[i+1]=x[i]+h*v[i]
        v[i+1]=v[i]-h*x[i+1]
    fig=plt.plot(x,v,label='Symplectic')

def eulerenergysymp(x0,v0,n,h,fig):
    t = np.zeros(n)
    x = np.zeros(n)
    v = np.zeros(n)
    x[0] = x0
    v[0] = v0
    for i in range (n-1):
```

```
        t[i+1]=h*(i+1)
        x[i+1]=x[i]+h*v[i]
        v[i+1]=v[i]-h*x[i+1]
    E = x*x+v*v
    fig=plt.plot(t,E,label='Symplectic')

x0 = 1 #m
v0 = 0 #m/s
k = 1 #N/m
m = 1 #kg

f=plt.figure()
plt.xlabel('Time (s)')
plt.ylabel('Position and Velocity')
plt.title('Explicit Euler approximation of spring motion')
euler(x0,v0,4000,0.01,f)
plt.legend(loc=4)
f.savefig('eulerexplicit.pdf',bbox_inches='tight')

g, axg = plt.subplots(2, constrained_layout=True)

h0=0.01
N0=2000

eulererror(x0,v0,N0,h0,axg)
eulererror(x0,v0,2*N0,h0/2,axg)
eulererror(x0,v0,4*N0,h0/4,axg)
eulererror(x0,v0,8*N0,h0/8,axg)
eulererror(x0,v0,16*N0,h0/16,axg)

axg[0].set_xlabel('Time (s)')
axg[1].set_xlabel('Time (s)')
axg[0].set_ylabel('Error in x (m)')
axg[1].set_ylabel('Error in v (m/s)')
axg[0].set_title('Error in simulated position (explicit)')
axg[1].set_title('Error in simulated velocity (explicit)')
axg[0].legend(loc=4)
axg[1].legend(loc=4)
g.savefig('eulerexpliciterror.pdf',bbox_inches='tight')

j = plt.figure()
plt.xlabel('Time (s)')
plt.ylabel('Energy (x^2+y^2)')
plt.title('Normalized energy of simulated spring (explicit)')
eulerenergy(x0,v0,N0,h0,j)
j.savefig('eulerexplicitenergy.pdf',bbox_inches='tight')

k=plt.figure()
plt.xlabel('Time (s)')
plt.ylabel('Position and Velocity')
plt.title('Implicit Euler approximation of spring motion')
```

```
eulerimplicit(x0,v0,4000,0.01,k)
plt.legend(loc=4)
k.savefig('eulerimplicit.pdf',bbox_inches='tight')

l, axl = plt.subplots(2, constrained_layout=True)

eulererrorimp(x0,v0,N0,h0,axl)
eulererrorimp(x0,v0,2*N0,h0/2,axl)
eulererrorimp(x0,v0,4*N0,h0/4,axl)
eulererrorimp(x0,v0,8*N0,h0/8,axl)
eulererrorimp(x0,v0,16*N0,h0/16,axl)

axl[0].set_xlabel('Time (s)')
axl[1].set_xlabel('Time (s)')
axl[0].set_ylabel('Error in x (m)')
axl[1].set_ylabel('Error in v (m/s)')
axl[0].set_title('Error in simulated position (implicit)')
axl[1].set_title('Error in simulated velocity (implicit)')
axl[0].legend(loc=4)
axl[1].legend(loc=4)
l.savefig('eulerimpliciterror.pdf',bbox_inches='tight')

m = plt.figure()
plt.xlabel('Time (s)')
plt.ylabel('Energy (x^2+y^2)')
plt.title('Normalized energy of simulated spring (implicit)')
eulerenergyimp(x0,v0,N0,h0,m)
m.savefig('eulerimplicitenergy.pdf',bbox_inches='tight')

n=plt.figure()

plt.xlabel('Position (m)')
plt.ylabel('Velocity (m/s)')
plt.title('Euler approximations in phase space')

eulerphaseexp(x0,v0,N0,h0,n)
eulerphaseimp(x0,v0,N0,h0,n)
plt.legend(loc=4)
n.savefig('eulerphasespace.pdf',bbox_inches='tight')

o=plt.figure()
plt.xlabel('Position (m)')
plt.ylabel('Velocity (m/s)')
plt.title('Symplectic deviation in phase space')

eulerphaseexp(x0,v0,50,20*h0,o)
#eulerphasesymp(x0,v0,2000,h0,o)
#eulerphasesymp(x0,v0,400,5*h0,o)
eulerphasesymp(x0,v0,1000,20*h0,o)
plt.legend(loc=4)
o.savefig('eulerphasesymp.pdf',bbox_inches='tight')
```

```
p=plt.figure()
plt.xlabel('Time (s)')
plt.ylabel('Energy (x^2+y^2)')
plt.title('Normalized energy of simulated spring (symplectic)')
eulerenergysymp(x0,v0,400,5*h0,p)
p.savefig('eulersymplecticenergy.pdf',bbox_inches='tight')
```

2 Numerical Model of Simple Harmonic Oscillator

Numerical methods can be used to integrate and model systems involving differential equations, such as a simple harmonic oscillator like a mass on a spring. This system is described by Newton's second law and the spring equation:

$$F = m\frac{d^2x}{dt^2} = -kx.$$

Using the explicit form of Euler's method of integrating differential equations, for a spring with mass and spring constant chosen so that $k = m$, we can derive

$$x_{i+1} = x_i + hv_i, \quad v_{i+1} = v_i - hx_i.$$

I wrote a Python function to find the position and velocity of a spring over time given the input initial position and velocity, the number of points N at which to find the value, and the interval h between points. Figure 1 shows the simulated approximation of position and velocity over time calculated for $x_0 = 1, v_0 = 0, N = 4000$, and $h = 0.01$. This shows several cycles, demonstrating that the error inherent to this approximation causes the calculated amplitude of the oscillation to increase over time.

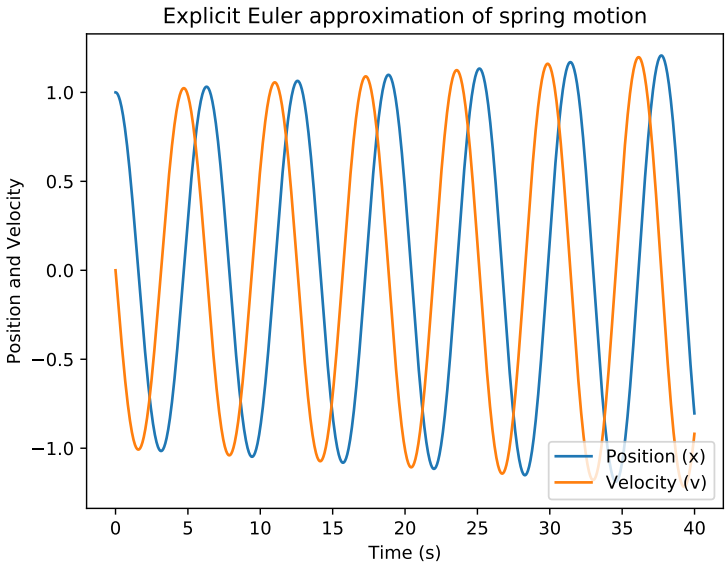

Figure 1: A plot showing position and velocity over time for an explicit Euler method simulation of a harmonic oscillator. N and h have been chosen to show several cycles, with the amplitude increasing by a notable amount each cycle.

The analytic solution to the system with these initial conditions is

$$x = \cos(t), v = -\sin(t).$$

The error of this simulation can be analyzed by subtracting the analytic solution from the calculated solution. Figure 2 shows the global error for several different values of h up to the same final time, demonstrating that the error as a function of time is proportional to h. As h increases by a factor of two, the error in x and v increases by a factor of two, with the error equaling zero when $x = 0$ or $v = 0$.

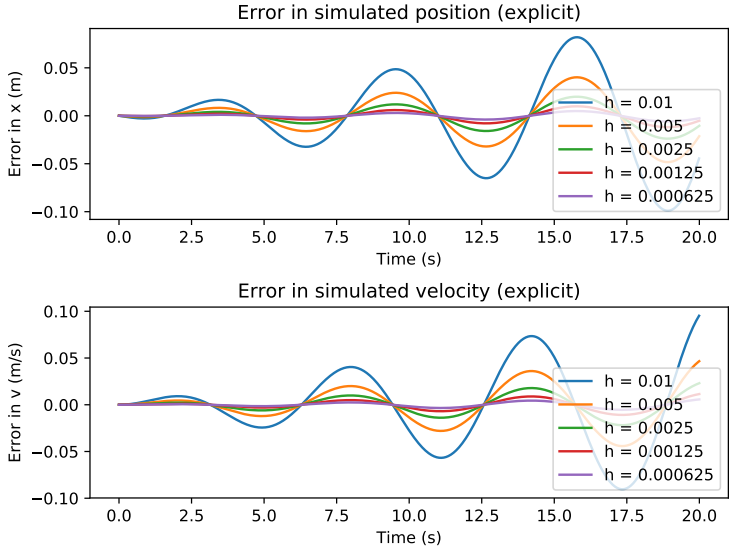

Figure 2: A plot showing global error over time for position and velocity using the explicit Euler method of simulating a simple harmonic oscillator. The error is plotted for several different values of h, each increasing by a factor of 2.

Because of the error in this simulation, the energy of the simulated system is not conserved. We can calculate the normalized total energy $E = x^2 + v^2$ over time, which increases over time for the explicit Euler method as shown in Figure 3.

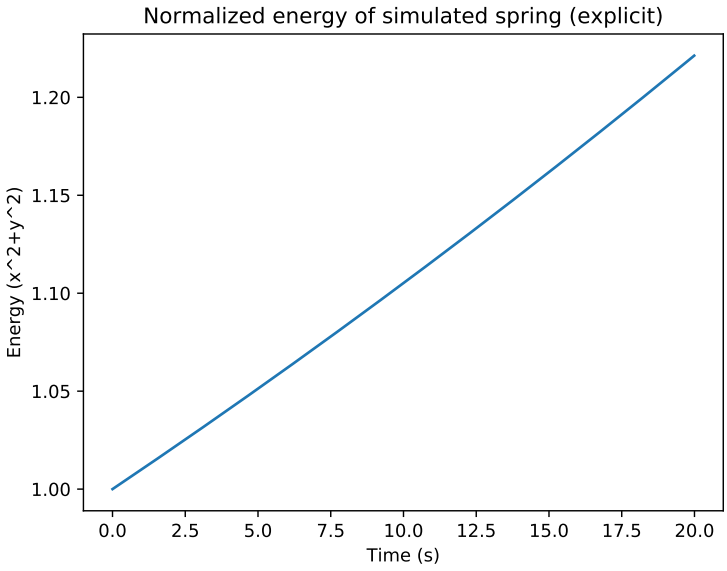

Figure 3: A plot showing normalized total energy over time for a harmonic oscillator simulated with the explicit Euler method. Over time, the energy increases approximately linearly, violating the conservation of energy expected for a real physical system.

The implicit Euler method uses the new position and velocity to compute the new position and velocity, rather than the old position and velocity. This gives the equations

$$x_{i+1} = x_i + hv_{i+1}, \quad v_{i+1} = v_i - hx_{i+1},$$

which can be solved to find

$$x_{i+1} = \frac{x_i + hv_i}{1+h^2}, \quad v_{i+1} = \frac{v_i - hx_i}{1+h^2}.$$

Using these equations, I simulated the same system with $x_0 = 1, v_0 = 0, N = 4000$, and $h = 0.01$, plotted in Figure 4.

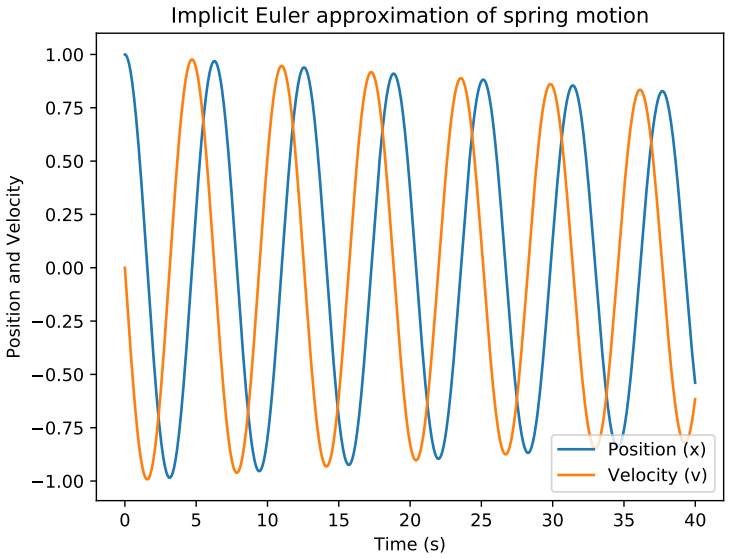

Figure 4: A plot showing position and velocity over time for an explicit Euler method simulation of a harmonic oscillator. N and h have been chosen to show several cycles, with the amplitude decreasing by a notable amount each cycle, and the parameters the same as in Figure 1.

I also plot the error and energy for the implicit Euler method, again using the same parameters as for the explicit method. These are shown in Figure 5 and Figure 6 respectively. The error is still proportional to h, taking the same form as before with roughly the same amplitude over time for the same h, and the energy decreases approximately linearly over time.

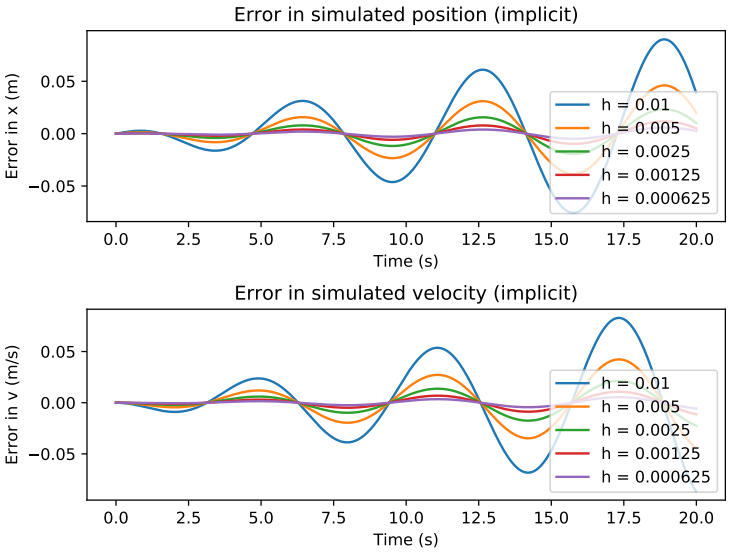

Figure 5: A plot showing global error over time for position and velocity using the implicit Euler method of simulating a simple harmonic oscillator. The error is plotted for several different values of h, each increasing by a factor of 2.

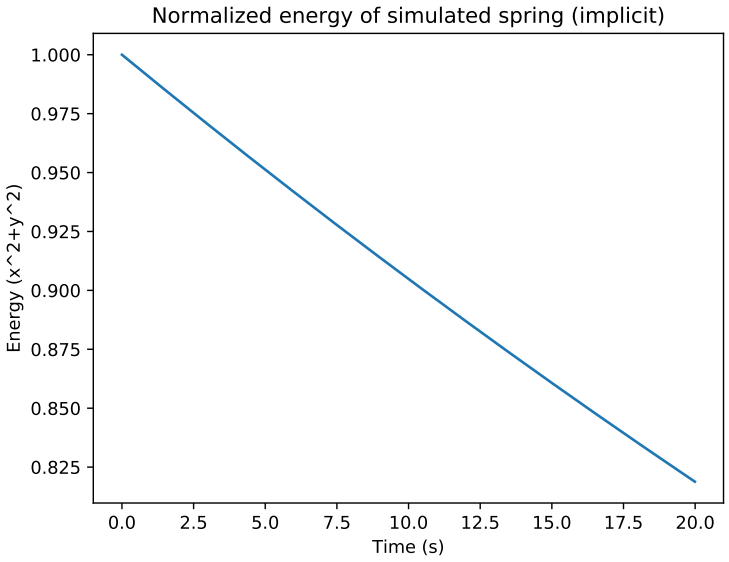

Figure 6: A plot showing normalized total energy over time for a harmonic oscillator simulated with the implicit Euler method. Over time, the energy decreases approximately linearly, similarly violating the conservation of energy expected for a real physical system, although in the opposite direction of the explicit method.

Phase space, as seen in Assignment 1, is a useful tool for analyzing the phase of oscillating systems. I have also plotted the results from the implicit and explicit Euler approximations in phase space, plotting v against x. Figure 7 shows these plots and the expected analytic solution. These accumulate no phase error, with the increasing or decreasing amplitude causing a circular spiral outward or inward.

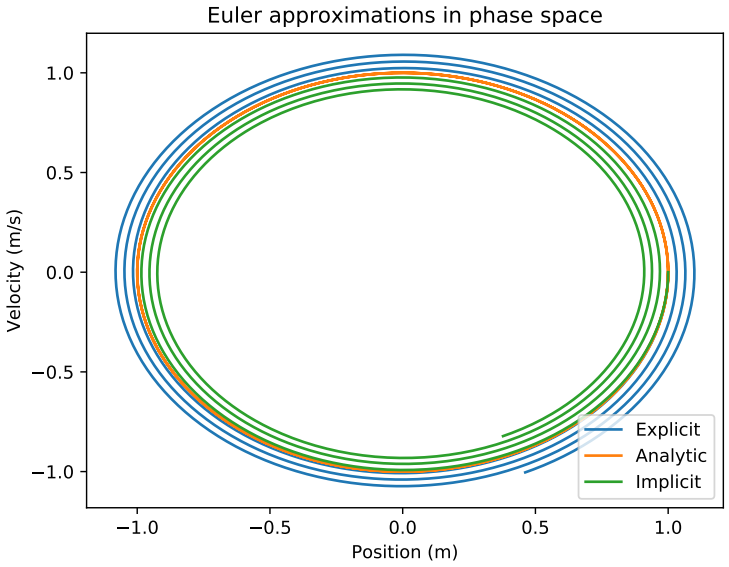

Figure 7: A plot showing the phase-space geometry of the explicit and implicit Euler approximations, and the expected analytic solution. Each approximation has x plotted against v. The explicit and implicit approximations spiral without changing phase as their amplitude increases or decreases over time.

Another method of approximating, the symplectic Euler method, conserves energy on the average much more effectively. This integrator uses the equations

$$x_{i+1} = x_i + hv_i, \quad v_{i+1} = v_i - hx_{i+1}.$$

Figure 8 shows this approximation in phase-space, for $h = 0.2$, compared to the explicit Euler approximation with the same h and the analytic solution. The symplectic approximation stays much closer to the analytic solution, but x and v become slightly out of phase with one another, as shown by the deformation of the symplectic ellipse from the analytic circle.

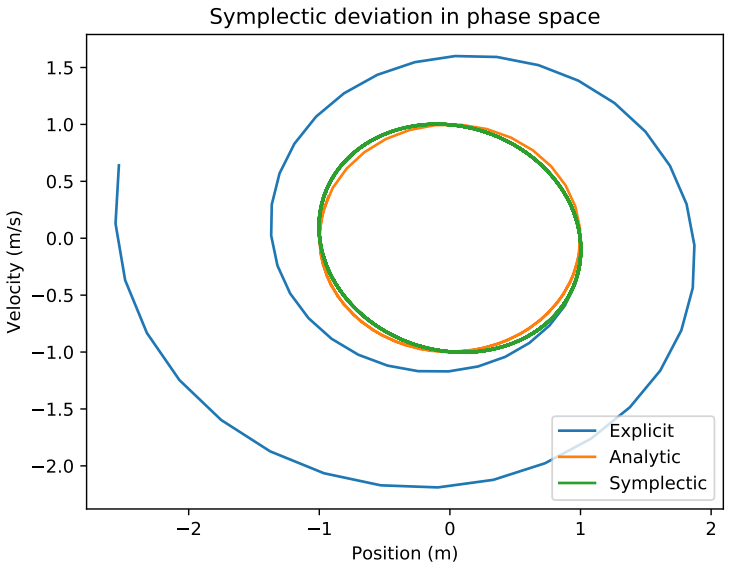

Figure 8: A plot showing the phase-space geometry of the symplectic and explicit Euler approximations, and the expected analytic solution. Each approximation has x plotted against v. While the explicit solution spirals outward, the symplectic solution remains close to the analytic solution, but slightly out of phase.

The error in the phase of this approximation shows up in the average energy calculated by the symplectic Euler method. The normalized total energy, calculated like with the previous approximations, is plotted in Figure 9. Unlike the explicit and implicit Euler methods, the symplectic energy is sinusoidal, oscillating around the expected central value of 1. This corresponds to the phase-space diagram, where the symplectic approximation deviated slightly from the circle of $x^2 + v^2 = E$, repeatedly cycling between lower and higher.

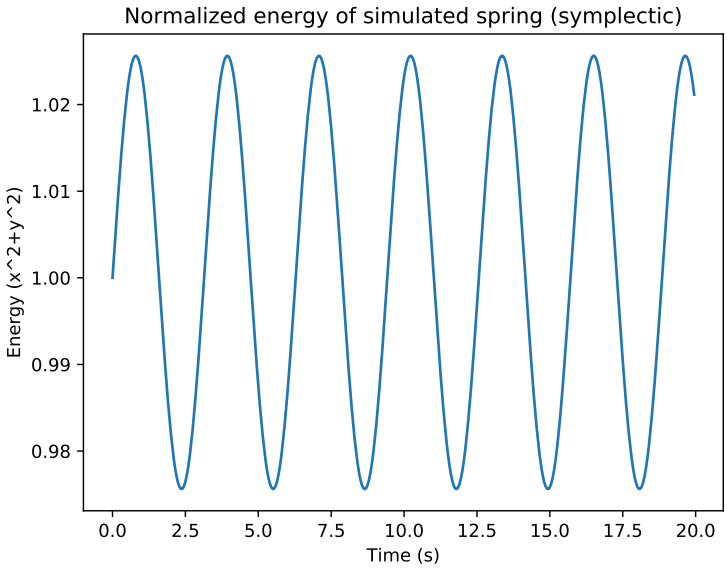

Figure 9: A plot showing the phase-space geometry of the explicit and implicit Euler approximations, and the expected analytic solution. Each approximation has x plotted against v. The explicit and implicit approximations spiral without changing phase as their amplitude increases or decreases over time.

Since the average energy is conserved, this makes the symplectic method more attractive for approximating this physical system. However, this method does introduce the phase error, so it is not perfect.